



User Guide

Amazon CloudWatch



Amazon CloudWatch: User Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon CloudWatch?	1
Accessing CloudWatch	1
Related AWS services	1
How CloudWatch works	2
Concepts	3
Namespaces	4
Metrics	4
Dimensions	6
Resolution	7
Statistics	8
Units	8
Periods	8
Aggregation	9
Percentiles	10
Alarms	11
Billing and costs	12
Resources	12
Getting set up	14
Sign up for an AWS account	14
Create a user with administrative access	14
Sign in to the Amazon CloudWatch console	16
Set up the AWS CLI	16
Getting started	17
See the pre-built cross-service dashboard	22
Remove a service from appearing in the cross-service dashboard	24
See a pre-built dashboard for a single AWS service	24
See a pre-built dashboard for a resource group	26
CloudWatch billing and cost	28
Analyze CloudWatch cost and usage data with Cost Explorer	28
To visualize and analyze CloudWatch cost and usage data	28
Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena	32
To analyze cost and usage data with AWS Cost and Usage Reports and Athena	33
Best practices for optimizing and reducing costs	36
CloudWatch metrics	36

CloudWatch alarms	44
CloudWatch Logs	47
Dashboards	51
Create a dashboard	52
CloudWatch cross-account observability dashboard	53
Cross-account cross-Region dashboards	54
Creating and using a cross-account cross-Region dashboard with the AWS Management Console	55
Create a cross-account cross-Region dashboard programmatically	56
Create flexible dashboards with dashboard variables	59
Types of dashboard variables	59
Tutorial: Create a Lambda dashboard with function name as the variable	60
Tutorial: Create a dashboard that uses a regular expression pattern to switch between Regions	62
Copy a variable to another dashboard	63
Create and work with widgets on CloudWatch dashboards	64
Add or remove a graph	64
Graph metrics manually on a CloudWatch dashboard	67
Edit a graph	68
Add an explorer widget to a CloudWatch dashboard	77
Add or remove a line widget	79
Add or remove a number widget	79
Add or remove a gauge widget	81
Add a custom widget to a CloudWatch dashboard	82
Add or remove a text widget	93
Add or remove an alarm widget	94
Add or remove a table widget	96
Link and unlink graphs	99
Sharing dashboards	100
Permissions required to share a dashboard	101
Permissions that are granted to people who you share the dashboard with	103
Share a single dashboard with specific users	104
Share a single dashboard publicly	105
Share all CloudWatch dashboards in the account by using SSO	106
Set up SSO for CloudWatch dashboard sharing	106
See how many of your dashboards are shared	107

See which of your dashboards are shared	108
Stop sharing one or more dashboards	108
Review shared dashboard permissions and change permission scope	109
Allowing people that you share with to see composite alarms	111
Allowing people that you share with to see logs table widgets	111
Allowing people that you share with to see custom widgets	113
Use live data	114
Viewing an animated dashboard	115
Add a dashboard to your favorites list	116
Change the period override setting or refresh interval	117
Change the time range or time zone format	118
Metrics	121
Basic monitoring and detailed monitoring	121
Query your metrics with CloudWatch Metrics Insights	124
Build your queries	125
Query components and syntax	126
Create alarms on Metrics Insights queries	135
Use Metrics Insights queries with metric math	140
Use natural language to generate and update CloudWatch Metrics Insights queries	141
SQL inference	144
Sample queries	145
Metrics Insights limits	153
Metrics Insights glossary	154
Troubleshooting Metrics Insights	154
Use metrics explorer to monitor resources by their tags and properties	155
CloudWatch agent configuration for metrics explorer	158
Use metric streams	158
Set up a metric stream	160
Statistics that can be streamed	171
Metric stream operation and maintenance	172
Monitor your metric streams with CloudWatch metrics	173
Trust between CloudWatch and Firehose	175
Metric streams output formats	176
Troubleshooting	205
View available metrics	206
Search for available metrics	209

Graphing metrics	211
Graph a metric	211
Merge two graphs into one	217
Use dynamic labels	218
Modify the time range or time zone format for a graph	221
Zooming in on a graph	223
Modify the y-axis for a graph	225
Create an alarm from a metric on a graph	226
Using anomaly detection	228
How anomaly detection works	230
Anomaly detection on metric math	231
Use metric math	232
Add a math expression to a CloudWatch graph	232
Metric math syntax and functions	233
Using IF expressions	273
Anomaly detection on metric math	277
Use search expressions in graphs	278
Search expression syntax	278
Search expression examples	285
Creating a graph with a search expression	288
Get statistics for a metric	290
CloudWatch statistics definitions	290
Get statistics for a specific resource	295
Aggregate statistics across resources	299
Aggregate statistics by Auto Scaling group	302
Aggregating statistics by AMI	304
Publish custom metrics	306
High-resolution metrics	306
Use dimensions	307
Publish single data points	308
Publish statistic sets	309
Publish the value zero	310
Stop publishing metrics	310
Alarms	311
Metric alarm states	312
Evaluating an alarm	312

Alarm actions	314
Lambda alarm actions	315
Configuring how alarms treat missing data	319
How alarm state is evaluated when data is missing	321
High-resolution alarms	324
Alarms on math expressions	325
Percentile-based alarms and low data samples	325
Common features of CloudWatch alarms	325
Alarm recommendations for AWS services	327
Find and create recommended alarms	327
Recommended alarms	329
Alarming on metrics	422
Create an alarm based on a static threshold	422
Create an alarm based on a metric math expression	424
Create an alarm based on a Metrics Insights query	427
Create an alarm based on a connected data source	428
Create an alarm based on anomaly detection	431
Modifying an anomaly detection model	435
Deleting an anomaly detection model	436
Alarming on logs	436
Create an alarm based on a log group-metric filter	437
Combining alarms	439
Create a composite alarm	441
Suppressing composite alarm actions	444
Acting on alarm changes	452
Notifying users on alarm changes	453
Alarm events and EventBridge	459
Managing alarms	473
Editing or deleting a CloudWatch alarm	473
Hide Auto Scaling alarms	474
Alarm use cases and examples	475
Create a billing alarm	475
Create a CPU usage alarm	478
Create a load balancer latency alarm	481
Create a storage throughput alarm	483
Create an alarm on Performance Insights counter metrics from an AWS database	485

Create alarms to stop, terminate, reboot, or recover an EC2 instance	488
Alarms and tagging	496
Application monitoring (APM)	498
Application Signals	498
Permissions required for Application Signals	500
Enable Application Signals	508
Monitor your application's operational health	566
Standard application metrics collected	607
Service level objectives (SLOs)	611
SLO concepts	612
Create an SLO	614
View and triage SLO status	617
Edit an existing SLO	618
Delete an SLO	619
Synthetic monitoring (canaries)	620
Required roles and permissions	622
Creating a canary	637
Groups	741
Test a canary locally	742
Troubleshooting a failed canary	764
Sample code for canary scripts	773
Canaries and X-Ray tracing	779
Running a canary on a VPC	780
Encrypting canary artifacts	781
Viewing canary statistics and details	783
CloudWatch metrics published by canaries	785
Edit or delete a canary	788
Start, stop, delete, or update runtime for multiple canaries	790
Monitoring canary events with Amazon EventBridge	791
CloudWatch RUM	795
IAM policies to use CloudWatch RUM	798
Set up an application to use CloudWatch RUM	799
Configuring the CloudWatch RUM web client	809
Regionalization	810
Use page groups	811
Specify custom metadata	812

Send custom events	818
Viewing the CloudWatch RUM dashboard	821
CloudWatch metrics that you can collect with CloudWatch RUM	823
Data protection and data privacy with CloudWatch RUM	834
Information collected by the CloudWatch RUM web client	836
Manage your applications that use CloudWatch RUM	868
CloudWatch RUM quotas	870
Troubleshooting	870
Perform launches and A/B experiments with CloudWatch Evidently	871
IAM policies to use Evidently	872
Create projects, features, launches, and experiments	874
Manage features, launches, and experiments	895
Adding code to your application	900
Project data storage	902
How Evidently calculates results	905
View launch results in the dashboard	908
View experiment results in the dashboard	908
How CloudWatch Evidently collects and stores data	909
Using service-linked roles	910
CloudWatch Evidently quotas	913
Tutorial: A/B testing with the Evidently sample application	914
Network monitoring	925
Using Internet Monitor	925
Supported Regions	927
Pricing	929
Components	930
Internet weather map	933
How Internet Monitor works	933
Use cases	941
Internet Monitor cross-account observability	942
Getting started	943
Examples with the CLI	959
Internet Monitor dashboard	968
Explore data using tools	979
Create alarms	999
EventBridge integration	1000

Troubleshoot errors	1001
Data protection and data privacy	1002
Identity and Access Management	1002
Quotas	1014
Using Network Monitor	1015
Network Monitor Key features	1015
Terminology and components	1016
Limitations and requirements	1016
How Network Monitor works	1016
Region availability	1019
Creating a monitor	1021
Working with monitors and probes	1026
Network Monitor dashboards	1034
Quotas	1040
Security	1041
Identify and Access management	1043
Pricing	1062
Infrastructure monitoring	1064
Container Insights	1064
Container Insights with enhanced observability for Amazon EKS	1065
Supported platforms	1066
CloudWatch agent container image	1066
Supported Regions	1067
Setting up Container Insights	1069
Viewing Container Insights metrics	1127
Metrics collected by Container Insights	1131
Performance log reference	1232
Container Insights Prometheus metrics monitoring	1267
Integration with Application Insights	1396
See Amazon ECS lifecycle events within Container Insights	1396
Troubleshooting Container Insights	1398
Building your own CloudWatch agent Docker image	1402
Deploying other CloudWatch agent features in your containers	1402
Lambda Insights	1403
Get started with Lambda Insights	1403
View your Lambda Insights metrics	1471

Integration with Application Insights	1472
Metrics collected by Lambda Insights	1472
Troubleshooting and known issues	1476
Example telemetry event	1477
Use Contributor Insights to analyze high-cardinality data	1479
Create a Contributor Insights rule	1480
Contributor Insights rule syntax	1485
Example rules	1490
Viewing Contributor Insights reports	1494
Graphing metrics generated by rules	1495
Using Contributor Insights built-in rules	1498
Detect common application problems with CloudWatch Application Insights	1498
What is Amazon CloudWatch Application Insights?	1499
How Application Insights works	1509
Get started	1525
Application Insights cross-account observability	1558
Work with component configurations	1558
Use CloudFormation templates	1630
Tutorial: Set up monitoring for SAP ASE	1643
Tutorial: Set up monitoring for SAP HANA	1652
Tutorial: Set up monitoring for SAP NetWeaver	1668
View and troubleshoot Application Insights	1686
Supported logs and metrics	1690
Using the resource health view	1787
Prerequisites	1787
Monitor across accounts and Regions	1790
CloudWatch cross-account observability	1793
Link monitoring accounts with source accounts	1796
Manage monitoring accounts and source accounts	1807
Cross-account cross-Region CloudWatch console	1810
Enabling cross-account cross-Region functionality	1810
(Optional) Integrate with AWS Organizations	1814
Troubleshooting	1815
Disabling and cleaning up after using cross-account	1816
Query metrics from other data sources	1817
Managing access to data sources	1818

Connect to a prebuilt data source with a wizard	1819
Amazon Managed Service for Prometheus	1820
Amazon OpenSearch Service	1821
Amazon RDS for PostgreSQL and Amazon RDS for MySQL	1821
Amazon S3 CSV files	1823
Microsoft Azure Monitor	1824
Prometheus	1825
Notification of Available Updates	1826
Create a custom connector to a data source	1826
Use a template	1827
Create a custom data source from scratch	1828
Use your custom data source	1834
How to pass arguments to your Lambda function	1835
Delete a connector to a data source	1835
Collect metrics, logs, and traces with the CloudWatch agent	1837
Installing the CloudWatch agent	1840
Installing the CloudWatch agent using the command line	1840
Install the CloudWatch agent using Systems Manager	1863
Installing the CloudWatch agent on new instances using AWS CloudFormation	1882
CloudWatch agent credentials preference	1888
Verifying the signature of the CloudWatch agent package	1890
Create the CloudWatch agent configuration file	1899
Create the CloudWatch agent configuration file with the wizard	1900
Manually create or edit the CloudWatch agent configuration file	1907
Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart	2007
Option 1: Install with IAM permissions on worker nodes	2008
Option 2: Install using IAM service account role (applies only to using the add-on)	2010
(Optional) Additional configuration	2012
Troubleshooting	2021
Metrics collected by the CloudWatch agent	2023
Metrics collected by the CloudWatch agent on Windows Server instances	2023
Metrics collected by the CloudWatch agent on Linux and macOS instances	2024
Memory metric definitions	2039
Common scenarios with the CloudWatch agent	2042
Running the CloudWatch agent as a different user	2042

How the CloudWatch agent handles sparse log files	2044
Adding custom dimensions to metrics collected by the CloudWatch agent	2045
Multiple CloudWatch agent configuration files	2046
Aggregating or rolling up metrics collected by the CloudWatch agent	2048
Collecting high-resolution metrics with the CloudWatch agent	2049
Sending metrics, logs, and traces to a different account	2050
Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent	2052
Troubleshooting the CloudWatch agent	2053
CloudWatch agent command line parameters	2054
Installing the CloudWatch agent using Run Command fails	2054
The CloudWatch agent won't start	2054
Verify that the CloudWatch agent is running	2054
The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region	2056
The CloudWatch agent won't start on Windows Server	2056
Where are the metrics?	2057
The CloudWatch agent takes a long time to run in a container or logs a hop limit error ..	2057
I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console	2058
CloudWatch agent files and locations	2058
Finding information about CloudWatch agent versions	2060
Logs generated by the CloudWatch agent	2061
Stopping and restarting the CloudWatch agent	2062
Embedding metrics within logs	2064
Publishing logs with the embedded metric format	2065
Using the client libraries	2065
Specification: Embedded metric format	2066
Using the PutLogEvents API to send manually-created embedded metric format logs	2075
Using the CloudWatch agent to send embedded metric format logs	2077
Using the embedded metric format with AWS Distro for OpenTelemetry	2084
Viewing your metrics and logs in the console	2084
Setting alarms on metrics created with the embedded metric format	2086
Services that publish CloudWatch metrics	2087
AWS usage metrics	2103
Visualizing your service quotas and setting alarms	2103
AWS API usage metrics	2105

CloudWatch usage metrics	2114
CloudWatch tutorials	2116
Scenario: Monitor estimated charges	2116
Step 1: Enable billing alerts	2117
Step 2: Create a billing alarm	2118
Step 3: Check the alarm status	2119
Step 4: Edit a billing alarm	2120
Step 5: Delete a billing alarm	2120
Scenario: Publish metrics	2121
Step 1: Define the data configuration	2121
Step 2: Add metrics to CloudWatch	2122
Step 3: Get statistics from CloudWatch	2123
Step 4: View graphs with the console	2123
Working with AWS SDKs	2125
Code examples	2127
Actions	2133
DeleteAlarms	2134
DeleteAnomalyDetector	2142
DeleteDashboards	2145
DescribeAlarmHistory	2148
DescribeAlarms	2153
DescribeAlarmsForMetric	2159
DescribeAnomalyDetectors	2171
DisableAlarmActions	2175
EnableAlarmActions	2186
GetDashboard	2195
GetMetricData	2197
GetMetricStatistics	2202
GetMetricWidgetImage	2212
ListDashboards	2216
ListMetrics	2219
PutAnomalyDetector	2234
PutDashboard	2237
PutMetricAlarm	2243
PutMetricData	2257
Scenarios	2272

Get started with alarms	2272
Get started with metrics, dashboards, and alarms	2274
Manage metrics and alarms	2350
Cross-service examples	2359
Monitor DynamoDB performance	2359
Security	2361
Data protection	2362
Encryption in transit	2362
Identity and access management	2363
Audience	2363
Authenticating with identities	2364
Managing access using policies	2367
How Amazon CloudWatch works with IAM	2370
Identity-based policy examples	2376
Troubleshooting	2381
CloudWatch dashboard permissions update	2383
AWS managed (predefined) policies for CloudWatch	2384
Customer managed policy examples	2416
Policy updates	2418
Using condition keys to limit access to CloudWatch namespaces	2440
Using condition keys to limit Contributor Insights users' access to log groups	2441
Using condition keys to limit alarm actions	2443
Using service-linked roles	2443
Using a service-linked role for CloudWatch RUM	2455
Using service-linked roles for Application Insights	2460
AWS managed policies for Application Insights	2472
Amazon CloudWatch permissions reference	2484
Compliance validation	2499
Resilience	2500
Infrastructure security	2500
Network isolation	2501
AWS Security Hub	2501
Interface VPC endpoints	2502
CloudWatch	2502
CloudWatch Synthetics	2504
Security considerations for Synthetics canaries	2506

Use secure connections	2507
Canary naming considerations	2507
Secrets and sensitive information in canary code	2507
Permissions considerations	2507
Stack traces and exception messages	2508
Scope your IAM roles narrowly	2508
Sensitive data redaction	2509
Logging API and console operations with AWS CloudTrail	2511
CloudWatch information in CloudTrail	2512
Example: CloudWatch log file entries	2513
CloudWatch data plane events in CloudTrail	2515
Query generation information in CloudTrail	2518
CloudWatch Internet Monitor in CloudTrail	2520
Example: CloudWatch Internet Monitor log file entries	2520
CloudWatch Synthetics information in CloudTrail	2522
Example: CloudWatch Synthetics log file entries	2523
Tagging your CloudWatch resources	2527
Supported resources in CloudWatch	2527
Managing tags	2528
Tag naming and usage conventions	2528
Grafana integration	2529
Service quotas	2530
Document history	2538

What is Amazon CloudWatch?

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications.

The CloudWatch home page automatically displays metrics about every AWS service you use. You can additionally create custom dashboards to display metrics about your custom applications, and display custom collections of metrics that you choose.

You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached. For example, you can monitor the CPU usage and disk reads and writes of your Amazon EC2 instances and then use that data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop under-used instances to save money.

With CloudWatch, you gain system-wide visibility into resource utilization, application performance, and operational health.

Accessing CloudWatch

You can access CloudWatch using any of the following methods:

- **Amazon CloudWatch console** – <https://console.aws.amazon.com/cloudwatch/>
- **AWS CLI** – For more information, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.
- **CloudWatch API** – For more information, see the [Amazon CloudWatch API Reference](#).
- **AWS SDKs** – For more information, see [Tools for Amazon Web Services](#).

Related AWS services

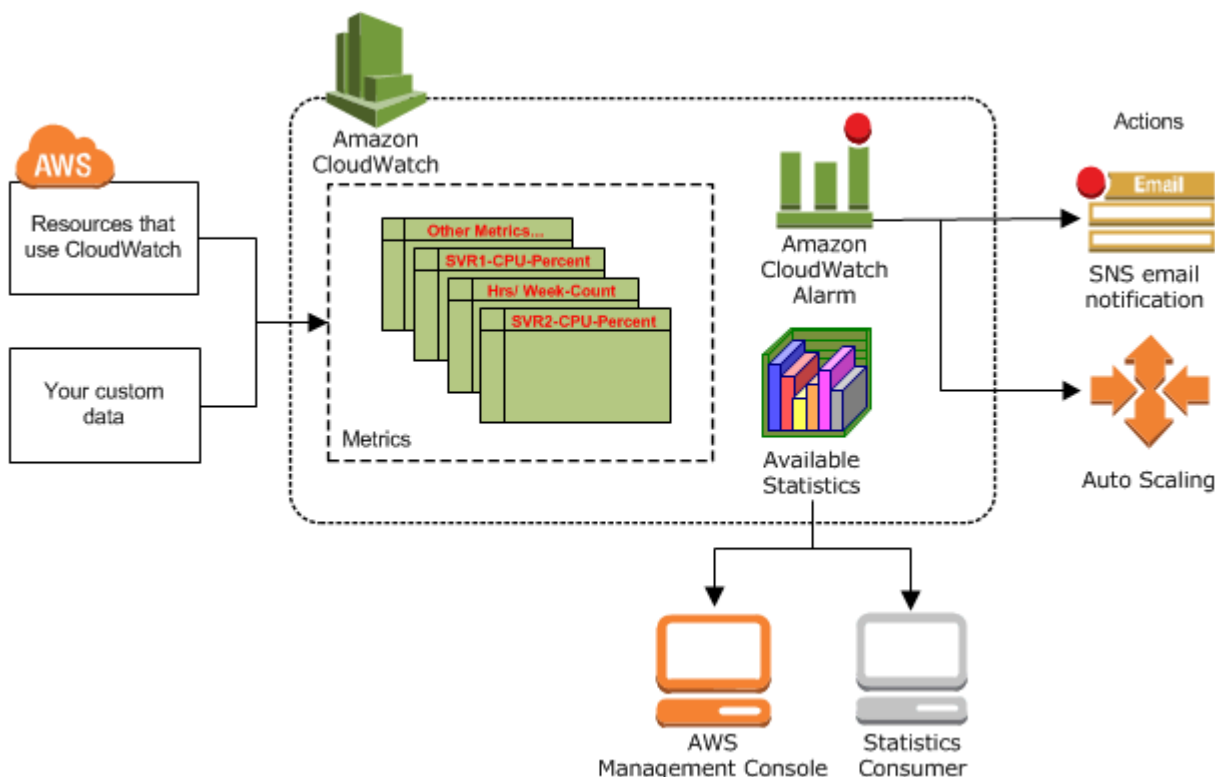
The following services are used along with Amazon CloudWatch:

- **Amazon Simple Notification Service (Amazon SNS)** coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. You use Amazon SNS with CloudWatch to send messages when an alarm threshold has been reached. For more information, see [Setting up Amazon SNS notifications](#).

- **Amazon EC2 Auto Scaling** enables you to automatically launch or terminate Amazon EC2 instances based on user-defined policies, health status checks, and schedules. You can use a CloudWatch alarm with Amazon EC2 Auto Scaling to scale your EC2 instances based on demand. For more information, see [Dynamic Scaling](#) in the *Amazon EC2 Auto Scaling User Guide*.
- **AWS CloudTrail** enables you to monitor the calls made to the Amazon CloudWatch API for your account, including calls made by the AWS Management Console, AWS CLI, and other services. When CloudTrail logging is turned on, CloudWatch writes log files to the Amazon S3 bucket that you specified when you configured CloudTrail. For more information, see [Logging Amazon CloudWatch API and console operations with AWS CloudTrail](#).
- **AWS Identity and Access Management (IAM)** is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). For more information, see [Identity and access management for Amazon CloudWatch](#).

How Amazon CloudWatch works

Amazon CloudWatch is basically a metrics repository. An AWS service—such as Amazon EC2—puts metrics into the repository, and you retrieve statistics based on those metrics. If you put your own custom metrics into the repository, you can retrieve statistics on these metrics as well.



You can use metrics to calculate statistics and then present the data graphically in the CloudWatch console. For more information about the other AWS resources that generate and send metrics to CloudWatch, see [AWS services that publish CloudWatch metrics](#).

You can configure alarm actions to stop, start, or terminate an Amazon EC2 instance when certain criteria are met. In addition, you can create alarms that initiate Amazon EC2 Auto Scaling and Amazon Simple Notification Service (Amazon SNS) actions on your behalf. For more information about creating CloudWatch alarms, see [Alarms](#).

AWS Cloud computing resources are housed in highly available data center facilities. To provide additional scalability and reliability, each data center facility is located in a specific geographical area, known as a *Region*. Each Region is designed to be completely isolated from the other Regions, to achieve the greatest possible failure isolation and stability. Metrics are stored separately in Regions, but you can use CloudWatch cross-Region functionality to aggregate statistics from different Regions. For more information, see [Cross-account cross-Region CloudWatch console](#) and [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Amazon CloudWatch concepts

The following terminology and concepts are central to your understanding and use of Amazon CloudWatch:

- [Namespaces](#)
- [Metrics](#)
- [Dimensions](#)
- [Resolution](#)
- [Statistics](#)
- [Percentiles](#)
- [Alarms](#)

For information about the service quotas for CloudWatch metrics, alarms, API requests, and alarm email notifications, see [CloudWatch service quotas](#).

Namespaces

A *namespace* is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

There is no default namespace. You must specify a namespace for each data point you publish to CloudWatch. You can specify a namespace name when you create a metric. These names must contain valid ASCII characters, and be 255 or fewer characters. Possible characters are: alphanumeric characters (0-9A-Za-z), period (.), hyphen (-), underscore (_), forward slash (/), hash (#), colon (:), and the space character. A namespace must contain at least one non-whitespace character.

The AWS namespaces typically use the following naming convention: `AWS/service`. For example, Amazon EC2 uses the `AWS/EC2` namespace. For the list of AWS namespaces, see [AWS services that publish CloudWatch metrics](#).

Metrics

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time. For example, the CPU usage of a particular EC2 instance is one metric provided by Amazon EC2. The data points themselves can come from any application or business activity from which you collect data.

By default, many AWS services provide metrics at no charge for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). For a charge, you can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. For custom metrics, you can add the data points in any order, and at any rate you choose. You can retrieve statistics about those data points as an ordered set of time-series data.

Metrics exist only in the Region in which they are created. Metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them. Data points older than 15 months expire on a rolling basis; as new data points come in, data older than 15 months is dropped.

Metrics are uniquely defined by a name, a namespace, and zero or more dimensions. Each data point in a metric has a time stamp, and (optionally) a unit of measure. You can retrieve statistics from CloudWatch for any metric.

For more information, see [View available metrics](#) and [Publish custom metrics](#).

Time stamps

Each metric data point must be associated with a time stamp. The time stamp can be up to two weeks in the past and up to two hours into the future. If you do not provide a time stamp, CloudWatch creates a time stamp for you based on the time the data point was received.

Time stamps are `dateTime` objects, with the complete date plus hours, minutes, and seconds (for example, 2016-10-31T23:59:59Z). For more information, see [dateTime](#). Although it is not required, we recommend that you use Coordinated Universal Time (UTC). When you retrieve statistics from CloudWatch, all times are in UTC.

CloudWatch alarms check metrics based on the current time in UTC. Custom metrics sent to CloudWatch with time stamps other than the current UTC time can cause alarms to display the **Insufficient Data** state or result in delayed alarms.

Metrics retention

CloudWatch retains metric data as follows:

- Data points with a period of less than 60 seconds are available for 3 hours. These data points are high-resolution custom metrics.
- Data points with a period of 60 seconds (1 minute) are available for 15 days
- Data points with a period of 300 seconds (5 minutes) are available for 63 days
- Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)

Data points that are initially published with a shorter period are aggregated together for long-term storage. For example, if you collect data using a period of 1 minute, the data remains available for 15 days with 1-minute resolution. After 15 days this data is still available, but is aggregated and is retrievable only with a resolution of 5 minutes. After 63 days, the data is further aggregated and is available with a resolution of 1 hour.

Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results

of a [list-metrics](#) command. The best way to retrieve these metrics is with the [get-metric-data](#) or [get-metric-statistics](#) commands in the AWS CLI.

Dimensions

A *dimension* is a name/value pair that is part of the identity of a metric. You can assign up to 30 dimensions to a metric.

Every metric has specific characteristics that describe it, and you can think of dimensions as categories for those characteristics. Dimensions help you design a structure for your statistics plan. Because dimensions are part of the unique identifier for a metric, whenever you add a unique name/value pair to one of your metrics, you are creating a new variation of that metric.

AWS services that send data to CloudWatch attach dimensions to each metric. You can use dimensions to filter the results that CloudWatch returns. For example, you can get statistics for a specific EC2 instance by specifying the InstanceId dimension when you search for metrics.

For metrics produced by certain AWS services, such as Amazon EC2, CloudWatch can aggregate data across dimensions. For example, if you search for metrics in the AWS/EC2 namespace but do not specify any dimensions, CloudWatch aggregates all data for the specified metric to create the statistic that you requested. CloudWatch does not aggregate across dimensions for your custom metrics.

Dimension combinations

CloudWatch treats each unique combination of dimensions as a separate metric, even if the metrics have the same metric name. You can only retrieve statistics using combinations of dimensions that you specifically published. When you retrieve statistics, specify the same values for the namespace, metric name, and dimension parameters that were used when the metrics were created. You can also specify the start and end times for CloudWatch to use for aggregation.

For example, suppose that you publish four distinct metrics named ServerStats in the DataCenterMetric namespace with the following properties:

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp:
2016-10-31T12:30:00Z, Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp:
2016-10-31T12:31:00Z, Value: 115
```

```
Dimensions: Server=Prod, Domain=Rio,          Unit: Count, Timestamp:
2016-10-31T12:32:00Z, Value: 95
Dimensions: Server=Beta, Domain=Rio,          Unit: Count, Timestamp:
2016-10-31T12:33:00Z, Value: 97
```

If you publish only those four metrics, you can retrieve statistics for these combinations of dimensions:

- Server=Prod, Domain=Frankfurt
- Server=Prod, Domain=Rio
- Server=Beta, Domain=Frankfurt
- Server=Beta, Domain=Rio

You can't retrieve statistics for the following dimensions or if you specify no dimensions. (The exception is by using the metric math **SEARCH** function, which can retrieve statistics for multiple metrics. For more information, see [Use search expressions in graphs.](#))

- Server=Prod
- Server=Beta
- Domain=Frankfurt
- Domain=Rio

Resolution

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling

PutMetricData more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Statistics

Statistics are metric data aggregations over specified periods of time. CloudWatch provides statistics based on the metric data points provided by your custom data or provided by other AWS services to CloudWatch. Aggregations are made using the namespace, metric name, dimensions, and the data point unit of measure, within the time period you specify.

For detailed definitions of the statistics supported by CloudWatch, see [CloudWatch statistics definitions](#).

Units

Each statistic has a unit of measure. Example units include Bytes, Seconds, Count, and Percent. For the complete list of the units that CloudWatch supports, see the [MetricDatum](#) data type in the *Amazon CloudWatch API Reference*.

You can specify a unit when you create a custom metric. If you do not specify a unit, CloudWatch uses None as the unit. Units help provide conceptual meaning to your data. Though CloudWatch attaches no significance to a unit internally, other applications can derive semantic information based on the unit.

Metric data points that specify a unit of measure are aggregated separately. When you get statistics without specifying a unit, CloudWatch aggregates all data points of the same unit together. If you have two otherwise identical metrics with different units, two separate data streams are returned, one for each unit.

Periods

A *period* is the length of time associated with a specific Amazon CloudWatch statistic. Each statistic represents an aggregation of the metrics data collected for a specified period of time. Periods are defined in numbers of seconds, and valid values for period are 1, 5, 10, 30, or any multiple of 60. For example, to specify a period of six minutes, use 360 as the period value. You can adjust how

the data is aggregated by varying the length of the period. The default value of a period is 60 seconds. A period can be as short as one second, and must be a multiple of 60 if it is greater than the default value of 60 seconds.

Only custom metrics that you define with a storage resolution of 1 second support sub-minute periods. Even though the option to set a period below 60 is always available in the console, you should select a period that aligns to how the metric is stored. For more information about metrics that support sub-minute periods, see [High-resolution metrics](#).

When you retrieve statistics, you can specify a period, start time, and end time. These parameters determine the overall length of time associated with the statistics. The default values for the start time and end time get you the last hour's worth of statistics. The values that you specify for the start time and end time determine how many periods CloudWatch returns. For example, retrieving statistics using the default values for the period, start time, and end time returns an aggregated set of statistics for each minute of the previous hour. If you prefer statistics aggregated in ten-minute blocks, specify a period of 600. For statistics aggregated over the entire hour, specify a period of 3600.

When statistics are aggregated over a period of time, they are stamped with the time corresponding to the beginning of the period. For example, data aggregated from 7:00pm to 8:00pm is stamped as 7:00pm. Additionally, data aggregated between 7:00pm and 8:00pm begins to be visible at 7:00pm, then the values of that aggregated data may change as CloudWatch collects more samples during the period.

Periods are also important for CloudWatch alarms. When you create an alarm to monitor a specific metric, you are asking CloudWatch to compare that metric to the threshold value that you specified. You have extensive control over how CloudWatch makes that comparison. Not only can you specify the period over which the comparison is made, but you can also specify how many evaluation periods are used to arrive at a conclusion. For example, if you specify three evaluation periods, CloudWatch compares a window of three data points. CloudWatch only notifies you if the oldest data point is breaching and the others are breaching or missing.

Aggregation

Amazon CloudWatch aggregates statistics according to the period length that you specify when retrieving statistics. You can publish as many data points as you want with the same or similar time stamps. CloudWatch aggregates them according to the specified period length. CloudWatch does not automatically aggregate data across Regions, but you can use metric math to aggregate metrics from different Regions.

You can publish data points for a metric that share not only the same time stamp, but also the same namespace and dimensions. CloudWatch returns aggregated statistics for those data points. You can also publish multiple data points for the same or different metrics, with any time stamp.

For large datasets, you can insert a pre-aggregated dataset called a *statistic set*. With statistic sets, you give CloudWatch the Min, Max, Sum, and SampleCount for a number of data points. This is commonly used when you need to collect data many times in a minute. For example, suppose you have a metric for the request latency of a webpage. It doesn't make sense to publish data with every webpage hit. We suggest that you collect the latency of all hits to that webpage, aggregate them once a minute, and send that statistic set to CloudWatch.

Amazon CloudWatch doesn't differentiate the source of a metric. If you publish a metric with the same namespace and dimensions from different sources, CloudWatch treats this as a single metric. This can be useful for service metrics in a distributed, scaled system. For example, all the hosts in a web server application could publish identical metrics representing the latency of requests they are processing. CloudWatch treats these as a single metric, allowing you to get the statistics for minimum, maximum, average, and sum of all requests across your application.

Percentiles

A *percentile* indicates the relative standing of a value in a dataset. For example, the 95th percentile means that 95 percent of the data is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.

Percentiles are often used to isolate anomalies. In a normal distribution, 95 percent of the data is within two standard deviations from the mean and 99.7 percent of the data is within three standard deviations from the mean. Any data that falls outside three standard deviations is often considered to be an anomaly because it differs so greatly from the average value. For example, suppose that you are monitoring the CPU utilization of your EC2 instances to ensure that your customers have a good experience. If you monitor the average, this can hide anomalies. If you monitor the maximum, a single anomaly can skew the results. Using percentiles, you can monitor the 95th percentile of CPU utilization to check for instances with an unusually heavy load.

Some CloudWatch metrics support percentiles as a statistic. For these metrics, you can monitor your system and applications using percentiles as you would when using the other CloudWatch statistics (Average, Minimum, Maximum, and Sum). For example, when you create an alarm, you can use percentiles as the statistical function. You can specify the percentile with up to ten decimal places (for example, p95.0123456789).

Percentile statistics are available for custom metrics as long as you publish the raw, unsummarized data points for your custom metric. Percentile statistics are not available for metrics when any of the metric values are negative numbers.

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can only retrieve percentile statistics for this data if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support percentile statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

CloudWatch also supports trimmed mean and other performance statistics, which can have a similar use as percentiles. For more information, see [CloudWatch statistics definitions](#).

Alarms

You can use an *alarm* to automatically initiate actions on your behalf. An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy. You can also add alarms to dashboards.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

When creating an alarm, select an alarm monitoring period that is greater than or equal to the metric's resolution. For example, basic monitoring for Amazon EC2 provides metrics for your instances every 5 minutes. When setting an alarm on a basic monitoring metric, select a period of

at least 300 seconds (5 minutes). Detailed monitoring for Amazon EC2 provides metrics for your instances with a resolution of 1 minute. When setting an alarm on a detailed monitoring metric, select a period of at least 60 seconds (1 minute).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see [Publish custom metrics](#).

For more information, see [Using Amazon CloudWatch alarms](#) and [Create an alarm from a metric on a graph](#).

Billing and costs

For complete information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

For information that can help you analyze your bill and possibly optimize and reduce costs, see [CloudWatch billing and cost](#).

Amazon CloudWatch resources

The following related resources can help you as you work with this service.

Resource	Description
Amazon CloudWatch FAQs	The FAQ covers the top questions developers have asked about this product.
AWS Developer Center	A central starting point to find documentation, code examples, release notes, and other information to help you build innovative applications with AWS.
AWS Management Console	The console allows you to perform most of the functions of Amazon CloudWatch and various other AWS offerings without programming.
Amazon CloudWatch Discussion Forums	Community-based forum for developers to discuss technical questions related to Amazon CloudWatch.

Resource	Description
AWS Support	The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
Amazon CloudWatch product information	The primary webpage for information about Amazon CloudWatch.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse, etc.

Getting set up

To use Amazon CloudWatch you need an AWS account. Your AWS account allows you to use services (for example, Amazon EC2) to generate metrics that you can view in the CloudWatch console, a point-and-click web-based interface. In addition, you can install and configure the AWS command line interface (CLI).

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Sign in to the Amazon CloudWatch console

To sign in to the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, use the navigation bar to change the Region to the Region where you have your AWS resources.
3. Even if this is the first time you are using the CloudWatch console, **Your Metrics** could already report metrics, because you have used an AWS product that automatically pushes metrics to Amazon CloudWatch for free. Other services require that you enable metrics.

If you do not have any alarms, the **Your Alarms** section will have a **Create Alarm** button.

Set up the AWS CLI

You can use the AWS CLI or the Amazon CloudWatch CLI to perform CloudWatch commands. Note that the AWS CLI replaces the CloudWatch CLI; we include new CloudWatch features only in the AWS CLI.

For information about how to install and configure the AWS CLI, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

For information about how to install and configure the Amazon CloudWatch CLI, see [Set Up the Command Line Interface](#) in the *Amazon CloudWatch CLI Reference*.

Getting started with Amazon CloudWatch

Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The CloudWatch overview home page appears.



The overview displays the following items, refreshed automatically.

- **Alarms by AWS service** displays a list of AWS services you use in your account, along with the state of alarms in those services. Next to that, two or four alarms in your account are displayed. The number depends on how many AWS services you use. The alarms shown are those in the ALARM state or those that most recently changed state.

These upper areas help you quickly assess the health of your AWS services, by seeing the alarm states in every service and the alarms that most recently changed state. This helps you monitor and quickly diagnose issues.

- Below these areas is the *default dashboard*, if one exists. The default dashboard is a custom dashboard that you have created and named **CloudWatch-Default**. This is a convenient way for you to add metrics about your own custom services or applications to the overview page, or to bring forward additional key metrics from AWS services that you most want to monitor.

Note

The automatic dashboards on the CloudWatch home page display only information from the current account, even if the account is a monitoring account set up for CloudWatch cross-account observability. For information about creating custom cross-account dashboards, see [CloudWatch cross-account observability dashboard](#).

From this overview, you can see a cross-service dashboard of metrics from multiple AWS service, or focus your view to a specific resource group or a specific AWS service. This enables you to narrow your view to a subset of resources in which you are interested. For more information, see the following sections.

See the automatic pre-built dashboard for a single service

To see the automatic pre-built dashboard for a single service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.

2. In the left navigation pane, choose **Dashboards**.
3. Choose the **Automatic dashboards** tab, then choose the service that you want to see.
4. To switch to viewing the alarms for this service, select the check box for **In alarm**, **Insufficient data**, or **OK** near the top of the screen where the service name is currently displayed.
5. When viewing metrics, you can focus on a particular metric in several ways:
 - a. To see more details about the metrics in any graph, hover on the graph, and choose the actions icon, **View in metrics**.

The graph appears in a new tab, with the relevant metrics listed below the graph. You can customize your view of this graph, changing the metrics and resources shown, the statistic, the period, and other factors to get a better understanding of the current situation.

- b. You can view log events from the time range shown in the graph. This may help you discover events that happened in your infrastructure that are causing an unexpected change in your metrics.

To see the log events, hover on the graph, and choose the actions icon, **View in logs**.

The CloudWatch Logs view appears in a new tab, displaying a list of your log groups. To see the log events in one of these log groups that occurred during the time range shown in the original graph, choose that log group.

6. When viewing alarms, you can focus on a particular alarm in several ways:

- To see more details about an alarm, hover on the alarm, and choose the actions icon, **View in alarms**.

The alarms view appears in a new tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

7. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
8. To change the time range shown in all graphs and alarms currently displayed, next to **Time range** at the top of the screen, choose the range. To select from more time range options than those displayed by default, choose **custom**.
9. To return to the cross-service dashboard, choose **Overview** in the list at the top of the screen that currently shows the service you are focusing on.

Alternatively, from any view, you can choose **CloudWatch** at the top of the screen to clear all filters and return to the overview page.

See the pre-built cross-service dashboard

You can switch to the Cross-service dashboard screen and interact with dashboards for all of the AWS services that you're using. The CloudWatch Console displays your dashboards in alphabetical order and shows one or two key metrics on each dashboard.

Note

If you're using five or more AWS services, the CloudWatch Console won't display the Cross-service dashboard on the Overview screen.

To open the Cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

You're directed to the Overview screen.

2. From the Overview screen, select the dropdown that reads **Overview**, and then choose **Cross service dashboard**.

You're directed to the Cross service dashboard screen.

3. (Optional) If you're using the original interface, scroll to the section **Cross-service dashboard**, and then choose **View Cross-service dashboard**.

You're directed to the Cross-service dashboard screen.

4. You can focus on a particular service in two ways:

- a. To see more key metrics for a service, choose its name from the list at the top of the screen, where **Cross service dashboard** is currently shown. Or, you can choose **View Service dashboard** next to the service name.

An automatic dashboard for that service is displayed, showing more metrics for that service. Additionally, for some services, the bottom of the service dashboard displays resources related to that service. You can choose one of those resources to that service console and focus further on that resource.

- b. To see all the alarms related to a service, choose the button on the right of the screen next to that service name. The text on this button indicates how many alarms you have created in this service, and whether any are in the ALARM state.

When the alarms are displayed, multiple alarms that have similar settings (such as dimensions, threshold, or period) may be shown in a single graph.

You can then view details about an alarm and see the alarm history. To do so, hover on the alarm graph, and choose the actions icon, **View in alarms**.

The alarms view appears in a new browser tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

5. You can focus on resources in a particular resource group. To do so, choose the resource group from the list at the top of the page where **All resources** is displayed.

For more information, see [See a pre-built dashboard for a resource group](#).

6. To change the time range shown in all graphs and alarms currently displayed, select the range you want next to **Time range** at the top of the screen. Choose **custom** to select from more time range options than those displayed by default.
7. Alarms are always refreshed once a minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose the refresh rate you want. You can also choose to turn off automatic refresh.

Remove a service from the cross-service dashboard

You can prevent a service's metrics from appearing in the cross-service dashboard. This helps you focus your cross-service dashboard on the services you most want to monitor.

If you remove a service from the cross-service dashboard, the alarms for that service still appear in the views of your alarms.

To remove a service's metrics from the cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.

2. At the top of the page, under **Overview**, choose the service you want to remove.

The view changes to show metrics from only that service.

3. Choose **Actions**, then clear the check box next to **Show on cross service dashboard**.

See a pre-built dashboard for a resource group

You can focus your view to display metrics and alarms from a single resource group. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or distinguish between your production and development environments. They also enable you to focus on each of these resource groups on the CloudWatch overview. For more information, see [What Is AWS Resource Groups?](#)

When you focus on a resource group, the display changes to show only the services where you have tagged resources as part of this resource group. The recent alarms area shows only the alarms that are associated with resources that are part of the resource group. Additionally, if you have created a

dashboard with the name **CloudWatch-Default-ResourceGroupName**, it is displayed in the **Default dashboard** area.

You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure explains only how to focus on a resource group.

To focus on a single resource group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. At the top of the page, where **All resources** is displayed, choose a resource group.
3. To see more metrics related to this resource group, near the bottom of the screen, choose **View cross service dashboard**.

The cross-service dashboard appears, showing only the services related to this resource group. For each service, one or two key metrics are displayed.

4. To change the time range shown in all graphs and alarms currently displayed, for **Time range** at the top of the screen, select a range. To select from more time range options than those displayed by default, choose **custom**.
5. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
6. To return to showing information about all the resources in your account, near the top of the screen where the name of the resource group is currently displayed, choose **All resources**.

See the pre-built cross-service dashboard

You can switch to the Cross-service dashboard screen and interact with dashboards for all of the AWS services that you're using. The CloudWatch Console displays your dashboards in alphabetical order and displays one or two key metrics from each service.

Note

If you're using five or more AWS services, the CloudWatch Console won't display the Cross-service dashboard on the Overview screen.

To open the Cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

You're directed to the Overview screen.

2. From the Overview screen, select the dropdown that reads **Overview**, and then choose **Cross service dashboard**.

You're directed to the Cross service dashboard screen.

3. (Optional) If you're using the original interface, scroll to the section **Cross-service dashboard**, and then choose **View Cross-service dashboard**.

You're directed to the Cross-service dashboard screen.

4. You can focus on a particular service in two ways:

- a. To see more key metrics for a service, choose its name from the list at the top of the screen, where **Cross service dashboard** is currently shown. Or, you can choose **View Service dashboard** next to the service name.

An automatic dashboard for that service is displayed, showing more metrics for that service. Additionally, for some services, the bottom of the service dashboard displays resources related to that service. You can choose one of those resources to that service console and focus further on that resource.

- b. To see all the alarms related to a service, choose the button on the right of the screen next to that service name. The text on this button indicates how many alarms you have created in this service, and whether any are in the ALARM state.

When the alarms are displayed, multiple alarms that have similar settings (such as dimensions, threshold, or period) may be shown in a single graph.

You can then view details about an alarm and see the alarm history. To do so, hover on the alarm graph, and choose the actions icon, **View in alarms**.

The alarms view appears in a new browser tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

5. You can focus on resources in a particular resource group. To do so, choose the resource group from the list at the top of the page where **All resources** is displayed.

For more information, see [See a pre-built dashboard for a resource group](#).

6. To change the time range shown in all graphs and alarms currently displayed, select the range you want next to **Time range** at the top of the screen. Choose **custom** to select from more time range options than those displayed by default.
7. Alarms are always refreshed once a minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose the refresh rate you want. You can also choose to turn off automatic refresh.

Remove a service from appearing in the cross-service dashboard

You can prevent a service's metrics from appearing in the cross-service dashboard. This helps you focus your cross-service dashboard on the services you most want to monitor.

If you remove a service from the cross-service dashboard, the alarms for that service still appear in the views of your alarms.

To remove a service's metrics from the cross-service dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.

2. At the top of the page, under **Overview**, choose the service you want to remove.

The view changes to show metrics from only that service.

3. Choose **Actions**, then clear the check box next to **Show on cross service dashboard**.

See a pre-built dashboard for a single AWS service

On the CloudWatch home page, you can focus the view to a single AWS service. You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows only how to focus on an AWS service.

To focus on a single service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The home page appears.

2. For **Overview**, where **Overview** is currently shown in the drop down menu, choose **Service dashboards**.
3. Choose the service that you want to focus on.

The view changes to display graphs of key metrics from the selected service.

4. To switch to viewing the alarms for this service, select the check box for **In alarm**, **Insufficient data**, or **OK** near the top of the screen where the service name is currently displayed.
5. When viewing metrics, you can focus on a particular metric in several ways:
 - a. To see more details about the metrics in any graph, hover on the graph, and choose the actions icon, **View in metrics**.

The graph appears in a new tab, with the relevant metrics listed below the graph. You can customize your view of this graph, changing the metrics and resources shown, the statistic, the period, and other factors to get a better understanding of the current situation.

- b. You can view log events from the time range shown in the graph. This may help you discover events that happened in your infrastructure that are causing an unexpected change in your metrics.

To see the log events, hover on the graph, and choose the actions icon, **View in logs**.

The CloudWatch Logs view appears in a new tab, displaying a list of your log groups. To see the log events in one of these log groups that occurred during the time range shown in the original graph, choose that log group.

6. When viewing alarms, you can focus on a particular alarm in several ways:
 - To see more details about an alarm, hover on the alarm, and choose the actions icon, **View in alarms**.

The alarms view appears in a new tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

7. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.

8. To change the time range shown in all graphs and alarms currently displayed, next to **Time range** at the top of the screen, choose the range. To select from more time range options than those displayed by default, choose **custom**.
9. To return to the cross-service dashboard, choose **Overview** in the list at the top of the screen that currently shows the service you are focusing on.

Alternatively, from any view, you can choose **CloudWatch** at the top of the screen to clear all filters and return to the overview page.

See a pre-built dashboard for a resource group

You can focus your view to display metrics and alarms from a single resource group. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or distinguish between your production and development environments. They also enable you to focus on each of these resource groups on the CloudWatch overview. For more information, see [What Is AWS Resource Groups?](#)

When you focus on a resource group, the display changes to show only the services where you have tagged resources as part of this resource group. The recent alarms area shows only the alarms that are associated with resources that are part of the resource group. Additionally, if you have created a dashboard with the name **CloudWatch-Default-ResourceGroupName**, it is displayed in the **Default dashboard** area.

You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows just how to focus on a resource group.

To focus on a single resource group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. At the top of the page, where **All resources** is displayed, choose a resource group.
3. To see more metrics related to this resource group, near the bottom of the screen, choose **View cross service dashboard**.

The cross-service dashboard appears, showing only the services related to this resource group. For each service, one or two key metrics are displayed.

4. To change the time range shown in all graphs and alarms currently displayed, for **Time range** at the top of the screen, select a range. To select from more time range options than those displayed by default, choose **custom**.

5. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
6. To return to showing information about all the resources in your account, near the top of the screen where the name of the resource group is currently displayed, choose **All resources**.

CloudWatch billing and cost

This section describes how Amazon CloudWatch features generate costs. It also provides methods that can help you analyze, optimize, and reduce CloudWatch costs. Throughout this section, we sometimes refer to pricing when describing CloudWatch features. For information about pricing, see [Amazon CloudWatch pricing](#).

Topics

- [Analyze CloudWatch cost and usage data with Cost Explorer](#)
- [Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena](#)
- [Best practices for optimizing and reducing costs](#)

Analyze CloudWatch cost and usage data with Cost Explorer

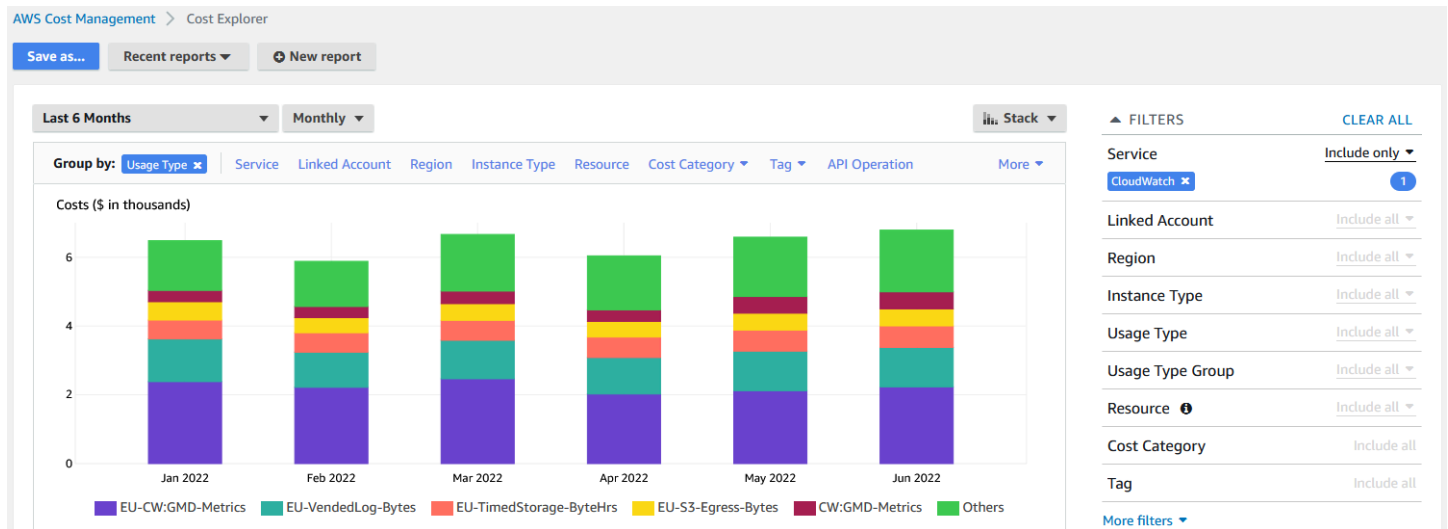
With AWS Cost Explorer, you can visualize and analyze cost and usage data for AWS services over time, including CloudWatch. For more information, see [Getting started with AWS Cost Explorer](#).

The following procedure describes how to use Cost Explorer to visualize and analyze CloudWatch cost and usage data.

To visualize and analyze CloudWatch cost and usage data

1. Sign in to the Cost Explorer console at <https://console.aws.amazon.com/cost-management/home#/custom>.
2. Under **FILTERS**, for **Service**, select **CloudWatch**.
3. For **Group by**, choose **Usage Type**. You can also group your results by other categories, such as the following:
 - **API Operation** – See which API operations generated the most costs.
 - **Region** – See which Regions generated the most costs.

The following image shows an example of the costs that CloudWatch features generated over six months.



To see which CloudWatch features generated the most costs, look at the values for UsageType. For example, EU-CW:GMD-Metrics represents the costs that CloudWatch bulk API requests generated.

Note

The strings for UsageType match specific features and Regions. For example, the first part of EU-CW:GMD-Metrics (EU) matches the Europe (Ireland) Region, and the second part of EU-CW:GMD-Metrics (GMD-Metrics) matches CloudWatch bulk API requests.

The entire string for UsageType can be formatted as follows: <Region>-CW:<Feature> or <Region>-<Feature>.

To enhance readability, the strings for UsageType in the tables throughout this document have been shortened to their string suffixes. For example, EU-CW:GMD-Metrics is shortened to GMD-Metrics.

The following table includes the names of each CloudWatch feature, lists the names of each subfeature, and lists the strings for UsageType.

CloudWatch feature	CloudWatch subfeature	UsageType
CloudWatch metrics	Custom metrics	MetricMonitorUsage
	Detailed monitoring	

CloudWatch feature	CloudWatch subfeature	UsageType
		MetricMonitorUsage
	<i>Embedded metrics</i>	MetricMonitorUsage
CloudWatch API requests	<i>API requests</i>	Requests
	<i>Bulk (Get)</i>	GMD-Metrics
	<i>Contributor Insights</i>	GIRR-Metrics
	<i>Bitmap image snapshot</i>	GMWI-Metrics
CloudWatch metric streams	<i>Metric streams</i>	MetricStreamUsage
CloudWatch dashboards	<i>Dashboard with 50 or fewer metrics</i>	DashboardsUsageHour-Basic
	<i>Dashboard with more than 50 metrics</i>	DashboardsUsageHour
CloudWatch alarms	<i>Standard (metric alarm)</i>	AlarmMonitorUsage
	<i>High resolution (metric alarm)</i>	HighResAlarmMonitorUsage
	<i>Metrics Insights query alarm</i>	MetricInsightAlarmUsage
	<i>Composite (aggregated alarm)</i>	CompositeAlarmMonitorUsage

CloudWatch feature	<i>CloudWatch subfeature</i>	UsageType
CloudWatch Application Signals	<i>Application Signals</i>	Application-Signals
CloudWatch custom logs	<i>Collect (ingest)</i>	DataProcessing-Bytes
	<i>Store (archive)</i>	TimedStorage-ByteHrs
	<i>Analyze (query)</i>	DataScanned-Bytes
CloudWatch Infrequent Access logs	<i>Collect (ingest)</i>	DataProcessingIA-Bytes
CloudWatch vended logs	<i>Delivery (Amazon CloudWatch Logs)</i>	VendedLog-Bytes
	<i>Delivery (CloudWatch Logs Infrequent Access Logs)</i>	VendedLogIA-Bytes
	<i>Delivery (Amazon Simple Storage Service)</i>	S3-Egress-ComprBytes S3-Egress-Bytes
	<i>Delivery (Amazon Data Firehose)</i>	FH-Egress-Bytes
Contributor Insights	<i>CloudWatch Logs (Rules)</i>	ContributorInsightRules
	<i>CloudWatch Logs (Events)</i>	ContributorInsightEvents

CloudWatch feature	CloudWatch subfeature	UsageType
	Amazon DynamoDB (Rules)	ContributorRulesManaged
	DynamoDB Events)	ContributorEventsManaged
Canaries (Synthetics)	Run	Canary-runs
Evidently	Events	Evidently-event
	Analysis Units	Evidently-eau
RUM	Events	RUM-event

Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena

Another way to analyze CloudWatch cost and usage data is by using AWS Cost and Usage Reports with Amazon Athena. AWS Cost and Usage Reports contain a comprehensive set of cost and usage data. You can create reports that track your costs and usage, and you can publish these reports to an S3 bucket of your choice. You also can download and delete your reports from your S3 bucket. For more information, see [What are AWS Cost and Usage Reports?](#) in the *AWS Cost and Usage Reports User Guide*.

Note

There is no charge for using AWS Cost and Usage Reports. You only pay for storage when you publish your reports to Amazon Simple Storage Service (Amazon S3). For more information, see [Quotas and restrictions](#) in the *AWS Cost and Usage Reports User Guide*.

Athena is a query service that you can use with AWS Cost and Usage Reports to analyze cost and usage data. You can query your reports in your S3 bucket without needing to download them first. For more information, see [What is Amazon Athena?](#) in the *Amazon Athena User Guide*. For more information, see [What is Amazon Athena?](#) in the *Amazon Athena User Guide*. For information about pricing, see [Amazon Athena pricing](#).

The following procedure describes the process for enabling AWS Cost and Usage Reports and integrating the service with Athena. The procedure contains two example queries that you can use to analyze CloudWatch cost and usage data.

Note

You can use any of the example queries in this document. All of the example queries in this document correspond to a database named ***costandusagereport***, and show results for the month of April and the year 2022. You can change this information. However, before you run a query, make sure that the name of your database matches the name of the database in the query.

To analyze cost and usage data with AWS Cost and Usage Reports and Athena

1. Enable AWS Cost and Usage Reports. For more information, see [Creating cost and usage reports](#) in the *AWS Cost and Usage Reports User Guide*.

Tip

When you create your reports, make sure to select **Include resource IDs**. Otherwise, your reports won't include the column `line_item_resource_id`. This line helps you further identify costs when analyzing cost and usage data.

2. Integrate AWS Cost and Usage Reports with Athena. For more information, see [Setting up Athena using AWS CloudFormation templates](#) in the *AWS Cost and Usage Reports User Guide*.
3. Query your cost and usage reports.

Example: Athena query

You can use the following query to show which CloudWatch features generated the most costs for a given month.

```
SELECT
CASE
-- Metrics
WHEN line_item_usage_type LIKE '%%MetricMonitorUsage%%' THEN 'Metrics (Custom, Detailed
  monitoring management portal EMF)'
WHEN line_item_usage_type LIKE '%%Requests%%' THEN 'Metrics (API Requests)'
WHEN line_item_usage_type LIKE '%%GMD-Metrics%%' THEN 'Metrics (Bulk API Requests)'
WHEN line_item_usage_type LIKE '%%MetricStreamUsage%%' THEN 'Metric Streams'
-- Dashboard
WHEN line_item_usage_type LIKE '%%DashboardsUsageHour%%' THEN 'Dashboards'
-- Alarms
WHEN line_item_usage_type LIKE '%%AlarmMonitorUsage%%' THEN 'Alarms (Standard)'
WHEN line_item_usage_type LIKE '%%HighResAlarmMonitorUsage%%' THEN 'Alarms (High
  Resolution)'
WHEN line_item_usage_type LIKE '%%MetricInsightAlarmUsage%%' THEN 'Alarms (Metrics
  Insights)'
WHEN line_item_usage_type LIKE '%%CompositeAlarmMonitorUsage%%' THEN 'Alarms
  (Composite)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessing-Bytes%%' THEN 'Logs (Collect - Data
  Ingestion)'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessingIA-Bytes%%' THEN 'Infrequent Access
  Logs (Collect - Data Ingestion)'
WHEN line_item_usage_type LIKE '%%TimedStorage-ByteHrs%%' THEN 'Logs (Storage -
  Archival)'
WHEN line_item_usage_type LIKE '%%DataScanned-Bytes%%' THEN 'Logs (Analyze - Logs
  Insights queries)'
-- Vended Logs
WHEN line_item_usage_type LIKE '%%VendedLog-Bytes%%' THEN 'Vended Logs (Delivered to
  CW)'
WHEN line_item_usage_type LIKE '%%VendedLogIA-Bytes%%' THEN 'Vended Infrequent Access
  Logs (Delivered to CW)'
WHEN line_item_usage_type LIKE '%%FH-Egress-Bytes%%' THEN 'Vended Logs (Delivered to
  Kinesis FH)'
WHEN (line_item_usage_type LIKE '%%S3-Egress-Bytes%%') OR (line_item_usage_type LIKE '%%
  S3-Egress-
  ComprBytes%%') THEN 'Vended Logs (Delivered to S3)'
-- Other
WHEN line_item_usage_type LIKE '%%Application-Signals%%' THEN 'Application Signals'
```

```

WHEN line_item_usage_type LIKE '%%Canary-runs%%' THEN 'Synthetics'
WHEN line_item_usage_type LIKE '%%Evidently%%' THEN 'Evidently'
WHEN line_item_usage_type LIKE '%%RUM-event%%' THEN 'RUM'
ELSE 'Others'
END AS UsageType,
-- REGEXP_EXTRACT(line_item_resource_id, '^(?:.+?:){5}(.)$',1) as ResourceID,
-- SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
('Tax', 'Credit', 'Refund', 'EdpDiscount', 'Fee', 'RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
GROUP BY
1
ORDER BY
TotalSpend DESC,
UsageType;

```

Example: Athena query

You can use the following query to show the results for UsageType and Operation. This shows you how CloudWatch features generated costs. The results also show the values for UsageQuantity and TotalSpend, so that you can see your total usage costs.

Tip

For more information about UsageType, add the following line to this query:

```
line_item_line_item_description
```

This line creates a column called **Description**.

```

SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,

```

```
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
('Tax', 'Credit', 'Refund', 'EdpDiscount', 'Fee', 'RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
```

Best practices for optimizing and reducing costs

CloudWatch metrics

Many AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon S3, and Amazon Data Firehose, automatically send metrics to CloudWatch at no charge. However, metrics that are grouped in the following categories can incur additional costs:

- *Custom metrics, detailed monitoring, and embedded metrics*
- *API requests*
- *Metric streams*

For more information, see [Using Amazon CloudWatch metrics](#).

Custom metrics, detailed monitoring, and embedded metrics

Custom metrics

You can create custom metrics to organize data points in any order and at any rate.

All custom metrics are prorated by the hour. They're metered only when they're sent to CloudWatch. For information about how metrics are priced, see [Amazon CloudWatch Pricing](#).

The following table lists the names of relevant subfeatures for CloudWatch metrics. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify metric-related costs.

Note

To get more details about the metrics that are listed in the following table while you're querying cost and usage data with Athena, match the strings for `Operation` with the results that are shown for `line_item_operation`.

<i>CloudWatch subfeature</i>	UsageType	Operation	Purpose
<i>Custom metrics</i>	MetricMonitorUsage	MetricStorage	Custom metrics
<i>Detailed monitoring</i>	MetricMonitorUsage	MetricStorage:AWS/ <i>{Service}</i>	Detailed monitoring
<i>Embedded metrics</i>	MetricMonitorUsage	MetricStorage:AWS/Logs-EMF	Logs embedded metrics
<i>Log filters</i>	MetricMonitorUsage	MetricStorage:AWS/CloudWatchLogs	Log group metric filters

Detailed monitoring

CloudWatch has two types of monitoring:

- **Basic monitoring**

Basic monitoring is free and automatically enabled for all AWS services that support the feature.

- **Detailed monitoring**

Detailed monitoring incurs costs and adds different enhancements depending on the AWS service. For each AWS service that supports detailed monitoring, you can choose whether to enable it for that service. For more information, see [Basic and detailed monitoring](#).

Note

Other AWS services support detailed monitoring and might refer to this feature using a different name. For example, detailed monitoring for Amazon S3 is referred to as ***request metrics***.

Similar to custom metrics, detailed monitoring is prorated by the hour and metered only when data is sent to CloudWatch. Detailed monitoring generates costs by the number of metrics that are sent to CloudWatch. To reduce costs, only enable detailed monitoring when necessary. For information about how detailed monitoring is priced, see [Amazon CloudWatch Pricing](#).

Example: Athena query

You can use the following query to show which EC2 instances have detailed monitoring enabled.

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation='MetricStorage:AWS/EC2'
AND line_item_line_item_type NOT IN
('Tax', 'Credit', 'Refund', 'EdpDiscount', 'Fee', 'RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
```

```

line_item_usage_type,
line_item_resource_id,
line_item_operation,
line_item_line_item_description
ORDER BY line_item_operation

```

Embedded metrics

With the CloudWatch embedded metric format, you can ingest application data as log data, so that you can generate actionable metrics. For more information, see [Ingesting high-cardinality logs and generating metrics with the CloudWatch embedded metric format](#).

Embedded metrics generate costs by the number of logs ingested, number of logs archived, and number of custom metrics generated.

The following table lists the names of relevant subfeatures for the CloudWatch embedded metric format. The table includes the strings for UsageType and Operation, which can help you analyze and identify costs.

CloudWatch subfeature	UsageType	Operation	Purpose
<i>Custom metrics</i>	MetricMonitorUsage	MetricStorage:AWS/Logs-EMF	Logs embedded metrics
<i>Logs ingestion</i>	DataProcessing-Bytes	PutLogEvents	Uploads a batch of log events to the specified log group or log stream
<i>Logs archival</i>	TimedStorage-ByteHrs	HourlyStorageMetering	Stores logs per hour and logs per byte in CloudWatch Logs

To analyze costs, use AWS Cost and Usage Reports with Athena so that you can identify which metrics are generating costs and determine how the costs are generated.

To make the most of costs generated by the CloudWatch embedded metric format, avoid creating metrics based on high-cardinality dimensions. This way, CloudWatch doesn't create a custom metric for each unique dimension combination. For more information, see [Dimensions](#).

If you're using CloudWatch Container Insights to leverage the embedded metric format, you can use AWS Distro for Open Telemetry as an alternative to make the most of metric-related costs. With Container Insights, you can collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. When you enable Container Insights, the CloudWatch agent sends your logs to CloudWatch, so it can use the logs to generate embedded metrics. However, the CloudWatch agent only sends a fixed number of metrics to CloudWatch, and you're charged for all available metrics, including any that you're not using. With AWS Distro for Open Telemetry, you can configure and customize which metrics and dimensions are sent to CloudWatch. This helps you reduce the volume of data and cost that Container Insights generates. For more information, see the following resources:

- [Using Container Insights](#)
- [AWS Distro for Open Telemetry](#)

API requests

CloudWatch has the following types of API requests:

- **API requests**
- **Bulk (Get)**
- **Contributor Insights**
- **Bitmap image snapshot**

API requests generate costs by the request type and number of metrics requested.

The following table lists the types of API requests and includes the strings for `UsageType` and `Operation`, which can help you analyze and identify API-related costs.

API request type	UsageType	Operation	Purpose
<i>API requests</i>	Requests	GetMetricStatistics	Retrieves statistics for the specified metrics

<i>API request type</i>	UsageType	Operation	Purpose
	Requests	ListMetrics	Lists the specified metrics
	Requests	PutMetricData	Publishes metric data points to CloudWatch
	Requests	GetDashboard	Displays details for the specified dashboards
	Requests	ListDashboards	Lists the dashboards in your account
	Requests	PutDashboard	Creates or updates a dashboard
	Requests	DeleteDashboards	Deletes all specified dashboards
<i>Bulk (Get)</i>	GMD-Metrics	GetMetricData	Retrieves CloudWatch metric values
<i>Contributor Insights</i>	GIRR-Metrics	GetInsightRuleReport	Returns time-series data that's collected by a Contributor Insights rule
<i>Bitmap image snapshot</i>	GMWI-Metrics	GetMetricWidgetImage	Retrieves a snapshot of one or more CloudWatch metrics as a bitmap image

To analyze costs, use Cost Explorer, and group your results by **API Operation**.

Costs for API requests vary, and you incur costs when you exceed the number of API calls provided to you under the AWS Free Tier limit.

Note

GetMetricData and GetMetricWidgetImage aren't included under the AWS Free Tier limit. For more information, see [Using the AWS Free Tier](#) in the *AWS Billing User Guide*.

The API requests that typically drive cost are Put and Get requests.

PutMetricData

PutMetricData generates costs every time that it's called and can incur significant costs depending on the use case. For more information, see [PutMetricData](#) in the *Amazon CloudWatch API Reference*.

To make the most of costs that are generated by PutMetricData, batch more data into your API calls. Depending on your use case, consider using CloudWatch Logs or the CloudWatch embedded metric format to inject metric data. For more information, see the following resources:

- [What is Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch Logs User Guide*
- [Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format](#)
- [Lowering costs and focusing on our customers with Amazon CloudWatch embedded custom metrics](#)

GetMetricData

GetMetricData can also generate significant costs. Common use cases that drive cost involve third-party monitoring tools that pull data to generate insights. For more information, see [GetMetricData](#) in the *Amazon CloudWatch API Reference*.

To reduce costs generated by GetMetricData, consider only pulling data that's monitored and used, or consider pulling data less often. Depending on your use case, you might consider using metric streams instead of GetMetricData, so that you can push data in near real time to third parties at a lower cost. For more information, see the following resources:

- [Using metric streams](#)
- [CloudWatch Metric Streams - Send AWS Metrics to Partners and to Your Apps in Real Time](#)

GetMetricStatistics

Depending on your use case, you might consider using `GetMetricStatistics` instead of `GetMetricData`. With `GetMetricData`, you can retrieve data quickly and at scale. However, `GetMetricStatistics` is included under the AWS Free Tier limit for up to one million API requests, which can help you reduce costs if you don't need to retrieve as many metrics and data points per call. For more information, see the following resources:

- [GetMetricStatistics](#) in the *Amazon CloudWatch API Reference*
- [Should I use GetMetricData or GetMetricStatistics?](#)

Note

External callers make API calls. Currently, the only way to identify these callers is by opening a technical support request to the CloudWatch team and asking for information about them. For information about creating a technical support request, see [How do I get technical support from AWS?](#)

CloudWatch metric streams

With CloudWatch metric streams, you can send metrics continuously to AWS destinations and third-party service provider destinations.

Metric streams generate costs by the number of metric updates. Metric updates always include values for the following statistics:

- Minimum
- Maximum
- Sample Count
- Sum

For more information, see [Statistics that can be streamed](#).

To analyze costs that are generated by CloudWatch metric streams, use AWS Cost and Usage Reports with Athena. This way, you can identify which metric streams are generating costs and determine how the costs are generated.

Example: Athena query

You can use the following query to track which metric streams generate costs by Amazon Resource Name (ARN).

```
SELECT
SPLIT_PART(line_item_resource_id, '/', 2) AS "Stream Name",
line_item_resource_id as ARN,
SUM(CAST(line_item_unblended_cost AS decimal(16,2))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_line_item_type NOT IN
('Tax', 'Credit', 'Refund', 'EdpDiscount', 'Fee', 'RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
AND line_item_usage_type LIKE '%%MetricStreamUsage%%'
GROUP BY line_item_resource_id
ORDER BY TotalSpend DESC
```

To reduce costs generated by CloudWatch metric streams, stream only the metrics that bring your business value. You also can stop or pause any metric stream that you're not using.

CloudWatch alarms

With CloudWatch alarms, you can create alarms based on a single metric, alarms based on a Metrics Insights query, and composite alarms which watch other alarms.

Note

Costs for metric and composite alarms are prorated by the hour. You incur costs for your alarms only while your alarms exist. To optimize cost, you be sure not to leave behind misconfigured or low-value alarms. To help with this, you can automate the cleanup of CloudWatch alarms that you no longer need. For more information, see [Automating Amazon CloudWatch Alarm Cleanup at Scale](#)

Metric alarms

Metric alarms have the following resolution settings:

- **Standard** (evaluated every 60 seconds)
- **High resolution** (evaluated every 10 seconds)

When you create a metric alarm, your costs are based on your alarm's resolution setting and the number of metrics that your alarm references. For example, a metric alarm that references one metric incurs one alarm-metric cost per hour. For more information, see [Using Amazon CloudWatch alarms](#).

If you create a metric alarm that contains a metric math expression, which references multiple metrics, you incur a cost for each alarm-metric that's referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see [Creating a CloudWatch alarm based on a metric math expression](#).

If you create an anomaly detection alarm, where your alarm analyzes past metric data to create a model of expected values, you incur a cost for each alarm-metric that's referenced in your alarm plus two additional metrics, one for the upper and lower band metrics that the anomaly detection model creates. For information about how to create an anomaly detection alarm, see [Creating a CloudWatch alarm based on anomaly detection](#).

Metrics Insights query alarms

Metric Insights query alarms are a specific type of metric alarm, only available with standard resolution (evaluated every 60 seconds).

When you create a Metric Insights query alarm, your costs are based on the number of metrics analyzed by the query that your alarm references. For example, a Metric Insights query alarm that references a query whose filter matches ten metrics incurs ten metrics analyzed cost per hour. For more information, see the pricing example on [Amazon CloudWatch Pricing](#).

If you create an alarm that contains both a Metrics Insights query and a metric math expression, it is reported as a Metrics Insights query alarm. If your alarm contains a metric math expression which references other metrics in addition to the metrics analyzed by the Metrics Insights query, you incur an additional cost for each alarm-metric that's referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see [Creating a CloudWatch alarm based on a metric math expression](#).

Composite alarms

Composite alarms contain rule expressions that specify how they should evaluate the states of other alarms to determine their own states. Composite alarms incur a standard cost per hour, regardless of how many other alarms they evaluate. Alarms that composite alarms reference in rule expressions incur separate costs. For more information, see [Creating a composite alarm](#).

Alarm usage types

The following table lists the names of relevant subfeatures for CloudWatch alarms. The table includes the strings for UsageType, which can help you analyze and identify alarm-related costs.

<i>CloudWatch subfeature</i>	UsageType
<i>Standard metric alarm</i>	AlarmMonitorUsage
<i>High-resolution metric alarm</i>	HighResAlarmMonitorUsage
<i>Metrics Insights query alarm</i>	MetricInsightAlarmUsage
<i>Composite alarm</i>	CompositeAlarmMonitorUsage

Reducing alarm costs

To optimize costs generated by metric math alarms that aggregate four or more metrics, you can aggregate data before the data is sent to CloudWatch. This way, you can create an alarm for a single metric instead of an alarm that aggregates data for multiple metrics. For more information, see [Publishing custom metrics](#).

To optimize costs generated by Metrics Insights query alarms, you can ensure that the filter used for the query matches only the metrics that you want to monitor.

The best way to reduce costs is to remove all unnecessary or unused alarms. For example, you can delete alarms that evaluate metrics emitted by AWS resources that no longer exist.

Example: Check for alarms in `INSUFFICIENT_DATA` state with `DescribeAlarms`

If you delete a resource, but not the metric alarms that the resource emits, the alarms still exist and typically will go into the `INSUFFICIENT_DATA` state. To check for alarms that are in the `INSUFFICIENT_DATA` state, use the following AWS Command Line Interface (AWS CLI) command.

```
$ aws cloudwatch describe-alarms --state-value INSUFFICIENT_DATA
```

Other ways to reduce costs include the following:

- Make sure to create alarms for the correct metrics.
- Make sure that you don't have any alarms enabled in Regions where you're not working.
- Remember that, although composite alarms reduce noise, they also generate additional costs.
- When deciding whether to create a standard alarm or high-resolution alarm, consider your use case and the value that each type of alarm brings.

CloudWatch Logs

Amazon CloudWatch Logs has the following log types:

- **Custom logs** (logs that you create for your applications)
- **Vended logs** (logs that other AWS services, such as Amazon Virtual Private Cloud (Amazon VPC) and Amazon Route 53, create on your behalf)

For more information about vended logs, see [Enabling logging from certain AWS services](#) in the *Amazon CloudWatch Logs User Guide*.

Custom and vended logs generate costs based on the number of logs that are *collected*, *stored*, and *analyzed*. Separately, vended logs generate costs for delivery to Amazon S3 and Firehose.

The following table lists the names of the CloudWatch Logs features and names of relevant subfeatures. The table includes the strings for UsageType and Operation, which can help you analyze and identify log-related costs.

CloudWatch Logs feature	CloudWatch Logs subfeature	UsageType	Operation	Purpose
Custom logs	Collect (ingest)	DataProcessing-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream

CloudWatch Logs feature	CloudWatch Logs subfeature	UsageType	Operation	Purpose
	<i>Store (archive)</i>	TimedStorage-Bytes	HourlyStorageMetering	Stores logs per hour and logs per byte in CloudWatch Logs
	<i>Analyze (Logs Insights queries)</i>	DataScanned-Bytes	StartQuery	Logs data scanned by CloudWatch Logs Insights queries
Vended logs	<i>Delivery (CloudWatch Logs)</i>	VendedLog-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream
	<i>Delivery (Amazon S3)</i>	S3-Egress-ComprBytes S3-Egress-Bytes	LogDelivery	Sends vended logs (CloudWatch, Amazon S3, or Firehose)
	<i>Delivery (Firehose)</i>	FH-Egress-Bytes	LogDelivery	Sends vended logs (CloudWatch, Amazon S3, or Firehose)

To analyze costs, use AWS Cost and Usage Reports with Athena, so that you can identify which logs are generating costs and determine how the costs are generated.

Example: Athena query

You can use the following query to track which logs generate costs by resource ID.


```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_resource_id AS ResourceID,
line_item_usage_type AS UsageType,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2022'
AND month='4'
AND line_item_operation IN
('PutLogEvents', 'HourlyStorageMetering', 'StartQuery', 'LogDelivery')
AND line_item_line_item_type NOT IN
('Tax', 'Credit', 'Refund', 'EdpDiscount', 'Fee', 'RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
ORDER BY
TotalSpend DESC
```

To make the most of costs that are generated by CloudWatch Logs, consider the following:

- Log only the events that bring your business value. This helps you generate fewer costs for ingestion.
- Change your log retention settings, so that you generate fewer costs for storage. For more information, see [Change log data retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.
- Run queries that CloudWatch Logs Insights automatically saves in your history. This way, you generate fewer costs for analysis. For more information, see [View running queries or query history](#) in the *Amazon CloudWatch Logs User Guide*.
- Use the CloudWatch agent to collect system and application logs and send them to CloudWatch. This way, you can collect only the log events that meet your criteria. For more information, see [Amazon CloudWatch Agent adds Support for Log Filter Expressions](#).

To reduce costs for vended logs, consider your use case, and then determine whether your logs should be sent to CloudWatch or Amazon S3. For more information, see [Logs sent to Amazon S3](#) in the *Amazon CloudWatch Logs User Guide*.

Tip

If you want to use metric filters, subscription filters, CloudWatch Logs Insights, and Contributor Insights, send vended logs to CloudWatch.

Alternatively, if you're working with VPC Flow Logs and using them for auditing and compliance purposes, send vended logs to Amazon S3.

For information about how to track charges that are generated by publishing VPC Flow Logs to S3 buckets, see [Using AWS Cost and Usage Reports and cost allocation tags to understand VPC Flow Logs data ingestion in Amazon S3](#).

For additional information about how to make the most of costs that are generated by CloudWatch Logs, see [Which log group is causing a sudden increase in my CloudWatch Logs bill?](#)

Using Amazon CloudWatch dashboards

Amazon CloudWatch dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the metrics and alarms for your AWS resources.

With dashboards, you can create the following:

- A single view for selected metrics and alarms to help you assess the health of your resources and applications across one or more Regions. You can select the color used for each metric on each graph, so that you can easily track the same metric across multiple graphs.
- An operational playbook that provides guidance for team members during operational events about how to respond to specific incidents.
- A common view of critical resource and application measurements that can be shared by team members for faster communication flow during operational events.

If you have multiple AWS accounts, you can set up *CloudWatch cross-account observability* and then create rich cross-account dashboards in your monitoring accounts. These dashboards can include graphs of metrics from source accounts and CloudWatch Logs Insights widgets with queries of log groups from source accounts. Additionally, alarms that you create in the monitoring account can watch metrics in source accounts. For more information, see [CloudWatch cross-account observability](#).

You can create dashboards from the console or using the AWS CLI or PutDashboard API operation. You can add dashboards to a favorites list, where you can access not only your favorited dashboards, but also your recently visited dashboards. For more information, see [Add a dashboard to your favorites list](#).

To access CloudWatch dashboards, you need one of the following:

- The AdministratorAccess policy
- The CloudWatchFullAccess policy
- A custom policy that includes one or more of these specific permissions:
 - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards

- `cloudwatch:PutDashboard` to be able to create or modify dashboards
- `cloudwatch:DeleteDashboards` to be able to delete dashboards

Contents

- [Creating a CloudWatch dashboard](#)
- [CloudWatch cross-account observability dashboard](#)
- [Cross-account cross-Region dashboards](#)
- [Create flexible dashboards with dashboard variables](#)
- [Create and work with widgets on CloudWatch dashboards](#)
- [Sharing CloudWatch dashboards](#)
- [Use live data](#)
- [Viewing an animated dashboard](#)
- [Add a CloudWatch dashboard to your favorites list](#)
- [Change the period override setting or refresh interval for the CloudWatch dashboard](#)
- [Change the time range or time zone format of a CloudWatch dashboard](#)

Creating a CloudWatch dashboard

To get started, create a CloudWatch dashboard. You can create multiple dashboards, and you can add dashboards to a favorites list. You aren't limited to the number of dashboards that you can have in your AWS account. All dashboards are global. They are not Region-specific.

The following procedure shows you how to create a dashboard from the CloudWatch console. You can use the `PutDashboard` API operation to create a dashboard from the command line interface. The API operation contains a JSON string that defines your dashboard content. For more information about creating a dashboard with the `PutDashboard` API operation, see [PutDashboard](#) in the *Amazon CloudWatch API Reference*.

Tip

If you're creating a new dashboard with the `PutDashboard` API operation, you can use the JSON string from a dashboard that already exists.

To create a dashboard from the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose **Create dashboard**.
3. In the **Create new dashboard** dialog box, enter a name for the dashboard, and then choose **Create dashboard**.

If you use the name **CloudWatch-Default** or **CloudWatch-Default-*ResourceGroupName***, the dashboard appears in the overview of the CloudWatch home page under **Default Dashboard**. For more information, see [Getting started with Amazon CloudWatch](#).

4. In the **Add to this dashboard** dialog box, do one of the following:
 - To add a graph to the dashboard, choose **Line** or **Stacked area**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**. If a metric doesn't appear in the dialog box because it hasn't published data in more than 14 days, you can add it manually. For more information, see [Graph metrics manually on a CloudWatch dashboard](#).
 - To add a number displaying a metric to the dashboard, choose **Number**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**.
 - To add a text block to the dashboard, choose **Text**, and then choose **Configure**. In the **New text widget** dialog box, for **Markdown**, format your text using [Markdown](#), and then choose **Create widget**.
5. (Optional) Choose **Add widget**, and then repeat step 4 to add another widget to the dashboard. You can repeat this step multiple times.

For each graph on the dashboard, there is an information icon at the upper right. Choose this icon to see the descriptions of the metrics in the graph.

6. Choose **Save dashboard**.

CloudWatch cross-account observability dashboard

If you have multiple AWS accounts, you can set up *CloudWatch cross-account observability* and then create rich cross-account dashboards in your monitoring accounts. You can seamlessly search, visualize, and analyze your metrics, logs, and traces without account boundaries.

For more information about setting up CloudWatch cross-account observability, see [CloudWatch cross-account observability](#).

With CloudWatch cross-account observability, you can do the following in a dashboard in a monitoring account:

- Search, view, and create graphs of metrics that reside in source accounts. A single graph can include metrics from multiple accounts.
- Create alarms in the monitoring account that watch metrics in source accounts.
- View the log events from log groups located in source accounts, and run CloudWatch Logs Insights queries of log groups in source accounts. A single CloudWatch Logs Insights query in a monitoring account can query multiple log groups in multiple source accounts at once.
- View nodes from source accounts in a trace map in X-Ray. You can then filter the map to specific source accounts.

When you are signed in to a monitoring account, a blue **Monitoring account** badge appears at the top right of every page that supports CloudWatch cross-account observability functionality.

Cross-account cross-Region dashboards

You can create *cross-account cross-Region dashboards*, which summarize your CloudWatch data from multiple AWS accounts and multiple Regions into one dashboard. From this high-level dashboard you can get a view of your entire application, and also drill down into more specific dashboards without having to sign in and out of accounts or switch Regions.

You can create cross-account cross-Region dashboards in the AWS Management Console and programmatically.

Prerequisite

Before you can create a cross-account cross-Region dashboard, you must enable at least one sharing account and at least one monitoring account. Additionally, to be able to use the CloudWatch console to create a cross-account dashboard, you must enable the console for cross-account functionality. For more information, see [Cross-account cross-Region CloudWatch console](#).

Creating and using a cross-account cross-Region dashboard with the AWS Management Console

You can use the AWS Management Console to create a cross-account cross-Region dashboard.

To create a cross-account cross-Region dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Dashboards**.
4. Choose a dashboard, or create a new one.
5. At the top of the screen, you can switch between accounts and Regions. As you create your dashboard, you can include widgets from multiple accounts and Regions. Widgets include graphs, alarms, and CloudWatch Logs Insights widgets.

Creating a graph with metrics from different accounts and Regions

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
4. Select the account and Region that you want to add metrics from. You can select your account and Region from the account and Region dropdown menus near the top-right of the screen.
5. Add the metrics you want to the graph. For more information, see [Graphing metrics](#).
6. Repeat steps 4-5 to add metrics from other accounts and Regions.
7. (Optional) Choose the **Graphed metrics** tab and add a metric math function that uses the metrics that you have chosen. For more information, see [Use metric math](#).

You can also set up a single graph to include multiple SEARCH functions. Each search can refer to a different account or Region.

8. When you are finished with the graph, choose **Actions, Add to dashboard**.

Select your cross-account dashboard, and choose **Add to dashboard**.

Adding an alarm from a different account to your cross-account dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. At the top of the page, choose the account where the alarm is located.
4. In the navigation pane, choose **Alarms**.
5. Select the check box next to the alarm that you want to add, and choose **Add to dashboard**.
6. Select the cross-account dashboard that you want to add it to, and choose **Add to dashboard**.

Create a cross-account cross-Region dashboard programmatically

You can use the AWS APIs and SDKs to create dashboards programmatically. For more information, see [PutDashboard](#).

To enable cross-account cross-Region dashboards, we have added new parameters to the dashboard body structure, as shown in the following table and examples. For more information about overall dashboard body structure, see [Dashboard Body Structure and Syntax](#).

Parameter	Use	Scope	Default
accountId	Specifies the ID of the account where the widget or the metric is located.	Widget or metric	Account that is currently logged in
region	Specifies the Region of the metric.	Widget or metric	Current Region selected in the console

The following examples illustrate the JSON source for widgets in a cross-account cross-Region dashboard.

This example sets the `accountId` field to the ID of the sharing account at the widget level. This specifies that all metrics in this widget will come from that sharing account and Region.

```
{
  "widgets": [
    {
```



```

    ...
    "properties": {
      "metrics": [
        ...
      ],
      "accountId": "111122223333",
      "region": "us-east-1"
    }
  }
]
}

```

This example sets the `accountId` field differently at the level of each metric. In this example, the different metrics in this metric math expression come from different sharing accounts and different Regions.

```

{
  "widgets": [
    {
      ...
      "properties": {
        "metrics": [
          [ { "expression": "SUM(METRICS())", "label": "[avg: ${AVG}]
Expression1", "id": "e1", "stat": "Sum" } ],
          [ "AWS/EC2", "CPUUtilization", { "id": "m2", "accountId":
"5555666677778888", "region": "us-east-1", "label": "[avg: ${AVG}] ApplicationALabel
" } ],
          [ ".", ".", { "id": "m1", "accountId": "9999000011112222", "region":
"eu-west-1", "label": "[avg: ${AVG}] ApplicationBLabel" } ]
        ],
        "view": "timeSeries",
        "region": "us-east-1", ---> home region of the metric. Not present in above
example
        "stacked": false,
        "stat": "Sum",
        "period": 300,
        "title": "Cross account example"
      }
    }
  ]
}

```

This example shows an alarm widget.

```
{
  "type": "metric",
  "x": 6,
  "y": 0,
  "width": 6,
  "height": 6,
  "properties": {
    "accountID": "111122223333",
    "title": "over50",
    "annotations": {
      "alarms": [
        "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
      ]
    },
    "view": "timeSeries",
    "stacked": false
  }
}
```

This example is for a CloudWatch Logs Insights widget.

```
{
  "type": "log",
  "x": 0,
  "y": 6,
  "width": 24,
  "height": 6,
  "properties": {
    "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp desc\n| limit 20",
    "accountId": "111122223333",
    "region": "us-east-1",
    "stacked": false,
    "view": "table"
  }
}
```

Another way to create dashboards programmatically is to first create one in the AWS Management Console, and then copy the JSON source of this dashboard. To do so, load the dashboard and choose **Actions, View/edit source**. You can then copy this dashboard JSON to use as a template to create similar dashboards.

Create flexible dashboards with dashboard variables

Use *dashboard variables* to create flexible dashboards that can quickly display different content in multiple widgets, depending on the value of an input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different AWS Regions.

After you create a dashboard that uses a variable, you can copy the same variable pattern to other existing dashboards.

Using dashboard variables improves the operational workflow for people who use your dashboards. It can also reduce your costs because you're using dashboard variables in one dashboard instead of creating multiple similar dashboards.

Note

If you share a dashboard that contains dashboard variables, the people that you share it with won't be able to change between the variable values.

Types of dashboard variables

The dashboard variable can be a *property variable* or a *pattern variable*.

- *Property variables* change all instances of a property in all widgets in the dashboard. This property can be any JSON property in the JSON source of a dashboard, such as `region`. Or it can be a dimension name for a metric, such as `InstanceID` or `FunctionName`.

For a tutorial that uses a property variable, see [Tutorial: Create a Lambda dashboard with function name as the variable](#).

For more information about the JSON source of dashboards, see [Dashboard Body Structure and Syntax](#). In the CloudWatch console, you can see the JSON source for any custom dashboard by choosing **Actions, View/edit source**.

- *Pattern variables* use a regular expression pattern to change all of a JSON property or only a certain part of it.

For a tutorial that uses a pattern variable, see [Tutorial: Create a dashboard that uses a regular expression pattern to switch between Regions](#).

Property variables apply to most use cases and are less complex to set up.

Topics

- [Tutorial: Create a Lambda dashboard with function name as the variable](#)
- [Tutorial: Create a dashboard that uses a regular expression pattern to switch between Regions](#)
- [Copy a variable to another dashboard](#)

Tutorial: Create a Lambda dashboard with function name as the variable

The steps in this procedure illustrate how to create a flexible dashboard that shows a variety of metric graphs, using a property variable. This includes a dropdown selection box on the dashboard that you can use to switch the metrics in all the graphs between different Lambda functions.

Other use case examples for this type of dashboard include using InstanceId as the variable to create a dashboard of metrics with a dropdown for instance IDs. Alternatively, you could create a dashboard that uses region as the variable to display the same set of metrics from different Regions.

To use a dashboard property variable to create a flexible Lambda dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards, Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard that display metrics for a Lambda function. When you create these widgets, specify **Lambda, By Function Name** for the widget metrics. For the function, specify one of the Lambda functions that you want to include in this dashboard.

For more information about adding widgets to a dashboard, see [Create and work with widgets on CloudWatch dashboards](#).

5. After you add the widgets, as you are viewing the dashboard, choose **Actions, Variables, Create a variable**.
6. Choose **Property variable**.
7. For **Property that the variable changes**, choose **FunctionName**.

8. For **Input type**, for this use case, we recommend choosing **Select menu (dropdown)**. This creates a dropdown menu in the dashboard where you can select the Lambda function name to display metrics for.

If this was for a dashboard that toggled between only two or three different values for a variable, then **Radio button** would be a good choice.

If you prefer to enter or paste in values for the variable, you would choose **Text input**. This option doesn't include a dropdown list or radio buttons.
9. When you choose **Select menu (dropdown)**, you must then choose whether to populate the menu by entering values, or using a metric search. For this use case, let's assume that you have a large number of Lambda functions and you don't want to enter all of them manually. Choose **Use the results of a metric search** and then do the following:
 - a. Choose **Pre-built queries, Lambda, Errors**.

(Choosing **Errors** does not add the **Errors** metric to the dashboard. However, it quickly populates the **FunctionName** variable selection box.)
 - b. Choose **By Function Name** and then choose **Search**.

Under the **Search** button, you will then see **FunctionName** selected. You also see a message about how many **FunctionName** dimension values were found to populate the input box.
10. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:
 - To customize the name of your variable, enter the name in **Custom variable name**.
 - To customize the label for the variable input field, enter the label in **Input label**.
 - To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.
11. Choose **Add variable**.

A **FunctionName** dropdown selection box appears near the top of the dashboard. You can select a Lambda function in this box and all the widgets that use the variable will display information about the selected function.

Later, if you add more widgets to the dashboard that watch Lambda metrics with the **FunctionName** dimension, they will automatically use the variable.

Tutorial: Create a dashboard that uses a regular expression pattern to switch between Regions

The steps in this procedure illustrate how to create a flexible dashboard that can switch between Regions. This tutorial uses a regular expression *pattern variable* instead of a property variable. For a tutorial that uses a property variable, see [Tutorial: Create a Lambda dashboard with function name as the variable](#).

For many use cases, you can create a dashboard that switches between Regions by using a property variable. But if the widgets rely on Amazon Resource Names (ARNs) that include Region names, you must use a pattern variable to change the Region names within the ARNs.

To use a dashboard pattern variable to create a flexible multi-Region dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards, Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard. When you add the widgets that you want to display Region-specific data, avoid specifying any dimensions with values that appear in only one Region. For example, for Amazon EC2 metrics, specify metrics that are aggregated instead of metrics that use **InstanceID** as a dimension.

For more information about adding widgets to a dashboard, see [Create and work with widgets on CloudWatch dashboards](#).

5. After you add the widgets, as you are viewing the dashboard, choose **Actions, Variables, Create a variable**.
6. Choose **Pattern variable**.
7. For **Property that the variable changes**, enter the name of the current dashboard Region, such as **us-east-2**.

You have the correct Region entered if the label below that box displays the widgets that will be impacted by the variable.

8. For **Input type**, for this use case, select **Radio button**.
9. For **Define how inputs are populated**, choose **Create a list of custom values**.

10. For **Create your custom values**, enter the Regions that you want to switch between, with one Region on each line. After each Region, enter a comma and then the label to display for that radio button. For example:

us-east-1, N. Virginia

us-east-2, Ohio

eu-west-3, Paris

As you fill in the custom values, the **Preview** pane updates to display what the radio buttons will look like.

11. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:

- To customize the name of your variable, enter the name in **Custom variable name**.
- To customize the label for the variable input field, enter the label in **Input label**. For this tutorial, enter **Region:**.

If you enter a value here, the **Preview** pane updates to display what the radio buttons will look like.

- To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.

12. Choose **Add variable**.

The dashboard appears, with a **Region:** label next to the radio buttons for the Regions near the top. When you switch between Regions, all the widgets that use the variable will display information about the selected Region.

Copy a variable to another dashboard

After you create a dashboard with useful variables, you can copy these variables to other existing dashboards. For more information about dashboard variables, see [Create flexible dashboards with dashboard variables](#).

To copy a dashboard variable to another dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Dashboards**, and then choose the name of the dashboard that has the variable that you want to copy. Enter a string to find dashboards with matching names, if needed.
3. Choose **Actions, Variables, Manage variables**.
4. Choose the radio button next to the variable that you want to copy, and choose **Copy to another dashboard**.
5. Choose the selection box and start typing the dashboard name that you want to copy the variable to.
6. Select the dashboard name and choose **Copy variable**.

Create and work with widgets on CloudWatch dashboards

Use the topics in this section to create and work with graphs, alarms, and text widgets in your dashboards.

Contents

- [Add or remove a graph from a CloudWatch dashboard](#)
- [Graph metrics manually on a CloudWatch dashboard](#)
- [Working with existing graphs](#)
- [Add a metrics explorer widget to a CloudWatch dashboard](#)
- [Add or remove a line widget on a CloudWatch dashboard](#)
- [Add or remove a number widget from a CloudWatch dashboard](#)
- [Add or remove a gauge widget from a CloudWatch dashboard](#)
- [Add a custom widget to a CloudWatch dashboard](#)
- [Add or remove a text widget from a CloudWatch dashboard](#)
- [Add or remove an alarm widget from a CloudWatch dashboard](#)
- [Add or remove a data table widget from a CloudWatch dashboard](#)
- [Link and unlink graphs on a CloudWatch dashboard](#)

Add or remove a graph from a CloudWatch dashboard

You can add graphs that contain one or more metrics to your CloudWatch dashboard. The types of graphs that you can add to your dashboard include **Line**, **Stacked area**, **Number**, **Gauge**, **Bar**,

and **Pie**. You can remove graphs from your dashboard when you don't need them anymore. The procedures in this section describe how to add and remove graphs from your dashboard. For information about how to edit a graph on your dashboard, see [Edit a graph on a CloudWatch dashboard](#).


To add a graph to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and then choose the type of graph that you want to add to your dashboard, then choose **Next**.
 - If you select **Line**, **Stacked area**, **Bar**, or **Pie**, choose **Metrics**.
4. In the **Browse** tab, search or browse for the metrics to graph, and select the ones that you want.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper part of the screen. The time ranges span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).

To set your own time range, choose **Custom**.

- (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. (Optional) To change your graph's widget type, use the dropdown that's next to the predefined time ranges.
 7. (Optional) In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.
 - a. To add a dynamic label, choose **Graphed metrics**, and then choose **Add dynamic labels**. Dynamic labels display a statistic about your metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of your labels. For more information, see [Use dynamic labels](#).
 - b. To change the color of a metric, choose the color square that's next to the metric.
 - c. To change the statistic, select the dropdown under **Statistic**, and then choose a new value. For more information, see [Statistics](#).

- d. To change the period, select the dropdown under the **Period** column, and then choose a new value.
8. If you are creating a gauge widget, you must choose the **Options** tab and specify the **Min** and **Max** values to use for the two ends of the gauge.
9. (Optional) To customize the Y-axis, choose **Options**. You can add a custom label under **Left Y-axis** in the label field. If your graph displays values on the right side of the Y-axis, you can customize that label, too. You also can set minimum and maximum limits on your Y-axis values, so that your graph only displays the value ranges that you specify.
10. (Optional) To add or edit horizontal annotations to line or stacked area graphs, or to add thresholds to gauge widgets, choose **Options**:
 - a. To add a horizontal annotation or threshold, choose **Add horizontal annotation** or **Add threshold**.
 - b. For **Label**, enter a label for the annotation then choose the check mark icon.
 - c. For **Value**, choose the pen and paper icon that's next to the current value, and enter your new value. After you enter your value, choose the check mark icon.
 - d. For **Fill**, select the dropdown and specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. To change the fill color, choose the color square that's next to the annotation.
 - e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.
 - f. To hide an annotation, clear the check box that's next to the annotation you want to hide.
 - g. To delete an annotation, choose **X** under **Actions**.

 **Note**

You can repeat these steps to add multiple horizontal annotations or thresholds to the same graph or gauge.

11. (Optional) To add or edit vertical annotations, choose **Options**:
 - a. To add a vertical annotation, choose **Add vertical annotation**.
 - b. For **Label**, choose the pen and paper icon that's next to the current annotation, and enter your new annotation. If you want to show only the date and time, leave the label field blank.
 - c. For **Date**, choose the current date and time, and enter the new date and time.

- d. For **Fill**, select the dropdown, and specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. To change the fill color, select the color square that's next to the annotation.
- e. To hide an annotation, clear the check box next to the annotation that you want to hide.
- f. To delete an annotation, choose **X** under **Actions**.

 **Note**

You can repeat these steps to add multiple vertical annotations to the same graph.

12. Choose **Create widget**.
13. Choose **Save dashboard**.

To remove a graph from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Graph metrics manually on a CloudWatch dashboard

If a metric hasn't published data in the past 14 days, you can't find it when searching for metrics to add to a graph on a CloudWatch dashboard. Use the following steps to add any metric manually to an existing graph.

To add a metric that you can't find in search to a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. The dashboard must already contain a graph where you want to add the metric. If it doesn't, create the graph and add any metric to it. For more information, see [Add or remove a graph from a CloudWatch dashboard](#).
4. Choose **Actions**, **View/edit source**.

A JSON block appears. The block specifies the widgets on the dashboard and their contents. The following is an example of one part of this block, which defines one graph.

```
{
    "type": "metric",
    "x": 0,
    "y": 0,
    "width": 6,
    "height": 3,
    "properties": {
        "view": "singleValue",
        "metrics": [
            [ "AWS/EBS", "VolumeReadOps", "VolumeId",
"vol-1234567890abcdef0" ]
        ],
        "region": "us-west-1"
    }
},
```

In this example, the following section defines the metric shown on this graph.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. Add a comma after the end bracket if there isn't already one and then add a similar bracketed section after the comma. In this new section, specify the namespace, metric name, and any necessary dimensions of the metric that you're adding to the graph. The following is an example.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

For more information about the formatting of metrics in JSON, see [Properties of a Metric Widget Object](#).

6. Choose **Update**.

Working with existing graphs

Follow the procedures in these sections to edit and modify your existing dashboard graph widgets.

Topics

- [Edit a graph on a CloudWatch dashboard](#)
- [Move or resize a graph on a CloudWatch dashboard](#)
- [Rename a graph on a CloudWatch dashboard](#)

Edit a graph on a CloudWatch dashboard

You can edit the graphs that you add to your CloudWatch dashboard. You can change a graph's title, statistic, or period. You can add, update, and remove metrics from your graphs. If your graph contains more than one metric, you can reduce clutter by hiding metrics that you aren't using. The procedures in this section describe how to edit a graph on your dashboard. For information about creating a graph, see [Add or remove a graph from a CloudWatch dashboard](#).


New interface

To edit a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pen and paper icon that's next to the current title. Enter the new title, and then choose **Apply**.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper are of the graph. The time ranges span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).

To set your own time range, choose **Custom**.

- (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. To change your graph's widget type, use dropdown that's next to the predefined time ranges.
 7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

- a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see [Use dynamic labels](#).
 - b. To change the color of a metric, choose the color square that's next to the metric.
 - c. To change the statistic, choose the statistic value under the **Statistic** column, and then choose a new value. For more information, see [Statistics](#).
 - d. To change the period, choose the period value under the **Period** column, and then choose a new value.
8. To add or edit horizontal annotations, choose **Options**:
- a. To add a horizontal annotation, choose **Add horizontal annotation**.
 - b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.
 - c. For **Value**, choose the pen and paper icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.
 - d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, another new label and value field appears.
-  **Tip**

You can change the fill color by choosing the colored square next to the annotation.
- e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.
 - f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
 - g. To delete an annotation, choose **X** under the **Actions** column.

Note

You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Options**:
 - a. To add a vertical annotation, choose **Add vertical annotation**.
 - b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

Tip

To show only the date and time, leave the label field blank.

- c. For **Date**, choose the current date and time. Then enter the new date and time.
- d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

Tip

You can change the fill color by choosing the color square next to the annotation.

Note

You can repeat these steps to add multiple vertical annotations to the same graph.

- e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
 - f. To delete an annotation, choose **X** under the **Actions** column.
10. To customize the Y-axis, choose **Options**. Under **Left Y-axis**, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too.

You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.

11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**. Choose **Legend** and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An **X** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Original interface

To edit a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit. Choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pencil icon that's next to the current title, and then enter the new title.
5. To change the graph's time range, choose one of the predefined time ranges in the upper area of the graph. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).
 - To set your own time range, choose **custom**.

6. To change your graph's widget type, select the **Graph options** tab. You can choose **Line**, **Stacked area**, **Number**, **Bar**, or **Pie**.

 **Tip**

You can change your graph's widget type by choosing the dropdown that's next to the predefined time ranges.

7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.
 - a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see [Use dynamic labels](#).
 - b. To change the color of a metric, choose the color square that's next to the metric.
 - c. To change the statistic, choose the statistic value under the **Statistic** column, and then choose a new value. For more information, see [Statistics](#).
 - d. To change the period, choose the period value under the **Period** column, and then choose a new value.
8. To add or edit horizontal annotations, choose **Graph options**:
 - a. To add a horizontal annotation, choose **Add horizontal annotation**.
 - b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.
 - c. For **Value**, choose the pencil icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.
 - d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

i Tip

You can change the fill color by choosing the color square next to the annotation.

- e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.
- f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
- g. To delete an annotation, choose **X** under the **Actions** column.

i Note

You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Graph options**:
 - a. To add a vertical annotation, choose **Add vertical annotation**.
 - b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

i Tip

To show only the date and time, leave the label field blank.

- c. For **Date**, choose the pencil icon next to the current date and time. Then enter the new date and time.
- d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

i Tip

You can change the fill color by choosing the color square next to the annotation.

Note

You can repeat these steps to add multiple vertical annotations to the same graph.

- e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.
 - f. To delete an annotation, choose **X** under the **Actions** column.
10. To customize the Y-axis, choose **Graph options**. Under **Left Y-axis**, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too. You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.
 11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit, and choose **Widget actions**. Choose **Legend**, and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An **X** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Move or resize a graph on a CloudWatch dashboard

You can arrange and resize graphs on your CloudWatch dashboard.

To move a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Do one of the following:
 - Hover over the title of the graph until the selection icon appears. Select and drag the graph to a new location on the dashboard.
 - To move the widget to either the top left or bottom left of the dashboard, choose the vertical ellipsis at the upper right of the widget to open the **Widget actions** menu. Then choose **Move**, and choose where to move the widget to.
4. Choose **Save dashboard**.

To resize a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To increase or decrease the size, hover over the graph and drag the lower right corner of the graph. Stop dragging the corner when you have the size that you want.
4. Choose **Save dashboard**.

To enlarge a graph temporarily

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the graph. Alternatively, hover over the title of the graph and choose **Widget actions**, **Enlarge**.

Rename a graph on a CloudWatch dashboard

You can change the default name that CloudWatch assigns to a graph on your dashboard.

To rename a graph on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.

3. Hover over the title of the graph and choose **Widget actions** and **Edit**.
4. On the **Edit graph** screen, near the top, choose the title of the graph.
5. For **Title**, enter a new name and choose **Ok** (check mark). In the lower-right corner of the **Edit graph** screen, choose **Update widget**.

Add a metrics explorer widget to a CloudWatch dashboard

Metrics explorer widgets include graphs of multiple resources that have the same tag, or share the same resource property such as an instance type. These widgets stay up to date, as resources that match are created or deleted. Adding metrics explorer widgets to your dashboard helps you to troubleshoot your environment more efficiently.

For example, you can monitor your fleet of EC2 instances by assigning tags that represent their environments, such as production or test. You can then use these tags to filter and aggregate the operational metrics, such as `CPUUtilization`, to understand the health and performance of the EC2 instances that are associated with each tag.

The following steps explain how to add a metrics explorer widget to a dashboard using the console. You can also add it programmatically or by using AWS CloudFormation. For more information, see [Metrics Explorer Widget Object Definition](#) and [AWS::CloudWatch::Dashboard](#).

To add a metrics explorer widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard where you want to add the metrics explorer widget.
4. Choose the **+** symbol.
5. Choose **Explorer** and then choose **Next**.

Note

You must be opted in to the new dashboard view to be able to add a Metrics Explorer widget. To opt in, choose **Dashboards** in the navigation pane, then choose **try out the new interface** in the banner at the top of the page.

6. Do one of the following:
 - To use a template, choose **Pre-filled Explorer widget** and then select a template to use.

- To create a custom visualization, choose **Empty Explorer widget**.

7. Choose **Create**.

If you used a template, the widget appears on your dashboard with the selected metrics. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

If you did not use a template, continue to the following steps.

8. In the new widget under **Explorer**, in the **Metrics** box, choose a single metric or all the available metrics from a service.

After you choose a metric, you can optionally repeat this step to add more metrics.

9. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name and then choose the statistic that you want.

10. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the **Production** tag and the M5 instance type, only the resources that match all of these selections are displayed. This is treated as an AND operation.

11. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

Depending on how you choose to aggregate, the result may be a single time series or multiple time series.

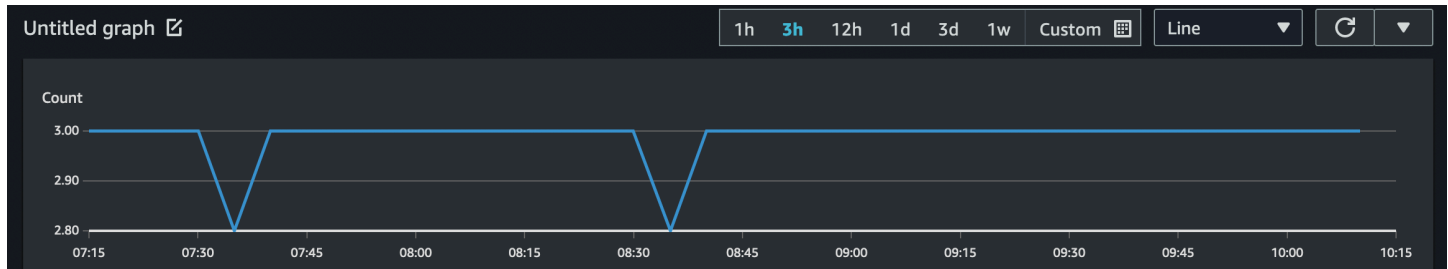
12. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.

13. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.

14. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

Add or remove a line widget on a CloudWatch dashboard

With the line widget, you can compare metrics over periods of time. You also can use the widget's mini-map zoom feature to inspect sections of line graphs without changing between zoomed-in and zoomed-out views. The procedures in this section describe how to add and remove a line widget on a CloudWatch dashboard. For information about using the widget's mini-map zoom feature with line graphs, see [Zooming in on a line or stacked area graph](#).



To add a line widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Line**.
4. Choose **Metrics**.
5. Choose **Browse**, and select the metric that you want to graph.
6. Choose **Create widget**, and then choose **Save dashboard**.

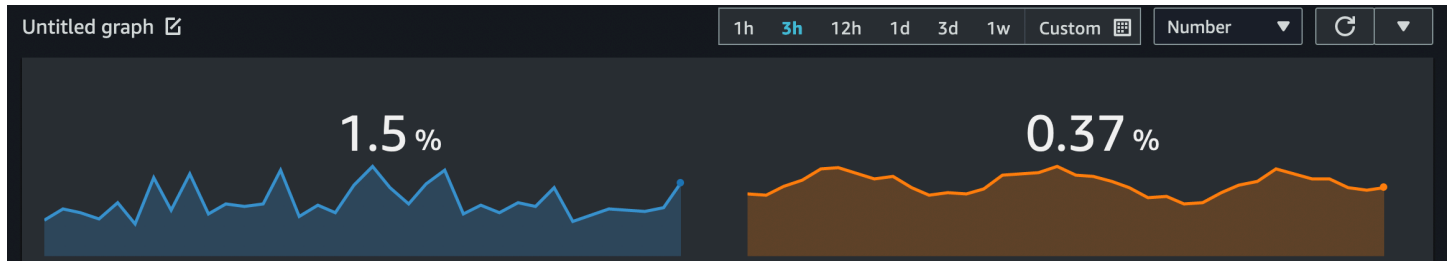
To remove a line widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the line widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Add or remove a number widget from a CloudWatch dashboard

With the number widget, you can look at the latest metric values and trends as soon as they appear. Because the number widget includes the sparkline feature, you can visualize the top and

bottom halves of metric trends in a single graph. The procedures in this section describe how to add and remove a number widget from a CloudWatch dashboard.



Note

Only the new interface supports the sparkline feature. When you create a number widget, the sparkline feature is automatically included.

To add a number widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Number**.
4. In the **Browse** tab, search or browse for the metric that you want to display.
5. (Optional) To change the color of the sparkline feature, choose **Graphed metrics**, and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
6. (Optional) To turn off the sparkline feature, choose **Options**. Under **Sparkline**, the check box.
7. (Optional) To change your number widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).

To set your own time range, choose **Custom**.

- (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
8. (Optional) To have the number widget display an aggregate (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).

To set your own time range, choose **Custom**.

- (Optional) To have this widget display an average of the metric value over the entire time range, instead of the most recent value, choose **Options, Time range value shows the value from the entire time range**.
9. Choose **Create widget**, and choose **Save dashboard**.

Tip

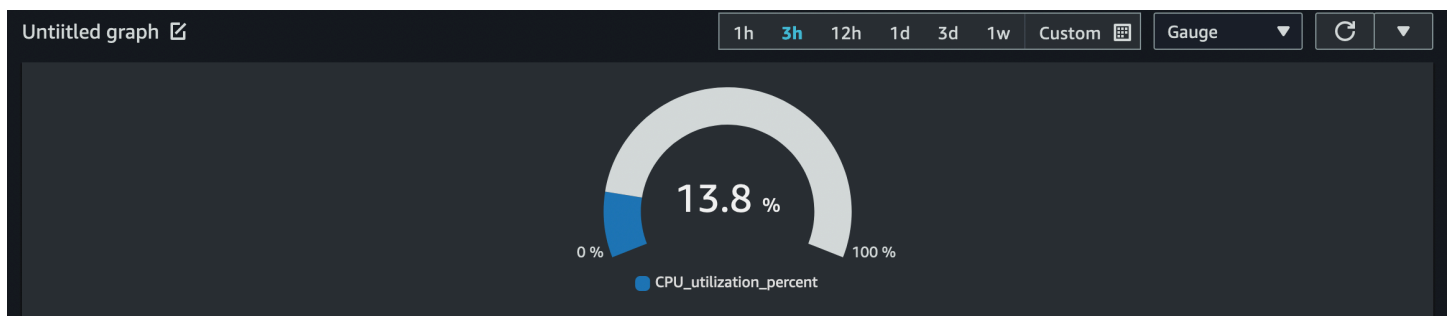
You can turn off the sparkline feature from the number widget on the dashboard screen. In the upper-right corner of the number widget that you want to modify, choose **Widget actions**. Select **Sparkline**, and then choose **Hide sparkline**.

To remove a number widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the number widget that you want to delete.
3. In the upper-right corner of the number widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Add or remove a gauge widget from a CloudWatch dashboard

With the gauge widget, you can visualize metric values that go between ranges. For example, you can use the gauge widget to graph percentages and CPU utilization, so that you can observe and diagnose any performance issues that occur. The procedures in this section describe how to add and remove a gauge widget from a CloudWatch dashboard.



Note

Only the new interface in the CloudWatch console supports creation of the gauge widget. You must set a gauge range when you create this widget.

To add a gauge widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, choose the **+** symbol, and then select **Gauge**.
4. Choose **Browse**, and then select the metric that you want to graph.
5. Choose **Options**. Under **Gauge range**, set values for **Min** and **Max**. For percentages, such as CPU utilization, we recommend that you set the values for Min to 0 and Max to 100.
6. (Optional) To change the color of the gauge widget, choose **Graphed metrics** and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
7. Choose **Create widget**, and choose **Save dashboard**.

To remove a gauge widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the gauge widget you want to delete.
3. In the upper-right corner of the gauge widget that you want to delete, choose **Widget actions**, and choose **Delete**.
4. Choose **Save dashboard**.

Add a custom widget to a CloudWatch dashboard

A *custom widget* is a CloudWatch dashboard widget that can call any AWS Lambda function with custom parameters. It then displays the returned HTML or JSON. Custom widgets are a simple way to build a custom data view on a dashboard. If you can write Lambda code and create HTML, you can create a useful custom widget. Additionally, Amazon provides several prebuilt custom widgets that you can create without any code.

When you create a Lambda function to use as a custom widget, we strongly recommend that you include the prefix **customWidget** in the function name. This helps you know which of your Lambda functions are safe to use when you add custom widgets to your dashboard.

Custom widgets behave like other widgets on your dashboard. They can be refreshed and auto-refreshed, resized, and moved around. They react to the time range of the dashboard.

If you have set up CloudWatch console cross-account functionality, you can add a custom widget created in one account to dashboards in other accounts. For more information, see [Cross-account cross-Region CloudWatch console](#).

You can also use custom widgets on your own website by using the CloudWatch dashboard sharing feature. For more information, see [Sharing CloudWatch dashboards](#).

Topics

- [Details about custom widgets](#)
- [Security and JavaScript](#)
- [Interactivity in the custom widget](#)
- [Create a custom widget](#)
- [Sample custom widgets](#)

Details about custom widgets

Custom widgets work as follows:

1. The CloudWatch dashboard calls the Lambda function containing the widget code. It passes in any custom parameters that are defined in the widget.
2. The Lambda function returns a string of HTML, JSON, or Markdown. Markdown is returned as JSON in the following format:

```
{"markdown": "markdown content"}
```

3. The dashboard displays the returned HTML or JSON.

If the function returns HTML, most HTML tags are supported. You can use Cascading Style Sheets (CSS) styles and Scalable Vector Graphics (SVG) to build sophisticated views.

The default style of HTML elements such as links and tables follow the styling of CloudWatch dashboards. You can customize this style by using inline styles, using the `<style>` tag. You can also deactivate the default styles by including a single HTML element with the class of `cwdb-no-default-styles`. The following example deactivates default styles: `<div class="cwdb-no-default-styles"></div>`.

Every call by a custom widget to Lambda includes a `widgetContext` element with the following contents, to provide the Lambda function developer with useful context information.

```
{
  "widgetContext": {
    "dashboardName": "Name-of-current-dashboard",
    "widgetId": "widget-16",
    "accountId": "012345678901",
    "locale": "en",
    "timezone": {
      "label": "UTC",
      "offsetISO": "+00:00",
      "offsetInMinutes": 0
    },
    "period": 300,
    "isAutoPeriod": true,
    "timeRange": {
      "mode": "relative",
      "start": 1627236199729,
      "end": 1627322599729,
      "relativeStart": 86400012,
      "zoom": {
        "start": 1627276030434,
        "end": 1627282956521
      }
    },
    "theme": "light",
    "linkCharts": true,
    "title": "Tweets for Amazon website problem",
    "forms": {
      "all": {}
    },
    "params": {
      "original": "param-to-widget"
    },
    "width": 588,
    "height": 369
  }
}
```

```
}  
}
```

Default CSS styling

Custom widgets provide the following default CSS styling elements:

- You can use the CSS class **btn** to add a button. It turns an anchor (<a>) into a button as in the following example:

```
<a class="btn" href="https://amazon.com">Open Amazon</a>
```

- You can use the CSS class **btn btn-primary** to add a primary button.
- The following elements are styled by default: **table**, **select**, **headers (h1, h2, and h3)**, **preformatted text (pre)**, **input**, and **text area**.

Using the describe parameter

We strongly recommend that you support the **describe** parameter in your functions, even if it just returns an empty string. If you don't support it, and it is called in the custom widget, it displays widget content as if it was documentation.

If you include the **describe** parameter, the Lambda function returns the documentation in Markdown format and does nothing else.

When you create a custom widget in the console, after you select the Lambda function a **Get documentation** button appears. If you choose this button, the function is invoked with the **describe** parameter and the function's documentation is returned. If the documentation is well-formatted in markdown, CloudWatch parses the first entry in the documentation that is surrounded by three single backtick characters (```) in YAML. Then, it automatically populates the documentation in the parameters. The following is an example of such well-formatted documentation.

```
``` yaml  
echo: <h1>Hello world</h1>
```
```

Security and JavaScript

For security reasons, JavaScript is not allowed in the returned HTML. Removing the JavaScript prevents permission escalation issues, where the writer of the Lambda function injects code that could run with higher permissions than the user viewing the widget on the dashboard.

If the returned HTML contains any JavaScript code or other known security vulnerabilities, it is cleaned from the HTML before it is rendered on the dashboard. For example, the `<iframe>` and `<use>` tags are not allowed and are removed.

Custom Widgets won't run by default in a dashboard. Instead, you must explicitly allow a custom widget to run if you trust the Lambda function that it invokes. You can choose to allow it once or allow always, for both individual widgets and entire dashboard. You can also deny permission for individual widgets and the entire dashboard.

Interactivity in the custom widget

Even though JavaScript is not allowed, there are other ways to allow interactivity with the returned HTML.

- Any element in the returned HTML can be tagged with special configuration in a `<cwdb-action>` tag, which can display information in pop-ups, ask for confirmation on clicks, and call any Lambda function when that element is chosen. For example, you can define buttons that call any AWS API using a Lambda function. The returned HTML can be set to either replace the existing Lambda widget's content, or display inside a modal.
- The returned HTML can include links that open new consoles, open other customer pages, or load other dashboards.
- The HTML can include the `title` attribute for an element, which gives additional information if the user hovers over that element.
- The element can include CSS selectors, such as `:hover`, which can invoke animations or other CSS effects. You can also show or hide elements in the page.

`<cwdb-action>` definition and usage

The `<cwdb-action>` element defines a behavior on the immediately previous element. The content of the `<cwdb-action>` is either HTML to display or a JSON block of parameters to pass to a Lambda function.

The following is an example of a `<cwdb-action>` element.

```
<cwdb-action
  action="call|html"
  confirmation="message"
  display="popup|widget"
  endpoint="<lambda ARN>"
  event="click|dblclick|mouseenter">

  html | params in JSON
</cwdb-action>
```

- **action**— Valid values are `call`, which calls a Lambda function, and `html`, which displays any HTML contained within `<cwdb-action>`. The default is `html`.
- **confirmation**— Displays a confirmation message that must be acknowledged before the action is taken, allowing the customer to cancel.
- **display**— Valid values are `popup` and `widget`, which replaces the content of the widget itself. The default is `widget`.
- **endpoint**— The Amazon Resource Name (ARN) of the Lambda function to call. This is required if `action` is `call`.
- **event**— Defines the event on the previous element that invokes the action. Valid values are `click`, `dblclick`, and `mouseenter`. The `mouseenter` event can be used only in combination with the `html` action. The default is `click`.

Examples

The following is an example of how to use the `<cwdb-action>` tag to create a button that reboots an Amazon EC2 instance using a Lambda function call. It displays the success or failure of the call in a pop-up.

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
  { "instanceId": "i-342389adbfe" }
</cwdb-action>
```

The next example displays more information in a pop-up.

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
```

```
<h1>Big title</h1>
  More info about <b>something important</b>.
</cwdb-action>
```

This example is a **Next** button that replaces the content of a widget with a call to a Lambda function.

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:nextPage">
  { "pageNum": 2 }
</cwdb-action>
```

Create a custom widget

To create a custom widget, you can use one of the samples provided by AWS, or you can create your own. The AWS samples include samples in both JavaScript and Python, and are created by a AWS CloudFormation stack. For a list of samples, see [Sample custom widgets](#).

To create a custom widget in a CloudWatch dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the + symbol.
4. Choose **Custom widget**.
5. Use one of the following methods:
 - To use a sample custom widget provided by AWS, do the following:
 - a. Select the sample in the dropdown box.

The AWS CloudFormation console launches in a new browser. In the AWS CloudFormation console, do the following:

- b. (Optional) Customize the AWS CloudFormation stack name.
 - c. Make selections for any parameters used by the sample.
 - d. Select **I acknowledge that AWS CloudFormation might create IAM resources**, and choose **Create stack**.
- To create your own custom widget provided by AWS, do the following:

- a. Choose **Next**.
- b. Choose to either select your Lambda function from a list, or enter its Amazon Resource Name (ARN). If you select it from a list, also specify the Region where the function is and the version to use.
- c. For **Parameters**, make selections for any parameters used by the function.
- d. Enter a title for the widget.
- e. For **Update on**, configure when the widget should be updated (when the Lambda function should be called again). This can be one or more of the following: **Refresh** to update it when the dashboard auto-refreshes, **Resize** to update it whenever the widget is resized, or **Time Range** to update it whenever the dashboard's time range is adjusted, including when graphs are zoomed into.
- f. If you are satisfied with the preview, choose **Create widget**.

Sample custom widgets

AWS provides sample custom widgets in both JavaScript and Python. You can create these sample widgets by using the link for each widget in this list. Alternatively, you can create and customize a widget by using the CloudWatch console. The links in this list open an AWS CloudFormation console and use an AWS CloudFormation quick-create link to create the custom widget.

You can also access the custom widget samples on [GitHub](#).

Following this list, complete examples of the Echo widget are shown for each language.

JavaScript

Sample custom widgets in JavaScript

- [Echo](#) – A basic echoer that you can use to test how HTML appears in a custom widget, without having to write a new widget.
- [Hello world](#) – A very basic starter widget.
- [Custom widget debugger](#) – A debugger widget that displays useful information about the Lambda runtime environment.
- [Query CloudWatch Logs Insights](#) – Run and edit CloudWatch Logs Insights queries.
- [Run Amazon Athena queries](#) – Run and edit Athena queries.
- [Call AWS API](#) – Call any read-only AWS API and display the results in JSON format.

- [Fast CloudWatch bitmap graph](#) – Render CloudWatch graphs using on the server side, for fast display.
- [Text widget from CloudWatch dashboard](#) – Displays the first text widget from the specified CloudWatch dashboard.
- [CloudWatch metric data as a table](#) – Displays raw CloudWatch metric data in a table.
- [Amazon EC2 table](#) – Displays the top EC2 instances by CPU utilization. This widget also includes a Reboot button, which is disabled by default.
- [AWS CodeDeploy deployments](#) – Displays CodeDeploy deployments.
- [AWS Cost Explorer report](#) – Displays a report on the cost of each AWS service for a selected time range.
- [Display content of external URL](#) – Displays the content of an externally accessible URL.
- [Display an Amazon S3 object](#) – Displays an object in an Amazon S3 bucket in your account.
- [Simple SVG pie chart](#) – Example of a graphical SVG-based widget.

Python

Sample custom widgets in Python

- [Echo](#) – A basic echoer which you can use to test how HTML appears in a custom widget, without having to write a new widget.
- [Hello world](#) – A very basic starter widget.
- [Custom widget debugger](#) – A debugger widget that displays useful information about the Lambda runtime environment.
- [Call AWS API](#) – Call any read-only AWS API and display the results in JSON format.
- [Fast CloudWatch bitmap graph](#) – Render CloudWatch graphs using on the server side, for fast display.
- [Send dashboard snapshot by email](#) – Take a snapshot of the current dashboard and send it to email recipients.
- [Send dashboard snapshot to Amazon S3](#) – Take a snapshot of the current dashboard and store it in Amazon S3.
- [Text widget from CloudWatch dashboard](#) – Displays the first text widget from the specified CloudWatch dashboard.
- [Display content of external URL](#) – Displays the content of an externally accessible URL.

- [RSS reader](#) – Displays RSS feeds.
- [Display an Amazon S3 object](#) – Displays an object in an Amazon S3 bucket in your account.
- [Simple SVG pie chart](#) – Example of a graphical SVG-based widget.

Echo widget in JavaScript

The following is the Echo sample widget in JavaScript.

```
const DOCS = `
## Echo
A basic echo script. Anything passed in the \\\`echo\\\` parameter is returned as
the content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
\\\`yaml
echo: <h1>Hello world</h1>
\\\`
`;

exports.handler = async (event) => {
  if (event.describe) {
    return DOCS;
  }

  let widgetContext = JSON.stringify(event.widgetContext, null, 4);
  widgetContext = widgetContext.replace(/</g, '&lt;');
  widgetContext = widgetContext.replace(/>/g, '&gt;');

  return `${event.echo || ''}<pre>${widgetContext}</pre>`;
};
```

Echo widget in Python

The following is the Echo sample widget in Python.

```
import json
```

```
DOCS = """
## Echo
A basic echo script. Anything passed in the ``echo`` parameter is returned as the
content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '&lt;')
    widgetContext = widgetContext.replace('>', '&gt;')

    return f'{echo}<pre>{widgetContext}</pre>'
```

Echo widget in Java

The following is the Echo sample widget in Java.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

    static String DOCS = ""
        + "## Echo\n"
        + "A basic echo script. Anything passed in the ``echo`` parameter is returned as
the content of the custom widget.\n"
```

```

+ "### Widget parameters\n"
+ "Param | Description\n"
+ "---|---\n"
+ "***echo** | The content to echo back\n\n"
+ "### Example parameters\n"
+ "```yaml\n"
+ "echo: <h1>Hello world</h1>\n"
+ "```\n";

Gson gson = new GsonBuilder().setPrettyPrinting().create();

@Override
public String handleRequest(Event event, Context context) {

    if (event.describe) {
        return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
gson.toJson(event.widgetContext) + "</pre>";
}
}

class Event {

    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}

```

Add or remove a text widget from a CloudWatch dashboard

A text widget contains a block of text in [Markdown](#) format. You can add, edit, or remove text widgets from your CloudWatch dashboard.

To add a text widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the + symbol.
4. Choose **Text**.
5. For **Markdown**, add and format your text using [Markdown](#) and choose **Create widget**.
6. To make the text widget transparent, choose **Transparent background**.
7. Choose **Save dashboard**.

To edit a text widget on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Edit**.
4. Update the text as needed and choose **Update widget**.
5. Choose **Save dashboard**.

To remove a text widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Delete**.
4. Choose **Save dashboard**.

Add or remove an alarm widget from a CloudWatch dashboard

To add an alarm widget to a dashboard, choose one of the following options:

- Add a single alarm in a widget, which displays the graph of the alarm's metric and also displays the alarm status.

- Add an *alarm status widget*, which displays the status of multiple alarms in a grid. Only the alarm names and current status are displayed, Graphs are not displayed. You can include up to 100 alarms in one alarm status widget.

To add a single alarm, including its graph, to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, select the alarm to add, and then choose **Add to Dashboard**.
3. Select a dashboard, choose a widget type (**Line**, **Stacked area**, or **Number**), and then choose **Add to dashboard**.
4. To see your alarm on the dashboard, choose **Dashboards** in the navigation pane and select the dashboard.
5. (Optional) To temporarily make an alarm graph larger, select the graph.
6. (Optional) To change the widget type, hover over the title of the graph, choose **Widget actions**, and then choose **Widget type**.

To add an alarm status widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the + symbol.
4. Choose **Alarm status**.
5. Select the check boxes next to the alarms that you want to add to the widget, and then choose **Create widget**.
6. Choose **Add to dashboard**.

To remove an alarm widget from a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the widget, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**. If you attempt to navigate away from the dashboard before you save your changes, you're prompted to either save or discard your changes.

Add or remove a data table widget from a CloudWatch dashboard

With the data table widget, you can see the raw datapoints of your metric and a quick summary of that raw data. Because the data table widget is not a chart to abstract the actual data away from you, it is easier to understand the datapoints being presented. The procedures in this section describe how to add and remove a data table widget from a CloudWatch dashboard.

| <input type="checkbox"/> | Label | Min | Max | Sum | Average | 11/20
06:00 | 11/20
00:00 | 11/19
18:00 | 11/19
12:00 | 11/
06:00 |
|--------------------------|---------------|------|-------|-------|---------|----------------|----------------|----------------|----------------|--------------|
| <input type="checkbox"/> | TestMetric295 | 991 | 1,000 | 12k | 998 | 996 | 1,000 | 997 | 999 | |
| <input type="checkbox"/> | TestMetric296 | 995 | 1,000 | 12k | 998 | 995 | 1,000 | 1,000 | 998 | |
| <input type="checkbox"/> | TestMetric297 | 991 | 1,000 | 12k | 998 | 998 | 1,000 | 999 | 997 | |
| <input type="checkbox"/> | TestMetric298 | 994 | 1,000 | 12k | 997 | 996 | 999 | 995 | 995 | |
| <input type="checkbox"/> | TestMetric3 | 993 | 1,000 | 12k | 998 | 1,000 | 999 | 999 | 1,000 | |
| <input type="checkbox"/> | TestMetric299 | 995 | 999 | 12k | 998 | 999 | 995 | 999 | 998 | |
| <input type="checkbox"/> | TestMetric30 | 994 | 999 | 12k | 998 | 999 | 998 | 999 | 999 | |
| <input type="checkbox"/> | StackMetric2 | 99 | 99.9 | 1.2k | 99.6 | 99.2 | 99.7 | 99.5 | 99.8 | |
| <input type="checkbox"/> | StackMetric20 | 99 | 100 | 1.19k | 99.5 | 100 | 99.1 | 99.4 | 99.4 | |
| <input type="checkbox"/> | StackMetric21 | 97.5 | 100 | 1.19k | 99.4 | 99.6 | 99.7 | 97.6 | 99.8 | |
| <input type="checkbox"/> | StackMetric22 | 99 | 99.9 | 1.2k | 99.6 | 99.2 | 99.7 | 99.5 | 99.8 | |

Table properties

A data table has a default set of properties that don't require any changes to be made to the options or source. These properties include a sticky label column, all summary columns enabled, datapoints rounded, and their units converted.

Each data table widget can have the following properties. The information about each property includes how to configure it in the JSON source of the dashboard. For more information about dashboard JSON, see [Dashboard Body Structure and Syntax](#).

Summary

Summary columns are a new property introduced with the data table widget. These columns are a specific subset of summaries of your current table. For example, the **Sum** summary is a sum of all displayed datapoints in its row. The summary columns are not the same as CloudWatch statistics. Represented in source as:

```
"table": {
  "summaryColumns": [
    "MIN",
    "MAX",
```



```

        "SUM",
        "AVG"
    ]
},

```

Thresholds

Use this to apply thresholds to your table. When a data point falls within a threshold, its cell is highlighted with the threshold color. Represented in source as:

```

"annotations": {
  "horizontal": [
    {
      "label": string,
      "value": int,
      "fill": "above" | "below"
    }
  ]
}

```

Unit in label column

To display what unit is associated with the metric, you can enable this option to display the unit in the label column beside the label. Represented in source as:

```

"yAxis": {
  "left": {
    "showUnits": true | false
  }
}

```

Invert rows and columns

This transforms the table so that the datapoints swap from columns to rows, and the metrics become columns. Represented in source as:

```

"table": {
  "layout": "vertical" | "horizontal"
}

```

Sticky summary columns

This makes the summary columns sticky, so that they remain in view while you scroll. The label is already sticky. Represented in source as:

```
"table": {
  "stickySummary": true | false
}
```

Display only summary columns

This prevents the columns of datapoints from being displayed, so that only the label and summary columns are displayed. Represented in source as:

```
"table": {
  "showTimeSeriesData": false | true
}
```

Live data

Displays the most recent data point, even if it is not yet fully aggregated. Represented in source as:

```
"liveData": true | false
```

Number widget format

Displays as many digits as can fit in the cell, before rounding and converting. Represented in source as:

```
"singleValueFullPrecision": true | false
```

To add a data table widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and then choose a dashboard.
3. Choose the **+** button, select **Data table**, and choose **Next**.
4. In the **Browse** tab, search or browse for the metrics that you want to display in the table widget. Then select the metrics.

5. (Optional) To change the layout of the table, choose the **Options** tab and select **Invert rows and columns**.

You can also use the **Options** tab to change what columns appear in the table and display the unit being used in the **Label** column.

i **Tip**

To display more accurate thresholds, choose **Show as many digits as can fit before rounding**.

6. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.
7. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.
8. (Optional) To have this widget keep using the time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
9. Choose **Create widget** and then choose **Save dashboard**.

To remove a table widget to a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the widget that you want to remove, choose **Widget actions**, **Delete**.
4. Choose **Save dashboard**.

Link and unlink graphs on a CloudWatch dashboard

You can link the graphs on your dashboard together, so that when you zoom in or zoom out on one graph, the other graphs zoom in or zoom out at the same time. You can unlink graphs to limit zoom to one graph.

To link the graphs on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose **Actions** and then **Link graphs**.

To unlink the graphs on a dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Clear **Actions** and then **Link graphs**.

Sharing CloudWatch dashboards

You can share your CloudWatch dashboards with people who do not have direct access to your AWS account. This enables you to share dashboards across teams, with stakeholders, and with people external to your organization. You can even display dashboards on big screens in team areas, or embed them in Wikis and other webpages.

Warning

All people who you share the dashboard with are granted the permissions listed in [Permissions that are granted to people who you share the dashboard with](#) for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.

The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share dashboards, you can designate who can view the dashboard in three ways:

- Share a single dashboard and designate as many as five email addresses of people who can view the dashboard. Each of these users creates their own password that they must enter to view the dashboard.

- Share a single dashboard publicly, so that anyone who has the link can view the dashboard.
- Share all the CloudWatch dashboards in your account and specify a third-party single sign-on (SSO) provider for dashboard access. All users who are members of this SSO provider's list can access all the dashboards in the account. To enable this, you integrate the SSO provider with Amazon Cognito. The SSO provider must support Security Assertion Markup Language (SAML). For more information about Amazon Cognito, see [What is Amazon Cognito?](#)

Sharing a dashboard doesn't incur charges, but widgets inside a shared dashboard incur charges at standard CloudWatch rates. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

When you share a dashboard, Amazon Cognito resources are created in the US East (N. Virginia) Region.

Important

Do not modify resource names and identifiers that are created by the dashboard sharing process. This includes Amazon Cognito and IAM resources. Modifying these resources can cause unexpected and incorrect functionality of shared dashboards.

Note

If you share a dashboard that has metric widgets with alarm annotations, the people that you share the dashboard with will not see those widgets. They will instead see a blank widget with text saying that the widget is not available. You will still see metric widgets with alarm annotations when you view the dashboard yourself.

Permissions required to share a dashboard

To be able to share dashboards using any of the following methods and to see which dashboards have already been shared, you must be signed on as a user or with an IAM role that has certain permissions.

To be able to share dashboards, your user or IAM role must include the permissions included in the following policy statement:

```

{
  "Effect": "Allow",
  "Action": [
    "iam:CreateRole",
    "iam:CreatePolicy",
    "iam:AttachRolePolicy",
    "iam:PassRole"
  ],
  "Resource": [
    "arn:aws:iam::*:role/service-role/CWDBSharing*",
    "arn:aws:iam::*:policy/*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "cognito-idp:*",
    "cognito-identity:*",
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:GetDashboard",
  ],
  "Resource": [
    "*"
    // or the ARNs of dashboards that you want to share
  ]
}

```

To be able to see which dashboards are shared, but not be able to share dashboards, a user or an IAM role can include a policy statement similar to the following:

```

{
  "Effect": "Allow",
  "Action": [
    "cognito-idp:*",
    "cognito-identity:*"
  ],

```

```
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:ListDashboards",
    ],
    "Resource": [
      "*"
    ]
  }
}
```

Permissions that are granted to people who you share the dashboard with

When you share a dashboard, CloudWatch creates an IAM role in the account which gives the following permissions to the people who you share the dashboard with:

- `cloudwatch:GetInsightRuleReport`
- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

Warning

All people who you share the dashboard with are granted these permissions for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.

The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share a dashboard, by default the permissions that CloudWatch creates restrict access to only the alarms and Contributor Insights rules that are on the dashboard when it is shared. If

you add new alarms or Contributor Insights rules to the dashboard and want them to also be seen by the people who you shared the dashboard with, you must update the policy to allow these resources.

Share a single dashboard with specific users

Use the steps in this section to share a dashboard with as many as five email addresses that you choose.

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms](#).

To share a dashboard with specific users

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Next to **Share your dashboard and require a username and password**, choose **Start sharing**.
6. Under **Add email addresses**, enter the email addresses that you want to share the dashboard with. You can include as many as five email addresses.
7. When you have all the email addresses entered, read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then paste this link into email and send it to the invited users. They automatically receive a separate email with their user name and a temporary password to use to connect to the dashboard.

Share a single dashboard publicly

Follow the steps in this section to share a dashboard publicly. This can be useful to display the dashboard on a big screen in a team room, or embed it in a Wiki page.

Important

Sharing a dashboard publicly makes it accessible to anyone who has the link, with no authentication. Do this only for dashboards that do not contain sensitive information.

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms](#).

To share a dashboard publicly

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Next to **Share your dashboard publicly**, choose **Start sharing**.
6. Enter **Confirm** in the text box.
7. Read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then share this link. Anyone you share the link with can access the dashboard, without providing credentials.

Share all CloudWatch dashboards in the account by using SSO

Use the steps in this section to share all the dashboards in your account with users by using single sign-on (SSO).

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see logs table widgets](#).

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see [Allowing people that you share with to see composite alarms](#).

To share your CloudWatch dashboards with users who are in an SSO provider's list

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions, Share dashboard**.
5. Choose **Go to CloudWatch Settings**.
6. If the SSO provider that you want isn't listed in **Available SSO providers**, choose **Manage SSO providers** and follow the instructions in [Set up SSO for CloudWatch dashboard sharing](#).

Then return to the CloudWatch console and refresh the browser. The SSO provider that you enabled should now appear in the list.

7. Choose the SSO provider that you want in the **Available SSO providers** list.
8. Choose **Save changes**.

Set up SSO for CloudWatch dashboard sharing

To set up dashboard sharing through a third-party single sign-on provider that supports SAML, follow these steps.

⚠ Important

We strongly recommend that you do not share dashboards using a non-SAML SSO provider. Doing so causes a risk of inadvertently allowing third parties to access your account's dashboards.

To set up an SSO provider to enable dashboard sharing

1. Integrate the SSO provider with Amazon Cognito. For more information, see [Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools](#).
2. Download the metadata XML file from your SSO provider.
3. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
4. In the navigation pane, choose **Settings**.
5. In the **Dashboard sharing** section, choose **Configure**.
6. Choose **Manage SSO providers**.

This opens the Amazon Cognito console in the US East (N. Virginia) Region (us-east-1). If you don't see any **User Pools**, the Amazon Cognito console might have opened in a different Region. If so, change the Region to **US East (N. Virginia) us-east-1** and proceed with the next steps.

7. Choose the **CloudWatchDashboardSharing** pool.
8. In the navigation pane, choose **Identity providers**.
9. Choose **SAML**.
10. Enter a name for your SSO provider in **Provider name**.
11. Choose **Select file**, and select the metadata XML file that you downloaded in step 1.
12. Choose **Create provider**.

See how many of your dashboards are shared

You can use the CloudWatch console to see how many of your CloudWatch dashboards are currently being shared with others.

To see how many of your dashboards are being shared

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Settings**.
3. The **Dashboard sharing** section displays how many dashboards are shared.
4. To see which dashboards are shared, choose *number* **dashboards shared** under **Username and password** and under **Public dashboards**.

See which of your dashboards are shared

You can use the CloudWatch console to see which of your dashboards are currently being shared with others.

To see which of your dashboards are being shared

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. In the list of dashboards, see the **Share** column. Dashboards that have the icon in this column filled in are currently being shared.
4. To see which users a dashboard is being shared with, choose the dashboard name, and then choose **Actions, Share dashboard**.

The **Share dashboard *dashboard name*** page displays how the dashboard is being shared. If you want, you can stop sharing the dashboard by choosing **Stop sharing**.

Stop sharing one or more dashboards

You can stop sharing a single shared dashboard, or stop sharing all shared dashboards at once.

To stop sharing a single dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Choose **Stop sharing**.
6. In the confirmation box, choose **Stop sharing**.

To stop sharing all shared dashboards

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. In the **Dashboard sharing** section, choose **Stop sharing all dashboards**.
4. In the confirmation box, choose **Stop sharing all dashboards**.

Review shared dashboard permissions and change permission scope

Use the steps in this section if you want to review the permissions of the users of your shared dashboards, or change the scope of shared dashboard permissions.

To review shared dashboard permissions

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. (Optional) To limit which alarms that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:DescribeAlarms` permission from its current position to a new Allow statement that lists the ARNs of only the alarms that you want to be seen by shared dashboard users. See the following example.

```
{
  "Effect": "Allow",
  "Action": "cloudwatch:DescribeAlarms",
  "Resource": [
    "AlarmARN1",
    "AlarmARN2"
  ]
}
```

If you do this, be sure to remove the `cloudwatch:DescribeAlarms` permission from a section of the current policy that looks like this:

```
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:GetInsightRuleReport",
    "cloudwatch:GetMetricData",
    "cloudwatch:DescribeAlarms",
    "ec2:DescribeTags"
  ],
  "Resource": "*"
}
```

8. (Optional) To limit the scope of what Contributor Insights rules that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:GetInsightRuleReport` from its current position to a new `Allow` statement that lists the ARNs of only the Contributor Insights rules that you want to be seen by shared dashboard users. See the following example.

```
{
  "Effect": "Allow",
  "Action": "cloudwatch:GetInsightRuleReport",
  "Resource": [
    "PublicContributorInsightsRuleARN1",
    "PublicContributorInsightsRuleARN2"
  ]
}
```

If you do this, be sure to remove `cloudwatch:GetInsightRuleReport` from a section of the current policy that looks like this:

```
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:GetInsightRuleReport",
    "cloudwatch:GetMetricData",
    "cloudwatch:DescribeAlarms",
    "ec2:DescribeTags"
  ],
  "Resource": "*"
}
```

Allowing people that you share with to see composite alarms

When you share a dashboard, by default the composite alarm widgets on the dashboard are not visible to the people who you share the dashboard with. For composite alarm widgets to be visible, you need to add a `DescribeAlarms: *` permission to the dashboard sharing policy. That permission would look like this:

```
{
  "Effect": "Allow",
  "Action": "cloudwatch:DescribeAlarms",
  "Resource": "*"
}
```

Warning

The preceding policy statement give access to all alarms in the account. To reduce the scope of `cloudwatch:DescribeAlarms`, you must use a Deny statement. You can add a Deny statement to the policy and specify the ARNs of the alarms that you want to lock down. That deny statement should look similar to the following:

```
{
  "Effect": "Allow",
  "Action": "cloudwatch:DescribeAlarms",
  "Resource": "*"
},
{
  "Effect": "Deny",
  "Action": "cloudwatch:DescribeAlarms",
  "Resource": [
    "SensitiveAlarm1ARN",
    "SensitiveAlarm1ARN"
  ]
}
```

Allowing people that you share with to see logs table widgets

When you share a dashboard, by default the CloudWatch Logs Insights widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both

CloudWatch Logs Insights widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see CloudWatch Logs widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the CloudWatch Logs widgets

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions, Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the log groups that you want shared. See the following example.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:FilterLogEvents",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:GetLogRecord",
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "SharedLogGroup1ARN",
        "SharedLogGroup2ARN"
    ]
},
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes those five permissions with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the log groups that you want shared. For example, if your Resource section for these permissions was the following:


```
"LambdaFunction1ARN",  
"LambdaFunction2ARN"  
]  
}
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes that permission with `*` as the resource, we strongly recommend that you change the policy and specify only the ARNs of the Lambda functions that you want shared. For example, if your Resource section for these permissions was the following:

```
"Resource": "*"
```

Change the policy to specify only the ARNs of the custom widgets that you want shared, as in the following example:

```
"Resource": [  
  "LambdaFunction1ARN",  
  "LambdaFunction2ARN"  
]
```

Use live data

You can choose whether your metric widgets display *live data*. Live data is data published within the last minute that has not been fully aggregated.

- If live data is turned **off**, only data points with an aggregation period of at least one minute in the past are shown. For example, when using 5-minute periods, the data point for 12:35 would be aggregated from 12:35 to 12:40, and displayed at 12:41.
- If live data is turned **on**, the most recent data point is shown as soon as any data is published in the corresponding aggregation interval. Each time you refresh the display, the most recent data point may change as new data within that aggregation period is published. If you use a cumulative statistic such as **Sum** or **Sample Count**, using live data may result in a dip at the end of your graph.

You can choose to enable live data for a whole dashboard, or for individual widgets on the dashboard.

To choose whether to use live data on your entire dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To permanently turn live data on or off for all widgets on the dashboard, do the following:
 - a. Choose **Actions, Settings, Bulk update live data**.
 - b. Choose **Live Data on** or **Live Data off**, and choose **Set**.
4. To temporarily override the live data settings of each widget, choose **Actions**. Then, under **Overrides**, next to **Live data**, do one of the following:
 - Choose **On** to temporarily turn on live data for all widgets.
 - Choose **Off** to temporarily turn off live data for all widgets.
 - Choose **Do not override** to preserve each widget's live data setting.

To choose whether to use live data on a single widget

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select a widget, and choose **Actions, Edit**.
4. Choose the **Graph options** tab.
5. Select or clear the check box under **Live Data**.

Viewing an animated dashboard

You can view an animated dashboard that replays CloudWatch metric data that was captured over time. This can help you see trends, make presentations, or analyze issues after they occur.

Animated widgets in the dashboard include line widgets, stacked area widgets, number widgets, and metrics explorer widgets. Pie graphs, bar charts, text widgets, and logs widgets are displayed in the dashboard but are not animated.

To view an animated dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.

3. Choose the name of the dashboard.
4. Choose **Actions, Replay dashboard**.
5. (Optional) By default, when you start the animation, it appears as a sliding window. If you want the animation to appear as a point-by-point animation instead, choose the magnifying glass icon while the animation is paused and reset the zoom.
6. To start the animation, choose the Play button. You can also choose the back and forward buttons to move to other points in time.
7. (Optional) To change the time window for the animation, choose the calendar and select the time period.
8. To change the speed of the animation, choose **Auto speed** and select the new speed.
9. When you are finished, choose **Exit animate**.

Add a CloudWatch dashboard to your favorites list

In the CloudWatch console, you can add dashboards, alarms, and log groups to a favorites list. You can access the favorites list from the navigation pane. The following procedure describes how to add a dashboard to the favorites list.

To add a dashboard to the favorites list

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. From the list of dashboards, select the star symbol next to the name of the dashboard that you want to favorite.
 - (Optional) You also can favorite a dashboard by selecting a dashboard from the list and choosing the star symbol next to the dashboard name.
4. To access the favorites list, choose **Favorites and recents** in the navigation pane. The menu contains two columns. One contains your favorite dashboards, alarms, and log groups, and the other column contains the dashboards, alarms, and log groups that you recently visited.

Tip

You can favorite dashboards, as well as alarms and log groups, from the **Favorites and recents** menu in the CloudWatch console navigation pane. Under the **Recently visited**

column, hover over the dashboard that you want to favorite, and choose the star symbol next to it.

Change the period override setting or refresh interval for the CloudWatch dashboard

You can specify how the period setting of graphs added to this dashboard are retained or modified.

When an auto period or persisted time range is applied to a widget, the overall time range of the graph can affect the periods that you have set.

- If the time range is one day or less, period settings are not changed.
- If the time range is between one day and three days, periods set to below five minutes are changed to 5 minutes.
- If the time range is more than three days, periods set to below one hour are changed to one hour.

The following steps explain how to use the console to change the period override options. You can also change them by using the `periodOverride` field in the JSON structure of the dashboard. For more information, see [Dashboard Body Overall Structure](#).

To change the period override options

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Actions**.
3. Under **Period override**, choose one of the following:
 - Choose **Auto** to have the period of the metrics on each graph automatically adapt to the dashboard's time range.
 - Choose **Do not override** to ensure that the period setting of each graph is always obeyed.
 - Choose one of the other options to cause graphs added to the dashboard to always adapt that chosen time as their period setting.

The **Period override** always reverts to **Auto** when the dashboard is closed or the browser is refreshed. Different settings for **Period override** can't be saved.

You can change how often the data on your CloudWatch dashboard is refreshed.

To change the dashboard refresh interval

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. On the **Refresh options** menu (upper-right corner), choose **10 Seconds**, **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Change the time range or time zone format of a CloudWatch dashboard

You can change the time range to display dashboard data over minutes, hours, days, or weeks. You also can change the time zone format to display dashboard data in UTC or local time. Local time is the time zone that's specified in your computer's operating system.

Note

If you create a dashboard with graphs that contain 100 or more high-resolution metrics, we recommend that you don't set the time range to longer than 1 hour. For more information, see [High-resolution metrics](#).

Note

If the time range of a dashboard is shorter than the period used for a widget on the dashboard, the following happens:

- The widget is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
- The start time of the period for this data point is adjusted backwards to ensure that at least one data point can be displayed.

New console

To change the dashboard time range

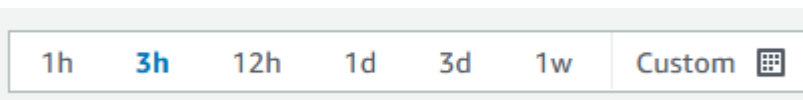
1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
 - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, or **1w**).
 - Alternatively, you can choose one of the following custom time range options:
 - Choose **Custom**, and then choose the **Relative** tab. Choose a time range from 1 minute to 15 months.
 - Choose **Custom**, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

Tip

If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper area of the dashboard, choose **Custom**.



4. In the upper-right corner of the box that appears, select **UTC** or **Local time** from the dropdown.
5. Choose **Apply**.

Old console

To change the dashboard time range

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
 - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).
 - Alternatively, you can choose one of the following custom time range options:
 - Choose the **custom** dropdown, and then choose the **Relative** tab. Select one of the predefined ranges, which span from 1 minute to 15 months.
 - Choose the **custom** dropdown, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

Tip

If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the dashboard screen, choose the **Custom** dropdown.
4. In the upper-right corner of the box that appears, select **UTC** or **Local timezone** from the dropdown.

Use Amazon CloudWatch metrics

Metrics are data about the performance of your systems. By default, many services provide free metrics for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). You can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. Amazon CloudWatch can load all the metrics in your account (both AWS resource metrics and application metrics that you provide) for search, graphing, and alarms.

Metric data is kept for 15 months, enabling you to view both up-to-the-minute data and historical data.

To graph metrics in the console, you can use CloudWatch Metrics Insights, a high-performance SQL query engine that you can use to identify trends and patterns within all your metrics in real time.

Contents

- [Basic monitoring and detailed monitoring](#)
- [Query your metrics with CloudWatch Metrics Insights](#)
- [Use metrics explorer to monitor resources by their tags and properties](#)
- [Use metric streams](#)
- [View available metrics](#)
- [Graphing metrics](#)
- [Using CloudWatch anomaly detection](#)
- [Use metric math](#)
- [Use search expressions in graphs](#)
- [Get statistics for a metric](#)
- [Publish custom metrics](#)

Basic monitoring and detailed monitoring

CloudWatch provides two categories of monitoring: *basic monitoring* and *detailed monitoring*.

Many AWS services offer basic monitoring by publishing a default set of metrics to CloudWatch with no charge to customers. By default, when you start using one of these AWS services, basic

monitoring is automatically enabled. For a list of services that offer basic monitoring, see [AWS services that publish CloudWatch metrics](#).

Detailed monitoring is offered by only some services. It also incurs charges. To use it for an AWS service, you must choose to activate it. For more information about pricing, see [Amazon CloudWatch pricing](#).

Detailed monitoring options differ based on the services that offer it. For example, Amazon EC2 detailed monitoring provides more frequent metrics, published at one-minute intervals, instead of the five-minute intervals used in Amazon EC2 basic monitoring. Detailed monitoring for Amazon S3 and Amazon Managed Streaming for Apache Kafka means more fine-grained metrics.

In different AWS services, detailed monitoring also has different names. For example, in Amazon EC2 it is called detailed monitoring, in AWS Elastic Beanstalk it is called enhanced monitoring, and in Amazon S3 it is called request metrics.

Using detailed monitoring for Amazon EC2 helps you better manage your Amazon EC2 resources, so that you can find trends and take action faster. For Amazon S3 request metrics are available at one-minute intervals to help you quickly identify and act on operational issues. On Amazon MSK, when you enable the PER_BROKER, PER_TOPIC_PER_BROKER, or PER_TOPIC_PER_PARTITION level monitoring, you get additional metrics that provide more visibility.

The following table lists the services that offer detailed monitoring. It also includes links to the documentation for those services that explain more about the detailed monitoring and provide instructions for how to activate it.

| Service | Documentation |
|--------------------|--|
| Amazon API Gateway | Dimensions for API Gateway metrics |
| Amazon CloudFront | Viewing additional CloudFront distribution metrics |
| Amazon EC2 | Enable or turn off detailed |

| Service | Documentation |
|-----------------------------|---|
| | monitoring for your instances |
| Elastic Beanstalk | Enhanced health reporting and monitoring |
| Amazon Kinesis Data Streams | Enhanced Shard-level Metrics |
| Amazon MSK | Amazon MSK Metrics for Monitoring with CloudWatch |
| Amazon S3 | Amazon S3 request metrics in CloudWatch |
| Amazon SES | Collect CloudWatch detailed monitoring metrics using Amazon SES event publishing. |

Additionally, CloudWatch offers out-of-the-box monitoring solutions with more detailed metrics and pre-created dashboards for some AWS services, as shown in the following table.

| Service | Feature documentation |
|---------|---------------------------------|
| Lambda | Lambda Insights |

| Service | Feature documentation |
|------------|--|
| Amazon ECS | Container Insights for Amazon ECS |
| Amazon EKS | Container Insights for Amazon EKS and Kubernetes |

Query your metrics with CloudWatch Metrics Insights

CloudWatch Metrics Insights is a powerful high-performance SQL query engine that you can use to query your metrics at scale. You can identify trends and patterns within all of your CloudWatch metrics in real time.

You can also set alarms on any Metrics Insights queries that return a single time series. This can be especially useful to create alarms that watch aggregated metrics across a fleet of your infrastructure or applications. Create the alarm once, and it dynamically adjusts as resources are added to or removed from the fleet.

You can perform a CloudWatch Metrics Insights query in the console with the CloudWatch Metrics Insights query editor. You can also perform a CloudWatch Metrics Insights query with the AWS CLI or an AWS SDK by running `GetMetricData` or `PutDashboard`. There's no charge for queries that you run with the CloudWatch Metrics Insights query editor. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

With the CloudWatch Metrics Insights query editor, you can choose from a variety of prebuilt sample queries and also create your own queries. As you create your queries, you can use a builder view to browse your existing metrics and dimensions. Alternatively, use an editor view to manually write queries.

You can also use natural language to create CloudWatch Metrics Insights queries. To do so, ask questions about or describe the data you're looking for. This AI-assisted capability generates a query based on your prompt and provides a line-by-line explanation of how the query works. For

more information, see [Use natural language to generate and update CloudWatch Metrics Insights queries](#).

With Metrics Insights, you can run queries at scale. With the **GROUP BY** clause, you can group your metrics in real time into separate time series per specific dimension value. Because Metrics Insights queries include an **ORDER BY** ability, you can use Metrics Insights to make "Top N" type queries. For example, "Top N" type queries can scan millions of metrics in your account and return the 10 instances that consume the most CPU. This can help you pinpoint and remedy latency issues in your applications.

Topics

- [Build your queries](#)
- [Metrics Insights query components and syntax](#)
- [Create alarms on Metrics Insights queries](#)
- [Use Metrics Insights queries with metric math](#)
- [Use natural language to generate and update CloudWatch Metrics Insights queries](#)
- [SQL inference](#)
- [Metrics Insights sample queries](#)
- [Metrics Insights limits](#)
- [Metrics Insights glossary](#)
- [Troubleshooting Metrics Insights](#)

Build your queries

You can run a CloudWatch Metrics Insights query using the CloudWatch console, the AWS CLI, or the AWS SDKs. Queries run in the console are free of charge. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

For more information about using the AWS SDKs to perform a Metrics Insights query, see [GetMetricData](#).

To run a query using the CloudWatch console, follow these steps:

To query your metrics using Metrics Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Metrics, All metrics**.
3. Choose the **Query** tab.
4. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. If you are satisfied with this query, you can skip the rest of this procedure. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
5. To create your own query, you can use the **Builder** view, the **Editor** view, and also use a combination of both. You can switch between the two views anytime and see your work in progress in both views.

In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

6. When you are satisfied with your query, choose **Run**.
7. (Optional) Another way to edit a query that you have graphed is to choose the **Graphed metrics** tab and choose the edit icon next to the query formula in the **Details** column.
8. (Optional) To remove a query from the graph, choose **Graphed metrics** and choose the **X** icon at the right side of the row that displays your query.

Metrics Insights query components and syntax

CloudWatch Metrics Insights syntax is as follows.

```
SELECT FUNCTION(metricName)  
FROM namespace | SCHEMA(...)  
[ WHERE labelKey OPERATOR labelValue [AND ... ] ]  
[ GROUP BY labelKey [ , ... ] ]  
[ ORDER BY FUNCTION() [ DESC | ASC ] ]  
[ LIMIT number ]
```

The possible clauses in a Metrics Insights query are as follows. None of the keywords are case sensitive, but the identifiers such as the names of metrics, namespaces, and dimensions are case sensitive.

SELECT

Required. Specifies the function to use to aggregate observations in each time bucket (determined by the provided period). Also specifies the name of the metric to query.

The valid values for **FUNCTION** are AVG, COUNT, MAX, MIN, and SUM.

- AVG calculates the average of the observations matched by the query.
- COUNT returns the count of the observations matched by the query.
- MAX returns the maximum value of the observations matched by the query.
- MIN returns the minimum value of the observations matched by the query.
- SUM calculates the sum of the observations matched by the query.

FROM

Required. Specifies the source of the metric. You can specify either the metric namespace that contains the metric that is to be queried, or a **SCHEMA** table function. Examples of metric namespaces include "AWS/EC2", "AWS/Lambda", and metric namespaces that you have created for your custom metrics.

Metric namespaces that include / or any other character that is not a letter, number, or underscore must be surrounded by double quotation marks. For more information, see [What needs quotation marks or escape characters?](#)

SCHEMA

An optional table function that can be used within a **FROM** clause. Use **SCHEMA** to scope down the query results to only the metrics that exactly match a list of dimensions, or to metrics that have no dimensions.

If you use a **SCHEMA** clause, it must contain at least one argument, and this first argument must be the metric namespace being queried. If you specify **SCHEMA** with only this namespace argument, the results are scoped down to only metrics that do not have any dimensions.

If you specify **SCHEMA** with additional arguments, the additional arguments after the namespace argument must be *label* keys. Label keys must be dimension names. If you specify one or more of these label keys, the results are scoped down to only those metrics that have that exact set of dimensions. The order of these label keys does not matter.

For example:

- **SELECT AVG(CPUUtilization) FROM "AWS/EC2"** matches all CPUUtilization metrics in the AWS/EC2 namespace, no matter their dimensions, and returns a single aggregated time series.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2")** matches only the CPUUtilization metrics in the AWS/EC2 namespace that do not have any dimensions defined.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId)** matches only the CPUUtilization metrics that were reported to CloudWatch with exactly one dimension, InstanceId.
- **SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)** matches only the RequestCount metrics that were reported to CloudWatch from AWS/ApplicationELB with exactly two dimensions, LoadBalancer and AvailabilityZone.

WHERE

Optional. Filters the results to only those metrics that match your specified expression using specific label values for one or more label keys. For example, **WHERE InstanceType = 'c3.4xlarge'** filters the results to only c3.4xlarge instance types, and **WHERE InstanceType != 'c3.4xlarge'** filters the results to all instance types except c3.4xlarge.

When you run a query in a monitoring account, you can use **WHERE AWS.AccountId** to limit results to only the account that you specify. For example, **WHERE AWS.AccountId=444455556666** queries metrics from only account 444455556666. To limit your query to only metrics in the monitoring account itself, use **WHERE AWS.AccountId=CURRENT_ACCOUNT_ID()**.

Label values must always be enclosed with single quotation marks.

Supported operators

The **WHERE** clause supports the following operators:

- **=** Label value must match the specified string.
- **!=** Label value must not match the specified string.
- **AND** Both conditions that are specified must be true to match. You can use multiple **AND** keywords to specify two or more conditions.

GROUP BY

Optional. Groups the query results into multiple time series, each one corresponding to a different value for the specified label key or keys. For example, using `GROUP BY InstanceId` returns a different time series for each value of `InstanceId`. Using `GROUP BY ServiceName, Operation` creates a different time series for each possible combination of the values of `ServiceName` and `Operation`.

With a **GROUP BY** clause, by default the results are ordered in alphabetical ascending order, using the sequence of labels specified in the **GROUP BY** clause. To change the order of the results, add an **ORDER BY** clause to your query.

When you run a query in a monitoring account, you can use `GROUP BY AWS.AccountId` to group the results based on the accounts they are from.

Note

If some of the matching metrics don't include a specific label key specified in the **GROUP BY** clause, a null group named `Other` is returned. For example, if you specify `GROUP BY ServiceName, Operation` and some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

ORDER BY

Optional. Specifies the order to use for the returned time series, if the query returns more than one time series. The order is based on the values found by the **FUNCTION** that you specify in the **ORDER BY** clause. The **FUNCTION** is used to calculate a single scalar value from each returned time series, and that value is used to determine the order.

You also specify whether to use ascending **ASC** or descending **DESC** order. If you omit this, the default is ascending **ASC**.

For example, adding an `ORDER BY MAX() DESC` clause orders the results by the maximum data point observed within the time range, in descending order: meaning that the time series that has the highest maximum data point is returned first.

The valid functions to use within an **ORDER BY** clause are `AVG()`, `COUNT()`, `MAX()`, `MIN()`, and `SUM()`.

If you use an **ORDER BY** clause with a **LIMIT** clause, the resulting query is a "Top N" query. **ORDER BY** is also useful for queries that might return a large number of metrics, because each query can return no more than 500 time series. If a query matches more than 500 time series, and you use an **ORDER BY** clause, the time series are sorted and then the 500 time series that come first in the sort order are the ones that are returned.

LIMIT

Optional. Limits the number of time series returned by the query to the value that you specify. The maximum value that you can specify is 500, and a query that does not specify a **LIMIT** can also return no more than 500 time series.

Using a **LIMIT** clause with an **ORDER BY** clause gives you a "Top N" query.

What needs quotation marks or escape characters?

In a query, label values must always be surrounded with single quotation marks. For example, **SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet'**.

Metric namespaces, metric names, and label keys that contain characters other than letters, numbers, and underscore (`_`) must be surrounded by double quote marks. For example, **SELECT MAX("My.Metric")**.

If one of these contains a double quotation mark or single quotation mark itself (such as `Bytes"Input"`), you must escape each quotation mark with a backslash, as in **SELECT AVG("Bytes \"Input\"")**.

If a metric namespace, metric name, or label key, contains a word that is a reserved keyword in Metrics Insights, these must also be enclosed in double quotation marks. For example, if you have a metric named `LIMIT`, you would use **SELECT AVG("LIMIT")**. It is also valid to enclose any namespace, metric name, or label in double quotation marks even if it does not include a reserved keyword.

For a complete list of reserved keywords, see [Reserved keywords](#).

Build a rich query step by step

This section illustrates building a full example that uses all possible clauses, step by step.

We start with the following query, which aggregates all of the Application Load Balancer RequestCount metrics that are collected with both the dimensions LoadBalancer and AvailabilityZone.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

Now, if we want to see metrics only from a specific load balancer, we can add a **WHERE** clause to limit the metrics returned to only those metrics where the value of the LoadBalancer dimension is app/load-balancer-1.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

The preceding query aggregates the RequestCount metrics from all Availability Zones for this load balancer into one time series. If we want to see different time series for each Availability Zone, we can add a **GROUP BY** clause.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

Next, we might want to order these results to see the highest values first. The following **ORDER BY** clause orders the time series in descending order, by the maximum value reported by each time series during the query time range:

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
```

Finally, if we are primarily interested in a "Top N" type of query, we can use a **LIMIT** clause. This final example limits the results to only the time series with the five highest MAX values.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

```
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

Cross-account query examples

These examples are valid when run in an account set up as a monitoring account in CloudWatch cross-account observability.

The following example searches all Amazon EC2 instances in the source account 123456789012 and returns the average.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = '123456789012'
```

The following example queries the CPUUtilization metric in AWS/EC2 in all the linked source accounts, and groups the results by account ID and instance type.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
GROUP BY AWS.AccountId, InstanceType
```

The following example queries the CPUUtilization in the monitoring account itself.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = CURRENT_ACCOUNT_ID()
```

Reserved keywords

The following are reserved keywords in CloudWatch Metrics Insights. If any of these words are in a namespace, metric name, or label key in a query, you must enclose them in double quote marks. Reserved keywords are not case sensitive.

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
"ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
"ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN"
"ASINH" "ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
```

```

"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE"
"AVERAGE" "AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE"
"BT" "BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N"
"CAST" "CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
"CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID"
"COLUMN" "COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
"CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
"COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
"CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
"CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE"
"DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
"DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES"
"DEL" "DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS"
"DICTIONARY" "DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE"
"DUAL" "DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING"
"END_FRAME" "END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
"EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
"FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
"FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
"GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
"HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE"
"ILIKE" "IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
"INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"

```

"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT"
 "INTERSECTION" "INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
 "IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
 "JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
 "JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS"
 "KILL" "KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
 "LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
 "LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
 "LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
 "LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
 "LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY"
 "LT" "MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
 "MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
 "MATERIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
 "MCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
 "MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
 "MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG"
 "MLOAD" "MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
 "MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM"
 "MULTISET" "NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
 "NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
 "NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
 "NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
 "NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
 "OCCURRENCES_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
 "OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
 "OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE"
 "OPTIMIZER_COSTS" "OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
 "OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
 "PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
 "PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
 "PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
 "PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
 "POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
 "PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE"
 "PRIVILEGES" "PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
 "PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
 "QUERYNO" "RADIANS" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
 "RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
 "REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
 "REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
 "RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
 "REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
 "RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"

```

"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
"ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
"ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
"SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL"
"SELECT" "SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER"
"SET" "SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT"
"SOME" "SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SQLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
"SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQRT" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
"STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
"SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"
"SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN"
"THRESHOLD" "TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION"
"TRANSLATE" "TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"
"UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"
"UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF"
"VARBINARY" "VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"
"VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN"
"WITHIN_GROUP" "WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"

```

Create alarms on Metrics Insights queries

You can create alarms on Metrics Insights queries. This helps you have alarms that track multiple resources without needing to be updated later. The query catches new resources and resources that change. For example, you can create an alarm that watches the CPU utilization of your fleet, and the alarm automatically evaluates new instances that you launch after creating the alarm.

In a monitoring account that is set up for CloudWatch cross-account observability, your Metrics Insights alarms can watch resources in source accounts and in the monitoring account itself. For more information about how to limit your alarm queries to a specific account or to group the results by account ID, see the WHERE and GROUP BY sections in [Metrics Insights query components and syntax](#).

Contents

- [Create a Metrics Insights alarm](#)
- [Partial data cases](#)

Create a Metrics Insights alarm

To create an alarm on a Metrics Insights query using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Choose the **Query** tab.
4. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
5. To create your own query, you can use the **Builder** view, the **Editor** view, or a combination of both. You can switch between the two views anytime and see your work in progress in both views.

In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

Important

To set an alarm on a Metrics Insights query, the query must return a single time series. If it contains a GROUP BY statement, the GROUP BY statement must be wrapped inside

a metric math expression that returns only one time series as the final result of the expression.

6. When you are satisfied with your query, choose **Run**.
7. Choose **Create alarm**.
8. Under **Conditions**, specify the following:
 - a. For **Whenever *metric* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
 - b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).
 - c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
9. Choose **Next**.
10. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.
11. To have the alarm perform Auto Scaling, EC2, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms and Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

12. When finished, choose **Next**.
13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

To create an alarm on a Metrics Insights query using the AWS CLI

- Use the `put-metric-alarm` command and specify a Metrics Insights query in the `metrics` parameter. For example, the following command sets an alarm that goes into ALARM state if any of your instances go above 50% in CPU utilization.

```
aws cloudwatch put-metric-alarm --alarm-name Metrics-Insights-alarm --
evaluation-periods 1 --comparison-operator GreaterThanThreshold --metrics
'[{"Id":"m1","Expression":"SELECT MAX(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
InstanceId)", "Period":60}]' --threshold 50
```

Partial data cases

If the Metrics Insights query used for the alarm matches more than 10,000 metrics, the alarm is evaluated based on the first 10,000 metrics that the query finds. This means that the alarm is being evaluated on partial data.

You can use the following methods to find whether a Metrics Insights alarm is currently evaluating its alarm state based on partial data:

- In the console, if you choose an alarm to see the **Details** page, the message **Evaluation warning: Not evaluating all data** appears on that page.
- You see the value `PARTIAL_DATA` in the `EvaluationState` field when you use the [describe-alarms](#) AWS CLI command or the [DescribeAlarms](#) API.

Alarms also publish events to Amazon EventBridge when it goes into the partial data state, so you can create an EventBridge rule to watch for these events. In these events, the `evaluationState` field has the value `PARTIAL_DATA`. The following is an example.

```
{
  "version": "0",
  "id": "12345678-3bf9-6a09-dc46-12345EXAMPLE",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2022-11-08T11:26:05Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:my-alarm-name"
  ],
  "detail": {
    "alarmName": "my-alarm-name",
    "state": {
      "value": "ALARM",
      "reason": "Threshold Crossed: 3 out of the last 3 datapoints [20000.0 (08/11/22 11:25:00), 20000.0 (08/11/22 11:24:00), 20000.0 (08/11/22 11:23:00)] were greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":\"2022-11-08T11:26:05.399+0000\",\"startDate\":\"2022-11-08T11:23:00.000+0000\",\"period\":60,\"recentDatapoints\":[20000.0,20000.0,20000.0],\"threshold\":0.0,\"evaluatedDatapoints\":[{\"timestamp\":\"2022-11-08T11:25:00.000+0000\",\"value\":20000.0}]}",
      "timestamp": "2022-11-08T11:26:05.401+0000",
      "evaluationState": "PARTIAL_DATA"
    },
    "previousState": {
      "value": "INSUFFICIENT_DATA",
      "reason": "Unchecked: Initial alarm creation",
      "timestamp": "2022-11-08T11:25:51.227+0000"
    },
    "configuration": {
      "metrics": [
        {
          "id": "m2",
          "expression": "SELECT SUM(PartialDataTestMetric) FROM partial_data_test",
          "returnData": true,
          "period": 60
        }
      ]
    }
  }
}
```

```
}  
  ]  
} ]  
} }  
}
```

If the query for the alarm includes a **GROUP BY** statement that initially returns more than 500 time series, the alarm is evaluated based on the first 500 time series that the query finds. However, if you use an **ORDER BY** clause, then all the time series that the query finds are sorted, and the 500 that have the highest or lowest values according to your **ORDER BY** clause are used to evaluate the alarm.

Use Metrics Insights queries with metric math

You can use a Metrics Insights query as an input to a metric math function. For more information about metric math, see [Use metric math](#).

A Metrics Insights query that does not include a **GROUP BY** clause returns a single time series. Therefore, its returned results can be used with any metric math function that takes a single time series as input.

A Metrics Insights query that includes a **GROUP BY** clause returns multiple time series. Therefore, its returned results can be used with any metric math function that takes an array of time series as input.

For example, the following query returns the total number of bytes downloaded for each bucket in the Region, as an array of time series:

```
SELECT SUM(BytesDownloaded)  
FROM SCHEMA("AWS/S3", BucketName, FilterId)  
WHERE FilterId = 'EntireBucket'  
GROUP BY BucketName
```

On a graph in the console or in a [GetMetricData](#) operation, the results of this query are q1. This query returns the result in bytes, so if you want to see the result as MB instead, you can use the following math function:

```
q1/1024/1024
```

Use natural language to generate and update CloudWatch Metrics Insights queries

Note

This feature is generally available in US East (N. Virginia), US West (Oregon), and Asia Pacific (Tokyo) for CloudWatch.

CloudWatch supports a natural language query capability to help you generate and update queries for [CloudWatch Metrics Insights](#) and [CloudWatch Logs Insights](#).

With this capability, you can ask questions about or describe the CloudWatch data you're looking for in plain English. The natural language capability generates a query based on a prompt that you enter and provides a line-by-line explanation of how the query works. You can also update your query to further investigate your data.

Depending on your environment, you can enter prompts like "Which Amazon Elastic Compute Cloud instance has the highest network out?" and "Show me the top 10 Amazon DynamoDB Tables by consumed reads."

To generate a CloudWatch Metrics Insights query with this capability, open the CloudWatch Metrics Insights query editor in the *builder* or *editor* view and choose **Generate query**.

Important

To use the natural language query capability, you must use the [CloudWatchFullAccess](#), [CloudWatchReadOnlyAccess](#), [CloudWatchFullAccessV2](#), [AdministratorAccess](#), or [ReadOnlyAccess](#) policy.

You can also include the `ccloudwatch:GenerateQuery` action in a new or existing customer managed or inline policy.

Example queries

The examples in this section describe how to generate and update queries using the natural language capability.

Note

For more information on the CloudWatch Metrics Insights query editor and syntax, see [CloudWatch Metrics Insights query components and syntax](#).

Example: Generate a natural language query

To generate a query using natural language, enter a prompt and choose **Generate new query**. This example shows a query that performs a basic search.

Prompt

The following is an example of a prompt that directs the capability to search for the top 10 DynamoDB Tables that consume the most read capacity.

```
Show top 10 DynamoDB Tables by consumed reads
```

Query

The following is an example of a query that the natural language capability generates based on the prompt. Notice how the prompt appears in a comment before the query. After the query, you can read an explanation that describes how the query works.

```
# Show top 10 DynamoDB Tables by consumed reads
SELECT SUM("ConsumedReadCapacityUnits")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query selects the sum of consumed read capacity units for each DynamoDB table,
groups the results by table name, orders the results from highest to lowest read
capacity consumption, and limits the results to the top 10 tables.
```

Note

To turn off the appearance of your prompt and the explanation of how the query works, use the gear icon in your editor.

Example: Update a natural language query

You can update a query by editing the initial prompt and then choosing **Update query**.

Updated prompt

The following example shows an updated version of the previous prompt. Instead of a prompt that searches for the top 10 DynamoDB Tables that consume the most read capacity, this prompt now directs the capability to sort the results by the number of bytes returned.

```
Sort by bytes returned instead
```

Updated query

The following is an example of the updated query. Notice how the updated prompt appears in a comment before the updated query. After the query, you can read an explanation that describes how the original query has been updated.

```
# Sort by bytes returned instead
SELECT SUM("ReturnedBytes")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query modifies the original query to select the sum of returned bytes instead
of consumed read capacity units, and orders the results from highest to lowest sum of
returned bytes, limiting the results to the top 10 tables.
```

Opting out of using your data for service improvement

The natural language prompt data you provide to train the AI model and generate relevant queries is used solely to provide and maintain your service. This data might be used to improve the quality of CloudWatch Metrics Insights. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see [AWS Service Terms](#) and [AWS responsible AI policy](#).

You can opt out of having your content used to develop or improve the quality of natural language queries by creating an AI service opt-out policy. To opt-out of data collection for all CloudWatch AI features, including the query generation capability, you must create an opt-out policy for CloudWatch. For more information, see [AI services opt-out policies](#) in the *AWS Organizations User Guide*.

SQL inference

CloudWatch Metrics Insights uses several mechanisms to infer the intention of a given SQL query.

Topics

- [Time bucketing](#)
- [Fields projection](#)
- [ORDER BY global aggregation](#)

Time bucketing

Time series data points resulting from a query are rolled up into time buckets based on the requested period. To aggregate values in standard SQL, an explicit **GROUP BY** clause must be defined to collect all the observations of a given period together. Because this is the standard way to query time series data, CloudWatch Metrics Insights infers time bucketing without the need to express an explicit **GROUP BY** clause.

For example, when a query is performed with a period of one minute, all the observations belonging to that minute until the next (excluded) are rolled up to the start time of the time bucket. This makes Metrics Insights SQL statements more concise and less verbose.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

The previous query returns a single time series (timestamp-value pairs), representing the average CPU utilization of all Amazon EC2 instances. Assuming the requested period is one minute, each data point returned represents the average of all observations measured within a specific one-minute interval (start time inclusive, end time exclusive). The timestamp related to the specific data point is the start time of the bucket

Fields projection

Metrics Insights queries always return the timestamp projection. You don't need to specify a timestamp column in the **SELECT** clause to get the timestamp of each corresponding data point value. For details about how timestamp is calculated, see [Time bucketing](#).

When using **GROUP BY**, each group name is also inferred and projected in the result, so that you can group the returned time series.


```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

The previous query returns a time series for each Amazon EC2 instance. Each time series is labelled after the value of the instance ID.

ORDER BY global aggregation

When using **ORDER BY, FUNCTION()** infers which aggregate function that you want to order by (the data point values of the queried metrics). The aggregate operation is performed across all the matched data points of each individual time series across the queried time window.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

The previous query returns the CPU utilization for each Amazon EC2 instance, limiting the result set to 10 entries. The results are ordered based on the maximum value of the individual time series within the requested time window. The **ORDER BY** clause is applied before **LIMIT**, so that the ordering is calculated against more than 10 time series.

Metrics Insights sample queries

This section contains examples of useful CloudWatch Metrics Insights queries that you can copy and use directly or copy and modify in query editor. Some of these examples are already available in the console, and you can access them by choosing **Add query** in the **Metrics** view.

Application Load Balancer examples

Total requests across all load balancers

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

Top 10 most active load balancers

```
SELECT MAX(ActiveConnectionCount)
```

```
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
LIMIT 10
```

AWS API usage examples

Top 20 AWS APIs by the number of calls in your account

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

CloudWatch APIs sorted by calls

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB examples

Top 10 tables by consumed reads

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by returned bytes

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by user errors

```
SELECT SUM(UserErrors)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store examples

Top 10 Amazon EBS volumes by bytes written

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Average Amazon EBS volume write time

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 examples

CPU utilization of EC2 instances sorted by highest

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY AVG() DESC
```

Average CPU utilization across the entire fleet

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

Top 10 instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

```
ORDER BY MAX() DESC
LIMIT 10
```

In this case, the CloudWatch agent is collecting a CPUUtilization metric per application. This query filters the average of this metric for a specific application name.

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service examples

Average CPU utilization across all ECS clusters

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

Top 10 clusters by memory utilization

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by CPU utilization

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by running tasks (Container Insights)

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights examples

Average CPU utilization across all EKS clusters

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

Top 10 clusters by node CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 clusters by pod memory utilization

```
SELECT AVG(pop_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 nodes by CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 pods by memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge examples

Top 10 rules by invocations

```
SELECT SUM(Invocations)
```

```
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by failed invocations

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by matched rules

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis examples

Top 10 streams by bytes written

```
SELECT SUM("PutRecords.Bytes")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 streams by earliest items in the stream

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda examples

Lambda functions ordered by number of invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
```

```
ORDER BY SUM() DESC
```

Top 10 Lambda functions by longest runtime

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Lambda functions by error count

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs examples

Top 10 log groups by incoming events

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Top 10 log groups by written bytes

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS examples

Top 10 Amazon RDS instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
```

```
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Amazon RDS clusters by writes

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service examples

Average latency by bucket

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

Top 10 buckets by bytes downloaded

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon Simple Notification Service examples

Total messages published by SNS topics

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

Top 10 topics by messages published

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```



```
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 topics by message delivery failures

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS examples

Top 10 queues by number of visible messages

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 most active queues

```
SELECT SUM(NumberOfMessagesSent)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 queues by age of earliest message

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights limits

CloudWatch Metrics Insights currently has the following limits:

- Currently, you can query only the most recent three hours of data.
- A single query can process no more than 10,000 metrics. This means that if the **SELECT**, **FROM**, and **WHERE** clauses match more than 10,000 metrics, the query only processes the first 10,000 of these metrics that it finds.
- A single query can return no more than 500 time series. This means that if the query would return more than 500 metrics, not all metrics will be returned in the query results. If you use an **ORDER BY** clause, then all the metrics being processed are sorted, and the 500 that have the highest or lowest values according to your **ORDER BY** clause are returned.

If you do not include an **ORDER BY** clause, you can't control which 500 matching metrics are returned.

- You can have as many as 200 Metrics Insights alarms per Region.
- Metrics Insights does not support high-resolution data, which is metric data reported with a granularity of less than one minute. If you request high-resolution data, the request does not fail, but the output is aggregated at one-minute granularity.
- Each [GetMetricData](#) operation can have only one query, but you can have multiple widgets in a dashboard that each include a query.

Metrics Insights glossary

label

In Metrics Insights, a label is a key-value pair that is used to scope a query to return a particular set of data, or to define criteria by which query results are to be separated into separate time series. A label key is similar to a column name in SQL. Currently, labels must be CloudWatch metric dimensions.

observation

An observation is a value recorded for a given metric at a given time.

Troubleshooting Metrics Insights

The results include "Other," but I don't have this as a dimension

This means that the query includes a **GROUP BY** clause that specifies a label key that is not used in some of the metrics that are returned by the query. In this case, a null group named `Other` is

returned. The metrics that do not include that label key are probably aggregated metrics that return values aggregated across all values of that label key.

For example, suppose we have the following query:

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

If some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

To prevent seeing "Other" in your results, use **SCHEMA** in your **FROM** clause, as in the following example:

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
GROUP BY Operation, ServiceName
```

This limits the returned results to only the metrics that have both the `Operation` and `ServiceName` dimensions.

The oldest timestamp in my graph has a lower metric value than the others

CloudWatch Metrics Insights currently supports the latest three hours of data only. When you graph with a period larger than one minute, there could be cases where the oldest data point differs from the expected value. This is because the Metrics Insights queries return only the most recent three hours of data. In this case, the oldest data point in the query returns only the observations that have been measured within the last three hours boundary, instead of returning all the observations within the period of that data point.

Use metrics explorer to monitor resources by their tags and properties

Metrics explorer is a tag-based tool that enables you to filter, aggregate, and visualize your metrics by tags and resource properties, to enhance observability for your services. This gives you a flexible and dynamic troubleshooting experience, so that you can create multiple graphs at a time and use these graphs to build your application health dashboards.

Metrics explorer visualizations are dynamic, so if a matching resource is created after you create a metrics explorer widget and add it to a CloudWatch dashboard, the new resource automatically appears in the explorer widget.

For example, if all of your EC2 production instances have the **production** tag, you can use metrics explorer to filter and aggregate metrics from all of these instances to understand their health and performance. If a new instance with a matching tag is later created, it's automatically added to the metrics explorer widget.

Note

Metrics explorer provides a point-in-time experience. Resources that have been terminated, or no longer exist with the property or tag that you specified are not displayed in the visualisation. However, you can still find the metrics for these resources in CloudWatch metrics views.

With metrics explorer, you can choose how to aggregate metrics from the resources that match the criteria, and whether to show them all in a single graph or on different graphs within one metrics explorer widget.

Metrics explorer includes templates that you can use to see useful visualization graphs with one click, and you can also extend these templates to create completely customized metrics explorer widgets.

Metrics explorer supports metrics emitted by AWS and EC2 metrics that are published by the CloudWatch agent, including memory, disk, and CPU metrics. To use metrics explorer to see the metrics that are published by the CloudWatch agent, you might have to update your CloudWatch agent configuration file. For more information, see [CloudWatch agent configuration for metrics explorer](#)

To create a visualization with metrics explorer and optionally add it to a dashboard, follow these steps.

To create a visualization with metrics explorer

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Explorer**.
3. Do one of the following:

- To use a template, select it in the box that currently shows **Empty Explorer**.

Depending on the template, the explorer might immediately display graphs of metrics. If it doesn't, choose one or more tags or properties in the **From** box and then data should appear. If it doesn't, use the options at the top of the page to display a longer time range in the graphs.

- To create a custom visualization, under **Metrics**, choose a single metric or all the available metrics from a service.

After you choose a metric, you can optionally repeat this step to add more metrics.

4. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name, and then choose the statistic that you want.
5. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the **Production** tag and the M5 instance type, only the resources that match all of these selections are displayed. It's treated as an AND operation.

6. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

Depending on how you choose to aggregate, the result may be a single time series or multiple time series.

7. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
8. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.
9. To add this visualization as a widget to a CloudWatch dashboard, choose **Add to dashboard**.

CloudWatch agent configuration for metrics explorer

To enable metrics explorer to discover EC2 metrics published by the CloudWatch agent, make sure that the CloudWatch agent configuration file contains the following values:

- In the `metrics` section, make sure that the `aggregation_dimensions` parameter includes `["InstanceId"]`. It can also contain other dimensions.
- In the `metrics` section, make sure that the `append_dimensions` parameter includes a `{"InstanceId": "${aws:InstanceId}"}` line. It can also contain other lines.
- In the `metrics` section, inside the `metrics_collected` section, check the sections for each resource type that you want metrics explorer to discover, such as the `cpu`, `disk`, and `memory` sections. Make sure that each of these sections has a `"resources": ["*"]` line..
- In the `cpu` section of the `metrics_collected` section, make sure there is a `"totalcpu": true` line.
- You must use the default `CWAgent` namespace for the metrics collected by the CloudWatch agent, instead of a custom namespace.

The settings in the previous list cause the CloudWatch agent to publish aggregate metrics for disks, CPUs, and other resources that can be plotted in metrics explorer for all the instances that use it.

These settings will republish the metrics that you had previously set up to be published with multiple dimensions, adding to your metric costs.

For more information about editing the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](#).

Use metric streams

You can use *metric streams* to continually stream CloudWatch metrics to a destination of your choice, with near-real-time delivery and low latency. Supported destinations include AWS destinations such as Amazon Simple Storage Service and several third-party service provider destinations.

There are three main usage scenarios for CloudWatch metric streams:

- **Custom setup with Firehose**— Create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream them to a data lake such as Amazon S3, or to any destination or endpoint supported by

Firehose including third-party providers. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.

- **Quick S3 setup**— Stream to Amazon Simple Storage Service with a quick setup process. By default, CloudWatch creates the resources needed for the stream. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported.
- **Quick AWS partner setup**— CloudWatch provides a quick setup experience for some third-party partners. You can use third-party service providers to monitor, troubleshoot, and analyze your applications using the streamed CloudWatch data. When you use the quick partner setup workflow, you need to provide only a destination URL and API key for your destination, and CloudWatch handles the rest of the setup. Quick partner setup is available for the following third-party providers:
 - Datadog
 - Dynatrace
 - New Relic
 - Splunk Observability Cloud
 - SumoLogic

You can stream all of your CloudWatch metrics, or use filters to stream only specified metrics. Each metric stream can include up to 1000 filters that either include or exclude metric namespaces or specific metrics. A single metric stream can have only include or exclude filters, but not both.

After a metric stream is created, if new metrics are created that match the filters in place, the new metrics are automatically included in the stream.

There is no limit on the number of metric streams per account or per Region, and no limit on the number of metric updates being streamed.

Each stream can use either JSON format, OpenTelemetry 1.0.0, or OpenTelemetry 0.7.0 format. You can edit the output format of a metric stream at any time, such as for upgrading from OpenTelemetry 0.7.0 to OpenTelemetry 1.0.0. For more information about output formats, see [Metric streams output formats](#).

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see [CloudWatch cross-account observability](#).

Metric streams always include the `Minimum`, `Maximum`, `SampleCount`, and `Sum` statistics. You can also choose to include additional statistics at an additional charge. For more information, see [Statistics that can be streamed](#).

Metric streams pricing is based on the number of metric updates. You also incur charges from Firehose for the delivery stream used for the metric stream. For more information, see [Amazon CloudWatch Pricing](#).

Topics

- [Set up a metric stream](#)
- [Statistics that can be streamed](#)
- [Metric stream operation and maintenance](#)
- [Monitor your metric streams with CloudWatch metrics](#)
- [Trust between CloudWatch and Firehose](#)
- [Metric streams output formats](#)
- [Troubleshooting](#)

Set up a metric stream

Use the steps in the following sections to set up a CloudWatch metric stream.

After a metric stream is created, the time it takes for metric data to appear at the destination depends on the configured buffering settings on the Firehose delivery stream. The buffering is expressed in maximum payload size or maximum wait time, whichever is reached first. If these are set to the minimum values (60 seconds, 1MB) the expected latency is within 3 minutes if the selected CloudWatch namespaces have active metric updates.

In a CloudWatch metric stream, data is sent every minute. Data might arrive at the final destination out of order. All specified metrics in the specified namespaces are sent in the metric stream, except metrics with a timestamp that is more than two days old.

For each combination of metric name and namespace that you stream, all dimension combinations of that metric name and namespace are streamed.

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see [CloudWatch cross-account observability](#).

To create and manage metric streams, you must be logged on to an account that has the **CloudWatchFullAccess** policy and the `iam:PassRole` permission, or an account that has the following list of permissions:

- `iam:PassRole`
- `cloudwatch:PutMetricStream`
- `cloudwatch>DeleteMetricStream`
- `cloudwatch:GetMetricStream`
- `cloudwatch:ListMetricStreams`
- `cloudwatch:StartMetricStreams`
- `cloudwatch:StopMetricStreams`

If you're going to have CloudWatch set up the IAM role needed for metric streams, you must also have the `iam:CreateRole` and `iam:PutRolePolicy` permissions.

Important

A user with the `cloudwatch:PutMetricStream` has access the CloudWatch metric data that is being streamed, even if they don't have the `cloudwatch:GetMetricData` permission.

Topics

- [Custom setup with Firehose](#)
- [Use Quick Amazon S3 setup](#)
- [Quick partner setup](#)

Custom setup with Firehose

Use this method to create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream

them to a data lake such as Amazon S3, or to any destination or endpoint supported by Firehose including third-party providers.

JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.

You can use the CloudWatch console, the AWS CLI, AWS CloudFormation, or the AWS Cloud Development Kit (AWS CDK) to set up a metric stream.

The Firehose delivery stream that you use for your metric stream must be in the same account and the same Region where you set up the metric stream. To achieve cross-Region functionality, you can configure the Firehose delivery stream to stream to a final destination that is in a different account or a different Region.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream using Firehose.

To set up a custom metric stream using Firehose

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Custom setup with Firehose**.
5. For **Select your Kinesis Data Firehose stream**, select the Firehose delivery stream to use. It must be in the same account. The default format for this option is OpenTelemetry 0.7.0, but you can change the format later in this procedure.

Then select the Firehose delivery stream to use under **Select your Firehose delivery stream**.

6. (Optional) You can choose **Select existing service role** to use an existing IAM role instead of having CloudWatch create a new one for you.

7. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.

8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

If you choose **All metrics**, all metrics from this account will be included in the stream.

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

If you choose **Select metrics**, do one of the following:

- To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
 - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.
9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

Streaming additional statistics incurs more charges. For more information, see [Statistics that can be streamed](#).

For definitions of the additional statistics, see [CloudWatch statistics definitions](#).

10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

AWS CLI or AWS API

Use the following steps to create a CloudWatch metric stream.

To use the AWS CLI or AWS API to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).
2. Create the Firehose delivery stream. For more information, see [Creating a Firehose stream](#).
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Firehose](#).
4. Use the `aws cloudwatch put-metric-stream` CLI command or the `PutMetricStream` API to create the CloudWatch metric stream.

AWS CloudFormation

You can use AWS CloudFormation to set up a metric stream. For more information, see [AWS::CloudWatch::MetricStream](#).

To use AWS CloudFormation to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).
2. Create the Firehose delivery stream. For more information, see [Creating a Firehose stream](#).
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Firehose](#).
4. Create the stream in AWS CloudFormation. For more information, see [AWS::CloudWatch::MetricStream](#).

AWS Cloud Development Kit (AWS CDK)

You can use AWS Cloud Development Kit (AWS CDK) to set up a metric stream.

To use the AWS CDK to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see [Creating a bucket](#).

2. Create the Firehose delivery stream. For more information, see [Creating an Amazon Data Firehose Delivery Stream](#).
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see [Trust between CloudWatch and Firehose](#).
4. Create the metric stream. The metric stream resource is available in AWS CDK as a Level 1 (L1) Construct named `CfnMetricStream`. For more information, see [Using L1 constructs](#).

Use Quick Amazon S3 setup

The **Quick S3 Setup** method works well if you want to quickly set up a stream to Amazon S3 and you don't need any formatting transformation beyond the supported JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format while you set up the stream.

Alternatively, if you want the final format to be Parquet format or Optimized Row Columnar (ORC), you should instead follow the steps in [Custom setup with Firehose](#).

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream Amazon S3 using Quick S3 setup.

To set up a metric stream using Quick S3 setup

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick S3 setup**. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format later in this procedure.
5. (Optional) Choose **Select existing resources** to use an existing S3 bucket or existing IAM roles instead of having CloudWatch create new ones for you.

6. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.
7. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

If you choose **All metrics**, all metrics from this account will be included in the stream.

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

If you choose **Select metrics**, do one of the following:

- To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
 - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.
8. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

Streaming additional statistics incurs more charges. For more information, see [Statistics that can be streamed](#).

For definitions of the additional statistics, see [CloudWatch statistics definitions](#).

9. (Optional) Customize the name of the new metric stream under **Metric stream name**.
10. Choose **Create metric stream**.

Quick partner setup

CloudWatch provides a quick setup experience for the following third-party partners. To use this workflow, you need to provide only a destination URL and API key for your destination. CloudWatch handles the rest of setup including creating the Firehose delivery stream and the necessary IAM roles.

Important

Before you use quick partner setup to create a metric stream, we strongly recommend that you read that partner's documentation, linked in the following list.

- [Datadog](#)
- [Dynatrace](#)
- [New Relic](#)
- [Splunk Observability Cloud](#)
- [SumoLogic](#)

When you set up a metric stream to one of these partners, the stream is created with some default settings, as listed in the following sections.

Topics

- [Set up a metric stream using quick partner setup](#)
- [Datadog stream defaults](#)
- [Dynatrace stream defaults](#)
- [New Relic stream defaults](#)
- [Splunk Observability Cloud stream defaults](#)
- [Sumo Logic stream defaults](#)

Set up a metric stream using quick partner setup

CloudWatch provides a quick setup option for some third-party partners. Before you start the steps in this section, you must have certain information for the partner. This information might include a destination URL and/or an API key for your partner destination. You should also read the

documentation at the partner's website linked in the previous section, and the defaults for that partner listed in the following sections.

To stream to a third-party destination not supported by quick setup, you can follow the instructions in [Follow the instructions in Custom setup with Firehose](#) to set up a stream using Firehose, and then send those metrics from Firehose to the final destination.

To use quick partner setup to create a metric stream to third-party provider

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick Amazon Web Services partner setup**.
5. Select the name of the partner that you want to stream metrics to.
6. For **Endpoint URL**, enter the destination URL.
7. For **Access Key** or **API Key**, enter the access key for the partner. Not all partners require an access key.
8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

If you choose **All metrics**, all metrics from this account will be included in the stream.

Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

If you choose **Select metrics**, do one of the following:

- To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
- To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.

9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

Streaming additional statistics incurs more charges. For more information, see [Statistics that can be streamed](#).

For definitions of the additional statistics, see [CloudWatch statistics definitions](#).

10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

Datadog stream defaults

Quick partner setup streams to Datadog use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 4 MBs
- **Firehose stream retry option** Duration of 60 seconds

When you use quick partner setup to create a metric stream to Datadog and you stream certain metrics, by default those metrics include some additional statistics. Streaming additional statistics can incur additional charges. For more information about statistics and their charges, see [Statistics that can be streamed](#).

The following list shows the metrics that have additional statistics streamed by default, if you choose to stream those metrics. You can choose to de-select these additional statistics before you start the stream.

- **Duration in AWS/Lambda:** p50, p80, p95, p99, p99.9
- **PostRuntimeExtensionDuration in AWS/Lambda:** p50, p99

- **FirstByteLatency and TotalRequestLatency in AWS/S3:** p50, p90, p95, p99, p99.9
- **ResponseLatency in AWS/Polly and TargetResponseTime in AWS/ApplicationELB:** p50, p90, p95, p99
- **Latency and IntegrationLatency in AWS/ApiGateway:** p90, p95, p99
- **Latency and TargetResponseTime in AWS/ELB:** p95, p99
- **RequestLatency in AWS/AppRunner:** p50, p95, p99
- **ActivityTime, ExecutionTime, LambdaFunctionRunTime, LambdaFunctionScheduleTime, LambdaFunctionTime, ActivityRunTime, and ActivityScheduleTime in AWS/States:** p95, p99
- **EncoderBitRate, ConfiguredBitRate, and ConfiguredBitRateAvailable in AWS/MediaLive:** p90
- **Latency in AWS/AppSync:** p90

Dynatrace stream defaults

Quick partner setup streams to Dynatrace use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 5 MBs
- **Firehose stream retry option** Duration of 600 seconds

New Relic stream defaults

Quick partner setup streams to New Relic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Splunk Observability Cloud stream defaults

Quick partner setup streams to Splunk Observability Cloud use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 300 seconds

Sumo Logic stream defaults

Quick partner setup streams to Sumo Logic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Statistics that can be streamed

Metric streams always include the following statistics: Minimum, Maximum, SampleCount, and Sum. You can also choose to include the following additional statistics in a metric stream. This choice is on a per-metric basis. For more information about these statistics, see [CloudWatch statistics definitions](#).

- Percentile values such as p95 or p99 (For streams with either JSON or OpenTelemetry format)
- Trimmed mean (Only for streams with the JSON format)
- Winsorized mean (Only for streams with the JSON format)
- Trimmed count (Only for streams with the JSON format)
- Trimmed sum (Only for streams with the JSON format)
- Percentile rank (Only for streams with the JSON format)
- Interquartile mean (Only for streams with the JSON format)

Streaming additional statistics incurs additional charges. Streaming between one and five of these additional statistics for a particular metric is billed as an additional metric update. Thereafter, each additional set of up to five of these statistics is billed as another metric update.

For example, suppose that for one metric you are streaming the following six additional statistics: p95, p99, p99.9, Trimmed mean, Winsorized mean, and Trimmed sum. Each update of this metric

is billed as three metric updates: one for the metric update which includes the default statistics, one for the first five additional statistics, and one for the sixth additional statistic. Adding up to four more additional statistics for a total of ten would not increase the billing, but an eleventh additional statistic would do so.

When you specify a metric name and namespace combination to stream additional statistics, all dimension combinations of that metric name and namespace are streamed with the additional statistics.

CloudWatch metric streams publishes a new metric, `TotalMetricUpdate`, which reflects the base number of metric updates plus extra metric updates incurred by streaming additional statistics. For more information, see [Monitor your metric streams with CloudWatch metrics](#).

For more information, see [Amazon CloudWatch Pricing](#).

Note

Some metrics do not support percentiles. Percentile statistics for these metrics are excluded from the stream and do not incur metric stream charges. An example of these statistics that do not support percentiles are some metrics in the AWS/ECS namespace.

The additional statistics that you configure are streamed only if they match the filters for the stream. For example, if you create a stream that has only EC2 and RDS in the include filters, and then your statistics configuration lists EC2 and Lambda, then the stream includes EC2 metrics with additional statistics, RDS metrics with only the default statistics, and doesn't include Lambda statistics at all.

Metric stream operation and maintenance

Metric streams are always in one of two states, **Running** or **Stopped**.

- **Running**— The metric stream is running correctly. There might not be any metric data streamed to the destination because of the filters on the stream.
- **Stopped**— The metric stream has been explicitly stopped by someone, and not because of an error. It might be useful to stop your stream to temporarily pause the streaming of data without deleting the stream.

If you stop and restart a metric stream, the metric data that was published to CloudWatch while the metric stream was stopped is not backfilled to the metric stream.

If you change the output format of a metric stream, in certain cases you might see a small amount of metric data written to the destination in both the old format and the new format. To avoid this situation, you can create a new Firehose delivery stream with the same configuration as your current one, then change to the new Firehose delivery stream and change the output format at the same time. This way, the Kinesis records with different output format are stored on Amazon S3 in separate objects. Later, you can direct the traffic back to the original Firehose delivery stream and delete the second delivery stream.

To view, edit, stop, and start your metric streams

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, Streams**.

The list of streams appears, and the **Status** column displays whether each stream is running or stopped.

3. To stop or start a metric stream, select the stream and choose **Stop** or **Start**.
4. To see the details about a metric stream, select the stream and choose **View details**.
5. To change the stream's output format, filters, destination Firehose stream, or roles, choose **Edit** and make the changes that you want.

If you change the filters, there might be some gaps in the metric data during the transition.

Monitor your metric streams with CloudWatch metrics

Metric streams emit CloudWatch metrics about their health and operation in the `AWS/CloudWatch/MetricStreams` namespace. The following metrics are emitted. These metrics are emitted with a `MetricStreamName` dimension and with no dimension. You can use the metrics with no dimensions to see aggregated metrics for all of your metric streams. You can use the metrics with the `MetricStreamName` dimension to see the metrics about only that metric stream.

For all of these metrics, values are emitted only for metric streams that are in the **Running** state.

| Metric | Description |
|-------------------|---|
| MetricUpdate | <p>The number of metric updates sent to the metric stream. If no metric updates are streamed during a time period, this metric is not emitted during that time period.</p> <p>If you stop the metric stream, this metric stops being emitted until the metric stream is started again.</p> <p>Valid Statistic: Sum</p> <p>Units: None</p> |
| TotalMetricUpdate | <p>This is calculated as MetricUpdate + a number based on additional statistics that are being streamed.</p> <p>For each unique namespace and metric name combination, streaming 1-5 additional statistics adds 1 to the TotalMetricUpdate , streaming 6-10 additional statistics adds 2 to TotalMetricUpdate , and so on.</p> <p>Valid Statistic: Sum</p> <p>Units: None</p> |
| PublishErrorRate | <p>The number of unrecoverable errors that occur when putting data into the Firehose delivery stream. If no errors occur during a time period, this metric is not emitted during that time period.</p> <p>If you stop the metric stream, this metric stops being emitted until the metric stream is started again.</p> <p>Valid Statistic: Average to see the rate of metric updates unable to be written. This value will be between 0.0 and 1.0.</p> <p>Units: None</p> |

Trust between CloudWatch and Firehose

The Firehose delivery stream must trust CloudWatch through an IAM role that has write permissions to Firehose. These permissions can be limited to the single Firehose delivery stream that the CloudWatch metric stream uses. The IAM role must trust the `streams.metrics.cloudwatch.amazonaws.com` service principal.

If you use the CloudWatch console to create a metric stream, you can have CloudWatch create the role with the correct permissions. If you use another method to create a metric stream, or you want to create the IAM role itself, it must contain the following permissions policy and trust policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:firehose:region:account-id:deliverystream/*"
    }
  ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "streams.metrics.cloudwatch.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Metric data is streamed by CloudWatch to the destination Firehose delivery stream on behalf of the source that owns the metric stream resource.

Metric streams output formats

The data in a CloudWatch metric stream can be in the JSON format or the OpenTelemetry format. Currently, both the OpenTelemetry 1.0.0 and 0.7.0 formats are supported.

Contents

- [JSON format](#)
 - [Which AWS Glue schema should I use for JSON output format?](#)
- [OpenTelemetry 1.0.0 format](#)
 - [Translations with OpenTelemetry 1.0.0 format](#)
 - [How to parse OpenTelemetry 1.0.0 messages](#)
- [OpenTelemetry 0.7.0 format](#)
 - [Translations with OpenTelemetry 0.7.0 format](#)
 - [How to parse OpenTelemetry 0.7.0 messages](#)

JSON format

In a CloudWatch metric stream that uses the JSON format, each Firehose record contains multiple JSON objects separated by a newline character (\n). Each object includes a single data point of a single metric.

The JSON format that is used is fully compatible with AWS Glue and with Amazon Athena. If you have a Firehose delivery stream and an AWS Glue table formatted correctly, the format can be automatically transformed into Parquet format or Optimized Row Columnar (ORC) format before being stored in S3. For more information about transforming the format, see [Converting Your Input Record Format in Firehose](#). For more information about the correct format for AWS Glue, see [Which AWS Glue schema should I use for JSON output format?](#)

In the JSON format, the valid values for `unit` are the same as for the value of `unit` in the `MetricDatum` API structure. For more information, see [MetricDatum](#). The value for the `timestamp` field is in epoch milliseconds, such as 1616004674229.

The following is an example of the format. In this example, the JSON is formatted for easy reading, but in practice the whole format is on a single line.

```
{
  "metric_stream_name": "MyMetricStream",
```



```
"account_id": "1234567890",
"region": "us-east-1",
"namespace": "AWS/EC2",
"metric_name": "DiskWriteOps",
"dimensions": {
  "InstanceId": "i-123456789012"
},
"timestamp": 1611929698000,
"value": {
  "count": 3.0,
  "sum": 20.0,
  "max": 18.0,
  "min": 0.0,
  "p99": 17.56,
  "p99.9": 17.8764,
  "TM(25%;75%)": 16.43
},
"unit": "Seconds"
}
```

Which AWS Glue schema should I use for JSON output format?

The following is an example of a JSON representation of the `StorageDescriptor` for an AWS Glue table, which would then be used by Firehose. For more information about `StorageDescriptor`, see [StorageDescriptor](#).

```
{
  "Columns": [
    {
      "Name": "metric_stream_name",
      "Type": "string"
    },
    {
      "Name": "account_id",
      "Type": "string"
    },
    {
      "Name": "region",
      "Type": "string"
    },
    {
      "Name": "namespace",
      "Type": "string"
    }
  ]
}
```

```

    },
    {
      "Name": "metric_name",
      "Type": "string"
    },
    {
      "Name": "timestamp",
      "Type": "timestamp"
    },
    {
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
    {
      "Name": "value",
      "Type":
"struct<min:double,max:double,count:double,sum:double,p99:double,p99.9:double>"
    },
    {
      "Name": "unit",
      "Type": "string"
    }
  ],
  "Location": "s3://my-s3-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}

```

The preceding example is for data written on Amazon S3 in JSON format. Replace the values in the following fields with the indicated values to store the data in Parquet format or Optimized Row Columnar (ORC) format.

- **Parquet:**

- `inputFormat`: `org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat`
- `outputFormat`: `org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat`
- `SerdeInfo.serializationLib`: `org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe`

- `parameters.classification: parquet`
- **ORC:**
 - `inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat`
 - `outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat`
 - `SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde`
 - `parameters.classification: orc`

OpenTelemetry 1.0.0 format

Note

With the OpenTelemetry 1.0.0 format, metric attributes are encoded as a list of `KeyValue` objects instead of the `StringKeyValue` type used in the 0.7.0 format. As a consumer, this is the only major change between the 0.7.0 and 1.0.0 formats. A parser generated from the 0.7.0 proto files won't parse metric attributes encoded in the 1.0.0 format. The same is true in reverse, a parser generated from the 1.0.0 proto files will not parse metric attributes encoded in the 0.7.0 format.

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see [OpenTelemetry](#).

For information about the full OpenTelemetry 1.0.0 specification, see [Release version 1.0.0](#).

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {  
  resource {  
    attributes {
```

```
    key: "cloud.provider"
    value {
      string_value: "aws"
    }
  }
  attributes {
    key: "cloud.account.id"
    value {
      string_value: "123456789012"
    }
  }
  attributes {
    key: "cloud.region"
    value {
      string_value: "us-east-1"
    }
  }
  attributes {
    key: "aws.exporter.arn"
    value {
      string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
    }
  }
}
scope_metrics {
  metrics {
    name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
    unit: "NoneTranslated"
    summary {
      data_points {
        start_time_unix_nano: 600000000000
        time_unix_nano: 1200000000000
        count: 1
        sum: 1.0
        quantile_values {
          value: 1.0
        }
        quantile_values {
          quantile: 0.95
          value: 1.0
        }
        quantile_values {
          quantile: 0.99
```

```
    value: 1.0
  }
  quantile_values {
    quantile: 1.0
    value: 1.0
  }
  attributes {
    key: "Namespace"
    value {
      string_value: "AWS/DynamoDB"
    }
  }
  attributes {
    key: "MetricName"
    value {
      string_value: "ConsumedReadCapacityUnits"
    }
  }
  attributes {
    key: "Dimensions"
    value {
      kvlist_value {
        values {
          key: "TableName"
          value {
            string_value: "MyTable"
          }
        }
      }
    }
  }
}
data_points {
  start_time_unix_nano: 700000000000
  time_unix_nano: 1300000000000
  count: 2
  sum: 5.0
  quantile_values {
    value: 2.0
  }
  quantile_values {
    quantile: 1.0
    value: 3.0
  }
}
```

```
    attributes {
      key: "Namespace"
      value {
        string_value: "AWS/DynamoDB"
      }
    }
  attributes {
    key: "MetricName"
    value {
      string_value: "ConsumedReadCapacityUnits"
    }
  }
  attributes {
    key: "Dimensions"
    value {
      kvlist_value {
        values {
          key: "TableName"
          value {
            string_value: "MyTable"
          }
        }
      }
    }
  }
}
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
```

```
}
```

`ResourceMetrics` is the top-level object to represent `MetricData` objects.

```
// A collection of ScopeMetrics from a Resource.
message ResourceMetrics {
  reserved 1000;

  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated ScopeMetrics scope_metrics = 2;

  // This schema_url applies to the data in the "resource" field. It does not apply
  // to the data in the "scope_metrics" field which have their own schema_url field.
  string schema_url = 3;
}
```

The Resource object

A Resource object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by AWS, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The Resource object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of attributes that describe the resource.
  // Attribute keys MUST be unique (it is not allowed to have more than one
```

```
// attribute with the same key).
repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

// dropped_attributes_count is the number of dropped attributes. If the value is 0,
then
// no attributes were dropped.
uint32 dropped_attributes_count = 2;
}
```

The ScopeMetrics object

The scope field will not be filled. We fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an Scope.
message ScopeMetrics {
  // The instrumentation scope information for the metrics in this message.
  // Semantically when InstrumentationScope isn't set, it is equivalent with
  // an empty instrumentation scope name (unknown).
  opentelemetry.proto.common.v1.InstrumentationScope scope = 1;

  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;

  // This schema_url applies to all metrics in the "metrics" field.
  string schema_url = 3;
}
```

The Metric object

The metric object contains some metadata and a Summary data field that contains a list of SummaryDataPoint.

For metric streams, the metadata is as follows:

- name will be `amazonaws.com/metric_namespace/metric_name`
- description will be blank
- unit will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see [Translations with OpenTelemetry 1.0.0 format](#) and [The Unified Code For Units of Measure](#).
- type will be SUMMARY


```
message Metric {
  reserved 4, 6, 8;

  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  // Data determines the aggregation type (if any) of the metric, what is the
  // reported value type for the data points, as well as the relationship to
  // the time interval over which they are reported.
  oneof data {
    Gauge gauge = 5;
    Sum sum = 7;
    Histogram histogram = 9;
    ExponentialHistogram exponential_histogram = 10;
    Summary summary = 11;
  }
}

message Summary {
  repeated SummaryDataPoint data_points = 1;
}
```

The SummaryDataPoint object

The SummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// SummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message SummaryDataPoint {
  reserved 1;

  // The set of key/value pairs that uniquely identify the timeseries from
  // where this point belongs. The list may be empty (may contain 0 elements).
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
```

```
repeated opentelemetry.proto.common.v1.KeyValue attributes = 7;

// StartTimeUnixNano is optional but strongly encouraged, see the
// the detailed comments above Metric.
//
// Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
// 1970.
fixed64 start_time_unix_nano = 2;

// TimeUnixNano is required, see the detailed comments above Metric.
//
// Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
// 1970.
fixed64 time_unix_nano = 3;

// count is the number of values in the population. Must be non-negative.
fixed64 count = 4;

// sum of the values in the population. If count is zero then this field
// must be zero.
//
// Note: Sum should only be filled out when measuring non-negative discrete
// events, and is assumed to be monotonic over the values of these events.
// Negative events *can* be recorded, but sum should not be filled out when
// doing so. This is specifically to enforce compatibility w/ OpenMetrics,
// see: https://github.com/OpenObservability/OpenMetrics/blob/main/specification/
OpenMetrics.md#summary
double sum = 5;

// Represents the value at a given quantile of a distribution.
//
// To record Min and Max values following conventions are used:
// - The 1.0 quantile is equivalent to the maximum value observed.
// - The 0.0 quantile is equivalent to the minimum value observed.
//
// See the following issue for more context:
// https://github.com/open-telemetry/opentelemetry-proto/issues/125
message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    //
```

```
// Quantile values must NOT be negative.
double value = 2;
}

// (Optional) list of values at different quantiles of the distribution calculated
// from the current snapshot. The quantiles must be strictly increasing.
repeated ValueAtQuantile quantile_values = 6;

// Flags that apply to this specific data point. See DataPointFlags
// for the available flags and their meaning.
uint32 flags = 8;
}
```

For more information, see [Translations with OpenTelemetry 1.0.0 format](#).

Translations with OpenTelemetry 1.0.0 format

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One attribute has the key `Namespace` and its value is the namespace of the metric
- One attribute has the key `MetricName` and its value is the name of the metric
- One pair has the key `Dimensions` and its value is a nested list of key-value pairs. Each pair in this list maps to a CloudWatch metric dimension, where the pair's key is the name of the dimension and its value is the value of the dimension.

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
 - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`

- `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
- `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
- `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see [The Unified Code For Units of Measure](#).

| CloudWatch | OpenTelemetry |
|-------------------|---------------|
| Second | s |
| Second or Seconds | s |
| Microseconds | us |
| Milliseconds | ms |
| Bytes | By |
| Kilobytes | kBy |
| Megabytes | MBy |
| Gigabytes | GBy |
| Terabytes | TBy |
| Bits | bit |
| Kilobits | kbit |
| Megabits | MBit |
| Gigabits | GBit |

| CloudWatch | OpenTelemetry |
|------------|---------------|
| Terabits | Tbit |
| Percent | % |
| Count | {Count} |
| None | 1 |

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 1.0.0 messages

This section provides information to help you get started with parsing OpenTelemetry 1.0.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 1.0.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
 - To use Java, add the following Maven dependency to your Java project: [OpenTelemetry Java >> 0.14.1](#).
 - To use any other language, follow these steps:
 - a. Make sure that your language is supported by checking the list at [Generating Your Classes](#).
 - b. Install the Protobuf compiler by following the steps at [Download Protocol Buffers](#).
 - c. Download the OpenTelemetry 0.7.0 ProtoBuf definitions at [Release version 1.0.0](#).
 - d. Confirm that you are in the root folder of the downloaded OpenTelemetry 0.7.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see [Generating Your Classes](#).

The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
```

```
opentelemetry/proto/common/v1/common.proto \  
opentelemetry/proto/resource/v1/resource.proto \  
opentelemetry/proto/metrics/v1/metrics.proto \  
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;  
  
import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.ArrayList;  
import java.util.List;  
  
public class MyOpenTelemetryParser {  
  
    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws  
    IOException {  
        List<ExportMetricsServiceRequest> result = new ArrayList<>();  
  
        ExportMetricsServiceRequest request;  
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`  
        records, each of them starting with a header with an  
        UnsignedVarInt32 indicating the record length in bytes:  
        -----  
        |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...  
        -----  
        */  
        while ((request =  
ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {  
            // Do whatever we want with the parsed message  
            result.add(request);  
        }  
  
        return result;  
    }  
}
```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array`. *number* 8-bit integers

```
const pb = require('google-protobuf')
const pbMetrics =
  require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
  const result = []

  // Loop until we've read all the data from the buffer
  while (data.length) {
    /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
       records, each of them starting with a header with an
       UnsignedVarInt32 indicating the record length in bytes:
       -----
       |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
       -----
    */
    const reader = new pb.BinaryReader(data)
    const messageLength = reader.decoder_.readUnsignedVarint32()
    const messageFrom = reader.decoder_.cursor_
    const messageTo = messageFrom + messageLength

    // Extract the current `ExportMetricsServiceRequest` message to parse
    const message = data.subarray(messageFrom, messageTo)

    // Parse the current message using the ProtoBuf library
    const parsed =
      pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

    // Do whatever we want with the parsed message
    result.push(parsed.toObject())

    // Shrink the remaining buffer, removing the already parsed data
```

```
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see [Download Protocol Buffers](#).

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

OpenTelemetry 0.7.0 format

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see [OpenTelemetry](#).

For information about the full OpenTelemetry 0.7.0 specification, see [v0.7.0 release](#).

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/MyMetricStream"
      }
    }
  }
}
```

```
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
      double_summary {
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
        }
        start_time_unix_nano: 1604948400000000000
        time_unix_nano: 1604948460000000000
        count: 1
        sum: 1.0
        quantile_values {
          quantile: 0.0
          value: 1.0
        }
        quantile_values {
          quantile: 0.95
          value: 1.0
        }
        quantile_values {
          quantile: 0.99
          value: 1.0
        }
        quantile_values {
          quantile: 1.0
          value: 1.0
        }
      }
    }
    data_points {
      labels {
        key: "Namespace"
```



```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
}
```

The Resource object

A Resource object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by AWS, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The Resource object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of labels that describe the resource.
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

The InstrumentationLibraryMetrics object

The `instrumentation_library` field will not be filled. We will fill only the `metrics` field that we are exporting.

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
  // The instrumentation library information for the metrics in this message.
  // If this field is not set then no library info is known.
  opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;
}
```

The Metric object

The metric object contains a `DoubleSummary` data field that contains a list of `DoubleSummaryDataPoint`.

```
message Metric {
  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  oneof data {
    IntGauge int_gauge = 4;
    DoubleGauge double_gauge = 5;
    IntSum int_sum = 6;
    DoubleSum double_sum = 7;
    IntHistogram int_histogram = 8;
    DoubleHistogram double_histogram = 9;
    DoubleSummary double_summary = 11;
  }
}

message DoubleSummary {
  repeated DoubleSummaryDataPoint data_points = 1;
}
```

The MetricDescriptor object

The `MetricDescriptor` object contains metadata. For more information, see [metrics.proto](#) on GitHub.

For metric streams, the MetricDescriptor has the following contents:

- name will be `amazonaws.com/metric_namespace/metric_name`
- description will be blank.
- unit will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see [Translations with OpenTelemetry 0.7.0 format](#) and [The Unified Code For Units of Measure](#).
- type will be SUMMARY.

The DoubleSummaryDataPoint object

The DoubleSummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
  // The set of labels that uniquely identify this timeseries.
  repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

  // start_time_unix_nano is the last time when the aggregation value was reset
  // to "zero". For some metric types this is ignored, see data types for more
  // details.
  //
  // The aggregation value is over the time interval (start_time_unix_nano,
  // time_unix_nano].
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  //
  // Value of 0 indicates that the timestamp is unspecified. In that case the
  // timestamp may be decided by the backend.
  fixed64 start_time_unix_nano = 2;

  // time_unix_nano is the moment when this aggregation value was reported.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
```

```
fixed64 count = 4;

// sum of the values in the population. If count is zero then this field
// must be zero.
double sum = 5;

// Represents the value at a given quantile of a distribution.
//
// To record Min and Max values following conventions are used:
// - The 1.0 quantile is equivalent to the maximum value observed.
// - The 0.0 quantile is equivalent to the minimum value observed.
message ValueAtQuantile {
  // The quantile of a distribution. Must be in the interval
  // [0.0, 1.0].
  double quantile = 1;

  // The value at the given quantile of a distribution.
  double value = 2;
}

// (Optional) list of values at different quantiles of the distribution calculated
// from the current snapshot. The quantiles must be strictly increasing.
repeated ValueAtQuantile quantile_values = 6;
}
```

For more information, see [Translations with OpenTelemetry 0.7.0 format](#).

Translations with OpenTelemetry 0.7.0 format

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One pair contains the namespace of the metric
- One pair contains the name of the metric
- For each dimension, CloudWatch stores the following pair:
`metricDatum.Dimensions[i].Name`, `metricDatum.Dimensions[i].Value`

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
 - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
 - `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
 - `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
 - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see [The Unified Code For Units of Measure](#).

| CloudWatch | OpenTelemetry |
|-------------------|---------------|
| Second | s |
| Second or Seconds | s |
| Microsecond | us |
| Milliseconds | ms |
| Bytes | By |
| Kilobytes | kBy |
| Megabytes | MBy |
| Gigabytes | GBy |

| CloudWatch | OpenTelemetry |
|------------|---------------|
| Terabytes | TBy |
| Bits | bit |
| Kilobits | kbit |
| Megabits | MBit |
| Gigabits | GBit |
| Terabits | Tbit |
| Percent | % |
| Count | {Count} |
| None | 1 |

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 0.7.0 messages

This section provides information to help you get started with parsing OpenTelemetry 0.7.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 0.7.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
 - To use Java, add the following Maven dependency to your Java project: [OpenTelemetry Java >> 0.14.1](#).
 - To use any other language, follow these steps:
 - a. Make sure that your language is supported by checking the list at [Generating Your Classes](#).
 - b. Install the Protobuf compiler by following the steps at [Download Protocol Buffers](#).

- c. Download the OpenTelemetry 0.7.0 ProtoBuf definitions at [v0.7.0 release](#).
- d. Confirm that you are in the root folder of the downloaded OpenTelemetry 0.7.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see [Generating Your Classes](#).

The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
        records, each of them starting with a header with an
        UnsignedVarInt32 indicating the record length in bytes:
        -----
        |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
        -----
        */
    }
}
```

```

    */
    while ((request =
ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
        // Do whatever we want with the parsed message
        result.add(request);
    }

    return result;
}
}

```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array` *number* 8-bit integers

```

const pb = require('google-protobuf')
const pbMetrics =
  require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
  const result = []

  // Loop until we've read all the data from the buffer
  while (data.length) {
    /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
    records, each of them starting with a header with an
    UnsignedVarInt32 indicating the record length in bytes:
    -----
    |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
    -----
    */
    const reader = new pb.BinaryReader(data)
    const messageLength = reader.decoder_.readUnsignedVarint32()
    const messageFrom = reader.decoder_.cursor_
    const messageTo = messageFrom + messageLength
  }
}

```

```
// Extract the current `ExportMetricsServiceRequest` message to parse
const message = data.subarray(messageFrom, messageTo)

// Parse the current message using the ProtoBuf library
const parsed =
  pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

// Do whatever we want with the parsed message
result.push(parsed.toObject())

// Shrink the remaining buffer, removing the already parsed data
data = data.subarray(messageTo)
}

return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see [Download Protocol Buffers](#).

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest Protocol Buffers` messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

Troubleshooting

If you're not seeing metric data at your final destination, check the following:

- Check that the metric stream is in the running state. For steps on how to use the CloudWatch console to do this, see [Metric stream operation and maintenance](#).
- Metrics published more than two days in the past are not streamed. To determine whether a particular metric will be streamed, graph the metric in the CloudWatch console and check how old the last visible datapoint is. If it is more than two days in the past, then it won't be picked up by metric streams.
- Check the metrics emitted by the metric stream. In the CloudWatch console, under **Metrics**, look at the **AWS/CloudWatch/MetricStreams** namespace for the **MetricUpdate**, **TotalMetricUpdate**, and **PublishErrorRate** metrics.
- If the **PublishErrorRate** metric is high, confirm that the destination that is used by the Firehose delivery stream exists and that the IAM role specified in the metric stream's configuration grants the `CloudWatch` service principal permissions to write to it. For more information, see [Trust between CloudWatch and Firehose](#).
- Check that the Firehose delivery stream has permission to write to the final destination.
- In the Firehose console, view the Firehose delivery stream that is used for the metric stream and check the **Monitoring** tab to see whether the Firehose delivery stream is receiving data.
- Confirm that you have configured your Firehose delivery stream with the correct details.
- Check any available logs or metrics for the final destination that the Firehose delivery stream writes to.
- To get more detailed information, enable CloudWatch Logs error logging on the Firehose delivery stream. For more information, see [Monitoring Amazon Data Firehose Using CloudWatch Logs](#).

View available metrics

Metrics are grouped first by namespace, and then by the various dimension combinations within each namespace. For example, you can view all EC2 metrics, EC2 metrics grouped by instance, or EC2 metrics grouped by Auto Scaling group.

Only the AWS services that you're using send metrics to Amazon CloudWatch.

For a list of AWS services that send metrics to CloudWatch, see [AWS services that publish CloudWatch metrics](#). From this page, you can also see the metrics and dimensions that are published by each of those services.

Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a [list-metrics](#) command. The best way to retrieve these metrics is with the [get-metric-data](#) or [get-metric-statistics](#) commands in the AWS CLI.

If the old metric you want to view has a current metric with similar dimensions, you can view that current similar metric and then choose the **Source** tab, and change the metric name and dimension fields to the ones that you want, and also change the time range to a time when the metric was being reported.

The following steps help you browse through the metric namespaces to find and view metrics. You can also search for metrics using targeted search terms. For more information, see [Search for available metrics](#).

If you are browsing in an account set up as a monitoring account in CloudWatch cross-account observability, you can view metrics from the source accounts linked to this monitoring account. When metrics from source accounts are displayed, the ID or label of the account that they are from is also displayed. For more information, see [CloudWatch cross-account observability](#).

To view available metrics by namespace and dimension using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Select a metric namespace (for example, **EC2** or **Lambda**).

4. Select a metric dimension (for example, **Per-Instance Metrics** or **By Function Name**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. By each metric name is an information button you can choose to see a popup with the metric definition.

If this is a monitoring account in CloudWatch cross-account observability, you also see the metrics from the source accounts linked to this monitoring account. The **Account label** and **Account id** columns in the table display which account each metric is from.

You can do the following:

- a. To sort the table, use the column heading.
- b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
- c. To filter by account, choose the account label or account ID and then choose **Add to search**.
- d. To filter by resource, choose the resource ID and then choose **Add to search**.
- e. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

To view available metrics by account namespace, dimension, or metric using the AWS CLI

Use the [list-metrics](#) command to list CloudWatch metrics. For a list of the namespaces, metrics, and dimensions for all services that publish metrics, see [AWS services that publish CloudWatch metrics](#).

The following example command lists all the metrics for Amazon EC2.

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

The following is example output.

```
{
  "Metrics" : [
    ...
    {
      "Namespace": "AWS/EC2",
      "Dimensions": [
        {
          "Name": "InstanceId",
          "Value": "i-1234567890abcdef0"
        }
      ]
    }
  ]
}
```

```

    }
  ],
  "MetricName": "NetworkOut"
},
{
  "Namespace": "AWS/EC2",
  "Dimensions": [
    {
      "Name": "InstanceId",
      "Value": "i-1234567890abcdef0"
    }
  ],
  "MetricName": "CPUUtilization"
},
{
  "Namespace": "AWS/EC2",
  "Dimensions": [
    {
      "Name": "InstanceId",
      "Value": "i-1234567890abcdef0"
    }
  ],
  "MetricName": "NetworkIn"
},
...
]
}

```

To list all the available metrics for a specified resource

The following example specifies the AWS/EC2 namespace and the InstanceId dimension to view the results for the specified instance only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions
Name=InstanceId,Value=i-1234567890abcdef0
```

To list a metric for all resources

The following example specifies the AWS/EC2 namespace and a metric name to view the results for the specified metric only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```


To retrieve metrics from linked source accounts in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from both the monitoring account and all linked source accounts. If you do not add `--include-linked-accounts`, the command returns only the monitoring account's metrics.

```
aws cloudwatch list-metrics --include-linked-accounts
```

To retrieve metrics from a source account in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from the source account with the ID 111122223333.

```
aws cloudwatch list-metrics --include-linked-accounts --owning-account "111122223333"
```

Search for available metrics

You can search within all of the metrics in your account using targeted search terms. Metrics are returned that have matching results within their namespace, metric name, or dimensions.

If this is a monitoring account in CloudWatch cross-account observability, you also search for metrics from the source accounts linked to this monitoring account.

Note

Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **All metrics** tab in the console, and they are not returned in the results of a [list-metrics](#) command. The best way to retrieve these metrics is with the [get-metric-data](#) or [get-metric-statistics](#) commands in the AWS CLI.

To search for available metrics in CloudWatch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the search field on the **All metrics** tab, enter a search term, such as a metric name, namespace, account ID, account label, dimension name or value, or resource name. This shows you all of the namespaces with metrics with this search term.

For example, if you search for **volume**, this shows the namespaces that contain metrics with this term in their name.

For more information on search, see [Use search expressions in graphs](#)

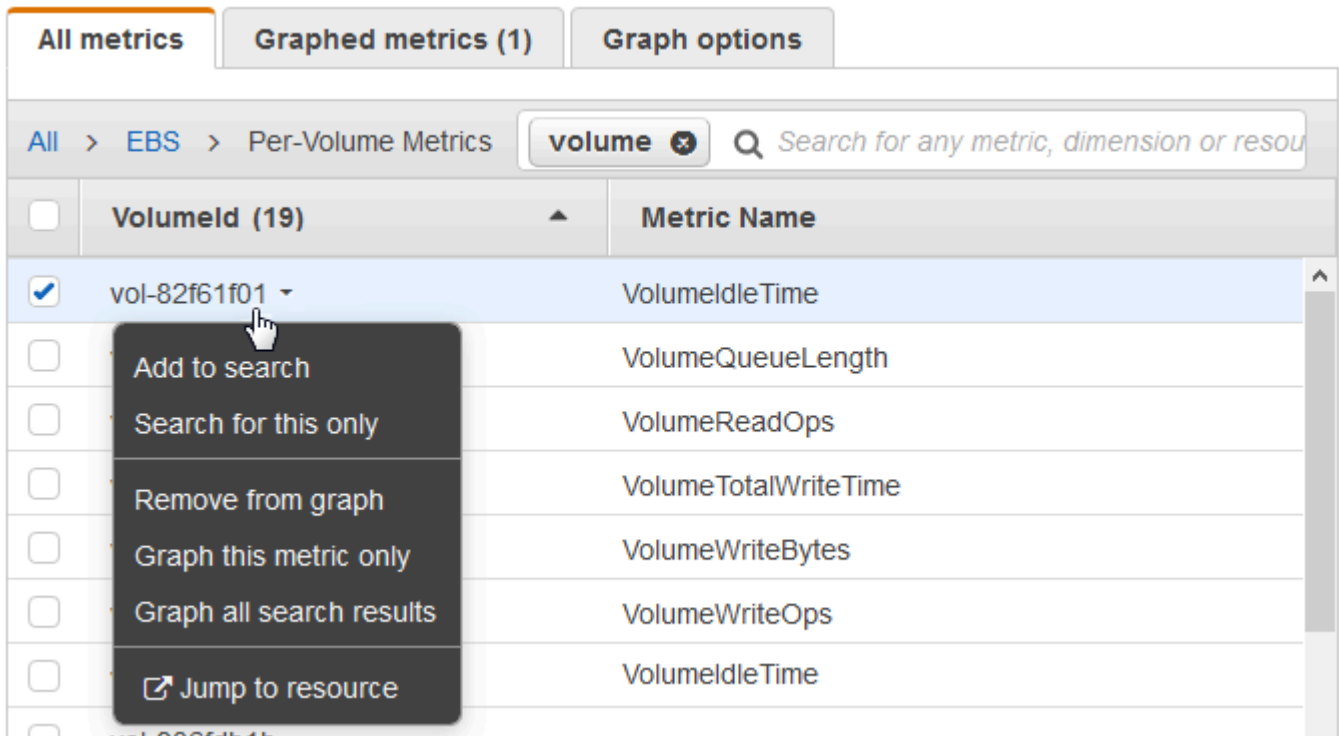
4. To graph all the search results, choose **Graph search**

or

Select a namespace to view the metrics from that namespace. You can then do the following:

- To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
- To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
- To view one of the resources on its console, choose the resource ID and then choose **Jump to resource**.
- To view help for a metric, select the metric name and choose **What is this?**.

The selected metrics appear on the graph.



5. (Optional) Select one of the buttons in the search bar to edit that part of the search term.

Graphing metrics

Use the CloudWatch console to graph metric data generated by other AWS services. This makes it more efficient to see the metric activity on your services. The following procedures describe how to graph metrics in CloudWatch.

Contents

- [Graph a metric](#)
- [Merge two graphs into one](#)
- [Use dynamic labels](#)
- [Modify the time range or time zone format for a graph](#)
- [Zoom in on a line graph or stacked area graph](#)
- [Modify the y-axis for a graph](#)
- [Create an alarm from a metric on a graph](#)

Graph a metric

You can select metrics and create graphs of the metric data using the CloudWatch console.

CloudWatch supports the following statistics on metrics: Average, Minimum, Maximum, Sum, and SampleCount. For more information, see [Statistics](#).

You can view your data at different levels of detail. For example, you can choose a one-minute view, which can be useful when troubleshooting. Or, choose a less detailed, one-hour view. That can be useful when viewing a broader time range (for example, 3 days) so that you can see trends over time. For more information, see [Periods](#).

If you are using an account that is set up as a monitoring account in CloudWatch cross-account observability, you can graph metrics from the source accounts linked to this monitoring account. For more information, see [CloudWatch cross-account observability](#).

Creating a graph

To graph a metric

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.

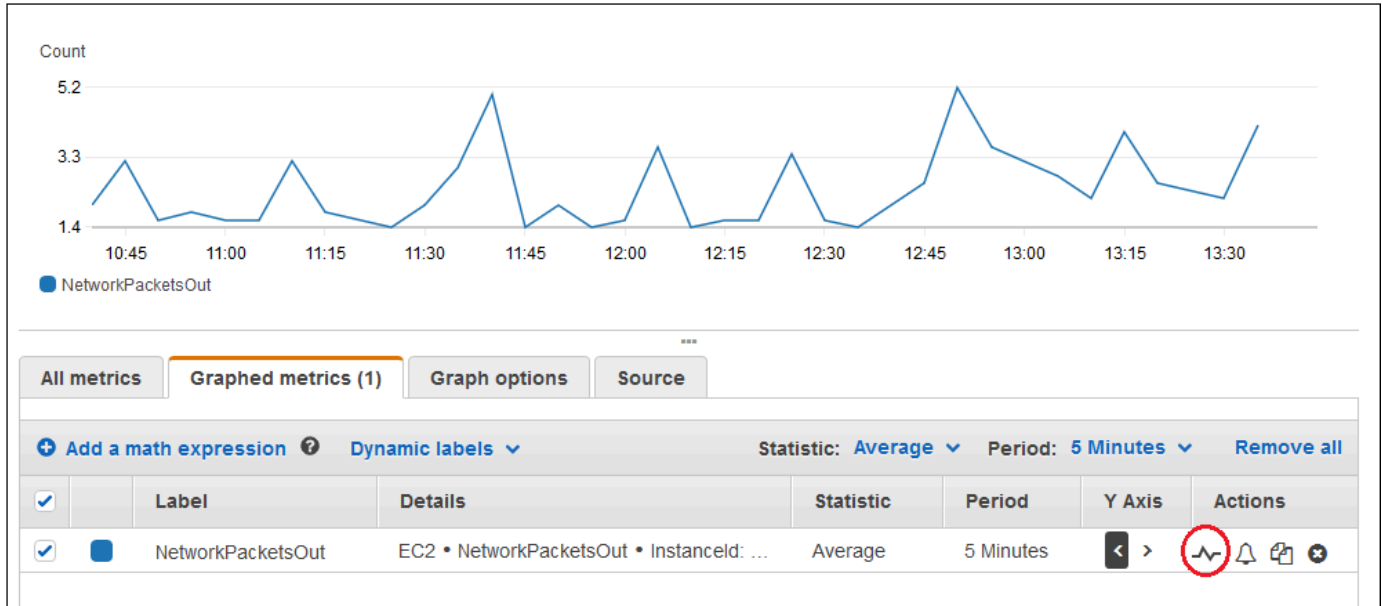
- In the **Browse** tab, enter a search term in the search field, such as a metric name, account ID, or resource name.

For example, if you search for the CPUUtilization metric, you see the namespaces and dimensions with this metric.

- Select one of the results for your search to view the metrics.
- To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
- (Optional) To change the type of graph, choose the **Options** tab. You can then choose between a line graph, stacked area chart, number display, gauge, bar chart, or pie chart.
- Choose the **Graphed metrics** tab.
- (Optional) To change the statistic used in the graph, choose the new statistic in the **Statistic** column next to the metric name.

For more information about CloudWatch statistics, see [CloudWatch statistics definitions](#). For more information about the **pxx percentile statistics**, see [Percentiles](#).

- (Optional) To add an anomaly detection band that shows expected values for the metric, choose the anomaly detection icon under **Actions** next to the metric.



CloudWatch uses up to two weeks of the metric's recent historical data to calculate a model for expected values. It then displays this range of expected values as a band on the graph. CloudWatch adds a new row under the metric to display the anomaly detection band math expression, labeled **ANOMALY_DETECTION_BAND**. If recent historical data exists, you

immediately see a preview anomaly detection band, which is an approximation of the anomaly detection band generated by the model. It takes up to 15 minutes for the actual anomaly detection band to appear.

By default, CloudWatch creates the upper and lower bounds of the band of expected values with a default value of 2 for the band threshold. To change this number, change the value at the end of the formula under **Details** for the band.

- (Optional) Choose **Edit model** to change how the anomaly detection model is calculated. You can exclude past and future time periods from being used in the training for calculating the model. It is critical to exclude unusual events system as system outage, deployments, and holidays from the training data. You can also specify the time zone to use for the model for daylight saving time changes.

For more information, see [Using CloudWatch anomaly detection](#).

To hide the model from the graph, remove the checkmark from the line with the ANOMALY_DETECTION_BAND function or choose the X icon. To delete the model entirely, choose **Edit model**, **Delete model**.

10. (Optional) As you choose metrics to graph, specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric, and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics**, **Add dynamic label**.

By default, the dynamic values that you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see [Use dynamic labels](#).

11. To view more information about the metric being graphed, pause the mouse over the legend.
12. Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.

- d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

13. To get a URL for your graph, choose **Actions, Share**. Copy the URL to save or share.

14. To add your graph to a dashboard, choose **Actions, Add to dashboard**.

Creating a graph of metrics from another data source

You can create a graph that displays resources from data sources other than CloudWatch. For more information about creating connections to these other data sources, see [Query metrics from other data sources](#).

To graph a metric from another data source

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.

If you haven't already created a connection to the data source that you want, select **Create and manage data sources**, then choose **Create and manage data sources**. For information about the rest of this data source creation process, see [Connect to a prebuilt data source with a wizard](#).

5. The wizard or query editor prompts you for the information necessary for the query. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus; and Prometheus data sources, a PromQL query editor box with a query helper appear.
6. When you have finished constructing the query, choose **Graph query**.

The graph is populated with metrics from the query.

7. (Optional) Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.
 - d. For **Axis**, specify whether the numbers in **Value** refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

8. (Optional) To add this graph to a dashboard, choose **Actions, Add to dashboard**.

Updating a graph

To update your graph

1. To change the name of the graph, choose the pencil icon.
2. To change the time range, select one of the predefined values or choose **custom**. For more information, see [Modify the time range or time zone format for a graph](#).
3. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
4. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

5. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.
 - b. For **Value**, enter the metric value where the horizontal annotation appears.
 - c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose **Above** or **Below** for the corresponding area to be filled. If you specify **Between**, another **Value** field appears, and the area of the graph between the two values is filled.
 - d. For **Axis**, specify whether the numbers in **Value** refer to the metric associated with the left y-axis or the right y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

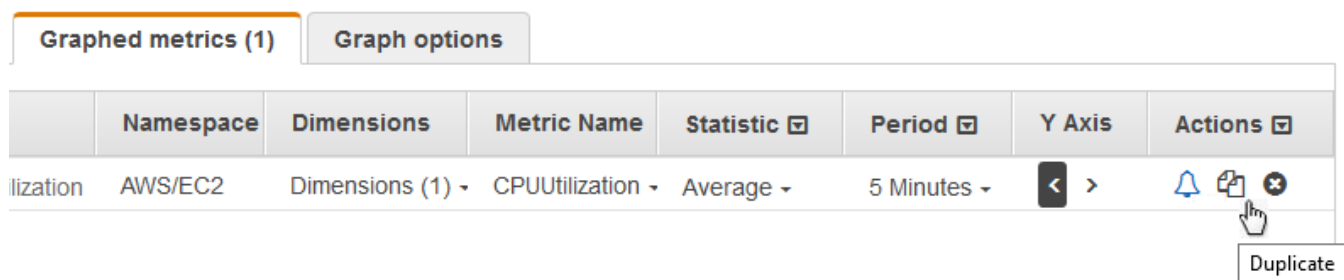
To delete an annotation, choose **x** in the **Actions** column.

6. To change the refresh interval, choose **Refresh options** and then select **Auto refresh** or choose **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Duplicating a metric

To duplicate a metric

1. Choose the **Graphed metrics** tab.
2. For **Actions**, choose the **Duplicate** icon.



3. Update the duplicate metric as needed.

Merge two graphs into one

You can merge two different graphs into one, and then the resulting graph shows both metrics. This can be useful if you already have different metrics displayed in different graphs and want to combine them, or you want to easily create a single graph with metrics from different Regions.

To merge a graph into another one, you use either the URL or JSON source of the graph that you want to merge in.

To merge two graphs into one

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Open the graph that you want to merge into another graph. To do so, you can choose **Metrics**, **All metrics**, and then choose a metric to graph. Or you can open a dashboard and then open one of the graphs on the dashboard by selecting the graph and choosing **Open in metrics** from the menu at the upper right of the graph.
3. After you have a graph open, do one of the following:
 - Copy the URL from the browser bar.
 - Choose the **Source** tab and then choose **Copy**.
4. Open the graph that you want to merge the previous graph into.
5. When you have the second graph open in the **Metrics** view, choose **Actions**, **Merge graph**.
6. Enter the URL or JSON that you previously copied, and choose **Merge**.
7. The merged graphs appear. The y-axis on the left is for the original graph, and the y-axis on the right is for the graph that you merged into it.

Note

If the graph that you merged into uses the **METRICS()** function, the metrics in the graph that was merged in are not included in the **METRICS()** calculation in the merged graph.

8. To save the merged graph to a dashboard, choose **Actions**, **Add to dashboard**.

Use dynamic labels

You can use dynamic labels with your graphs. Dynamic labels add a dynamically updated value to the label for the selected metric. You can add a wide range of values to the labels, as shown in the following tables.

The dynamic value shown in the label is derived from the time range currently shown on the graph. The dynamic part of the label automatically updates when either the dashboard or the graph is refreshed.

If you use a dynamic label with a search expression, the dynamic label applies to every metric returned by the search.

You can use the CloudWatch console to add a dynamic value to a label, edit the label, change the position of the dynamic value within the label column, and make other customizations.

Dynamic labels

Within a dynamic label, you can use the following values relating to properties of the metric:

| Dynamic label live value | Description |
|--|--|
| <code>\${AVG}</code> | The average of the values in the time range currently shown in the graph. |
| <code>\${DATAPOINT_COUNT}</code> | The number of data points in the time range that is currently shown in the graph. |
| <code>\${FIRST}</code> | The oldest of the metric values in the time range that is currently shown in the graph. |
| <code>\${FIRST_LAST_RANGE}</code> | The difference between the metric values of the oldest and newest data points that are currently shown in the graph. |
| <code>\${FIRST_LAST_TIME_RANGE}</code> | The absolute time range between the oldest and newest data points that are currently shown in the graph. |
| <code>\${FIRST_TIME}</code> | The timestamp of the oldest data point in the time range that is currently shown in the graph. |

| Dynamic label live value | Description |
|--------------------------------------|--|
| <code>\${FIRST_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the oldest data point in the time range that is currently shown in the graph. |
| <code>\${LABEL}</code> | The representation of the default label for a metric. |
| <code>\${LAST}</code> | The most recent of the metric values in the time range that is currently shown in the graph. |
| <code>\${LAST_TIME}</code> | The timestamp of the newest data point in the time range that is currently shown in the graph. |
| <code>\${LAST_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the newest data point in the time range that is currently shown in the graph. |
| <code>\${MAX}</code> | The maximum of the values in the time range currently shown in the graph. |
| <code>\${MAX_TIME}</code> | The timestamp of the data point that has the highest metric value, of the data points that are currently shown in the graph. |
| <code>\${MAX_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the data point with the highest value, of those data points that are currently shown in the graph. |
| <code>\${MIN}</code> | The minimum of the values in the time range currently shown in the graph. |
| <code>\${MIN_MAX_RANGE}</code> | The difference in metric values between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph. |
| <code>\${MIN_MAX_TIME_RANGE}</code> | The absolute time range between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph. |

| Dynamic label live value | Description |
|---|---|
| <code>\${MIN_TIME}</code> | The timestamp of the data point that has the lowest metric value, of the data points that are currently shown in the graph. |
| <code>\${MIN_TIME_RELATIVE}</code> | The absolute time difference between now and the timestamp of the data point with the lowest value, of those data points that are currently shown in the graph. |
| <code>\${PROP('AccountId')}</code> | The AWS account ID of the metric. |
| <code>\${PROP('AccountLabel')}</code> | The label specified for the source account that owns this metric, in CloudWatch cross-account observability. |
| <code>\${PROP('Dim.<i>dimension</i> _name ')}</code> | The value of the specified dimension. Replace <i>dimension _name</i> with the case-sensitive name of your dimension. |
| <code>\${PROP('MetricName')}</code> | The name of the metric. |
| <code>\${PROP('Namespace')}</code> | The namespace of the metric. |
| <code>\${PROP('Period')}</code> | The period of the metric, in seconds. |
| <code>\${PROP('Region')}</code> | The AWS Region where the metric is published. |
| <code>\${PROP('Stat')}</code> | The metric statistic that is being graphed. |
| <code>\${SUM}</code> | The sum of the values in the time range currently shown in the graph. |

For example, suppose you have a search expression `SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum')`, which finds the Errors for each of your Lambda functions. If you set the label to be `[max: ${MAX} Errors for Function Name ${LABEL}]`, the label for each metric is `[max: number Errors for Function Name Name]`.

You can add as many as six dynamic values to a label. You can use the `${LABEL}` placeholder only once within each label.

Modify the time range or time zone format for a graph

This section describes how you can modify the date, time, and time zone format on a CloudWatch metrics graph. It also describes how you can zoom in on a graph to apply a specific time range. For information about creating a graph, see [Graph a metric](#).

Note

If the time range of a dashboard is shorter than the period used for a graph on the dashboard, the following happens:

- The graph is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
- The start time of the period for this data point is adjusted backwards to ensure that at least one data point can be displayed.

Set a relative time range

New interface

To specify a relative time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Relative** tab in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, **Months**.
4. After you specify a time range, choose **Apply**.

Original interface

To specify a relative time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Relative** in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, or **Months**.

Set an absolute time range

New interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Absolute** tab in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Original interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Absolute** in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Set the time zone format

New interface

To specify the time zone for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local time zone**.
4. After you make your changes, choose **Apply**.

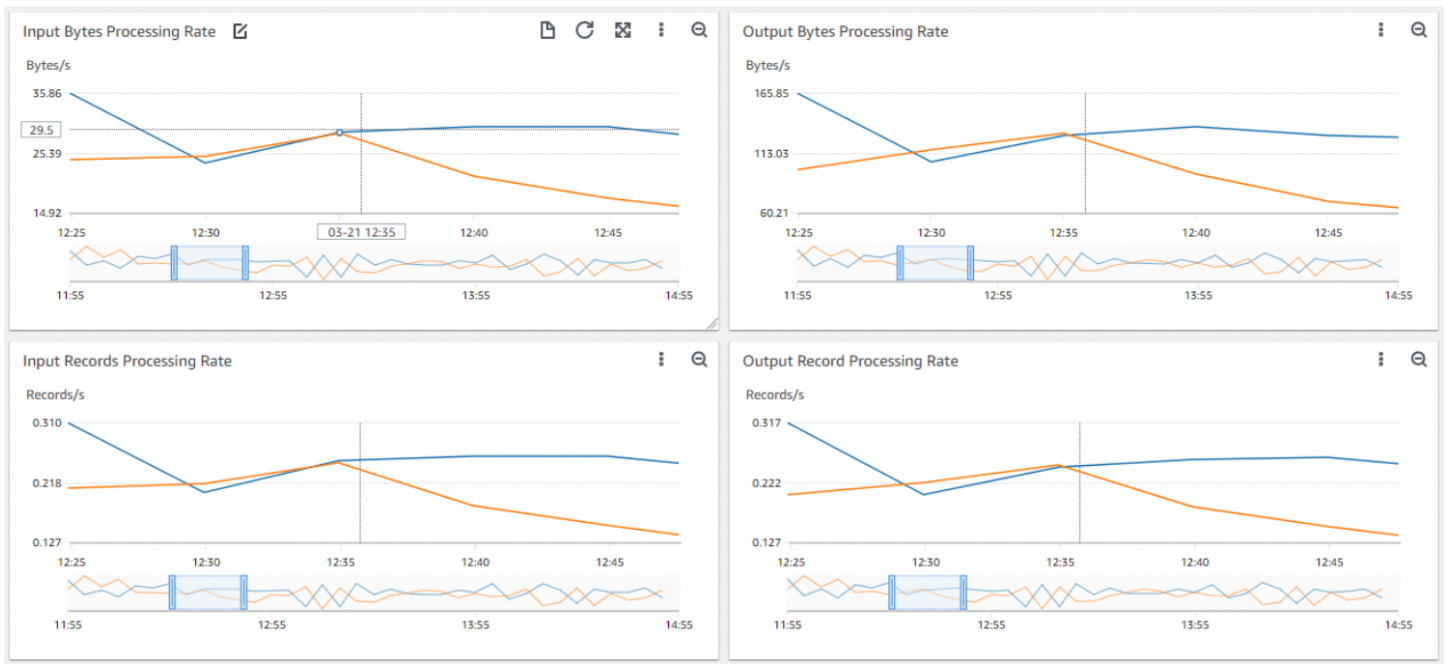
Original interface

To specify the time zone for a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local timezone**.

Zoom in on a line graph or stacked area graph

In the CloudWatch console, you can use the mini-map zoom feature to focus on sections of line graphs and stacked area graphs without changing between zoomed-in and zoomed-out views. For example, you can use the mini-map zoom feature to focus on a peak in a line graph, so that you can compare the spike against other metrics in your dashboard from the same timeline. The procedures in this section describe how to use the zoom feature.



In the preceding image, the zoom feature focuses on a spike in a line graph that's related to the input bytes processing rate while also showing other line graphs in the dashboard that focus on sections from the same timeline.

New interface

To zoom in on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
3. Choose **Browse**, and then select a metric or metrics to graph.
4. Choose **Options**, and select **Line** under **Widget type**.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Original interface

To zoom in on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
3. Choose **All metrics**, and then select a metric to graph.
4. Choose **Graph options**. Under **Widget type** select **Line**.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Tip

If you already created a dashboard that contains a line graph or stacked area graph, you can go to the dashboard and begin using the zoom feature.

Modify the y-axis for a graph

You can set custom bounds for the y-axis on a graph to help you see the data better. For example, you can change the bounds on a CPUUtilization graph to 100 percent so that it's easy to see whether the CPU is low (the plotted line is near the bottom of the graph) or high (the plotted line is near the top of the graph).

You can switch between two different y-axes for your graph. This is useful if the graph contains metrics that have different units or that differ greatly in their range of values.

To modify the y-axis on a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. On the **Graph options** tab, specify the **Min** and **Max** values for **Left Y Axis**. The value of **Min** can't be greater than the value of **Max**.

All metrics **Graphed metrics (1)** **Graph options**

Left Y Axis

Limits Min Max

Right Y Axis

Limits Min Max

- To create a second y-axis, specify the **Min** and **Max** values for **Right Y Axis**.
- To switch between the two y-axes, choose the **Graphed metrics** tab. For **Y Axis**, choose **Left Y Axis** or **Right Y Axis**.

Graphed metrics (1) **Graph options**

| | Namespace | Dimensions | Metric Name | Statistic | Period | Y Axis | Actions |
|----------|-----------|----------------|----------------|-----------|-----------|--|--|
| lization | AWS/EC2 | Dimensions (1) | CPUUtilization | Average | 5 Minutes | <input type="radio"/> Left Y Axis
<input checked="" type="radio"/> Right Y Axis | <input type="button" value="Alert"/> <input type="button" value="Share"/> <input type="button" value="Close"/> |

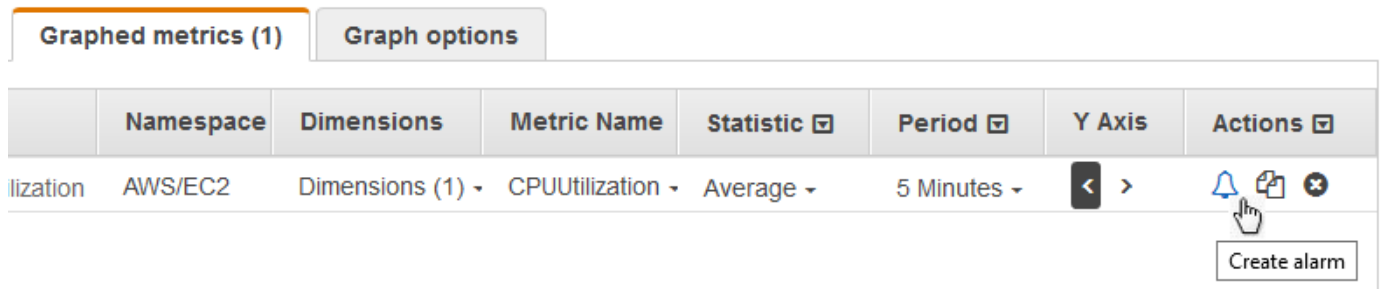
Create an alarm from a metric on a graph

You can graph a metric and then create an alarm from the metric on the graph, which has the benefit of populating many of the alarm fields for you.

To create an alarm from a metric on a graph

- Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
- In the navigation pane, choose **Metrics**.
- Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).

- The **All metrics** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
- To create an alarm for the metric, choose the **Graphed metrics** tab. For **Actions**, choose the alarm icon.



- Under **Conditions**, choose **Static** or **Anomaly detection** to specify whether to use a static threshold or anomaly detection model for the alarm.

Depending on your choice, enter the rest of the data for the alarm conditions.

- Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

- For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
- Choose **Next**.
- Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

- To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
- When finished, choose **Next**.

13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Using CloudWatch anomaly detection

When you enable *anomaly detection* for a metric, CloudWatch applies statistical and machine learning algorithms. These algorithms continuously analyze metrics of systems and applications, determine normal baselines, and surface anomalies with minimal user intervention.

The algorithms generate an anomaly detection model. The model generates a range of expected values that represent normal metric behavior.

You can enable anomaly detection using the AWS Management Console, the AWS CLI, AWS CloudFormation, or the AWS SDK. You can enable anomaly detection on metrics vended by AWS and also on custom metrics. In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

You can use the model of expected values in two ways:

- Create anomaly detection alarms based on a metric's expected value. These types of alarms don't have a static threshold for determining alarm state. Instead, they compare the metric's value to the expected value based on the anomaly detection model.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or both.

For more information, see [Create a CloudWatch alarm based on anomaly detection](#).

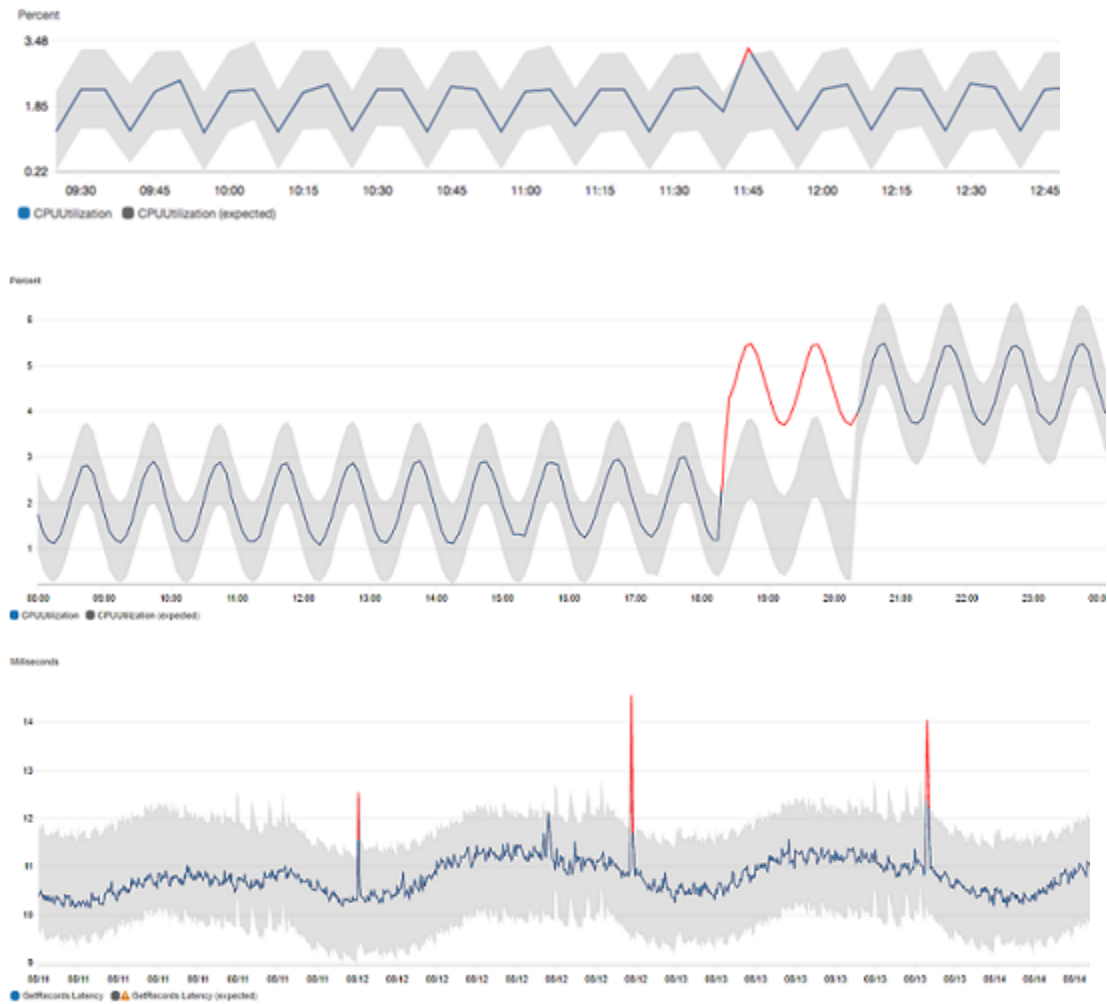
- When viewing a graph of metric data, overlay the expected values onto the graph as a band. This makes it visually clear which values in the graph are out of the normal range. For more information, see [Creating a graph](#).

You can also retrieve the upper and lower values of the model's band by using the `GetMetricData` API request with the `ANOMALY_DETECTION_BAND` metric math function. For more information, see [GetMetricData](#).

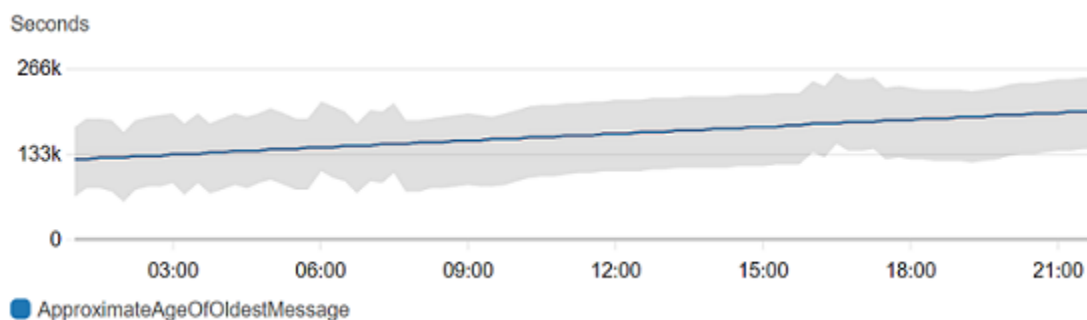
In a graph with anomaly detection, the expected range of values is shown as a gray band. If the metric's actual value goes beyond this band, it is shown as red during that time.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes could be hourly, daily, or weekly, as shown in the following examples.

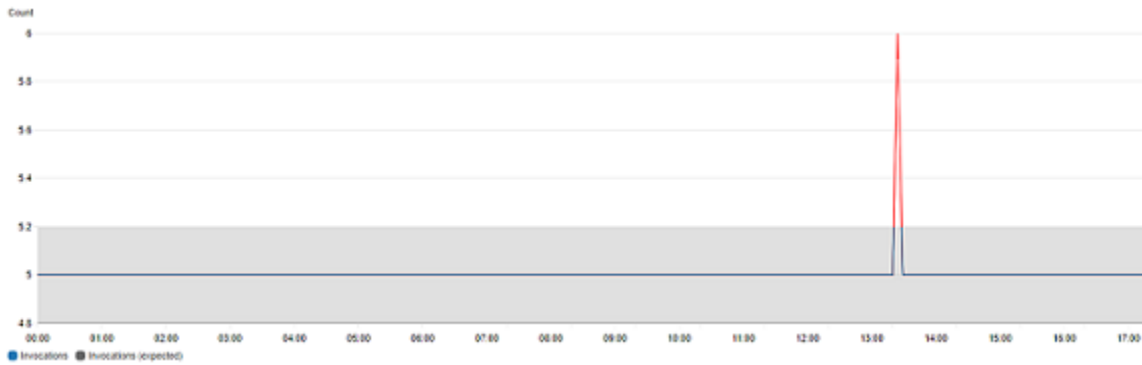
CPU with Anomaly Detection



The longer-range trends could be downward or upward.



Anomaly detections also works well with metrics with flat patterns.



How CloudWatch anomaly detection works

When you enable anomaly detection for a metric, CloudWatch applies machine learning algorithms to the metric's past data to create a model of the metric's expected values. The model assesses both trends and hourly, daily, and weekly patterns of the metric. The algorithm trains on up to two weeks of metric data, but you can enable anomaly detection on a metric even if the metric does not have a full two weeks of data.

You specify a value for the anomaly detection threshold that CloudWatch uses along with the model to determine the "normal" range of values for the metric. A higher value for the anomaly detection threshold produces a thicker band of "normal" values.

The machine learning model is specific to a metric and a statistic. For example, if you enable anomaly detection for a metric using the AVG statistic, the model is specific to the AVG statistic.

When CloudWatch creates a model for many common metrics from AWS services, it ensures that the band doesn't extend outside of logical values. For example, the band for `MemoryUtilization` of an EC2 instance will stay between 0 and 100, and the bands tracking `CloudFront Requests`, which can't be negative, will never extend below zero.

After you create a model, CloudWatch anomaly detection continually evaluates the model and makes adjustments to it to ensure that it is as accurate as possible. This includes re-training the model to adjust if the metric values evolve over time or have sudden changes, and also includes predictors to improve the models of metrics that are seasonal, spiky, or sparse.

After you enable anomaly detection on a metric, you can choose to exclude specified time periods of the metric from being used to train the model. This way, you can exclude deployments or other unusual events from being used for model training, ensuring the most accurate model is created.

Using anomaly detection models for alarms incurs charges on your AWS account. For more information, see [Amazon CloudWatch Pricing](#).

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on the output metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions. For information about functions that are not supported, see [Using metric math](#) in the *Amazon CloudWatch User Guide*.

You can create anomaly detection models based on metric math expressions similar to how you already create anomaly detection models. From the CloudWatch console, you can apply anomaly detection to metric math expressions and select anomaly detection as a threshold type for these expressions.

Note

Anomaly detection on metric math only can be enabled and edited in the latest version of the metrics user interface. When you create anomaly detectors based on metric math expressions in the new version of the interface, you can view them in the old version, but not edit them.

For information about how to create alarms and models for anomaly detection and metric math, see the following sections:

- [Creating a CloudWatch alarm based on anomaly detection](#)
- [Creating a CloudWatch alarm based on a metric math expression](#)

You also can create, delete, and discover anomaly detection models based on metric math expressions using the CloudWatch API with `PutAnomalyDetector`, `DeleteAnomalyDetector`, and `DescribeAnomalyDetectors`. For information about these API actions, see the following sections in the *Amazon CloudWatch API Reference*.

- [PutAnomalyDetector](#)
- [DeleteAnomalyDetector](#)

- [DescribeAnomalyDetectors](#)

For information about how anomaly detection alarms are priced, see [Amazon CloudWatch pricing](#).

Use metric math

Metric math enables you to query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. Using AWS Lambda metrics as an example, you could divide the `Errors` metric by the `Invocations` metric to get an error rate. Then add the resulting time series to a graph on your CloudWatch dashboard.

You can also perform metric math programmatically, using the `GetMetricData` API operation. For more information, see [GetMetricData](#).

Add a math expression to a CloudWatch graph

You can add a math expression to a graph on your CloudWatch dashboard. Each graph is limited to using a maximum of 500 metrics and expressions, so you can add a math expression only if the graph has 499 or fewer metrics. This applies even if not all the metrics are displayed on the graph.

To add a math expression to a graph

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Create or edit a graph. There needs to be at least one metric in the graph.
3. Choose **Graphed metrics**.
4. Choose **Math expression, Start with empty expression**. A new line appears for the expression.
5. In the new line, under the **Details** column, enter the math expression. The tables in the **Metric Math Syntax and Functions** section list the functions that you can use in the expression.

To use a metric or the result of another expression as part of the formula for this expression, use the value shown in the **Id** column: for example, `m1+m2` or `e1-MIN(e1)`.

You can change the value of **Id**. It can include numbers, letters, an underscore, and must start with a lowercase letter. Changing the value of **Id** to a more meaningful name can also make a graph easier to understand; for example, changing `m1` and `m2` to `errors` and `requests`.

Tip

Choose the down arrow next to **Math Expression** to see a list of supported functions, which you can use when creating your expression.

6. For the **Label** column of the expression, enter a name that describes what the expression is calculating.

If the result of an expression is an array of time series, each of those time series is displayed on the graph with a separate line, with different colors. Immediately under the graph is a legend for each line in the graph. For a single expression that produces multiple time series, the legend captions for those time series are in the format *Expression-Label Metric-Label*. For example, if the graph includes a metric with a label of **Errors** and an expression **FILL(METRICS(), 0)** that has a label of **Filled With 0:**, one line in the legend would be **Filled With 0: Errors**. To have the legend show only the original metric labels, set *Expression-Label* to be empty.

When one expression produces an array of time series on the graph, you can't change the colors used for each of those time series.

7. After you have added the desired expressions, you can simplify the graph by hiding some of the original metrics. To hide a metric or expression, clear the check box to the left of the **Id** field.

Metric math syntax and functions

The following sections explain the functions available for metric math. All functions must be written in uppercase letters (such as **AVG**), and the **Id** field for all metrics and math expressions must start with a lowercase letter.

The final result of any math expression must be a single time series or an array of time series. Some functions produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value in the time series.

Data type abbreviations

Some functions are valid for only certain types of data. The abbreviations in the following list are used in the tables of functions to represent the types of data supported for each function:

- **S** represents a scalar number, such as 2, -5, or 50.25.
- **TS** is a time series (a series of values for a single CloudWatch metric over time): for example, the `CPUUtilization` metric for instance `i-1234567890abcdef0` over the last three days.
- **TS[]** is an array of time series, such as the time series for multiple metrics.
- **String[]** is an array of strings.

The METRICS() function

The **METRICS()** function returns all the metrics in the request. Math expressions aren't included.

You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. For example, the expression **SUM(METRICS())** returns a time series (TS) that is the sum of the values of all the graphed metrics. **METRICS()/100** returns an array of time series, each of which is a time series showing each data point of one of the metrics divided by 100.

You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their **Id** field. You can also use **SUM([METRICS("4xx"), METRICS("5xx")])** to match multiple strings.

Basic arithmetic functions

The following table lists the basic arithmetic functions that are supported. Missing values in a time series are treated as 0. If the value of a data point causes a function to attempt to divide by zero, the data point is dropped.

| Operation | Arguments | Examples |
|---------------------------------|-----------|---------------|
| Arithmetic operators: + - * / ^ | S, S | PERIOD(m1)/60 |
| | S, TS | 5 * m1 |
| | TS, TS | m1 - m2 |

| Operation | Arguments | Examples |
|---------------------|-----------|---|
| | S, TS[] | SUM(100/[m1, m2]) |
| | TS, TS[] | AVG(METRICS())
METRICS()*100 |
| Unary subtraction - | S | -5*m1 |
| | TS | -m1 |
| | TS[] | SUM(-[m1, m2]) |

Comparison and logical operators

You can use comparison and logical operators with either a pair of time series or a pair of single scalar values. When you use a comparison operator with a pair of time series, the operators return a time series where each data point is either 0 (false) or 1 (true). If you use a comparison operator on a pair of scalar values, a single scalar value is returned, either 0 or 1.

When comparison operators are used between two time series, and only one of the time series has a value for a particular time stamp, the function treats the missing value in the other time series as **0**.

You can use logical operators in conjunction with comparison operators, to create more complex functions.

The following table lists the operators that are supported.

| Type of operator | Supported operators |
|----------------------|---------------------------|
| Comparison operators | ==
!=
<=
>=
< |

| Type of operator | Supported operators |
|-------------------|---------------------|
| | > |
| Logical operators | AND or &&
OR or |

To see how these operators are used, suppose we have two time series: **metric1** has values of [30, 20, 0, 0] and **metric2** has values of [20, -, 20, -] where - indicates that there is no value for that timestamp.

| Expression | Output |
|---------------------------------|------------|
| (metric1 < metric2) | 0, 0, 1, 0 |
| (metric1 >= 30) | 1, 0, 0, 0 |
| (metric1 > 15 AND metric2 > 15) | 1, 0, 0, 0 |

Functions supported for metric math

The following table describes the functions that you can use in math expressions. Enter all functions in uppercase letters.

The final result of any math expression must be a single time series or an array of time series. Some functions in tables in the following sections produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value of that data point.

In the following table, every example in the **Examples** column is an expression that results in a single time series or an array of time series. These examples show how functions that return scalar numbers can be used as part of a valid expression that produces a single time series.

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|-------------------------------|-------------|--------------|---|--|------------------------------|
| ABS | TS
TS[] | TS
TS[] | Returns the absolute value of each data point. | ABS(m1-m2)
MIN(ABS([m1, m2]))
ABS(METRICS()) | ✓ |
| ANOMALY_DETECTION_BAND | TS
TS, S | TS[] | Returns an anomaly detection band for the specified metric. The band consists of two time series, one representing the upper limit of the "normal" expected value of the metric, and the other representing the lower limit. The function can take two arguments . The first is the ID of the metric to create the band for. The second argument is the number of standard deviations to use for the band. If you don't specify this argument, the default of 2 is used. For more information, see Using CloudWatch anomaly detection . | ANOMALY_DETECTION_BAND(m1)
ANOMALY_DETECTION_BAND(m1,4) | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------|------------|--------------|--|---|------------------------------|
| AVG | TS
TS[] | S
TS | <p>The AVG of a single time series returns a scalar representing the average of all the data points in the metric. The AVG of an array of time series returns a single time series. Missing values are treated as 0.</p> | <p>SUM([m1,m2])/AVG(m2)</p> <p>AVG(METRICS())</p> | ✓ |

Note

We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `AVG(m2)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|-------------|------------|--------------|--|--|------------------------------|
| | | | <p>number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.</p> <p>To use this function with alarms, especially alarms that have Auto Scaling actions, we recommend that you set the alarm to use M out of N datapoints, where $M < N$.</p> | | |
| CEIL | TS
TS[] | TS
TS[] | Returns the ceiling of each metric. The ceiling is the smallest integer greater than or equal to each value. | CEIL(m1)
CEIL(METRICS())
SUM(CEIL(METRICS())) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------------------|------------|--------------|---|---|------------------------------|
| DATAPOINT_COUNT | TS
TS[] | S
TS | Returns a count of the data points that reported values. This is useful for calculating averages of sparse metrics. | SUM(m1) / DATAPOINT_COUNT(m1)

DATAPOINT_COUNT(METRICS()) | ✓ |

Note

We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--------------------------|----------|------------------------------|
| | | | extra data is requested. | | |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------------------|--|--|--|---|------------------------------|
| DB_PERF_INSIGHTS | String,
String,
String

String,
String,
String[] | TS (if given a single string)

TS[] (if given an array of strings) | Returns Performance Insights Counter metrics for databases such as Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). This function returns the same amount of data that you can get by directly querying the Performance Insights APIs. You can use these metrics in CloudWatch for graphing and creating alarms. | DB_PERF_INSIGHTS('RDS', 'db-ABCDE FGHIJKLMN OPQRSTUVWXYZ1', 'os.cpuUtilization.user.avg')

DB_PERF_INSIGHTS('DOCDB', 'db-ABCDEFGHIJKLMN OPQRSTUVWXYZ1', ['os.cpuUtilization.idle.avg', 'os.cpuUtilization.user.max']) | |

⚠ Important

When you use this function, you must specify the Unique Database Resource ID of the database. This is different than

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|----------|------------------------------|
| | | | <p>the database identifier.</p> <p>To find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the Configuration tab. The Resource ID is displayed in the Configuration section.</p> <p>DB_PERF_INSIGHTS also brings in the DBLoad metric at sub-minute intervals.</p> <p>Performance Insights metrics retrieved with this function are not stored in CloudWatch. Therefore, some CloudWatch features such as cross-acc</p> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|---|----------|------------------------------|
| | | | <p>ount observability, anomaly detection, metric streams, metrics explorer, and Metric Insights don't work with Performance Insights metrics that you retrieve with DB_PERF_INSIGHTS.</p> <p>A single request using the DB_PERF_INSIGHTS function can retrieve the following numbers of data points.</p> <ul style="list-style-type: none"> • 1080 data points for high-resolution periods (1s, 10s, 30s) • 1440 data points for standard-resolution periods (1m, 5m, 1hr, 1d) <p>The DB_PERF_INSIGHTS function supports only the following period lengths:</p> <ul style="list-style-type: none"> • 1 second | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|----------|------------------------------|
| | | | <ul style="list-style-type: none"> • 10 seconds • 30 seconds • 1 minute • 5 minutes • 1 hour • 1 day <p>For more information about Amazon RDS Performance Insights counter metrics, see Performance Insights counter metrics.</p> <p>For more information about Amazon DocumentDB Performance Insights counter metrics, see Performance Insights for counter metrics.</p> <div style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>High-resolution metrics with sub-minute granularity retrieved by DB_PERF_I</p> </div> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|---|----------|------------------------------|
| | | | <p>NSIGHTS are only applicable to the DBLoad metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see Monitoring OS metrics with Enhanced Monitoring. You can create a high-resolution alarm using the DB_PERF_I NSIGHTS function for a maximum time range of three hours. You can use the CloudWatc</p> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------------|------------|--------------|--|-----------------------------|------------------------------|
| | | | <p>h console to graph metrics retrieved with the DB_PERF_INSIGHTS function for any time range.</p> | | |
| DIFF | TS
TS[] | TS
TS[] | Returns the difference between each value in the time series and the preceding value from that time series. | DIFF(m1) | ✓ |
| DIFF_TIME | TS
TS[] | TS
TS[] | Returns the difference in seconds between the timestamp of each value in the time series and the timestamp of the preceding value from that time series. | DIFF_TIME(METRICS()) | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------|---|--------------|--|--|------------------------------|
| FILL | TS,
[S
REPEAT

LINEAR],
TS[],
[TS
 S
REPEAT

LINEAR] | TS
TS[] | <p>Fills the missing values of a time series. There are several options for the values to use as the filler for missing values:</p> <ul style="list-style-type: none"> You can specify a value to use as the filler value. You can specify a metric to use as the filler value. You can use the REPEAT keyword to fill missing values with the most recent actual value of the metric before the missing value. You can use the LINEAR keyword to fill the missing values with values that create a linear interpolation between the values at the beginning and the end of the gap. | FILL(m1,10)


FILL(METRICS(), 0)

FILL(METRICS(), m1)

FILL(m1, MIN(m1))

FILL(m1, REPEAT)

FILL(METRICS(), LINEAR) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|----------|------------------------------|
| | | | <p> Note</p> <p>When you use this function in an alarm, you can encounter an issue if your metrics are being published with a slight delay, and the most recent minute never has data. In this case, FILL replaces that missing data point with the requested value. That causes the latest data point for the metric to always be the FILL value, which can result in the alarm being stuck in either OK state or ALARM state. You can</p> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|-----------------------------|------------|--------------|---|--|------------------------------|
| | | | work around this by using a M out of N alarm. For more information, see Evaluating an alarm . | | |
| FIRST
LAST | TS[] | TS | Returns the first or last time series from an array of time series. This is useful when used with the Sort function. It can also be used to get the high and low thresholds from the ANOMALY_DETECTION_BAND function. | IF(FIRST(SORT(METRICS(), AVG, DESC))>100, 1, 0) Looks at the top metric from an array, which is sorted by AVG. It then returns a 1 or 0 for each data point, depending on whether that data point value is more than 100.

LAST(ANOMALY_DETECTION_BAND(m1)) returns the upper bound of the anomaly prediction band. | ✓ |
| FLOOR | TS
TS[] | TS
TS[] | Returns the floor of each metric. The floor is the largest integer less than or equal to each value. | FLOOR(m1)

FLOOR(METRICS()) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------------------------|--|--------------|--|--|------------------------------|
| IF | IF expression | TS | Use IF along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series. For more information, see Using IF expressions . | For examples, see Using IF expressions . | ✓ |
| INSIGHT_RULE_METRIC | INSIGHT_RULE_METRIC (ruleName, metricName) | TS | Use INSIGHT_RULE_METRIC to extract statistics from a rule in Contributor Insights. For more information, see Graphing metrics generated by rules . | | |
| LAMBDA | LAMBDA (LambdaFunctionName[, optionalArg]*) | TS
TS[] | Calls a Lambda function to query metrics from a data source that is not CloudWatch. For more information, see How to pass arguments to your Lambda function . | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|--------------|------------|--------------|---|-----------------------|------------------------------|
| LOG | TS
TS[] | TS
TS[] | The LOG of a time series returns the natural logarithm value of each value in the time series. | LOG(METRICS()) | ✓ |
| LOG10 | TS
TS[] | TS
TS[] | The LOG10 of a time series returns the base-10 logarithm value of each value in the time series. | LOG10(m1) | ✓ |


| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------|----------|--------------|---|---|------------------------------|
| MAX | TS | S | <p>The MAX of a single time series returns a scalar representing the maximum value of all data points in the metric.</p> <p>If you input an array of time series, the MAX function creates and returns a time series that consists of the highest value for each data point, among the time series that were used as the input.</p> | <p>MAX(m1)/m1</p> <p>MAX(METRICS())</p> | ✓ |
| | TS[] | TS | | | |

Note

We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `MAX(m2)`. Whenever an

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|---------------------|----------|--------------|---|-----------------------------------|------------------------------|
| | | | alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | | |
| METRIC_COUNT | TS[] | S | Returns the number of metrics in the time series array. | m1/METRIC_COUNT(METRICS()) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------------|----------------|--------------|---|---|------------------------------|
| METRICS | null
string | TS[] | <p>The METRICS() function returns all CloudWatch metrics in the request. Math expressions aren't included.</p> <p>You can use METRICS() within a larger expression that produces a single time series or an array of time series.</p> <p>You can use the METRICS() function with a string to return only the graphed metrics that contain that string in their Id field. For example, the expression SUM(METRICS("errors")) returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their Id field. You can also use SUM([METRICS("4xx"), METRICS("5xx")]) to match multiple strings.</p> | <p>AVG(METRICS())</p> <p>SUM(METRICS("errors"))</p> | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|------------|------------|--------------|--|--|------------------------------|
| MIN | TS
TS[] | S
TS | <p>The MIN of a single time series returns a scalar representing the minimum value of all data points in the metric.</p> <p>If you input an array of time series, the MIN function creates and returns a time series that consists of the lowest value for each data point, among the time series that were used as the input.</p> <p>If you input an array of time series, the MIN function creates and returns a time series that consists of the lowest value for each data point, among the time series that were used as the input.</p> <div data-bbox="634 1654 987 1837" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>We recommend that you do</p> </div> | m1-MIN(m1)

MIN(METRICS()) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|---|----------|------------------------------|
| | | | <p>not use this function in CloudWatch alarms if you want the function to return a scalar. For example, MIN(m2) Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.</p> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|--|----------|--------------|---|---|------------------------------|
| MINUTE
HOUR
DAY
DATE
MONTH
YEAR
EPOCH | TS | TS | <p>These functions take the period and range of the time series and return a new non-sparse time series where each value is based on its timestamp.</p> <ul style="list-style-type: none"> • MINUTE returns a non-sparse time series of integers between 0 and 59 that represent the UTC minute of each timestamp in the original time series. • HOUR returns a non-sparse time series of integers between 0 and 23 that represent the UTC hour of each timestamp in the original time series. • DAY returns a non-sparse time series of integers between 1 and 7 that represent the UTC day of the week of each timestamp in the | <p>MINUTE(m1)</p> <p>IF(DAY(m1)<6,m1) returns metrics only from weekdays, Monday to Friday.</p> <p>IF(MONTH(m1) == 4,m1) returns only metrics published in April.</p> | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|----------|------------------------------|
| | | | <p>original time series.
1 represents Monday and 7 represents Sunday.</p> <ul style="list-style-type: none"> • DATE returns a non-sparse time series of integers between 1 and 31 that represent the UTC day of the month of each timestamp in the original time series. • MONTH returns a non-sparse time series of integers between 1 and 12 that represent the UTC month of each timestamp in the original time series.
1 represents January and 12 represents December. • YEAR returns a non-sparse time series of integers that represent the UTC year of each timestamp in the original time series. | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|---------------|----------|--------------|---|----------------------|------------------------------|
| | | | <ul style="list-style-type: none"> EPOCH returns a non-sparse time series of integers that represent the UTC time in seconds since the Epoch of each timestamp in the original time series. The Epoch is January 1, 1970. | | |
| PERIOD | TS | S | Returns the period of the metric in seconds. Valid input is metrics, not the results of other expressions. | m1/PERIOD(m1) | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|-------------|------------|--------------|--|---|------------------------------|
| RATE | TS
TS[] | TS
TS[] | Returns the rate of change of the metric per second. This is calculated as the difference between the latest data point value and the previous data point value, divided by the time difference in seconds between the two values. | RATE(m1)
RATE(METRICS()) | ✓ |

⚠ Important

Setting alarms on expressions that use the RATE function on metrics with sparse data can behave unpredictably, because the range of data points fetched when evaluating the alarm can vary based on when the data

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|-----------------------------|----------|------------------------------|
| | | | points were last published. | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|---------------------|----------|--------------|---|--------------------------------|------------------------------|
| REMOVE_EMPTY | TS[] | TS[] | Removes any time series that have no data points from an array of time series. The result is an array of time series where each time series contains at least one data point. | REMOVE_EMPTY(METRICS()) | ✓ |

Note

We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|----------|------------------------------|
| | | | <div style="border: 1px solid #ccc; border-radius: 10px; padding: 5px; background-color: #e0f0ff;"> differently when extra data is requested. </div> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|--------------------|------------|--------------|---|-----------------------------|------------------------------|
| RUNNING_SUM | TS
TS[] | TS
TS[] | Returns a time series with the running sum of the values in the original time series. | RUNNING_SUM([m1,m2]) | ✓ |

Note

We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|---------------|-------------------|----------------|---|----------|------------------------------|
| SEARCH | Search expression | One or more TS | <p>Returns one or more time series that match a search criteria that you specify. The SEARCH function enables you to add multiple related time series to a graph with one expression. The graph is dynamically updated to include new metrics that are added later and match the search criteria. For more information, see Use search expressions in graphs.</p> <p>You can't create an alarm based on a SEARCH expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.</p> <p>If you are signed in to a monitoring account in CloudWatch cross-</p> | | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------------------|---------------------------|--------------|--|----------|------------------------------|
| | | | account observability, the SEARCH function finds metrics in the source accounts and the monitoring account. | | |
| SERVICE_QUOTA | TS that is a usage metric | TS | Returns the service quota for the given usage metric. You can use this to visualize how your current usage compares to the quota, and to set alarms that warn you when you approach the quota. For more information, see AWS usage metrics . | | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|--------------|---------------------------------|----------------|--|---|------------------------------|
| SLICE | (TS[], S, S)
or
(TS[], S) | TS[]

TS | <p>Retrieves part of an array of time series. This is especially useful when combined with SORT. For example, you can exclude the top result from an array of time series.</p> <p>You can use two scalar arguments to define the set of time series that you want returned. The two scalars define the start (inclusive) and end (exclusive) of the array to return. The array is zero-indexed, so the first time series in the array is time series 0. Alternatively, you can specify just one value, and CloudWatch returns all time series starting with that value.</p> | <p>SLICE(SORT(METRICS ()), SUM, DESC), 0, 10) returns the 10 metrics from the array of metrics in the request that have the highest SUM value.</p> <p>SLICE(SORT(METRICS ()), AVG, ASC), 5) sorts the array of metrics by the AVG statistic, then returns all the time series except for the 5 with the lowest AVG.</p> | ✓ |

| Function | Arguments | Return type* | Description | Examples | Supported for cross-account? |
|-------------|---|--------------|---|---|------------------------------|
| SORT | (TS[], FUNCTION_NAME)
SORT_ORDER)

(TS[], FUNCTION_NAME)
SORT_ORDER, S) | TS[] | <p>Sorts an array of time series according to the function you specify. The function you use can be AVG, MIN, MAX, or SUM. The sort order can be either ASC for ascending (lowest values first) or DESC to sort the higher values first. You can optionally specify a number after the sort order which acts as a limit. For example, specifying a limit of 5 returns only the top 5 time series from the sort.</p> <p>When this math function is displayed on a graph, the labels for each metric in the graph are also sorted and numbered.</p> | <p>SORT(METRICS(), AVG, DESC, 10) calculates the average value of each time series, sorts the time series with the highest values at the beginning of the sort, and returns only the 10 time series with the highest averages.</p> <p>SORT(METRICS(), MAX, ASC) sorts the array of metrics by the MAX statistic, then returns all of them in ascending order.</p> | ✓ |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|---------------|------------|--------------|---|--|------------------------------|
| STDDEV | TS
TS[] | S
TS | The STDDEV of a single time series returns a scalar representing the standard deviation of all data points in the metric. The STDDEV of an array of time series returns a single time series. | m1/STDDEV(m1)
STDDEV(METRICS()) | ✓ |

Note

We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `STDDEV(m2)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|---|----------|------------------------------|
| | | | <p>points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.</p> | | |

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|----------|----------|--------------|--|--|------------------------------|
| SUM | TS | S | The SUM of a single time series returns a scalar representing the sum of the values of all data points in the metric. The SUM of an array of time series returns a single time series. | SUM(METRICS())/SUM(m1)

SUM([m1,m2])

SUM(METRICS("errors"))/SUM(METRICS("requests"))*100 | ✓ |
| | TS[] | TS | | | |

Note

We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `SUM(m1)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data

| Function | Argument | Return type* | Description | Examples | Supported for cross-account? |
|--------------------|----------|--------------|--|--|------------------------------|
| | | | points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. | | |
| TIME_SERIES | S | TS | Returns a non-sparse time series where every value is set to a scalar argument. | TIME_SERIES(MAX(m1))

TIME_SERIES(5*AVG(m1))

TIME_SERIES(10) | ✓ |

*Using a function that returns only a scalar number is not valid, as all final results of expressions must be a single time series or an array of time series. Instead, use these functions as part of a larger expression that returns a time series.

Using IF expressions

Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series.


IF uses the following arguments:

```
IF(condition, trueValue, falseValue)
```

The condition evaluates to FALSE if the value of the condition data point is 0, and to TRUE if the value of the condition is any other value, whether that value is positive or negative. If the condition is a time series, it is evaluated separately for every timestamp.

The following lists the valid syntaxes. For each of these syntaxes, the output is a single time series.

- **IF(TS *Comparison Operator* S, S | TS, S / TS)**

 **Note**

If the TS *comparison operator* S is TRUE but *metric2* doesn't have a corresponding data point, the output will be 0.

- **IF(TS, TS, TS)**
- **IF(TS, S, TS)**
- **IF(TS, TS, S)**
- **IF(TS, S, S)**
- **IF(S, TS, TS)**

The following sections provide more details and examples for these syntaxes.

IF(TS *Comparison Operator* S, scalar2 | metric2, scalar3 | metric3)

The corresponding output time series value:

- has the value of **scalar2** or **metric2**, if **TS *Comparison Operator* S** is TRUE
- has the value of **scalar3** or **metric3**, if **TS *Comparison Operator* S** is FALSE
- has the value of **0** if the **TS *Comparison Operator*** is TRUE and the corresponding data point in **metric2** doesn't exist.
- has the value of **0** if the **TS *Comparison Operator*** is FALSE and the corresponding data point in **metric3** doesn't exist.
- is an empty time series, if the corresponding data point of does not exist in **metric3**, or if **scalar3/metric3** is omitted from the expression

IF(metric1, metric2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist in **metric3**
- is dropped, if the corresponding data point of **metric1** is FALSE and **metric3** is omitted from the expression.
- is dropped, if the corresponding data point of **metric1** is missing.

The following table shows an example for this syntax.

| Metric or function | Values |
|--------------------------------------|-------------------|
| (metric1) | [1, 1, 0, 0, -] |
| (metric2) | [30, -, 0, 0, 30] |
| (metric3) | [0, 0, 20, -, 20] |
| IF(metric1, metric2, metric3) | [30, 0, 20, 0, -] |

IF(metric1, scalar2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **scalar2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist on **metric3**, or if **metric3** is omitted from the expression.

| Metric or function | Values |
|--------------------|-----------------|
| (metric1) | [1, 1, 0, 0, -] |

| Metric or function | Values |
|-------------------------------|-------------------|
| scalar2 | 5 |
| (metric3) | [0, 0, 20, -, 20] |
| IF(metric1, scalar2, metric3) | [5, 5, 20, -, -] |

IF(metric1, metric2, scalar3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **scalar3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped if the corresponding data point in **metric1** does not exist.

| Metric or function | Values |
|-------------------------------|-------------------|
| (metric1) | [1, 1, 0, 0, -] |
| (metric2) | [30, -, 0, 0, 30] |
| scalar3 | 5 |
| IF(metric1, metric2, scalar3) | [30, 0, 5, 5, -] |

IF(scalar1, metric2, metric3)

The corresponding output time series value:

- has the value of **metric2**, if **scalar1** is TRUE.
- has the value of **metric3**, if **scalar1** is FALSE.
- is an empty time series, if **metric3** is omitted from the expression.

Use case examples for IF expressions

The following examples illustrate the possible uses of the **IF** function.

- To display only the low values of a metric:

IF(metric1<400, metric1)

- To change each data point in a metric to one of two values, to show relative highs and lows of the original metric:

IF(metric1<400, 10, 2)

- To display a 1 for each timestamp where latency is over the threshold, and display a 0 for all other data points:

IF(latency>threshold, 1, 0)

Use metric math with the GetMetricData API operation

You can use `GetMetricData` to perform calculations using math expressions, and also retrieve large batches of metric data in one API call. For more information, see [GetMetricData](#).

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions.

Anomaly detection on metric math doesn't support the following functions:

- Expressions that contain more than one **ANOMALY_DETECTION_BAND** in the same line.
- Expressions that contain more than 10 metrics or math expressions.
- Expressions that contain the **METRICS** expression.
- Expressions that contain the **SEARCH** function.
- Expressions that use the **DP_PERF_INSIGHTS** function.
- Expressions that use metrics with different periods.
- Metric math anomaly detectors that use high-resolution metrics as input.

For more information about this feature, see [Using CloudWatch anomaly detection](#) in the *Amazon CloudWatch User Guide*.

Use search expressions in graphs

Search expressions are a type of math expression that you can add to CloudWatch graphs. Search expressions enable you to quickly add multiple related metrics to a graph. They also enable you to create dynamic graphs that automatically add appropriate metrics to their display, even if those metrics don't exist when you first create the graph.

For example, you can create a search expression that displays the `AWS/EC2 CPUUtilization` metric for all instances in the Region. If you later launch a new instance, the `CPUUtilization` of the new instance is automatically added to the graph.

When you use a search expression in a graph, the search finds the search expression in metric names, namespaces, dimension names, and dimension values. You can use Boolean operators for more complex and powerful searches. A search expression can find only metrics that have reported data within the past two weeks.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

If you are using a monitoring account in CloudWatch cross-account observability, your search expressions can find metrics in the source accounts linked to that monitoring account.

Topics

- [CloudWatch search expression syntax](#)
- [CloudWatch search expression examples](#)
- [Create a CloudWatch graph with a search expression](#)

CloudWatch search expression syntax

A valid search expression has the following format.

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic')
```

For example:

```
SEARCH( '{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average' )
```

- The first part of the query after the word `SEARCH`, enclosed in curly braces, is the *metric schema* to be searched. The metric schema contains a metric namespace and one or more dimension names. Including a metric schema in a search query is optional. If specified, the metric schema must contain a namespace and can optionally contain one or more dimension names that are valid in that namespace.

You don't need to use quote marks inside the metric schema unless a namespace or dimension name includes spaces or non-alphanumeric characters. In that case, you must enclose the name that contains those characters with double quotes.

- The `SearchTerm` is also optional, but a valid search must contain either the metric schema, the `SearchTerm`, or both. The `SearchTerm` usually contains one or more account IDs, metric names or dimension values. The `SearchTerm` can include multiple terms to search for, by both partial match and exact match. It can also contain Boolean operators.

Using an account ID in a `SearchTerm` works only in accounts that are set up as monitoring accounts for CloudWatch cross-account observability. The syntax for an account ID in `SearchTerm` is `:aws.AccountId = "444455556666"`. You can also use `'LOCAL'` to specify the monitoring account itself: `:aws.AccountId = 'LOCAL'`

For more information, see [CloudWatch cross-account observability](#).

The `SearchTerm` can include one or more designators, such as `MetricName=` as in this example, but using designators isn't required.

The metric schema and `SearchTerm` must be enclosed together in a pair of single quote marks.

- The `Statistic` is the name of any valid CloudWatch statistic. It must be enclosed by single quotes. For more information, see [Statistics](#).

The preceding example searches the `AWS/EC2` namespace for any metrics that have `InstanceId` as a dimension name. It returns all `CPUUtilization` metrics that it finds, with the graph showing the `Average` statistic.

A search expression can find only metrics that have reported data within the past two weeks.

Search expression limits

The maximum search expression query size is 1024 characters. You can have as many as 100 search expressions on one graph. A graph can display as many as 500 time series.

CloudWatch search expressions: Tokenization

When you specify a `SearchTerm`, the search function searches for *tokens*, which are substrings that CloudWatch automatically generates from full metric names, dimension names, dimension values, and namespaces. CloudWatch generates tokens distinguished by the camel-case capitalization in the original string. Numeric characters also serve as the start of new tokens, and non-alphanumeric characters serve as delimiters, creating tokens before and after the non-alphanumeric characters.

A continuous string of the same type of token delimiter character results in one token.

All generated tokens are in lowercase. The following table shows some examples of tokens generated.

| Original string | Tokens generated |
|-------------------|---|
| CustomCount1 | customcount1 , custom, count, 1 |
| SDBFailure | sdbfailure , sdb, failure |
| Project2-trial333 | project2trial333 , project, 2, trial, 333 |

CloudWatch search expressions: Partial matches

When you specify a `SearchTerm`, the search term is also tokenized. CloudWatch finds metrics based on partial matches, which are matches of a single token generated from the search term to a single token generated from a metric name, namespace, dimension name, or dimension value.

Partial match searches to match a single token are case insensitive. For example, using any of the following search terms can return the `CustomCount1` metric:

- count
- Count
- COUNT

However, using `couNT` as a search term doesn't find `CustomCount1` because the capitalization in the search term `couNT` is tokenized into `cou` and `NT`.

Searches can also match composite tokens, which are multiple tokens that appear consecutively in the original name. To match a composite token, the search is case sensitive. For example, if the original term is `CustomCount1`, searches for `CustomCount` or `Count1` are successful, but searches for `customcount` or `count1` aren't.

CloudWatch search expressions: Exact matches

You can define a search to find only exact matches of your search term by using double quotes around the part of the search term that requires an exact match. These double-quotes are enclosed in the single-quotes used around the entire search term. For example, `SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum')` finds the exact string `CustomCount1` if it exists as a metric name, dimension name, or dimension value in the namespace named `MyNamespace`. However, the searches `SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum')` or `SEARCH(' {MyNamespace}, "Custom" ', 'Maximum')` do not find this string.

You can combine partial match terms and exact match terms in a single search expression. For example, `SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')` returns the Elastic Load Balancing metric named `ConsumedLCUs` as well as all Elastic Load Balancing metrics or dimensions that contain the token `flow`.

Using exact match is also a good way to find names with special characters, such as non-alphanumeric characters or spaces, as in the following example.

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ', 'Maximum')
```

CloudWatch search expressions: Excluding a metric schema

All examples shown so far include a metric schema, in curly braces. Searches that omit a metric schema are also valid.

For example, `SEARCH(' "CPUUtilization" ', 'Average')` returns all metric names, dimension names, dimension values, and namespaces that are an exact match for the string `CPUUtilization`. In the AWS metric namespaces, this can include metrics from several services including Amazon EC2, Amazon ECS, SageMaker, and others.

To narrow this search to only one AWS service, the best practice is to specify the namespace and any necessary dimensions in the metric schema, as in the following example. Although this narrows the search to the AWS/EC2 namespace, it would still return results of other metrics if you have defined CPUUtilization as a dimension value for those metrics.

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average')
```

Alternatively you could add the namespace in the SearchTerm as in the following example. But in this example, the search would match any AWS/EC2 string, even if it was a custom dimension name or value.

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average')
```

CloudWatch search expressions: Specifying property names in the search

The following exact match search for "CustomCount1" returns all metrics with exactly that name.

```
SEARCH(' "CustomCount1" ', 'Maximum')
```

But it also returns metrics with dimension names, dimension values, or namespaces of CustomCount1. To structure your search further, you can specify the property name of the type of object that you want to find in your searches. The following example searches all namespaces and returns metrics named CustomCount1.

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum')
```

You can also use namespaces and dimension name/value pairs as property names, as in the following examples. The first of these examples also illustrates that you can use property names with partial match searches as well.

```
SEARCH(' InstanceType=micro ', 'Average')
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average')
```

CloudWatch search expressions: Non-alphanumeric characters

Non-alphanumeric characters serve as delimiters, and mark where the names of metrics, dimensions, namespaces, and search terms are to be separated into tokens. When terms are

tokenized, non-alphanumeric characters are stripped out and don't appear in the tokens. For example, `Network-Errors_2` generates the tokens `network`, `errors`, and `2`.

Your search term can include any non-alphanumeric characters. If these characters appear in your search term, they can specify composite tokens in a partial match. For example, all of the following searches would find metrics named either `Network-Errors-2` or `NetworkErrors2`.

```
network/errors
network+errors
network-errors
Network_Errors
```

When you're doing an exact value search, any non-alphanumeric characters used in the exact search must be the correct characters that appear in the string being searched for. For example, if you want to find `Network-Errors-2`, searching for `"Network-Errors-2"` is successful, but a search for `"Network_Errors_2"` isn't.

When you perform an exact match search, the following characters must be escaped with a backslash.

```
" \ ( )
```

For example, to find the metric name `Europe\France Traffic(Network)` by exact match, use the search term `"Europe\\France Traffic\\(Network\\)"`

CloudWatch search expressions: Boolean operators

Search supports the use of the Boolean operators `AND`, `OR`, and `NOT` within the `SearchTerm`. Boolean operators are enclosed in the single quote marks that you use to enclose the entire search term. Boolean operators are case sensitive, so `and`, `or`, and `not` aren't valid as Boolean operators.

You can use `AND` explicitly in your search, such as `SEARCH('{AWS/EC2,InstanceId} network AND packets ', 'Average ')`. Not using any Boolean operator between search terms implicitly searches them as if there were an `AND` operator, so `SEARCH('{AWS/EC2,InstanceId} network packets ', 'Average ')` yields the same search results.

Use `NOT` to exclude subsets of data from the results. For example, `SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average ')` returns the `CPUUtilization` for all your instances, except for the instance

i-1234567890123456. You can also use a NOT clause as the only search term. For example, **SEARCH('NOT Namespace=AWS ', 'Maximum')** yields all your custom metrics (metrics with namespaces that don't include AWS).

You can use multiple NOT phrases in a query. For example, **SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average')** returns the CPUUtilization of all instances in the Region, except for those with dimension values of ProjectA or ProjectB.

You can combine Boolean operators for more powerful and detailed searches, as in the following examples. Use parentheses to group the operators.

Both of the next two examples return all metric names containing ReadOps from both the EC2 and EBS namespaces.

```
SEARCH( ' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum' )
```

```
SEARCH( ' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum' )
```

The following example narrows the previous search to only results that include ProjectA, which could be the value of a dimension.

```
SEARCH( ' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum' )
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions, and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH( ' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND (ProjectA OR ProjectB)) ', 'Average' )
```

CloudWatch search expressions: Using math expressions

You can use a search expression within a math expressions in a graph.

For example, **SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum'))** returns the sum of the Errors metric of all your Lambda functions.

Using separate lines for your search expression and math expression might yield more useful results. For example, suppose that you use the following two expressions in a graph. The first line displays separate `Errors` lines for each of your Lambda functions. The ID of this expression is `e1`. The second line adds another line showing the sum of the errors from all of the functions.

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum')
SUM(e1)
```

CloudWatch search expression examples

The following examples illustrate more search expression uses and syntax. Let's start with a search for `CPUUtilization` across all instances in the Region and then look at variations.

This example displays one line for each instance in the Region, showing the `CPUUtilization` metric from the `AWS/EC2` namespace.

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average')
```

Changing `InstanceId` to `InstanceType` changes the graph to show one line for each instance type used in the Region. Data from all instances of each type is aggregated into one line for that instance type.

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average')
```

The following example aggregates the `CPUUtilization` by instance type and displays one line for each instance type that includes the string `micro`.

```
SEARCH(' {AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',
'Average')
```

This example narrows the previous example, changing the `InstanceType` to an exact search for `t2.micro` instances.

```
SEARCH(' {AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',
'Average')
```

The following search removes the `{metric schema}` part of the query, so the `CPUUtilization` metric from all namespaces appears in the graph. This can return quite a few results because the

graph includes multiple lines for the CPUUtilization metric from each AWS service, aggregated along different dimensions.

```
SEARCH('MetricName="CPUUtilization" ', 'Average')
```

To narrow these results a bit, you can specify two specific metric namespaces.

```
SEARCH('MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average')
```

The preceding example is the only way to do a search of specific multiple namespaces with one search query, as you can specify only one metric schema in each query. However, to add more structure, you could use two queries in the graph, as in the following example. This example also adds more structure by specifying a dimension to use to aggregate the data for Amazon ECS.

```
SEARCH('{AWS/ECS ClusterName}, MetricName="CPUUtilization" ', 'Average')  
SEARCH(' {AWS/EBS} MetricName="CPUUtilization" ', 'Average')
```

The following example returns the Elastic Load Balancing metric named ConsumedLCUs as well as all Elastic Load Balancing metrics or dimensions that contain the token flow.

```
SEARCH('{AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH('{AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND  
(ProjectA OR ProjectB)) ', 'Average')
```

The following example displays all of your custom metrics, excluding metrics generated by AWS services.

```
SEARCH('NOT Namespace=AWS ', 'Average')
```

The following example displays metrics with metric names, namespaces, dimension names, and dimension values that contain the string Errors as part of their name.

```
SEARCH('Errors', 'Average')
```

The following example narrows that search to exact matches. For example, this search finds the metric name `Errors` but not metrics named `ConnectionErrors` or `errors`.

```
SEARCH(' "Errors" ', 'Average')
```

The following example shows how to specify names that contain spaces or special characters in the metric schema part of the search term.

```
SEARCH('{ "Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum')
```

CloudWatch cross-account observability search expression examples

CloudWatch cross-account observability examples

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use the **SEARCH** function to return metrics from specified source accounts. For more information, see [CloudWatch cross-account observability](#).

The following example retrieves all Lambda metrics from the account with the account ID 111122223333.

```
SEARCH(' AWS/Lambda :aws.AccountId = "111122223333" ', 'Average')
```

The following example retrieves all AWS/EC2 metrics from two accounts: 111122223333 and 777788889999.

```
SEARCH(' AWS/EC2 :aws.AccountId = ("111122223333" OR "777788889999") ', 'Average')
```

The following example retrieves all AWS/EC2 metrics from the source account 111122223333 and from the monitoring account itself.

```
SEARCH(' AWS/EC2 :aws.AccountId = ("111122223333" OR 'LOCAL') ', 'Average')
```

The following example retrieves the SUM of the `MetaDataToken` metric from the account 444455556666 with the `InstanceId` dimension.

```
SEARCH('{AWS/EC2,InstanceId} :aws.AccountId=444455556666 MetricName=\"MetadataNoToken\n\"', 'Sum')
```

Create a CloudWatch graph with a search expression

On the CloudWatch console, you can access search capability when you add a graph to a dashboard, or by using the **Metrics** view.

You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To add a graph with a search expression to an existing dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose either **Line** or **Stacked area** and choose **Configure**.
5. On the **Graphed metrics** tab, choose **Add a math expression**.
6. For **Details**, enter the search expression that you want. For example, `SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')`
7. (Optional) To add another search expression or math expression to the graph, choose **Add a math expression**
8. (Optional) After you add a search expression, you can specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics** and then **Dynamic labels**.

By default, the dynamic values you add to the label appear at the beginning of the label. You can then click the **Label** value for the metric to edit the label. For more information, see [Use dynamic labels](#).
9. (Optional) To add a single metric to the graph, choose the **All metrics** tab and drill down to the metric you want.
10. (Optional) To change the time range shown on the graph, choose either **custom** at the top of the graph or one of the time periods to the left of **custom**.
11. (Optional) Horizontal annotations help dashboard users quickly see when a metric has spiked to a certain level or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:
 - a. For **Label**, enter a label for the annotation.

- b. For **Value**, enter the metric value where the horizontal annotation appears.
- c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose Above or Below for the corresponding area to be filled. If you specify Between, another Value field appears, and the area of the graph between the two values is filled.
- d. For **Axis**, specify whether the numbers in Value refer to the metric associated with the left y-axis or the right y-axis if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

12. (Optional) Vertical annotations help you mark milestones in a graph, such as operational events or the beginning and end of a deployment. To add a vertical annotation, choose **Graph options** and then **Add vertical annotation**:

- a. For **Label**, enter a label for the annotation. To show only the date and time on the annotation, keep the **Label** field blank.
- b. For **Date**, specify the date and time where the vertical annotation appears.
- c. For **Fill**, specify whether to use fill shading before or after a vertical annotation or between two vertical annotations. For example, choose Before or After for the corresponding area to be filled. If you specify Between, another Date field appears, and the area of the graph between the two values is filled.

Repeat these steps to add multiple vertical annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

13. Choose **Create widget**.

14. Choose **Save dashboard**.

To use the Metrics view to graph searched metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the search field, enter the tokens to search for: for example, **cpuutilization t2.small**.

Results that match your search appear.

4. To graph all of the metrics that match your search, choose **Graph search**.

or

To refine your search, choose one of the namespaces that appeared in your search results.

5. If you selected a namespace to narrow your results, you can do the following:
 - a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
 - b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
 - c. To view help for a metric, select the metric name and choose **What is this?**

The selected metrics appear on the graph.

6. (Optional) Select one of the buttons in the search bar to edit that part of the search term.
7. (Optional) To add the graph to a dashboard, choose **Actions** and then **Add to dashboard**.

Get statistics for a metric

CloudWatch statistics definitions

Statistics are metric data aggregations over specified periods of time. When you graph or retrieve the statistics for a metric, you specify the *Period* of time, such as five minutes, to use to calculate each statistical value. For example, if the **Period** is five minutes, the **Sum** is the sum of all sample values collected during the five-minute period, while the **Minimum** is the lowest value collected during the five-minute period.

CloudWatch supports the following statistics for metrics.

- **SampleCount** is the number of data points during the period.

- **Sum** is the sum of the values of the all data points collected during the period.
- **Average** is the value of $\text{Sum}/\text{SampleCount}$ during the specified period.
- **Minimum** is the lowest value observed during the specified period.
- **Maximum** is the highest value observed during the specified period.
- **Percentile (p)** indicates the relative standing of a value in a dataset. For example, **p95** is the 95th percentile and means that 95 percent of the data within the period is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.
- **Trimmed mean (TM)** is the mean of all values that are between two specified boundaries. Values outside of the boundaries are ignored when the mean is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. The numbers can be absolute values or percentages. For example, **tm90** calculates the average after removing the 10% of data points with the highest values. **TM(2%:98%)** calculates the average after removing the 2% lowest data points and the 2% highest data points. **TM(150:1000)** calculates the average after removing all data points that are lower than or equal to 150, or higher than 1000.
- **Interquartile mean (IQM)** is the trimmed mean of the *interquartile range*, or the middle 50% of values. It is equivalent to **TM(25%:75%)**.
- **Winsorized mean (WM)** is similar to trimmed mean. However, with winsorized mean, the values that are outside the boundary are not ignored, but instead are considered to be equal to the value at the edge of the appropriate boundary. After this normalization, the average is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. For example, **wm98** calculates the average while treating the 2% of the highest values to be equal to the value at the 98th percentile. **WM(10%:90%)** calculates the average while treating the highest 10% of data points to be the value of the 90% boundary, and treating the lowest 10% of data points to be the value of the 10% boundary.
- **Percentile rank (PR)** is the percentage of values that meet a fixed threshold. For example, **PR(:300)** returns the percentage of data points that have a value of 300 or less. **PR(100:2000)** returns the percentage of data points that have a value between 100 and 2000.

Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

- **Trimmed count (TC)** is the number of data points in the chosen range for a trimmed mean statistic. For example, **tc90** returns the number of data points not including any data points that fall in the highest 10% of the values. **TC(0.005:0.030)** returns the number of data points with values between 0.005 (exclusive) and 0.030 (inclusive).

- **Trimmed sum (TS)** is the sum of the values of data points in a chosen range for a trimmed mean statistic. It is equivalent to (Trimmed Mean) * (Trimmed count). For example, **ts90** returns the sum of the data points not including any data points that fall in the highest 10% of the values. **TS(80%:)** returns the sum of the data point values, not including any data points with values in the lowest 80% of the range of values.

Note

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, if you define two boundaries as fixed values instead of percentages, the calculation includes values equal to the higher boundary, but does not include values equal to the lower boundary.

Syntax

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, the following syntax rules apply:

- Using parentheses with one or two numbers with percent signs defines the boundaries to use as the values in the data set that fall in between the two percentiles that you specify. For example, **TM(10%:90%)** uses only the values between the 10th and 90th percentiles. **TM(:95%)** uses the values from the lowest end of the data set up to the 95th percentile, ignoring the 5% of data points with the highest values.
- Using parenthesis with one or two numbers without percent signs defines the boundaries to use as the values in the data set that fall in between the explicit values that you specify. For example, **TC(80:500)** uses only the values that are between 80 (exclusive) and 500 (inclusive). **TC(:0.5)** uses only the values that equal 0.5 or are lower.
- Using one number without parentheses calculates using percentages, ignoring data points that are higher than the specified percentile. For example, **tm99** calculates the mean while ignoring the 1% of the data points with the highest value. It is the same as **TM(:99%)**.
- Trimmed mean, Trimmed Count, Trimmed Sum, and Winsorized Mean can all be abbreviated using uppercase letters when specifying a range, such as **TM(5%:95%)**, **TM(100:200)**, or **TM(:95%)**. They can only be abbreviated using lowercase letters when you specifying only one number, such as **tm99**.

Statistics use cases

- **Trimmed mean** is most useful for metrics with a large sample size, such as webpage latency. For example, **tm99** disregards extreme high outliers that could result from network problems or human errors, to give a more accurate number for the average latency of typical requests. Similarly, **TM(10%:)** disregards the lowest 10% of latency values, such as those resulting from cache hits. And **TM(10%:99%)** excludes both of these types of outliers. We recommend that you use trimmed mean for monitoring latency.
- It is a good idea to keep watch on trimmed count whenever you are using trimmed mean, to make sure that the number of values being used in your trimmed mean calculations are enough to be statistically significant.
- Percentile rank enables you to put values into "bins" of ranges, and you can use this to manually create a histogram. To do this, break your values down into various bins, such as **PR(:1)**, **PR(1:5)**, **PR(5:10)**, and **PR(10:)**. Put each of these bins into a visualization as bar charts, and you have a histogram.

Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

Percentiles versus trimmed mean

A percentile such as **p99** and a trimmed mean such as **tm99** measure similar, but not identical values. Both **p99** and **tm99** ignore the 1% of the data points with the highest values, which are considered outliers. After that, **p99** is the maximum value of the remaining 99%, while **tm99** is the *average* of the remaining 99%. If you are looking at the latency of web requests, **p99** tells you the worst customer experience, ignoring outliers, while **tm99** tells you the average customer experience, ignoring outliers.

Trimmed mean is a good latency statistic to watch if you are looking to optimize your customer experience.

Requirements to use percentiles, trimmed mean, and some other statistics

CloudWatch needs raw data points to calculate the following statistics:

- Percentiles
- Trimmed mean
- Interquartile mean

- Winsorized mean
- Trimmed sum
- Trimmed count
- Percentile rank

If you publish data for a custom statistics using a statistic set instead of raw data, you can retrieve these types of statistics for this data only if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support these types of statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

Additionally, these type of statistics are not available for metrics when any of the metric values are negative numbers.

The following examples show you how to get statistics for the CloudWatch metrics for your resources, such as your EC2 instances.

Examples

- [Get statistics for a specific resource](#)
- [Aggregate statistics across resources](#)
- [Aggregate statistics by Auto Scaling group](#)
- [Aggregate statistics by Amazon Machine Image \(AMI\)](#)

Get statistics for a specific resource

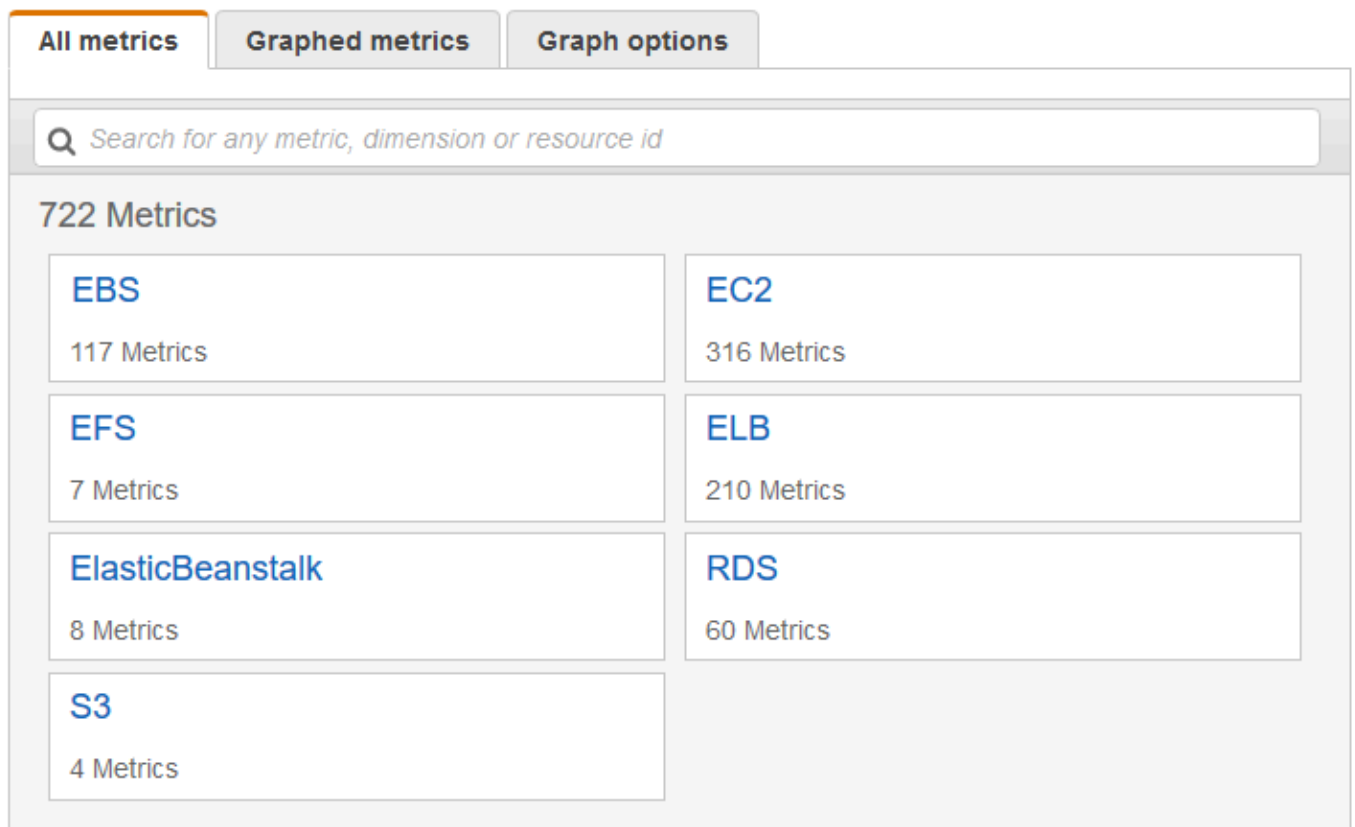
The following example shows you how to determine the maximum CPU utilization of a specific EC2 instance.

Requirements

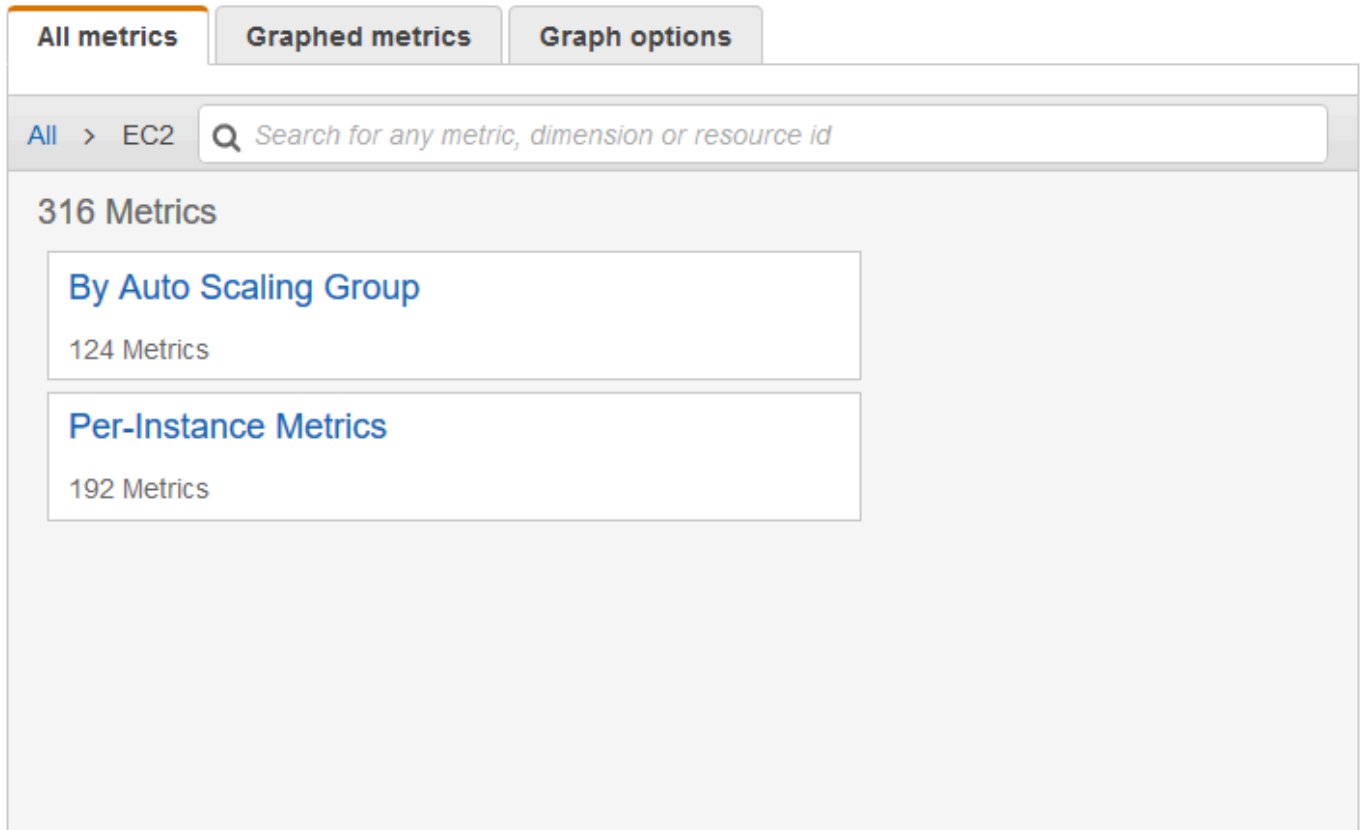
- You must have the ID of the instance. You can get the instance ID using the Amazon EC2 console or the [describe-instances](#) command.
- By default, basic monitoring is enabled, but you can enable detailed monitoring. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide*.

To display the average CPU utilization for a specific instance using the console

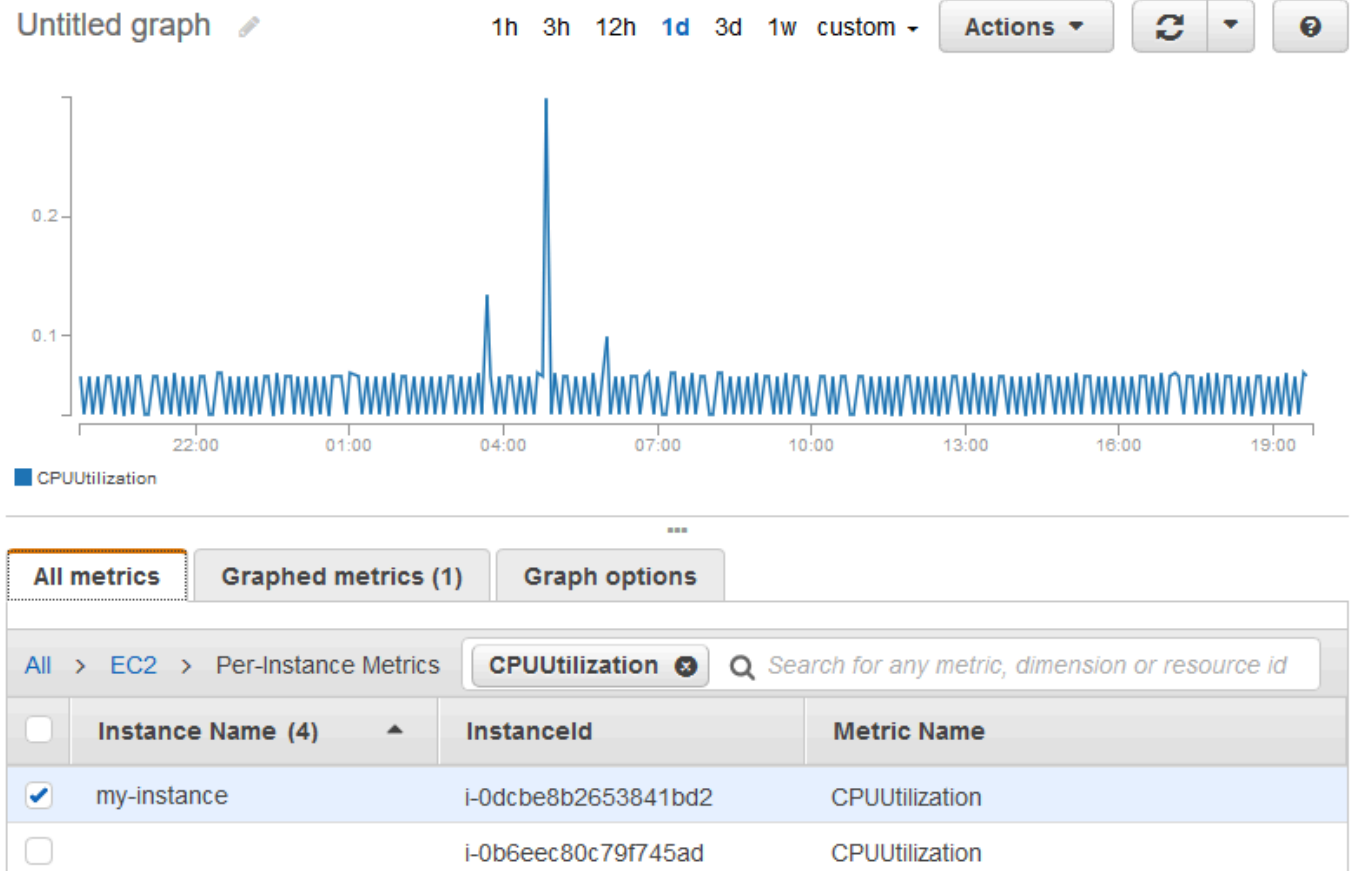
1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select the **EC2** metric namespace.



4. Select the **Per-Instance Metrics** dimension.



5. In the search field, enter **CPUUtilization** and press Enter. Select the row for the specific instance, which displays a graph for the CPUUtilization metric for the instance. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



- To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p99.999**).

| | Label | Namespace | Dimensions | Metric Name | Statistic | Period |
|--|----------------|-----------|----------------|----------------|-----------|-----------|
| | CPUUtilization | AWS/EC2 | Dimensions (1) | CPUUtilization | Average | 5 Minutes |

- To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value, and then choose a different value.

To get the CPU utilization per EC2 instance using the AWS CLI

Use the [get-metric-statistics](#) command as follows to get the CPUUtilization metric for the specified instance.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Maximum \
--start-time 2016-10-18T23:18:00 --end-time 2016-10-19T23:18:00 --period 360
```

The returned statistics are 6-minute values for the requested 24-hour time interval. Each value represents the maximum CPU utilization percentage for the specified instance for a particular 6-minute time period. The data points aren't returned in chronological order. The following shows the beginning of the example output (the full output includes data points for every 6 minutes of the 24-hour period).

```
{
  "Datapoints": [
    {
      "Timestamp": "2016-10-19T00:18:00Z",
      "Maximum": 0.33000000000000002,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2016-10-19T03:18:00Z",
      "Maximum": 99.670000000000002,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2016-10-19T07:18:00Z",
      "Maximum": 0.34000000000000002,
      "Unit": "Percent"
    },
    ...
  ],
  "Label": "CPUUtilization"
}
```

Aggregate statistics across resources

You can aggregate the metrics for AWS resources across multiple resources. Metrics are completely separate between Regions, but you can use metric math to aggregate similar metrics across Regions. For more information, see [Use metric math](#).

For example, you can aggregate statistics for your EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. Therefore, you must enable detailed monitoring (at an additional charge), which provides data in 1-minute periods. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide*.

This example shows you how to get the average CPU usage for your EC2 instances. Because no dimension is specified, CloudWatch returns statistics for all dimensions in the AWS/EC2 namespace. To get statistics for other metrics, see [AWS services that publish CloudWatch metrics](#).

⚠ Important

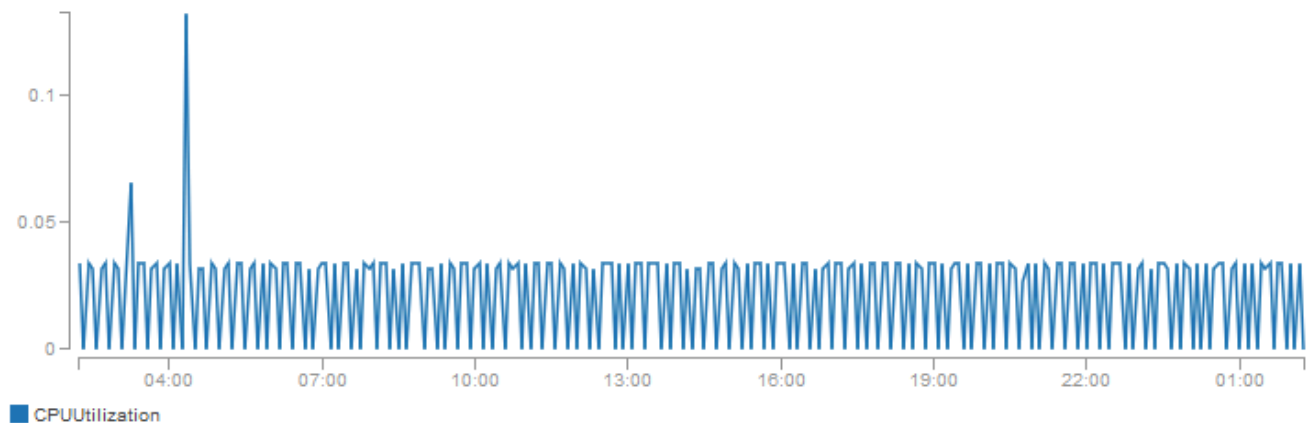
This technique for retrieving all dimensions across an AWS namespace doesn't work for custom namespaces that you publish to CloudWatch. With custom namespaces, you must specify the complete set of dimensions that are associated with any given data point to retrieve statistics that include the data point.

To display average CPU utilization for your EC2 instances

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and choose **Across All Instances**.
4. Select the row that contains CPUUtilization, which displays a graph for the metric for all your EC2 instances. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.

Untitled graph 1h 3h 12h **1d** 3d 1w custom ▾

Actions ▾



All metrics

Graphed metrics (1)

Graph options

All > EC2 > Across All Instances

| <input type="checkbox"/> | Metric Name (7) ▲ |
|-------------------------------------|--|
| <input checked="" type="checkbox"/> | CPUUtilization |
| <input type="checkbox"/> | DiskReadBytes |
| <input type="checkbox"/> | DiskReadOps |

5. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
6. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get average CPU utilization across your EC2 instances using the AWS CLI

Use the [get-metric-statistics](#) command as follows:

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization
--statistics "Average" "SampleCount" \
--start-time 2016-10-11T23:18:00 --end-time 2016-10-12T23:18:00 --period 3600
```

The following is example output:

```
{
  "Datapoints": [
    {
      "SampleCount": 238.0,
      "Timestamp": "2016-10-12T07:18:00Z",
      "Average": 0.038235294117647062,
      "Unit": "Percent"
    },
    {
      "SampleCount": 240.0,
      "Timestamp": "2016-10-12T09:18:00Z",
      "Average": 0.16670833333333332,
      "Unit": "Percent"
    },
    {
      "SampleCount": 238.0,
      "Timestamp": "2016-10-11T23:18:00Z",
      "Average": 0.041596638655462197,
      "Unit": "Percent"
    },
    ...
  ],
  "Label": "CPUUtilization"
}
```

Aggregate statistics by Auto Scaling group

You can aggregate statistics for the EC2 instances in an Auto Scaling group. Metrics are completely separate between Regions, but you can use CloudWatch metric math to aggregate and transform metrics from multiple Regions. You can also use the cross-account dashboard to perform metric math on metrics from different accounts.

This example shows you how to get the total bytes written to disk for one Auto Scaling group. The total is computed for 1-minute periods for a 24-hour interval across all EC2 instances in the specified Auto Scaling group.

To display DiskWriteBytes for the instances in an Auto Scaling group using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and then choose **By Auto Scaling Group**.

- Select the row for the **DiskWriteBytes** metric and the specific Auto Scaling group, which displays a graph for the metric for the instances in the Auto Scaling group. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



...

All metrics Graphed metrics (1) Graph options

All > EC2 > By Auto Scaling Group

| <input type="checkbox"/> | AutoScalingGroupName (28) | Metric Name |
|-------------------------------------|---------------------------|----------------|
| <input type="checkbox"/> | my-asg | DiskReadBytes |
| <input type="checkbox"/> | my-asg | DiskReadOps |
| <input checked="" type="checkbox"/> | my-asg | DiskWriteBytes |
| <input type="checkbox"/> | my-asg | DiskWriteOps |

- To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
- To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get DiskWriteBytes for the instances in an Auto Scaling group using the AWS CLI

Use the [get-metric-statistics](#) command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name DiskWriteBytes
--dimensions Name=AutoScalingGroupName,Value=my-asg --statistics "Sum" "SampleCount" \
--start-time 2016-10-16T23:18:00 --end-time 2016-10-18T23:18:00 --period 360
```

The following is example output.

```
{
  "Datapoints": [
    {
      "SampleCount": 18.0,
      "Timestamp": "2016-10-19T21:36:00Z",
      "Sum": 0.0,
      "Unit": "Bytes"
    },
    {
      "SampleCount": 5.0,
      "Timestamp": "2016-10-19T21:42:00Z",
      "Sum": 0.0,
      "Unit": "Bytes"
    }
  ],
  "Label": "DiskWriteBytes"
}
```

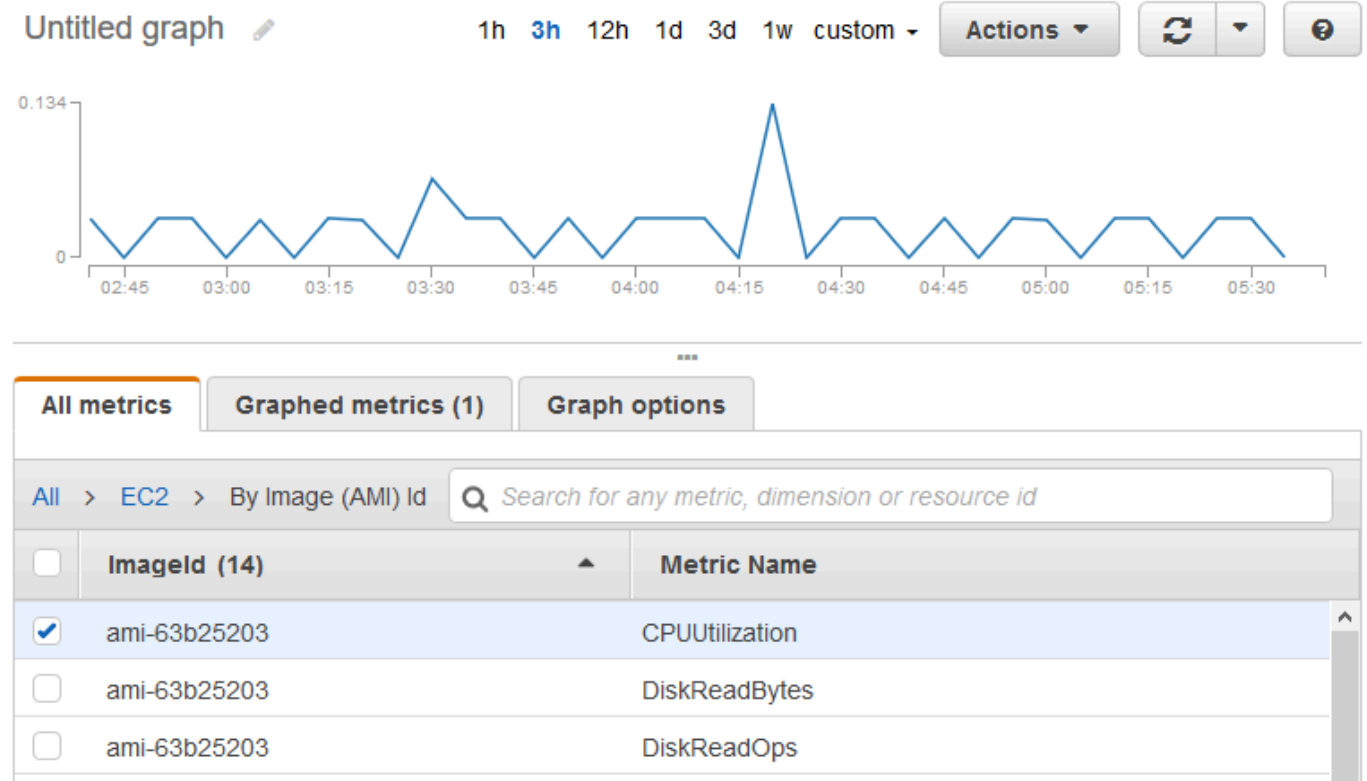
Aggregate statistics by Amazon Machine Image (AMI)

You can aggregate statistics for the EC2 instances that have detailed monitoring enabled. Instances that use basic monitoring aren't included. For more information, see [Enable or Disable Detailed Monitoring for Your Instances](#) in the *Amazon EC2 User Guide*.

This example shows you how to determine average CPU utilization for all instances that use the specified AMI. The average is over 60-second time intervals for a one-day period.

To display the average CPU utilization by AMI using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the **EC2** namespace and then choose **By Image (AMI) Id**.
4. Select the row for the `CPUUtilization` metric and the specific AMI, which displays a graph for the metric for the specified AMI. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**.



- To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
- To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.

To get the average CPU utilization by AMI using the AWS CLI

Use the [get-metric-statistics](#) command as follows.

```
aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization \
--dimensions Name=ImageId,Value=ami-3c47a355 --statistics Average \
--start-time 2016-10-10T00:00:00 --end-time 2016-10-11T00:00:00 --period 3600
```

The operation returns statistics that are one-hour values for the one-day interval. Each value represents an average CPU utilization percentage for EC2 instances running the specified AMI. The following is example output.

```
{
  "Datapoints": [
```

```
{
  "Timestamp": "2016-10-10T07:00:00Z",
  "Average": 0.041000000000000009,
  "Unit": "Percent"
},
{
  "Timestamp": "2016-10-10T14:00:00Z",
  "Average": 0.079579831932773085,
  "Unit": "Percent"
},
{
  "Timestamp": "2016-10-10T06:00:00Z",
  "Average": 0.0360000000000000011,
  "Unit": "Percent"
},
...
],
"Label": "CPUUtilization"
}
```

Publish custom metrics

You can publish your own metrics to CloudWatch using the AWS CLI or an API. You can view statistical graphs of your published metrics with the AWS Management Console.

CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points called a *statistic set*.

Topics

- [High-resolution metrics](#)
- [Use dimensions](#)
- [Publish single data points](#)
- [Publish statistic sets](#)
- [Publish the value zero](#)
- [Stop publishing metrics](#)

High-resolution metrics

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Use dimensions

In custom metrics, the `--dimensions` parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 30 dimensions assigned to one metric, and each dimension is defined by a name and value pair.

How you specify a dimension is different when you use different commands. With [put-metric-data](#), you specify each dimension as `MyName=MyVaLue`, and with [get-metric-statistics](#) or [put-metric-alarm](#) you use the format `Name=MyName, Value=MyVaLue`. For example, the following command publishes a `Buffers` metric with two dimensions named `InstanceId` and `InstanceType`.

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

This command retrieves statistics for that same metric. Separate the `Name` and `Value` parts of a single dimension with commas, but if you have multiple dimensions, use a space between one dimension and the next.

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-time 2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period 60
```

If a single metric includes multiple dimensions, you must specify a value for every defined dimension when you use [get-metric-statistics](#). For example, the Amazon S3 metric `BucketSizeBytes` includes the dimensions `BucketName` and `StorageType`, so you must specify both dimensions with [get-metric-statistics](#).

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time
2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace
AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=MyBucketName
Name=StorageType,Value=StandardStorage --output table
```

To see what dimensions are defined for a metric, use the [list-metrics](#) command.

Publish single data points

To publish a single data point for a new or existing metric, use the [put-metric-data](#) command with one value and time stamp. For example, the following actions each publish one data point.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 5 --timestamp 2016-10-20T12:00:02.000Z
```

If you call this command with a new metric name, CloudWatch creates a metric for you. Otherwise, CloudWatch associates your data with the existing metric that you specified.

Note

When you create a metric, it can take up to 2 minutes before you can retrieve statistics for the new metric using the [get-metric-statistics](#) command. However, it can take up to 15 minutes before the new metric appears in the list of metrics retrieved using the [list-metrics](#) command.

Although you can publish data points with time stamps as granular as one-thousandth of a second, CloudWatch aggregates the data to a minimum granularity of 1 second. CloudWatch records the average (sum of all items divided by number of items) of the values received for each period, as well as the number of samples, maximum value, and minimum value for the same time period. For

example, the `PageViewCount` metric from the previous examples contains three data points with time stamps just seconds apart. If you have your period set to 1 minute, CloudWatch aggregates the three data points because they all have time stamps within a 1-minute period.

You can use the `get-metric-statistics` command to retrieve statistics based on the data points that you published.

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

The following is example output.

```
{
  "Datapoints": [
    {
      "SampleCount": 3.0,
      "Timestamp": "2016-10-20T12:00:00Z",
      "Average": 3.6666666666666665,
      "Maximum": 5.0,
      "Minimum": 2.0,
      "Sum": 11.0,
      "Unit": "None"
    }
  ],
  "Label": "PageViewCount"
}
```

Publish statistic sets

You can aggregate your data before you publish to CloudWatch. When you have multiple data points per minute, aggregating data minimizes the number of calls to `put-metric-data`. For example, instead of calling `put-metric-data` multiple times for three data points that are within 3 seconds of each other, you can aggregate the data into a statistic set that you publish with one call, using the `--statistic-values` parameter.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService
--statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can't retrieve percentile statistics for this data unless one of the following conditions is true:

- The `SampleCount` of the statistic set is 1
- The `Minimum` and the `Maximum` of the statistic set are equal

Publish the value zero

When your data is more sporadic and you have periods that have no associated data, you can choose to publish the value zero (0) for that period or no value at all. If you use periodic calls to `PutMetricData` to monitor the health of your application, you might want to publish zero instead of no value. For example, you can set a CloudWatch alarm to notify you if your application fails to publish metrics every five minutes. You want such an application to publish zeros for periods with no associated data.

You might also publish zeros if you want to track the total number of data points or if you want statistics such as minimum and average to include data points with the value 0.

Stop publishing metrics

To stop publishing custom metrics to CloudWatch, change your application's or service's code to stop using `PutMetricData`. CloudWatch doesn't pull metrics from applications, it only receives what is pushed to it, so to stop publishing your metrics you must stop them at the source.

Using Amazon CloudWatch alarms

You can create *metric* and *composite* alarms in Amazon CloudWatch.

- A *metric alarm* watches a single CloudWatch metric or the result of a math expression based on CloudWatch metrics. The alarm performs one or more actions based on the value of the metric or expression relative to a threshold over a number of time periods. The action can be sending a notification to an Amazon SNS topic, performing an Amazon EC2 action or an Amazon EC2 Auto Scaling action, or creating an OpsItem or incident in Systems Manager.
- A *composite alarm* includes a rule expression that takes into account the alarm states of other alarms that you have created. The composite alarm goes into ALARM state only if all conditions of the rule are met. The alarms specified in a composite alarm's rule expression can include metric alarms and other composite alarms.

Using composite alarms can reduce alarm noise. You can create multiple metric alarms, and also create a composite alarm and set up alerts only for the composite alarm. For example, a composite might go into ALARM state only when all of the underlying metric alarms are in ALARM state.

Composite alarms can send Amazon SNS notifications when they change state, and can create Systems Manager OpsItems or incidents when they go into ALARM state, but can't perform EC2 actions or Auto Scaling actions.

Note

You can create as many alarms as you want in your AWS account.

You can add alarms to dashboards, so you can monitor and receive alerts about your AWS resources and applications across multiple regions. After you add an alarm to a dashboard, the alarm turns gray when it's in the `INSUFFICIENT_DATA` state and red when it's in the `ALARM` state. The alarm is shown with no color when it's in the `OK` state.

You also can favorite recently visited alarms from the *Favorites and recents* option in the CloudWatch console navigation pane. The *Favorites and recents* option has columns for your favorited alarms and recently visited alarms.

An alarm invokes actions only when the alarm changes state. The exception is for alarms with Auto Scaling actions. For Auto Scaling actions, the alarm continues to invoke the action once per minute that the alarm remains in the new state.

An alarm can watch a metric in the same account. If you have enabled cross-account functionality in your CloudWatch console, you can also create alarms that watch metrics in other AWS accounts. Creating cross-account composite alarms is not supported. Creating cross-account alarms that use math expressions is supported, except that the `ANOMALY_DETECTION_BAND`, `INSIGHT_RULE`, and `SERVICE_QUOTA` functions are not supported for cross-account alarms.

Note

CloudWatch doesn't test or validate the actions that you specify, nor does it detect any Amazon EC2 Auto Scaling or Amazon SNS errors resulting from an attempt to invoke nonexistent actions. Make sure that your alarm actions exist.

Metric alarm states

A metric alarm has the following possible states:

- **OK** – The metric or expression is within the defined threshold.
- **ALARM** – The metric or expression is outside of the defined threshold.
- **INSUFFICIENT_DATA** – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

Evaluating an alarm

When you create an alarm, you specify three settings to enable CloudWatch to evaluate when to change the alarm state:

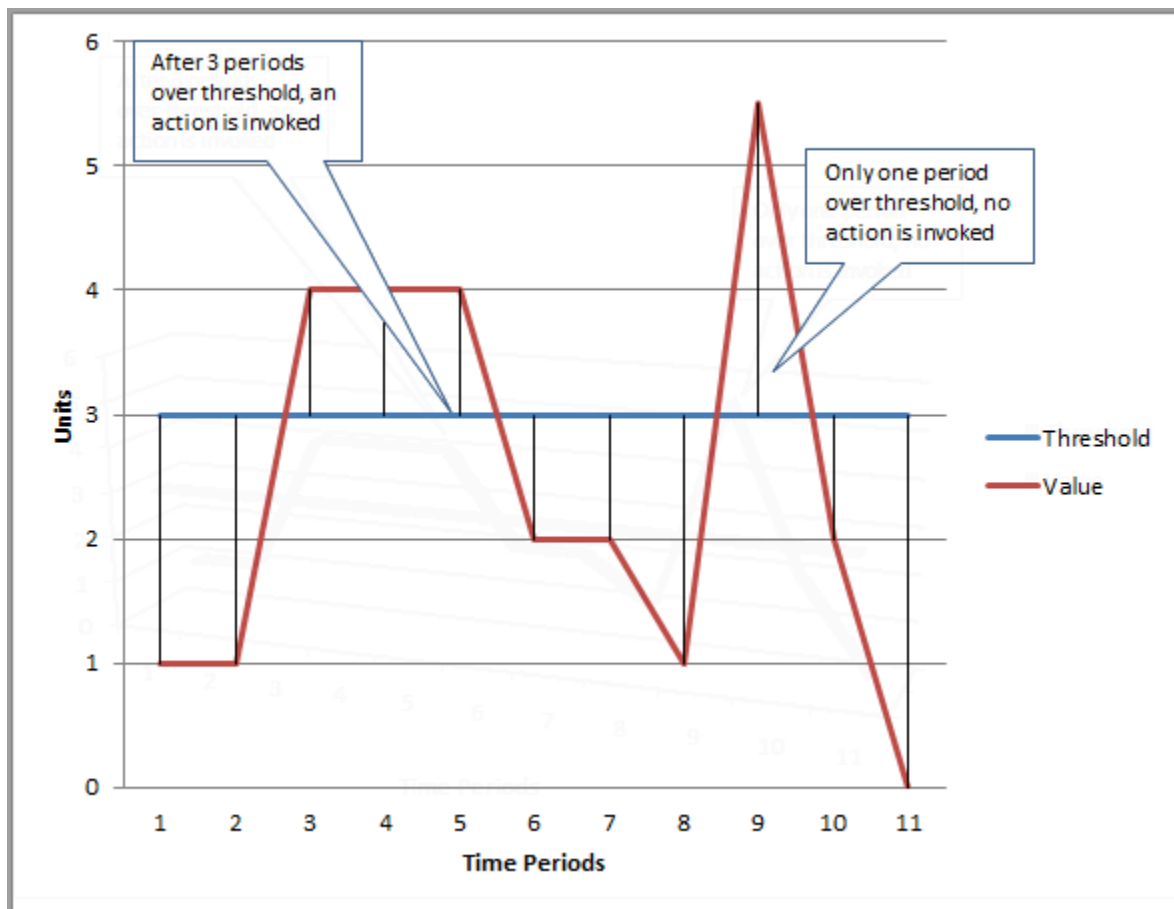
- **Period** is the length of time to use to evaluate the metric or expression to create each individual data point for an alarm. It is expressed in seconds.
- **Evaluation Periods** is the number of the most recent periods, or data points, to evaluate when determining alarm state.
- **Datapoints to Alarm** is the number of data points within the Evaluation Periods that must be breaching to cause the alarm to go to the ALARM state. The breaching data points don't have to

be consecutive, but they must all be within the last number of data points equal to **Evaluation Period**.

For any period of one minute or longer, an alarm is evaluated every minute and the evaluation is based on the window of time defined by the **Period** and **Evaluation Periods**. For example, if the **Period** is 5 minutes (300 seconds) and **Evaluation Periods** is 1, then at the end of minute 5 the alarm evaluates based on data from minutes 1 to 5. Then at the end of minute 6, the alarm is evaluated based on the data from minutes 2 to 6.

If the alarm period is 10 seconds or 30 seconds, the alarm is evaluated every 10 seconds.

In the following figure, the alarm threshold for a metric alarm is set to three units. Both **Evaluation Period** and **Datapoints to Alarm** are 3. That is, when all existing data points in the most recent three consecutive periods are above the threshold, the alarm goes to ALARM state. In the figure, this happens in the third through fifth time periods. At period six, the value dips below the threshold, so one of the periods being evaluated is not breaching, and the alarm state changes back to OK. During the ninth time period, the threshold is breached again, but for only one period. Consequently, the alarm state remains OK.



When you configure **Evaluation Periods** and **Datapoints to Alarm** as different values, you're setting an "M out of N" alarm. **Datapoints to Alarm** is ("M") and **Evaluation Periods** is ("N"). The evaluation interval is the number of evaluation periods multiplied by the period length. For example, if you configure 4 out of 5 data points with a period of 1 minute, the evaluation interval is 5 minutes. If you configure 3 out of 3 data points with a period of 10 minutes, the evaluation interval is 30 minutes.

Note

If data points are missing soon after you create an alarm, and the metric was being reported to CloudWatch before you created the alarm, CloudWatch retrieves the most recent data points from before the alarm was created when evaluating the alarm.

Alarm actions

You can specify what actions an alarm takes when it changes state between the OK, ALARM, and INSUFFICIENT_DATA states.

Most actions can be set for the transition into each of the three states. Except for Auto Scaling actions, the actions happen only on state transitions, and are not performed again if the condition persists for hours or days. You can use the fact that multiple actions are allowed for an alarm to send an email when a threshold is breached, and then another when the breaching condition ends. This helps you verify that your scaling or recovery actions are triggered when expected and are working as desired.

The following are supported as alarm actions.

- Notify one or more subscribers by **using an Amazon Simple Notification Service topic**. Subscribers can be applications as well as persons. For more information about Amazon SNS, see [What is Amazon SNS?](#)
- **Invoke a Lambda function**. This is the easiest way for you to automate custom actions on alarm state changes.
- Alarms based on EC2 metrics can also **perform EC2 actions**, such as stopping, terminating, rebooting, or recovering an EC2 instance. For more information, see [Create alarms to stop, terminate, reboot, or recover an EC2 instance](#).
- Alarms can perform actions to **scale an Auto Scaling group**. For more information, see [Step and simple scaling policies for Amazon EC2 Auto Scaling](#).

- Alarms can **create OpsItems in Systems Manager Ops Center or create incidents in AWS Systems Manager Incident Manager**. These actions are performed only when the alarm goes into ALARM state. For more information, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Lambda alarm actions

CloudWatch alarms guarantees an asynchronous invocation of the Lambda function for a given state change, except in the following cases:

- When the function doesn't exist.
- When CloudWatch is not authorized to invoke the Lambda function.

If CloudWatch can't reach the Lambda service or the message is rejected for another reason, CloudWatch retries until the invocation is successful. Lambda queues the message and handles execution retries. For more information about this execution model, including information about how Lambda handles errors, see [Asynchronous invocation](#) in the AWS Lambda Developer Guide.

You can invoke a Lambda function in the same account, or in other AWS accounts.

When you specify an alarm to invoke a Lambda function as an alarm action, you can choose to specify the function name, function alias, or a specific version of a function.

When you specify a Lambda function as an alarm action, you must create a resource policy for the function to allow the CloudWatch service principal to invoke the function.

One way to do this is by using the AWS CLI, as in the following example:

```
aws lambda add-permission \  
--function-name my-function-name \  
--statement-id AlarmAction \  
--action 'lambda:InvokeFunction' \  
--principal lambda.alarms.cloudwatch.amazonaws.com \  
--source-account 111122223333 \  
--source-arn arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name
```

Alternatively, you can create a policy similar to one of the following examples and then assign it to the function.

The following example specifies the account where the alarm is located, so that only alarms in that account (111122223333) can invoke the function.

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [{
    "Sid": "AlarmAction",
    "Effect": "Allow",
    "Principal": {
      "Service": "lambda.alarms.cloudwatch.amazonaws.com"
    },
    "Action": "lambda:InvokeFunction",
    "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
    "Condition": {
      "StringEquals": {
        "AWS:SourceAccount": "111122223333"
      }
    }
  }]
}
```

The following example has a narrower scope, allowing only the specified alarm in the specified account to invoke the function.

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "AlarmAction",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.alarms.cloudwatch.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name"
        }
      }
    }
  ]
}
```

```
    ]]  
  }  
}
```

We don't recommend creating a policy that doesn't specify a source account, because such policies are vulnerable to confused deputy issues.

Event object sent from CloudWatch to Lambda

When you configure a Lambda function as an alarm action, CloudWatch delivers a JSON payload to the Lambda function when it invokes the function. This JSON payload serves as the event object for the function. You can extract data from this JSON object and use it in your function. The following is an example of an event object from a metric alarm.

```
{  
  'source': 'aws.cloudwatch',  
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:444455556666:alarm:lambda-demo-metric-  
alarm',  
  'accountId': '444455556666',  
  'time': '2023-08-04T12:36:15.490+0000',  
  'region': 'us-east-1',  
  'alarmData': {  
    'alarmName': 'lambda-demo-metric-alarm',  
    'state': {  
      'value': 'ALARM',  
      'reason': 'test',  
      'timestamp': '2023-08-04T12:36:15.490+0000'  
    },  
    'previousState': {  
      'value': 'INSUFFICIENT_DATA',  
      'reason': 'Insufficient Data: 5 datapoints were unknown.',  
      'reasonData':  
        '{"version":"1.0","queryDate":"2023-08-04T12:31:29.591+0000","statistic":"Average","period":60  
[],"threshold":5.0,"evaluatedDatapoints":[{"timestamp":"2023-08-04T12:30:00.000+0000"},  
{"timestamp":"2023-08-04T12:29:00.000+0000"},  
{"timestamp":"2023-08-04T12:28:00.000+0000"},  
{"timestamp":"2023-08-04T12:27:00.000+0000"},  
{"timestamp":"2023-08-04T12:26:00.000+0000"}]}'  
      'timestamp': '2023-08-04T12:31:29.595+0000'  
    },  
    'configuration': {  
      'description': 'Metric Alarm to test Lambda actions',  
      'metrics': [  

```

```

    {
      'id': '1234e046-06f0-a3da-9534-EXAMPLEe4c',
      'metricStat': {
        'metric': {
          'namespace': 'AWS/Logs',
          'name': 'CallCount',
          'dimensions': {
            'InstanceId': 'i-12345678'
          }
        },
        'period': 60,
        'stat': 'Average',
        'unit': 'Percent'
      },
      'returnData': True
    }
  ]
}
}
}

```

The following is an example of an event object from a composite alarm.

```

{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:SuppressionDemo.Main',
  'accountId': '111122223333',
  'time': '2023-08-04T12:56:46.138+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'CompositeDemo.Main',
    'state': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-east-1:111122223333:alarm:CompositeDemo.FirstChild","state":{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}'
    },
    'timestamp': '2023-08-04T12:56:46.138+0000'
  },
  'previousState': {
    'value': 'ALARM',
  }
}

```

```
    'reason': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04 August, 2023 12:54:46 UTC',
    'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-east-1:111122223333:alarm:CompositeDemo.FirstChild","state":{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
    'timestamp': '2023-08-04T12:54:46.138+0000',
    'actionsSuppressedBy': 'WaitPeriod',
    'actionsSuppressedReason': 'Actions suppressed by WaitPeriod'
  },
  'configuration': {
    'alarmRule': 'ALARM(CompositeDemo.FirstChild) OR ALARM(CompositeDemo.SecondChild)',
    'actionsSuppressor': 'CompositeDemo.ActionsSuppressor',
    'actionsSuppressorWaitPeriod': 120,
    'actionsSuppressorExtensionPeriod': 180
  }
}
```

Configuring how CloudWatch alarms treat missing data

Sometimes, not every expected data point for a metric gets reported to CloudWatch. For example, this can happen when a connection is lost, a server goes down, or when a metric reports data only intermittently by design.

CloudWatch enables you to specify how to treat missing data points when evaluating an alarm. This helps you to configure your alarm so that it goes to ALARM state only when appropriate for the type of data being monitored. You can avoid false positives when missing data doesn't indicate a problem.

Similar to how each alarm is always in one of three states, each specific data point reported to CloudWatch falls under one of three categories:

- Not breaching (within the threshold)
- Breaching (violating the threshold)
- Missing

For each alarm, you can specify CloudWatch to treat missing data points as any of the following:

- `notBreaching` – Missing data points are treated as "good" and within the threshold
- `breaching` – Missing data points are treated as "bad" and breaching the threshold
- `ignore` – The current alarm state is maintained
- `missing` – If all data points in the alarm evaluation range are missing, the alarm transitions to `INSUFFICIENT_DATA`.

The best choice depends on the type of metric and the purpose of the alarm. For example, if you are creating an application rollback alarm using a metric that continually reports data, you might want to treat missing data points as breaching, because it might indicate that something is wrong. But for a metric that generates data points only when an error occurs, such as `ThrottledRequests` in Amazon DynamoDB, you would want to treat missing data as `notBreaching`. The default behavior is `missing`.

Important

Alarms configured on Amazon EC2 metrics can temporarily enter the `INSUFFICIENT_DATA` state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions, we recommend that you configure those alarms to treat missing data as `missing`, and to have these alarms trigger only when in the `ALARM` state.

Choosing the best option for your alarm prevents unnecessary and misleading alarm condition changes, and also more accurately indicates the health of your system.

Important

Alarms that evaluate metrics in the `AWS/DynamoDB` namespace always ignore missing data even if you choose a different option for how the alarm should treat missing data. When an `AWS/DynamoDB` metric has missing data, alarms that evaluate that metric remain in their current state.

How alarm state is evaluated when data is missing

Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as **Evaluation Periods**. The exact number of data points it attempts to retrieve depends on the length of the alarm period and whether it is based on a metric with standard resolution or high resolution. The time frame of the data points that it attempts to retrieve is the *evaluation range*.

Once CloudWatch retrieves these data points, the following happens:

- If no data points in the evaluation range are missing, CloudWatch evaluates the alarm based on the most recent data points collected. The number of data points evaluated is equal to the **Evaluation Periods** for the alarm. The extra data points from farther back in the evaluation range are not needed and are ignored.
- If some data points in the evaluation range are missing, but the total number of existing data points that were successfully retrieved from the evaluation range is equal to or more than the alarm's **Evaluation Periods**, CloudWatch evaluates the alarm state based on the most recent real data points that were successfully retrieved, including the necessary extra data points from farther back in the evaluation range. In this case, the value you set for how to treat missing data is not needed and is ignored.
- If some data points in the evaluation range are missing, and the number of actual data points that were retrieved is lower than the alarm's number of **Evaluation Periods**, CloudWatch fills in the missing data points with the result you specified for how to treat missing data, and then evaluates the alarm. However, all real data points in the evaluation range are included in the evaluation. CloudWatch uses missing data points only as few times as possible.

Note

A particular case of this behavior is that CloudWatch alarms might repeatedly re-evaluate the last set of data points for a period of time after the metric has stopped flowing. This re-evaluation might cause the alarm to change state and re-execute actions, if it had changed state immediately prior to the metric stream stopping. To mitigate this behavior, use shorter periods.

The following tables illustrate examples of the alarm evaluation behavior. In the first table, **Datapoints to Alarm** and **Evaluation Periods** are both 3. CloudWatch retrieves the 5 most recent

data points when evaluating the alarm, in case some of the most recent 3 data points are missing. 5 is the evaluation range for the alarm.

Column 1 shows the 5 most recent data points, because the evaluation range is 5. These data points are shown with the most recent data point on the right. 0 is a non-breaching data point, X is a breaching data point, and - is a missing data point.

Column 2 shows how many of the 3 necessary data points are missing. Even though the most recent 5 data points are evaluated, only 3 (the setting for **Evaluation Periods**) are necessary to evaluate the alarm state. The number of data points in Column 2 is the number of data points that must be "filled in", using the setting for how missing data is being treated.

In columns 3-6, the column headers are the possible values for how to treat missing data. The rows in these columns show the alarm state that is set for each of these possible ways to treat missing data.

| Data points | # of data points that must be filled | MISSING | IGNORE | BREACHING | NOT BREACHING |
|-------------|--------------------------------------|-------------------|----------------------|-----------|---------------|
| 0 - X - X | 0 | OK | OK | OK | OK |
| 0 - - - - | 2 | OK | OK | OK | OK |
| - - - - - | 3 | INSUFFICIENT_DATA | Retain current state | ALARM | OK |
| 0 X X - X | 0 | ALARM | ALARM | ALARM | ALARM |
| - - X - - | 2 | ALARM | Retain current state | ALARM | OK |

In the second row of the preceding table, the alarm stays OK even if missing data is treated as breaching, because the one existing data point is not breaching, and this is evaluated along with two missing data points which are treated as breaching. The next time this alarm is evaluated, if the data is still missing it will go to ALARM, as that non-breaching data point will no longer be in the evaluation range.

The third row, where all five of the most recent data points are missing, illustrates how the various settings for how to treat missing data affect the alarm state. If missing data points are considered breaching, the alarm goes into ALARM state, while if they are considered not breaching, then the alarm goes into OK state. If missing data points are ignored, the alarm retains the current state it had before the missing data points. And if missing data points are just considered as missing, then the alarm does not have enough recent real data to make an evaluation, and goes into INSUFFICIENT_DATA.

In the fourth row, the alarm goes to ALARM state in all cases because the three most recent data points are breaching, and the alarm's **Evaluation Periods** and **Datapoints to Alarm** are both set to 3. In this case, the missing data point is ignored and the setting for how to evaluate missing data is not needed, because there are 3 real data points to evaluate.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see [Avoiding premature transitions to alarm state](#).

In the next table, the **Period** is again set to 5 minutes, and **Datapoints to Alarm** is only 2 while **Evaluation Periods** is 3. This is a 2 out of 3, M out of N alarm.

The evaluation range is 5. This is the maximum number of recent data points that are retrieved and can be used in case some data points are missing.

| Data points | # of missing data points | MISSING | IGNORE | BREACHING | NOT BREACHING |
|-------------|--------------------------|---------|----------------------|-----------|---------------|
| 0 - X - X | 0 | ALARM | ALARM | ALARM | ALARM |
| 0 0 X 0 X | 0 | ALARM | ALARM | ALARM | ALARM |
| 0 - X - - | 1 | OK | OK | ALARM | OK |
| - - - - 0 | 2 | OK | OK | ALARM | OK |
| - - - X - | 2 | ALARM | Retain current state | ALARM | OK |

In rows 1 and 2, the alarm always goes to ALARM state because 2 of the 3 most recent data points are breaching. In row 2, the two oldest data points in the evaluation range are not needed because none of the 3 most recent data points are missing, so these two older data points are ignored.

In rows 3 and 4, the alarm goes to ALARM state only if missing data is treated as breaching, in which case the two most recent missing data points are both treated as breaching. In row 4, these two missing data points that are treated as breaching provide the two necessary breaching data points to trigger the ALARM state.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see the following section.

Avoiding premature transitions to alarm state

CloudWatch alarm evaluation includes logic to try to avoid false alarms, where the alarm goes into ALARM state prematurely when data is intermittent. The example shown in row 5 in the tables in the previous section illustrate this logic. In those rows, and in the following examples, the **Evaluation Periods** is 3 and the evaluation range is 5 data points. **Datapoints to Alarm** is 3, except for the M out of N example, where **Datapoints to Alarm** is 2.

Suppose an alarm's most recent data is - - - - X, with four missing data points and then a breaching data point as the most recent data point. Because the next data point may be non-breaching, the alarm does not go immediately into ALARM state when the data is either - - - - X or - - - X - and **Datapoints to Alarm** is 3. This way, false positives are avoided when the next data point is non-breaching and causes the data to be - - - X 0 or - - X - 0.

However, if the last few data points are - - X - -, the alarm goes into ALARM state even if missing data points are treated as missing. This is because alarms are designed to always go into ALARM state when the oldest available breaching datapoint during the **Evaluation Periods** number of data points is at least as old as the value of **Datapoints to Alarm**, and all other more recent data points are breaching or missing. In this case, the alarm goes into ALARM state even if the total number of datapoints available is lower than M (**Datapoints to Alarm**).

This alarm logic applies to M out of N alarms as well. If the oldest breaching data point during the evaluation range is at least as old as the value of **Datapoints to Alarm**, and all of the more recent data points are either breaching or missing, the alarm goes into ALARM state no matter the value of M (**Datapoints to Alarm**).

High-resolution alarms

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of

60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see [Publish custom metrics](#).

Alarms on math expressions

You can set an alarm on the result of a math expression that is based on one or more CloudWatch metrics. A math expression used for an alarm can include as many as 10 metrics. Each metric must be using the same period.

For an alarm based on a math expression, you can specify how you want CloudWatch to treat missing data points. In this case, the data point is considered missing if the math expression doesn't return a value for that data point.

Alarms based on math expressions can't perform Amazon EC2 actions.

For more information about metric math expressions and syntax, see [Use metric math](#).

Percentile-based CloudWatch alarms and low data samples

When you set a percentile as the statistic for an alarm, you can specify what to do when there is not enough data for a good statistical assessment. You can choose to have the alarm evaluate the statistic anyway and possibly change the alarm state. Or, you can have the alarm ignore the metric while the sample size is low, and wait to evaluate it until there is enough data to be statistically significant.

For percentiles between 0.5 (inclusive) and 1.00 (exclusive), this setting is used when there are fewer than $10/(1-\text{percentile})$ data points during the evaluation period. For example, this setting would be used if there were fewer than 1000 samples for an alarm on a p99 percentile. For percentiles between 0 and 0.5 (exclusive), the setting is used when there are fewer than $10/\text{percentile}$ data points.

Common features of CloudWatch alarms

The following features apply to all CloudWatch alarms:

- There is no limit to the number of alarms that you can create. To create or update an alarm, you use the CloudWatch console, the [PutMetricAlarm](#) API action, or the [put-metric-alarm](#) command in the AWS CLI.

- Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters
- You can list any or all of the currently configured alarms, and list any alarms in a particular state by using the CloudWatch console, the [DescribeAlarms](#) API action, or the [describe-alarms](#) command in the AWS CLI.
- You can disable and enable alarms by using the [DisableAlarmActions](#) and [EnableAlarmActions](#) API actions, or the [disable-alarm-actions](#) and [enable-alarm-actions](#) commands in the AWS CLI.
- You can test an alarm by setting it to any state using the [SetAlarmState](#) API action or the [set-alarm-state](#) command in the AWS CLI. This temporary state change lasts only until the next alarm comparison occurs.
- You can create an alarm for a custom metric before you've created that custom metric. For the alarm to be valid, you must include all of the dimensions for the custom metric in addition to the metric namespace and metric name in the alarm definition. To do this, you can use the [PutMetricAlarm](#) API action, or the [put-metric-alarm](#) command in the AWS CLI.
- You can view an alarm's history using the CloudWatch console, the [DescribeAlarmHistory](#) API action, or the [describe-alarm-history](#) command in the AWS CLI. CloudWatch preserves alarm history for 30 days. Each state transition is marked with a unique timestamp. In rare cases, your history might show more than one notification for a state change. The timestamp enables you to confirm unique state changes.
- You can favorite alarms from the *Favorites and recents* option in the CloudWatch console navigation pane by hovering over the alarm that you want to favorite and choosing the star symbol next to it.
- The number of evaluation periods for an alarm multiplied by the length of each evaluation period can't exceed one day.

Note

Some AWS resources don't send metric data to CloudWatch under certain conditions. For example, Amazon EBS might not send metric data for an available volume that is not attached to an Amazon EC2 instance, because there is no metric activity to be monitored for that volume. If you have an alarm set for such a metric, you might notice its state change to `INSUFFICIENT_DATA`. This might indicate that your resource is inactive, and might not necessarily mean that there is a problem. You can specify how each alarm treats missing data. For more information, see [Configuring how CloudWatch alarms treat missing data](#).

Best practice alarm recommendations for AWS services

CloudWatch provides out-of-the box alarm recommendations. These are CloudWatch alarms that we recommend that you create for metrics that are published by other AWS services. These recommendations can help you identify the metrics that you should set alarms for to follow best practices for monitoring. The recommendations also suggest the alarm thresholds to set. Following these recommendations can help you not miss important monitoring of your AWS infrastructure.

To find the alarm recommendations, you use the metrics section of the CloudWatch console, and select the alarm recommendations filter toggle. If you navigate to the recommended alarms in the console and then create a recommended alarm, CloudWatch can pre-fill some of the alarm settings. For some recommended alarms, the alarm threshold value is also pre-filled. You can also use the console to download infrastructure-as-code alarm definitions for recommended alarms, and then use this code to create the alarm in AWS CloudFormation, the AWS CLI, or Terraform.

You can also see the list of recommended alarms in [Recommended alarms](#).

You are charged for the alarms that you create, at the same rate as any other alarms that you create in CloudWatch. Using the recommendations incurs no extra charges. For more information, see [Amazon CloudWatch Pricing](#).

Find and create recommended alarms

Follow these steps to find the metrics that CloudWatch recommends that you set alarms for, and optionally to create one of these alarms. The first procedure explains how to find the metrics that have recommended alarms, and how to create one of these alarms.

You can also get a bulk download of infrastructure-as-code alarm definitions for all recommended alarms in an AWS namespace, such as AWS/Lambda or AWS/S3. Those instructions are later in this topic.

To find the metrics with recommended alarms, and create a single recommended alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Above the **Metrics** table, Choose **Alarm recommendations**.

The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.

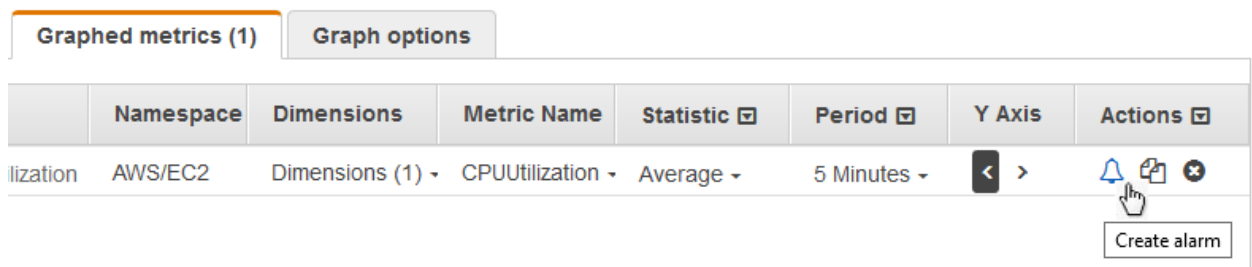
4. Choose the namespace for a service.

The list of metrics under this namespace is filtered to include only those that have alarm recommendations.

5. To see the alarm intent and recommended threshold for a metric, choose **View details**.

6. To create an alarm for one of the metrics, do one of the following:

- To use the console to create the alarm, do the following:
 - a. Select the checkbox for the metric and choose the **Graphed metrics** tab.
 - b. Choose the alarm icon.



The alarm creation wizard appears, with the metric name, statistic, and period filled in based on the alarm recommendation. If the recommendation includes a specific threshold value, that value is also pre-filled.

- c. Choose **Next**.
- d. Under **Notification**, select an SNS topic to notify when the alarm transitions to ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

- e. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
- f. When finished, choose **Next**.
- g. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
- h. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

- To download an infrastructure-as-code alarm definition to use in either AWS CloudFormation, AWS CLI, or Terraform, choose **Download alarm code** and select the format that you want. The downloaded code will have the recommended settings for the metric name, statistic, and threshold.

To download infrastructure-as-code alarm definitions for all recommended alarms for an AWS service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Above the **Metrics** table, Choose **Alarm recommendations**.

The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.

4. Choose the namespace for a service.

The list of metrics under this namespace is filtered to include only those that have alarm recommendations.

5. The **Download alarm code** displays how many alarms are recommended for the metrics in this namespace. To download infrastructure-as-code alarm definitions for all recommended alarms, choose **Download alarm code** and then choose the code format that you want.

Recommended alarms

The following sections list the metrics that we recommend that you set best practice alarms for. For each metric, the dimensions, alarm intent, recommended threshold, threshold justification, and the period length and number of datapoints is also displayed.

Some metrics might appear twice in the list. This happens when different alarms are recommended for different combinations of dimensions of that metric.

Datapoints to alarm is the number of data points that must be breaching to send the alarm into ALARM state. **Evaluation periods** is the number of periods that are taken into account when the alarm is evaluated. If these numbers are the same, the alarm goes into ALARM state only when that number of consecutive periods have values that breach the threshold. If **Datapoints to alarm** is lower than **Evaluation periods**, then it is an "M out of N" alarm and the alarm goes into ALARM

state if at least **Datapoints to alarm** data points are breaching within any **Evaluation periods** set of data points. For more information, see [Evaluating an alarm](#).

Topics

- [Amazon API Gateway](#)
- [Amazon EC2 Auto Scaling](#)
- [Amazon CloudFront](#)
- [Amazon Cognito](#)
- [Amazon DynamoDB](#)
- [Amazon EBS](#)
- [Amazon EC2](#)
- [Amazon ElastiCache](#)
- [Amazon EC2 \(AWS/ElasticGPUs\)](#)
- [Amazon ECS](#)
- [Amazon ECS with Container Insights](#)
- [Amazon EFS](#)
- [Amazon EKS with Container Insights](#)
- [Amazon Kinesis Data Streams](#)
- [Lambda](#)
- [Lambda Insights](#)
- [Amazon VPC \(AWS/NATGateway\)](#)
- [AWS Private Link \(AWS/PrivateLinkEndpoints\)](#)
- [AWS Private Link \(AWS/PrivateLinkServices\)](#)
- [Amazon RDS](#)
- [Amazon Route 53 Public Data Plane](#)
- [Amazon S3](#)
- [S3ObjectLambda](#)
- [Amazon SNS](#)
- [Amazon SQS](#)
- [AWS VPN](#)

Amazon API Gateway

4XXError

Dimensions: ApiName, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a resource was removed or a client is requesting one that doesn't exist. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4XX errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow [this guide](#) to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4XX errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5XXError

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the

API gateway and the backend API. This [documentation](#) can help you troubleshoot the cause of 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. you can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect low traffic volume for the REST API stage. This can be an indicator of an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of traffic volume drops for each resource and method. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the REST API resource and method in the stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API resource and method in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: Apild, Stage

Alarm description: This alarm helps to detect low traffic volume for the HTTP API stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, we recommend that you create alternative alarms to this in order to have more fine-grained monitoring of traffic volume drops for each route. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: Apild, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the HTTP API route in the stage. This can indicate an issue with the application calling the API such as using incorrect

endpoints. It could also indicate an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API route in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

IntegrationLatency

Dimensions: Apild, Stage

Alarm description: This alarm helps to detect if there is high integration latency for the API requests in a stage. You can correlate the IntegrationLatency metric value with the corresponding latency metric of your backend such as the Duration metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues, or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the integration latency.

Intent: This alarm can detect when the API Gateway requests in a stage have a high integration latency. We recommend this alarm for WebSocket APIs, and we consider it optional for HTTP

APIs because they already have separate alarm recommendations for the Latency metric. If you have detailed CloudWatch metrics enabled and you have different integration latency performance requirements per route, we recommend that you create alternative alarms in order to have more fine-grained monitoring of the integration latency for each route.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

IntegrationLatency

Dimensions: Apild, Stage, Route

Alarm description: This alarm helps to detect if there is high integration latency for the WebSocket API requests for a route in a stage. You can correlate the IntegrationLatency metric value with the corresponding latency metric of your backend such as the Duration metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues.

Intent: This alarm can detect when the API Gateway requests for a route in a stage have high integration latency.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage

Alarm description: This alarm detects high latency in a stage. Find the IntegrationLatency metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for issues. Consider also enabling CloudWatch Logs and checking for errors that might be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the latency. If applicable, refer to the [troubleshooting with Lambda](#) or [troubleshooting for edge-optimized API endpoints](#) guides.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements for each method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each resource and method.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all API workloads. However, you can use it as a starting point for the threshold. You can then choose different

threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine what the expected baseline latency is for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm detects high latency for a resource and method in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that might be causing the high latency. You can also refer to the [troubleshooting with Lambda](#) or [troubleshooting for edge-optimized API endpoints](#) guides if applicable.

Intent: This alarm can detect when the API Gateway requests for a resource and method in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: Apild, Stage

Alarm description: This alarm detects high latency in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that may be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route and narrow down the source of the latency. You can also refer to the [troubleshooting with Lambda integrations guide](#) if applicable.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements per route, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each route.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make it less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: Apild, Stage, Resource, Method

Alarm description: This alarm detects high latency for a route in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and should be investigated for performance issues. Consider also enabling CloudWatch logs and checking for any errors that might be causing the high latency. You can also refer to the [troubleshooting with Lambda integrations guide](#) if applicable.

Intent: This alarm is used to detect when the API Gateway requests for a route in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

4xx

Dimensions: Apild, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors can also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow [this guide](#) to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5xx

Dimensions: Apild, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the API gateway and the backend API. This [documentation](#) can help you troubleshoot the cause for 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine what the acceptable error rate is for the application workload, and then you can tune the threshold accordingly. Frequently occurring 5xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

MessageCount

Dimensions: Apild, Stage

Alarm description: This alarm helps to detect low traffic volume for the WebSocket API stage. This can indicate an issue when clients call the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, it is better to create alternative alarms to this one, in order to have more fine-grained monitoring of traffic volume drops for each route. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic.

Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

MessageCount

Dimensions: Apild, Stage, Route

Alarm description: This alarm helps detect low traffic volume for the WebSocket API route in the stage. This can indicate an issue with the clients calling the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API route in the stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

ClientError

Dimensions: Apild, Stage

Alarm description: This alarm detects a high rate of client errors. This can indicate an issue in the authorization or message parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow [this guide](#) to troubleshoot this issue.

Intent: This alarm can detect high rates of client errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. You can tune the threshold to suit the traffic of the requests as well as to suit your acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ExecutionError

Dimensions: Apild, Stage

Alarm description: This alarm helps to detect a high rate of execution errors. This can be caused by 5xx errors from your integration, permission issues, or other factors preventing successful invocation of the integration, such as the integration being throttled or deleted.

Consider enabling CloudWatch Logs for your API and checking the logs for the type and cause of the errors. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the errors. This [documentation](#) can also help you troubleshoot the cause of any connection errors.

Intent: This alarm can detect high rates of execution errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting execution errors. You can tune the threshold to suit the traffic of the requests, as well as to suit your acceptable error rates. You can analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring execution errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EC2 Auto Scaling

GroupInServiceCapacity

Dimensions: AutoScalingGroupName

Alarm description: This alarm helps to detect when the capacity in the group is below the desired capacity required for your workload. To troubleshoot, check your scaling activities for launch failures and confirm that your desired capacity configuration is correct.

Intent: This alarm can detect a low availability in your auto scaling group because of launch failures or suspended launches.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The threshold value should be the minimum capacity required to run your workload. In most cases, you can set this to match the GroupDesiredCapacity metric.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Amazon CloudFront

5xxErrorRate

Dimensions: DistributionId, Region=Global

Alarm description: This alarm monitors the percentage of 5xx error responses from your origin server, to help you detect if the CloudFront service is having issues. See [Troubleshooting error responses from your origin](#) for information to help you understand the problems with your server. Also, [turn on additional metrics](#) to get detailed error metrics.

Intent: This alarm is used to detect problems with serving requests from the origin server, or problems with communication between CloudFront and your origin server.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the tolerance for 5xx responses. You can analyze historical data and trends, and then set the threshold accordingly. Because 5xx errors can be caused by transient issues, we recommend that you set the threshold to a value greater than 0 so that the alarm is not too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

OriginLatency

Dimensions: DistributionId, Region=Global

Alarm description: The alarm helps to monitor if the origin server is taking too long to respond. If the server takes too long to respond, it might lead to a timeout. Refer to [find and fix delayed responses from applications on your origin server](#) if you experience consistently high OriginLatency values.

Intent: This alarm is used to detect problems with the origin server taking too long to respond.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 80% of the origin response timeout, and use the result as the threshold value. If this metric is consistently close to the origin response timeout value, you might start experiencing 504 errors.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionValidationErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor validation errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem. See [Restrictions on edge functions](#) to understand the common misconfigurations for CloudFront Functions.

Intent: This alarm is used to detect validation errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: A value greater than 0 indicates a validation error. We recommend setting the threshold to 0 because validation errors imply a problem when CloudFront functions

hand off back to CloudFront. For example, CloudFront needs the HTTP Host header in order to process a request. There is nothing stopping a user from deleting the Host header in their CloudFront functions code. But when CloudFront gets the response back and the Host header is missing, CloudFront throws a validation error.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionExecutionErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor execution errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem.

Intent: This alarm is used to detect execution errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend to set the threshold to 0 because an execution error indicates a problem with the code that occurs at runtime.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionThrottles

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you to monitor if your CloudFront function is throttled. If your function is throttled, it means that it is taking too long to execute. To avoid function throttles, consider optimizing the function code.

Intent: This alarm can detect when your CloudFront function is throttled so that you can react and resolve the issue for a smooth customer experience.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend setting the threshold to 0, to allow quicker resolution of the function throttles.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Cognito

SignUpThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled requests. If users are consistently getting throttled, you should increase the limit by requesting a service quota increase. Refer to [Quotas in Amazon Cognito](#) to learn how to request a quota increase. To take actions proactively, consider tracking the [usage quota](#).

Intent: This alarm helps to monitor the occurrence of throttled sign-up requests. This can help you know when to take actions to mitigate any degradation in sign-up experience. Sustained throttling of requests is a negative user sign-up experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the

threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SignInThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled user authentication requests. If users are consistently getting throttled, you might need to increase the limit by requesting a service quota increase. Refer to [Quotas in Amazon Cognito](#) to learn how to request a quota increase. To take actions proactively, consider tracking the [usage quota](#).

Intent: This alarm helps to monitor the occurrence of throttled sign-in requests. This can help you know when to take actions to mitigate any degradation in sign-in experience. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TokenRefreshThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: You can set the threshold value can to suit the traffic of the request as well as to match acceptable throttling for token refresh requests. Throttling is used to protect your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then you can tune your alarm threshold to be higher than your acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Intent: This alarm helps to monitor the occurrence of throttled token refresh requests. This can help you know when to take actions to mitigate any potential problems, to ensure a smooth user experience and the health and reliability of your authentication system. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Threshold value can also be set/tuned to suit the traffic of the request as well as acceptable throttling for token refresh requests. Throttling are there for protecting your system from too many requests, however it is important to monitor if you are under provisioned for your normal traffic as well and see if it is causing the impact. Historical data can also be analyzed to see what is the acceptable throttling for the application workload and threshold can be tuned higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FederationThrottles

Dimensions: UserPool, UserPoolClient, IdentityProvider

Alarm description: This alarm monitors the count of throttled identity federation requests. If you consistently see throttling, it might indicate that you need to increase the limit by requesting a service quota increase. Refer to [Quotas in Amazon Cognito](#) to learn how to request a quota increase.

Intent: This alarm helps to monitor the occurrence of throttled identity federation requests. This can help you take proactive responses to performance bottlenecks or misconfigurations, and ensure a smooth authentication experience for your users. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You can set the threshold to suit the traffic of the request as well as to match the acceptable throttling for identity federation requests. Throttling is used for protecting your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then set the threshold to a value above your acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon DynamoDB

AccountProvisionedReadCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's read capacity is reaching its provisioned limit. You can raise the account quota for read capacity utilization if this occurs. You can view your current quotas for read capacity units and request increases using [Service Quotas](#).

Intent: The alarm can detect if the account's read capacity utilization is approaching its provisioned read capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle read requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AccountProvisionedWriteCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's write capacity is reaching its provisioned limit. You can raise the account quota for write capacity utilization if this occurs. You can view your current quotas for write capacity units and request increases using [Service Quotas](#).

Intent: This alarm can detect if the account's write capacity utilization is approaching its provisioned write capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle write requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that the action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AgeOfOldestUnreplicatedRecord

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the delay in replication to a Kinesis data stream. Under normal operation, AgeOfOldestUnreplicatedRecord should be only milliseconds. This number grows based on unsuccessful replication attempts caused by customer-controlled configuration choices. Customer-controlled configuration examples that lead to unsuccessful replication attempts are an under-provisioned Kinesis data stream capacity that leads to excessive throttling, or a manual update to the Kinesis data stream's access policies that prevents DynamoDB from adding data to the data stream. To keep this metric as low as possible, you need to ensure the right provisioning of Kinesis data stream capacity and make sure that DynamoDB's permissions are unchanged.

Intent: This alarm can monitor unsuccessful replication attempts and the resulting delay in replication to the Kinesis data stream.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the desired replication delay measured in milliseconds. This value depends on your workload's requirements and expected performance.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

FailedToReplicateRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the number of records that DynamoDB failed to replicate to your Kinesis data stream. Certain items larger than 34 KB might expand in size to change data records that are larger than the 1 MB item size limit of Kinesis Data Streams. This size expansion occurs when these larger than 34 KB items include a large number of

Boolean or empty attribute values. Boolean and empty attribute values are stored as 1 byte in DynamoDB, but expand up to 5 bytes when they're serialized using standard JSON for Kinesis Data Streams replication. DynamoDB can't replicate such change records to your Kinesis data stream. DynamoDB skips these change data records, and automatically continues replicating subsequent records.

Intent: This alarm can monitor the number of records that DynamoDB failed to replicate to your Kinesis data stream because of the item size limit of Kinesis Data Streams.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect any records that DynamoDB failed to replicate.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName

Alarm description: This alarm detects if there are high number of read requests getting throttled for the DynamoDB table. To troubleshoot the issue, see [Troubleshooting throttling issues in Amazon DynamoDB](#).

Intent: This alarm can detect sustained throttling for read requests to the DynamoDB table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor whether you are under provisioned and not causing consistent throttling. You can also analyze

historical data to find the acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of read requests getting throttled for the Global Secondary Index of the DynamoDB table. To troubleshoot the issue, see [Troubleshooting throttling issues in Amazon DynamoDB](#).

Intent: The alarm can detect sustained throttling for read requests for the Global Secondary Index of the DynamoDB Table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find an acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual acceptable throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLatency

Dimensions: TableName, ReceivingRegion

Alarm description: The alarm detects if the replica in a Region for the global table is lagging behind the source Region. The latency can increase if an AWS Region becomes degraded and you have a replica table in that Region. In this case, you can temporarily redirect your application's read and write activity to a different AWS Region. If you are using 2017.11.29 (Legacy) of global tables, you should verify that write capacity units (WCUs) are identical for each of the replica tables. You can also make sure to follow recommendations in [Best practices and requirements for managing capacity](#).

Intent: The alarm can detect if the replica table in a Region is falling behind replicating the changes from another Region. This could cause your replica to diverge from the other replicas. It's useful to know the replication latency of each AWS Region and alert if that replication latency increases continually. The replication of the table applies to global tables only.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Replication latencies longer than 3 minutes are generally a cause for investigation. Review the criticality and requirements of replication delay and analyze historical trends, and then select the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

SuccessfulRequestLatency

Dimensions: TableName, Operation

Alarm description: This alarm detects a high latency for the DynamoDB table operation (indicated by the dimension value of the Operation in the alarm). See [this troubleshooting document](#) for troubleshooting latency issues in Amazon DynamoDB.

Intent: This alarm can detect a high latency for the DynamoDB table operation. Higher latency for the operations can negatively impact the overall efficiency of the system.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: DynamoDB provides single-digit millisecond latency on average for singleton operations such as GetItem, PutItem, and so on. However, you can set the threshold based on acceptable tolerance for the latency for the type of operation and table involved in the workload. You can analyze historical data of this metric to find the usual latency for the table operation, and then set the threshold to a number which represents critical delay for the operation.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

SystemErrors

Dimensions: TableName

Alarm description: This alarm detects a sustained high number of system errors for the DynamoDB table requests. If you continue to get 5xx errors, open the [AWS Service Health Dashboard](#) to check for operational issues with the service. You can use this alarm to get notified in case there is a prolonged internal service issue from DynamoDB and it helps you correlate with the issue your client application is facing. Refer [Error handling for DynamoDB](#) for more information.

Intent: This alarm can detect sustained system errors for the DynamoDB table requests. System errors indicate internal service errors from DynamoDB and helps correlate to the issue that the client is having.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected traffic, accounting for an acceptable level of system errors. You can also analyze historical data to find the acceptable

error count for the application workload, and then tune the threshold accordingly. System errors should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ThrottledPutRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the records getting throttled by your Kinesis data stream during the replication of change data capture to Kinesis. This throttling happens because of insufficient Kinesis data stream capacity. If you experience excessive and regular throttling, you might need to increase the number of Kinesis stream shards proportionally to the observed write throughput of your table. To learn more about determining the size of a Kinesis data stream, see [Determining the Initial Size of a Kinesis Data Stream](#).

Intent: This alarm can monitor the number of records that that were throttled by your Kinesis data stream because of insufficient Kinesis data stream capacity.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You might experience some throttling during exceptional usage peaks, but throttled records should remain as low as possible to avoid higher replication latency (DynamoDB retries sending throttled records to the Kinesis data stream). Set the threshold to a number which can help you catch regular excessive throttling. You can also analyze historical data of this metric to find the acceptable throttling rates for the application workload. Tune the threshold to a value that the application can tolerate based on your use case.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD**UserErrors****Dimensions:** None

Alarm description: This alarm detects a sustained high number of user errors for the DynamoDB table requests. You can check client application logs during the issue time frame to see why the requests are invalid. You can check [HTTP status code 400](#) to see the type of error you are getting and take action accordingly. You might have to fix the application logic to create valid requests.

Intent: This alarm can detect sustained user errors for the DynamoDB table requests. User errors for requested operations mean that the client is producing invalid requests and it is failing.

Statistic: Sum**Recommended threshold:** Depends on your situation

Threshold justification: Set the threshold to zero to detect any client side errors. Or you can set it to a higher value if you want to avoid the alarm triggering for a very lower number of errors. Decide based on your use case and traffic for the requests.

Period: 60**Datapoints to alarm:** 10**Evaluation periods:** 10**Comparison Operator:** GREATER_THAN_THRESHOLD**WriteThrottleEvents****Dimensions:** TableName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for the DynamoDB table. See [Troubleshooting throttling issues in Amazon DynamoDB](#) to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests to the DynamoDB table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable level of throttling for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

WriteThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for Global Secondary Index of the DynamoDB table. See [Troubleshooting throttling issues in Amazon DynamoDB](#) to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests for the Global Secondary Index of DynamoDB Table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected Write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable throttling level for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EBS

VolumeStalledIOCheck

Dimensions: Volumeld, Instanceld

Alarm description: This alarm helps you monitor the IO performance of your Amazon EBS volumes. This check detects underlying issues with the Amazon EBS infrastructure, such as hardware or software issues on the storage subsystems underlying the Amazon EBS volumes, hardware issues on the physical host that impact the reachability of the Amazon EBS volumes from your Amazon EC2 instance, and can detect connectivity issues between the instance and the Amazon EBS volumes. If the Stalled IO Check fails, you can either wait for AWS to resolve the issue, or you can take action such as replacing the affected volume or stopping and restarting the instance to which the volume is attached. In most cases, when this metric fails, Amazon EBS will automatically diagnose and recover your volume within a few minutes.

Intent: This alarm can detect the status of your Amazon EBS volumes to determine when these volumes are impaired and can not complete I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EC2

CPUUtilization

Dimensions: InstanceId

Alarm description: This alarm helps to monitor the CPU utilization of an EC2 instance. Depending on the application, consistently high utilization levels might be normal. But if performance is degraded, and the application is not constrained by disk I/O, memory, or network resources, then a maxed-out CPU might indicate a resource bottleneck or application performance problems. High CPU utilization might indicate that an upgrade to a more CPU intensive instance is required. If detailed monitoring is enabled, you can change the period to 60 seconds instead of 300 seconds. For more information, see [Enable or turn off detailed monitoring for your instances](#).

Intent: This alarm is used to detect high CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 70-80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some systems, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your system, and set the threshold accordingly.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

StatusCheckFailed

Dimensions: InstanceId

Alarm description: This alarm helps to monitor both system status checks and instance status checks. If either type of status check fails, then this alarm should be in ALARM state.

Intent: This alarm is used to detect the underlying problems with instances, including both system status check failures and instance status check failures.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

StatusCheckFailed_AttachedEBS

Dimensions: InstanceId

Alarm description: This alarm helps you monitor whether the Amazon EBS volumes attached to an instance are reachable and able to complete I/O operations. This status check detects underlying issues with the compute or Amazon EBS infrastructure such as the following:

- Hardware or software issues on the storage subsystems underlying the Amazon EBS volumes
- Hardware issues on the physical host that impact reachability of the Amazon EBS volumes
- Connectivity issues between the instance and Amazon EBS volumes

When the attached EBS status check fails, you can either wait for Amazon to resolve the issue, or you can take an action such as replacing the affected volumes or stopping and restarting the instance.

Intent: This alarm is used to detect unreachable Amazon EBS volumes attached to an instance. These can cause failures in I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon ElastiCache

CPUUtilization

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm helps to monitor the CPU utilization for the entire ElastiCache instance, including the database engine processes and other processes running on the instance. AWS ElastiCache supports two engine types: Memcached and Redis. When you reach high CPU utilization on a Memcached node, you should consider scaling up your instance type or adding new cache nodes. For Redis, if your main workload is from read requests, you should consider adding more read replicas to your cache cluster. If your main workload is from write requests, you should consider adding more shards to distribute the workload across more primary nodes if you're running in clustered mode, or scaling up your instance type if you're running Redis in non-clustered mode.

Intent: This alarm is used to detect high CPU utilization of ElastiCache hosts. It is useful to get a broad view of the CPU usage across the entire instance, including non-engine processes.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the percentage that reflects a critical CPU utilization level for your application. For Memcached, the engine can use up to `num_threads` cores. For Redis, the engine is largely single-threaded, but might use additional cores if available to accelerate I/O. In most cases, you can set the threshold to about 90% of your available CPU. Because Redis is single-threaded, the actual threshold value should be calculated as a fraction of the node's total capacity.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CurrConnections

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm detects high connection count, which might indicate heavy load or performance issues. A constant increase of `CurrConnections` might lead to exhaustion of the 65,000 available connections. It may indicate that connections improperly closed on the application side and were left established on the server side. You should consider using connection pooling or idle connection timeouts to limit the number of connections made to the cluster, or for Redis, consider tuning [tcp-keepalive](#) on your cluster to detect and terminate potential dead peers.

Intent: The alarm helps you identify high connection counts that could impact the performance and stability of your ElastiCache cluster.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the acceptable range of connections for your cluster. Review the capacity and the expected workload of your ElastiCache cluster and analyze the historical connection counts during regular usage to establish a baseline, and then select a threshold accordingly. Remember that each node can support up to 65,000 concurrent connections.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseMemoryUsagePercentage

Dimensions: CacheClusterId

Alarm description: This alarm helps you monitor the memory utilization of your cluster. When your `DatabaseMemoryUsagePercentage` reaches 100%, the Redis `maxmemory` policy is

triggered and evictions might occur based on the policy selected. If no object in the cache matches the eviction policy, write operations fail. Some workloads expect or rely on evictions, but if not, you will need to increase the memory capacity of your cluster. You can scale your cluster out by adding more primary nodes, or scale it up by using a larger node type. Refer to [Scaling ElastiCache for Redis clusters](#) for details.

Intent: This alarm is used to detect high memory utilization of your cluster so that you can avoid failures when writing to your cluster. It is useful to know when you'll need to scale up your cluster if your application does not expect to experience evictions.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on your application's memory requirements and the memory capacity of your ElastiCache cluster, you should set the threshold to the percentage that reflects the critical level of memory usage of the cluster. You can use historical memory usage data as reference for acceptable memory usage threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EngineCPUUtilization

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the CPU utilization of a Redis engine thread within the ElastiCache instance. Common reasons for high engine CPU are long-running commands that consume high CPU, a high number of requests, an increase of new client connection requests in a short time period, and high evictions when the cache doesn't have enough memory to hold new data. You should consider [Scaling ElastiCache for Redis clusters](#) by adding more nodes or scaling up your instance type.

Intent: This alarm is used to detect high CPU utilization of the Redis engine thread. It is useful if you want to monitor the CPU usage of the database engine itself.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Set the threshold to a percentage that reflects the critical engine CPU utilization level for your application. You can benchmark your cluster using your application and expected workload to correlate EngineCPUUtilization and performance as a reference, and then set the threshold accordingly. In most cases, you can set the threshold to about 90% of your available CPU.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLag

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the replication health of your ElastiCache cluster. A high replication lag means that the primary node or the replica can't keep up the pace of the replication. If your write activity is too high, consider scaling your cluster out by adding more primary nodes, or scaling it up by using a larger node type. Refer to [Scaling ElastiCache for Redis clusters](#) for details. If your read replicas are overloaded by the amount of read requests, consider adding more read replicas.

Intent: This alarm is used to detect a delay between data updates on the primary node and their synchronization to replica node. It helps to ensure data consistency of a read replica cluster node.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to your application's requirements and the potential impact of replication lag. You should consider your application's expected write rates and network conditions for the acceptable replication lag.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EC2 (AWS/ElasticGPUs)

GPUConnectivityCheckFailed

Dimensions: InstanceId, EGPUId

Alarm description: This alarm helps to detect connection failures between the instance and the Elastic Graphics accelerator. Elastic Graphics uses the instance network to send OpenGL commands to a remotely attached graphics card. In addition, a desktop running an OpenGL application with an Elastic Graphics accelerator is usually accessed using remote access technology. It is important to distinguish between a performance problem related to the OpenGL rendering or the desktop remote access technology. To learn more about the issue, see [Investigate application performance issues](#).

Intent: This alarm is used to detect connectivity issues from the instance to the Elastic Graphics accelerator.

Statistic: Maximum

Recommended threshold: 0.0

Threshold justification: The threshold value of 1 indicates that the connectivity has failed.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

GPUHealthCheckFailed

Dimensions: InstanceId, EGPUId

Alarm description: This alarm helps you know when the status of the Elastic graphics accelerator is unhealthy. If the accelerator is not healthy, see the troubleshooting steps at [Resolve Unhealthy status issues](#).

Intent: This alarm is used to detect if the Elastic Graphics accelerator is not healthy.

Statistic: Maximum

Recommended threshold: 0.0

Threshold justification: The threshold value of 1 indicates a status check failure.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS

CPUReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high CPU reservation of the ECS cluster. High CPU reservation might indicate that the cluster is running out of registered CPUs for the task. To troubleshoot, you can add more capacity, you can scale the cluster, or you can set up auto scaling.

Intent: The alarm is used to detect whether the total number of CPU units reserved by tasks on the cluster is reaching the total CPU units registered for the cluster. This helps you know when to scale up the cluster. Reaching the total CPU units for the cluster can result in running out of CPU for tasks. If you have EC2 capacity providers managed scaling turned on, or you have associated Fargate to capacity providers, then this alarm is not recommended.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Set the threshold for CPU reservation to 90%. Alternatively, you can choose a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CPUUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high CPU utilization of the ECS service. If there is no ongoing ECS deployment, a maxed-out CPU utilization might indicate a resource bottleneck or application performance problems. To troubleshoot, you can increase the CPU limit.

Intent: This alarm is used to detect high CPU utilization for the ECS service. Consistent high CPU utilization can indicate a resource bottleneck or application performance problems.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: The service metrics for CPU utilization might exceed 100% utilization. However, we recommend that you monitor the metric for high CPU utilization to avoid impacting other services. Set the threshold to about 90-95%. We recommend that you update your task definitions to reflect actual usage to prevent future issues with other services.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

MemoryReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high memory reservation of the ECS cluster. High memory reservation might indicate a resource bottleneck for the cluster. To troubleshoot, analyze the service task for performance to see if memory utilization of the task can be optimized. Also, you can register more memory or set up auto scaling.

Intent: The alarm is used to detect whether the total memory units reserved by tasks on the cluster is reaching the total memory units registered for the cluster. This can help you know when to scale up the cluster. Reaching the total memory units for the cluster can cause the cluster to be unable to launch new tasks. If you have EC2 capacity providers managed scaling turned on or you have associated Fargate to capacity providers, this alarm is not recommended.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Set the threshold for memory reservation to 90%. You can adjust this to a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

HTTPCode_Target_5XX_Count

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high server-side error count for the ECS service. This can indicate that there are errors that cause the server to be unable to serve requests. To troubleshoot, check your application logs.

Intent: This alarm is used to detect a high server-side error count for the ECS service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Calculate the value of about 5% of your average traffic and use this value as a starting point for the threshold. You can find the average traffic by using the RequestCount metric. You can also analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TargetResponseTime

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high target response time for ECS service requests. This can indicate that there are problems that cause the service to be unable to serve requests in time. To troubleshoot, check the CPUUtilization metric to see if the service is running out of CPU, or check the CPU utilization of other downstream services that your service depends on.

Intent: This alarm is used to detect a high target response time for ECS service requests.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Review the criticality and requirements of the target response time of the service and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS with Container Insights

EphemeralStorageUtilized

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high ephemeral storage utilized of the Fargate cluster. If ephemeral storage is consistently high, you can check ephemeral storage usage and increase the ephemeral storage.

Intent: This alarm is used to detect high ephemeral storage usage for the Fargate cluster. Consistent high ephemeral storage utilized can indicate that the disk is full and it might lead to failure of the container.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the ephemeral storage size. You can adjust this value based on your acceptable ephemeral storage utilization of the Fargate cluster. For some systems, a consistently high ephemeral storage utilized might be normal, while for others, it might lead to failure of the container.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

RunningTaskCount

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a low running task count of the ECS service. If the running task count is too low, it can indicate that the application can't handle the service load and it might lead to performance issues. If there is no running task, the Amazon ECS service might be unavailable or there might be deployment issues.

Intent: This alarm is used to detect whether the number of running tasks are too low. A consistent low running task count can indicate ECS service deployment or performance issues.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: You can adjust the threshold based on the minimum running task count of the ECS service. If the running task count is 0, the Amazon ECS service will be unavailable.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

instance_filesystem_utilization

Dimensions: InstanceId, ContainerInstanceId, ClusterName

Alarm description: This alarm helps you detect a high file system utilization of the ECS cluster. If the file system utilization is consistently high, check the disk usage.

Intent: This alarm is used to detect high file system utilization for the Amazon ECS cluster. A consistent high file system utilization can indicate a resource bottleneck or application performance problems, and it might prevent running new tasks.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: You can set the threshold for file system utilization to about 90-95%. You can adjust this value based on the acceptable file system capacity level of the Amazon ECS cluster. For some systems, a consistently high file system utilization might be normal and not indicate a problem, while for others, it might be a cause of concern and might lead to performance issues and prevent running new tasks.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EFS

PercentIOLimit

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that the workload stays within the I/O limit available to the file system. If the metric reaches its I/O limit consistently, consider moving the

application to a file system that uses Max I/O performance as mode. For troubleshooting, check clients that are connected to the file system and applications of the clients that throttles the file system.

Intent: This alarm is used to detect how close the file system is to reach the I/O limit of the General Purpose performance mode. Consistent high I/O percentage can be an indicator of the file system cannot scale with respect to I/O requests enough and the file system can be a resource bottleneck for the applications that use the file system.

Statistic: Average

Recommended threshold: 100.0

Threshold justification: When the file system reaches its I/O limit, it may respond to read and write requests slower. Therefore, it is recommended that the metric is monitored to avoid impacting applications that use the file system. The threshold can be set around 100%. However, this value can be adjusted to a lower value based on file system characteristics.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

BurstCreditBalance

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that there is available burst credit balance for the file system usage. When there is no available burst credit, applications access to the the file system will be limited due to low throughput. If the metric drops to 0 consistently, consider changing the throughput mode to [Elastic or Provisioned throughput mode](#).

Intent: This alarm is used to detect low burst credit balance of the file system. Consistent low burst credit balance can be an indicator of the slowing down in throughput and increase in I/O latency.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: When the file system run out of burst credits and even if the baseline throughput rate is lower, EFS continues to provide a metered throughput of 1 MiBps to all file systems. However, it is recommended that the metric is monitored for low burst credit balance to avoid the file system acting as resource bottleneck for the applications. The threshold can be set around 0 bytes.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EKS with Container Insights

node_cpu_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high CPU utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need for replacing your worker nodes with instances that have greater CPU or a need to scale the system horizontally.

Intent: This alarm helps to monitor the CPU utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_filesystem_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high file system utilization in the worker nodes of the EKS cluster. If the utilization is consistently high, you might need to update your worker nodes to have larger disk volume, or you might need to scale horizontally.

Intent: This alarm helps to monitor the filesystem utilization of the worker nodes in the EKS cluster. If the utilization reaches 100%, it can lead to application failure, disk I/O bottlenecks, pod eviction, or the node to become unresponsive entirely.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: If there's sufficient disk pressure (meaning that the disk is getting full), nodes are marked as not healthy, and the pods are evicted from the node. Pods on a node with disk pressure are evicted when the available file system is lower than the eviction thresholds set on the kubelet. Set the alarm threshold so that you have enough time to react before the node is evicted from the cluster.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_memory_utilization

Dimensions: ClusterName

Alarm description: This alarm helps in detecting high memory utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need to scale the number of pod replicas, or optimize your application.

Intent: This alarm helps to monitor the memory utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_cpu_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high CPU utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the CPU limit for the affected pod.

Intent: This alarm helps to monitor the CPU utilization of the pods belonging to a Kubernetes Service in the EKS cluster, so that you can quickly identify if a service's pod is consuming higher CPU than expected.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_memory_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high memory utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the memory limit for the affected pod.

Intent: This alarm helps to monitor the memory utilization of the pods in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

Dimensions: StreamName

Alarm description: This alarm can detect if iterator maximum age is too high. For real-time data processing applications, configure data retention according to tolerance of the delay. This is usually within minutes. For applications that process historic data, use this metric to monitor catchup speed. A quick solution to stop data loss is to increase the retention period while you troubleshoot the issue. You can also increase the number of workers processing records in your consumer application. The most common causes for gradual iterator age increase are insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. See [link](#) for more details.

Intent: This alarm is used to detect if data in your stream is going to expire because of being preserved too long or because record processing is too slow. It helps you avoid data loss after reaching 100% of the stream retention time.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the stream retention period and tolerance of processing delay for the records. Review your requirements and analyze historical trends, and then set the threshold to the number of milliseconds that represents a critical processing delay. If an iterator's age passes 50% of the retention period (by default, 24 hours, configurable up to 365 days), there is a risk for data loss because of record expiration. You can monitor the metric to make sure that none of your shards ever approach this limit.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

GetRecords.Success

Dimensions: StreamName

Alarm description: This metric increments whenever your consumers successfully read data from your stream. GetRecords doesn't return any data when it throws an exception. The most common exception is ProvisionedThroughputExceededException because request rate for the stream is too high, or because available throughput is already served for the given second. Reduce the frequency or size of your requests. For more information, see Streams [Limits](#) in the Amazon Kinesis Data Streams Developer Guide, and [Error Retries and Exponential Backoff in AWS](#).

Intent: This alarm can detect if the retrieval of records from the stream by consumers is failing. By setting an alarm on this metric, you can proactively detect any issues with data consumption, such as increased error rates or a decline in successful retrievals. This allows you to take timely actions to resolve potential problems and maintain a smooth data processing pipeline.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of retrieving records from the stream, set the threshold based on your application's tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical GetRecords metric data as reference for the acceptable failure rate. You should also consider retries when

setting the threshold because failed records can be retried. This helps to prevent transient spikes from triggering unnecessary alerts.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecord.Success

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed PutRecord operations breaches the threshold. Investigate the data producer logs to find the root causes of the failures. The most common reason is insufficient provisioned throughput on the shard that caused the ProvisionedThroughputExceededException. It happens because the request rate for the stream is too high, or the throughput attempted to be ingested into the shard is too high. Reduce the frequency or size of your requests. For more information, see Streams [Limits](#) and [Error Retries and Exponential Backoff in AWS](#).

Intent: This alarm can detect if ingestion of records into the stream is failing. It helps you identify issues in writing data to the stream. By setting an alarm on this metric, you can proactively detect any issues of producers in publishing data to the stream, such as increased error rates or a decrease in successful records being published. This enables you to take timely actions to address potential problems and maintain a reliable data ingestion process.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of data ingestion and processing to your service, set the threshold based on your application's tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical PutRecord metric data as reference for the acceptable failure rate. You should also consider retries when setting the threshold because failed records can be retried.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecords.FailedRecords

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed PutRecords exceeds the threshold. Kinesis Data Streams attempts to process all records in each PutRecords request, but a single record failure does not stop the processing of subsequent records. The main reason for these failures is exceeding the throughput of a stream or an individual shard. Common causes are traffic spikes and network latencies that cause records to arrive to the stream unevenly. You should detect unsuccessfully processed records and retry them in a subsequent call. Refer to [Handling Failures When Using PutRecords](#) for more details.

Intent: This alarm can detect consistent failures when using batch operation to put records to your stream. By setting an alarm on this metric, you can proactively detect an increase in failed records, enabling you to take timely actions to address the underlying problems and ensure a smooth and reliable data ingestion process.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the number of failed records reflecting the tolerance of the the application for failed records. You can use historical data as reference for the acceptable failure value. You should also consider retries when setting the threshold because failed records can be retried in subsequent PutRecords calls.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadProvisionedThroughputExceeded

Dimensions: StreamName

Alarm description: The alarm tracks the number of records that result in read throughput capacity throttling. If you find that you are being consistently throttled, you should consider

adding more shards to your stream to increase your provisioned read throughput. If there is more than one consumer application running on the stream, and they share the `GetRecords` limit, we recommend that you register new consumer applications via Enhanced Fan-Out. If adding more shards does not lower the number of throttles, you may have a “hot” shard that is being read from more than other shards are. Enable enhanced monitoring, find the “hot” shard, and split it.

Intent: This alarm can detect if consumers are throttled when they exceed your provisioned read throughput (determined by the number of shards you have). In that case, you won't be able to read from the stream, and the stream can start backing up.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried and hence setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact reading from the stream and should trigger the alarm. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: `GREATER_THAN_THRESHOLD`

SubscribeToShardEvent.MillisBehindLatest

Dimensions: `StreamName`, `ConsumerName`

Alarm description: This alarm detects when the delay of record processing in the application breaches the threshold. Transient problems such as API operation failures to a downstream application can cause a sudden increase in the metric. You should investigate if they consistently happen. A common cause is the consumer is not processing records fast enough because of insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. Blocking calls in critical path is often the cause of slowdowns in record processing. You can increase your parallelism by increasing the number of shards. You should also confirm underlying processing nodes have sufficient physical resources during peak demand.

Intent: This alarm can detect delay in the subscription to shard event of the stream. This indicates a processing lag and can help identify potential issues with the consumer application's performance or the overall stream's health. When the processing lag becomes significant, you should investigate and address any bottlenecks or consumer application inefficiencies to ensure real-time data processing and minimize data backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the delay that your application can tolerate. Review your application's requirements and analyze historical trends, and then select a threshold accordingly. When the `SubscribeToShard` call succeeds, your consumer starts receiving `SubscribeToShardEvent` events over the persistent connection for up to 5 minutes, after which time you need to call `SubscribeToShard` again to renew the subscription if you want to continue to receive records.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: `GREATER_THAN_THRESHOLD`

WriteProvisionedThroughputExceeded

Dimensions: `StreamName`

Alarm description: This alarm detects when the number of records resulting in write throughput capacity throttling reached the threshold. When your producers exceed your provisioned write throughput (determined by the number of shards you have), they are throttled and you won't be able to put records to the stream. To address consistent throttling, you should consider adding shards to your stream. This raises your provisioned write throughput and prevents future throttling. You should also consider partition key choice when ingesting records. Random partition key is preferred because it spreads records evenly across the shards of the stream, whenever possible.

Intent: This alarm can detect if your producers are being rejected for writing records because of throttling of the stream or shard. If your stream is in Provisioned mode, then setting this alarm helps you proactively take actions when the data stream reaches its limits, allowing you

to optimize the provisioned capacity or take appropriate scaling actions to avoid data loss and maintain smooth data processing.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried, so setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact writing to the stream, and you should set the alarm threshold to detect this. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda

ClaimedAccountConcurrency

Dimensions: None

Alarm description: This alarm helps to monitor if the concurrency of your Lambda functions is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. [Request a concurrency increase](#) in this Region.
2. Identify and reduce any unused reserved concurrency or provisioned concurrency.
3. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
4. Increase the batch size of the functions, so that more messages are processed by each function invocation.

Intent: This alarm can proactively detect if the concurrency of your Lambda functions is approaching the Region-level concurrency quota of your account, so that you can act on it. Functions are throttled if `ClaimedAccountConcurrency` reaches the Region-level

concurrency quota of the account. If you are using Reserved Concurrency (RC) or Provisioned Concurrency (PC), this alarm gives you more visibility on concurrency utilization than an alarm on `ConcurrentExecutions` would.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 90% of the concurrency quota set for the account in the Region, and use the result as the threshold value. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you should check the quota of your account from the Service Quotas dashboard.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Errors

Dimensions: FunctionName

Alarm description: This alarm detects high error counts. Errors includes the exceptions thrown by the code as well as exceptions thrown by the Lambda runtime. You can check the logs related to the function to diagnose the issue.

Intent: The alarm helps detect high error counts in function invocations.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value can depend on the tolerance for errors in your application. Understand the criticality of the invocations that the function is handling. For some applications, any error might be unacceptable, while other applications might allow for a certain margin of error.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Throttles

Dimensions: FunctionName

Alarm description: This alarm detects a high number of throttled invocation requests. Throttling occurs when there is no concurrency is available for scale up. There are several approaches to resolve this issue. 1) Request a concurrency increase from AWS Support in this Region. 2) Identify performance issues in the function to improve the speed of processing and therefore improve throughput. 3) Increase the batch size of the function, so that more messages are processed by each function invocation.

Intent: The alarm helps detect a high number of throttled invocation requests for a Lambda function. It is important to know if requests are constantly getting rejected due to throttling and if you need to improve Lambda function performance or increase concurrency capacity to avoid constant throttling.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value of the threshold can depend on the tolerance of the application. Set the threshold according to its usage and scaling requirements of the function.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Duration

Dimensions: FunctionName

Alarm description: This alarm detects long duration times for processing an event by a Lambda function. Long durations might be because of changes in function code making the function take longer to execute, or the function's dependencies taking longer.

Intent: This alarm can detect a long running duration of a Lambda function. High runtime duration indicates that a function is taking a longer time for invocation, and can also impact the concurrency capacity of invocation if Lambda is handling a higher number of events. It is critical to know if the Lambda function is constantly taking longer execution time than expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The threshold for the duration depends on your application and workloads and your performance requirements. For high-performance requirements, set the threshold to a shorter time to see if the function is meeting expectations. You can also analyze historical data for duration metrics to see if the time taken matches the performance expectation of the function, and then set the threshold to a longer time than the historical average. Make sure to set the threshold lower than the configured function timeout.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ConcurrentExecutions

Dimensions: FunctionName

Alarm description: This alarm helps to monitor if the concurrency of the function is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. Request a concurrency increase in this Region.
2. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
3. Increase the batch size of the functions, so that more messages are processed by each function invocation.

To get better visibility on reserved concurrency and provisioned concurrency utilization, set an alarm on the new metric `ClaimedAccountConcurrency` instead.

Intent: This alarm can proactively detect if the concurrency of the function is approaching the Region-level concurrency quota of your account, so that you can act on it. A function is throttled if it reaches the Region-level concurrency quota of the account.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the concurrency quota set for the account in the Region. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you can check the quota of your account, as it can be increased by contacting AWS support.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda Insights

We recommend setting best-practice alarms for the following Lambda Insights metrics.

memory_utilization

Dimensions: function_name

Alarm description: This alarm is used to detect if the memory utilization of a lambda function is approaching the configured limit. For troubleshooting, you can try to 1) Optimize your code. 2) Rightly size your memory allocation by accurately estimating the memory requirements. You can refer to [Lambda Power Tuning](#) for the same. 3) Use connection pooling. Refer to [Using Amazon RDS Proxy with Lambda](#) for the connection pooling for RDS database. 4) You can also consider designing your functions to avoid storing large amounts of data in memory between invocations.

Intent: This alarm is used to detect if the memory utilization for the Lambda function is approaching the configured limit.

Statistic: Average

Threshold Suggestion: 90.0

Threshold Justification: Set the threshold to 90% to get an alert when memory utilization exceeds 90% of the allocated memory. You can adjust this to a lower value if you have a concern for the workload for memory utilization. You can also check the historical data for this metric and set the threshold accordingly.

Period: 60

Datapoints to alarm: 10

Evaluation Periods: 10

ComparisonOperator: GREATER_THAN_THRESHOLD

Amazon VPC (AWS/NATGateway)

ErrorPortAllocation

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when the NAT Gateway is unable to allocate ports to new connections. To resolve this issue, see [Resolve port allocation errors on NAT Gateway](#).

Intent: This alarm is used to detect if the NAT gateway could not allocate a source port.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the value of ErrorPortAllocation is greater than zero, that means too many concurrent connections to a single popular destination are open through NATGateway.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

PacketsDropCount

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when packets are dropped by NAT Gateway. This might happen because of an issue with NAT Gateway, so check [AWS service health dashboard](#) for the status of AWS NAT Gateway in your Region. This can help you correlate the network issue related to traffic using NAT gateway.

Intent: This alarm is used to detect if packets are being dropped by NAT Gateway.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of 0.01 percent of the total traffic on the NAT Gateway and use that result as the threshold value. Use historical data of the traffic on NAT Gateway to determine the threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

AWS Private Link (AWS/PrivateLinkEndpoints)

PacketsDropped

Dimensions: VPC Id, VPC Endpoint Id, Endpoint Type, Subnet Id, Service Name

Alarm description: This alarm helps to detect if the endpoint or endpoint service is unhealthy by monitoring the number of packets dropped by the endpoint. Note that packets larger than 8500 bytes that arrive at the VPC endpoint are dropped. For troubleshooting, see [connectivity problems between an interface VPC endpoint and an endpoint service](#).

Intent: This alarm is used to detect if the endpoint or endpoint service is unhealthy.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the use case. If you want to be aware of the unhealthy status of the endpoint or endpoint service, you should set the threshold low so that you get a chance to fix the issue before a huge data loss. You can use historical data to understand the tolerance for dropped packets and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

AWS Private Link (AWS/PrivateLinkServices)

RstPacketsSent

Dimensions: Service Id, Load Balancer Arn, Az

Alarm description: This alarm helps you detect unhealthy targets of an endpoint service based on the number of reset packets that are sent to endpoints. When you debug connection errors with a consumer of your service, you can validate whether the service is resetting connections with the RstPacketsSent metric, or if something else is failing on the network path.

Intent: This alarm is used to detect unhealthy targets of an endpoint service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The threshold depends on the use case. If your use case can tolerate targets being unhealthy, you can set the threshold high. If the use case can't tolerate unhealthy targets you can set the threshold very low.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon RDS

CPUUtilization

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor consistent high CPU utilization. CPU utilization measures non-idle time. Consider using [Enhanced Monitoring](#) or [Performance Insights](#) to review which [wait time](#) is consuming the most of the CPU time (guest, irq, wait, nice, and so on) for MariaDB, MySQL, Oracle, and PostgreSQL. Then evaluate which queries consume the highest amount of CPU. If you can't tune your workload, consider moving to a larger DB instance class.

Intent: This alarm is used to detect consistent high CPU utilization in order to prevent very high response time and time-outs. If you want to check micro-bursting of CPU utilization you can set a lower alarm evaluation time.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Random spikes in CPU consumption might not hamper database performance, but sustained high CPU can hinder upcoming database requests. Depending on the overall database workload, high CPU at your RDS/Aurora instance can degrade the overall performance.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseConnections

Dimensions: DBInstanceIdentifier

Alarm description: This alarm detects a high number of connections. Review existing connections and terminate any that are in `sleep` state or that are improperly closed. Consider using connection pooling to limit the number of new connections. Alternatively, increase the DB instance size to use a class with more memory and hence a higher default value for

`max_connections` or increase the `max_connections` value in [RDS](#) and Aurora [MySQL](#) and [PostgreSQL](#) for the current class if it can support your workload.

Intent: This alarm is used to help prevent rejected connections when the maximum number of DB connections is reached. This alarm is not recommended if you frequently change DB instance class, because doing so changes the memory and default maximum number of connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The number of connections allowed depends on the size of your DB instance class and database engine-specific parameters related to processes/connections. You should calculate a value between 90-95% of the maximum number of connections for your database and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EBSByteBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a low percentage of throughput credits remaining. For troubleshooting, check [latency problems in RDS](#).

Intent: This alarm is used to detect a low percentage of throughput credits remaining in the burst bucket. Low byte balance percentage can cause throughput bottleneck issues. This alarm is not recommended for Aurora PostgreSQL instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: A throughput credit balance below 10% is considered to be poor and you should set the threshold accordingly. You can also set a lower threshold if your application can tolerate a lower throughput for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

EBSIOBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low percentage of IOPS credits remaining. For troubleshooting, see [latency problems in RDS](#).

Intent: This alarm is used to detect a low percentage of I/O credits remaining in the burst bucket. Low IOPS balance percentage can cause IOPS bottleneck issues. This alarm is not recommended for Aurora instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: An IOPS credits balance below 10% is considered to be poor and you can set the threshold accordingly. You can also set a lower threshold, if your application can tolerate a lower IOPS for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

FreeableMemory

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low freeable memory which can mean that there is a spike in database connections or that your instance may be under high memory pressure. Check for memory pressure by monitoring the CloudWatch metrics for SwapUsage` in addition to FreeableMemory. If the instance memory consumption is frequently too high, this indicates that you should check your workload or upgrade your instance class. For Aurora reader

DB instance, consider adding additional reader DB instances to the cluster. For information about troubleshooting Aurora, see [freeable memory issues](#).

Intent: This alarm is used to help prevent running out of memory which can result in rejected connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the workload and instance class, different values for the threshold can be appropriate. Ideally, available memory should not go below 25% of total memory for prolonged periods. For Aurora, you can set the threshold close to 5%, because the metric approaching 0 means that the DB instance has scaled up as much as it can. You can analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_THRESHOLD

FreeLocalStorage

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low free local storage. Aurora PostgreSQL-Compatible Edition uses local storage for storing error logs and temporary files. Aurora MySQL uses local storage for storing error logs, general logs, slow query logs, audit logs, and non-InnoDB temporary tables. These local storage volumes are backed by Amazon EBS Store and can be extended by using a larger DB instance class. For troubleshooting, check Aurora [PostgreSQL-Compatible](#) and [MySQL-Compatible](#).

Intent: This alarm is used to detect how close the Aurora DB instance is to reaching the local storage limit, if you do not use Aurora Serverless v2 or higher. Local storage can reach capacity when you store non-persistent data, such as temporary table and log files, in the local storage. This alarm can prevent an out-of-space error that occurs when your DB instance runs out of local storage.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 10%-20% of the amount of storage available based on velocity and trend of volume usage, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

FreeStorageSpace

Dimensions: DBInstanceIdentifier

Alarm description: This alarm watches for a low amount of available storage space. Consider scaling up your database storage if you frequently approach storage capacity limits. Include some buffer to accommodate unforeseen increases in demand from your applications. Alternatively, consider enabling RDS storage auto scaling. Additionally, consider freeing up more space by deleting unused or outdated data and logs. For further information, check [RDS run out of storage document](#) and [PostgreSQL storage issues document](#).

Intent: This alarm helps prevent storage full issues. This can prevent downtime that occurs when your database instance runs out of storage. We do not recommend using this alarm if you have storage auto scaling enabled, or if you frequently change the storage capacity of the database instance.

Statistic: Minimum

Recommended threshold: Depends on your situation

Threshold justification: The threshold value will depend on the currently allocated storage space. Typically, you should calculate the value of 10 percent of the allocated storage space and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

MaximumUsedTransactionIDs

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps prevent transaction ID wraparound for PostgreSQL. Refer to the troubleshooting steps in [this blog](#) to investigate and resolve the issue. You can also refer to [this blog](#) to familiarize yourself further with autovacuum concepts, common issues and best practices.

Intent: This alarm is used to help prevent transaction ID wraparound for PostgreSQL.

Statistic: Average

Recommended threshold: 1.0E9

Threshold justification: Setting this threshold to 1 billion should give you time to investigate the problem. The default `autovacuum_freeze_max_age` value is 200 million. If the age of the oldest transaction is 1 billion, autovacuum is having a problem keeping this threshold below the target of 200 million transaction IDs.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high read latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to [troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck](#). For Aurora, you can switch to an instance class that has [I/O-Optimized storage configuration](#). See [Planning I/O in Aurora](#) for guidance.

Intent: This alarm is used to detect high read latency. Database disks normally have a low read/write latency, but they can have issues that can cause high latency operations.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Read latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have higher latency for read operations. Review the criticality and requirements of read latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicaLag

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you understand the number of seconds a replica is behind the primary instance. A PostgreSQL Read Replica reports a replication lag of up to five minutes if there are no user transactions occurring on the source database instance. When the ReplicaLag metric reaches 0, the replica has caught up to the primary DB instance. If the ReplicaLag metric returns -1, then replication is currently not active. For guidance related to RDS PostgreSQL, see [replication best practices](#) and for troubleshooting ReplicaLag and related errors, see [troubleshooting ReplicaLag](#).

Intent: This alarm can detect the replica lag which reflects the data loss that could happen in case of a failure of the primary instance. If the replica gets too far behind the primary and the primary fails, the replica will be missing data that was in the primary instance.

Statistic: Maximum

Recommended threshold: 60.0

Threshold justification: Typically, the acceptable lag depends on the application. We recommend no more than 60 seconds.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

WriteLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high write latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to [troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck](#). For Aurora, you can switch to an instance class that has [I/O-Optimized storage configuration](#). See [Planning I/O in Aurora](#) for guidance.

Intent: This alarm is used to detect high write latency. Although database disks typically have low read/write latency, they may experience problems that cause high latency operations. Monitoring this will assure you the disk latency is as low as expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Write latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have a higher latency for write operations. Review the criticality and requirements of write latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DBLoad

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high DB load. If the number of processes exceed the number of vCPUs, the processes start queuing. When the queuing increases, the performance is impacted. If the DB load is often above the maximum vCPU, and the primary

wait state is CPU, the CPU is overloaded. In this case, you can monitor CPUUtilization, DBLoadCPU and queued tasks in Performance Insights/Enhanced Monitoring. You might want to throttle connections to the instance, tune any SQL queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve.

Intent: This alarm is used to detect a high DB load. High DB load can cause performance issues in the DB instance. This alarm is not applicable to serverless DB instances.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The maximum vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance. Depending on the maximum vCPU, different values for the threshold can be appropriate. Ideally, DB load should not go above vCPU line.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

AuroraVolumeBytesLeftTotal

Dimensions: DBClusterIdentifier

Alarm description: This alarm helps to monitor low remaining total volume. When the total volume left reaches the size limit, the cluster reports an out-of-space error. Aurora storage automatically scales with the data in the cluster volume and expands up to 128 TiB or 64 TiB depending on the [DB engine version](#). Consider reducing storage by dropping tables and databases that you no longer need. For more information, check [storage scaling](#).

Intent: This alarm is used to detect how close the Aurora cluster is to the volume size limit. This alarm can prevent an out-of-space error that occurs when your cluster runs out of space. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 10%-20% of the actual size limit based on velocity and trend of volume usage increase, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

AuroraBinlogReplicaLag

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor the error state of Aurora writer instance replication. For more information, see [Replicating Aurora MySQL DB clusters across AWS Regions](#). For troubleshooting, see [Aurora MySQL replication issues](#).

Intent: This alarm is used to detect whether the writer instance is in an error state and can't replicate the source. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: -1.0

Threshold justification: We recommend that you use -1 as the threshold value because Aurora MySQL publishes this value if the replica is in an error state.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

BlockedTransactions

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a high blocked transaction count in an Aurora DB instance. Blocked transactions can end in either a rollback or a commit. High concurrency,

idles in transaction, or long running transactions can lead to blocked transactions. For troubleshooting, see [Aurora MySQL](#) documentation.

Intent: This alarm is used to detect a high count of blocked transactions in an Aurora DB instance in order to prevent transaction rollbacks and performance degradation.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 5% of all transactions of your instance using the `ActiveTransactions` metric and use that result as the threshold value. You can also review the criticality and requirements of blocked transactions and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

BufferCacheHitRatio

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you monitor a consistent low cache hit ratio of the Aurora cluster. A low hit ratio indicates that your queries on this DB instance are frequently going to disk. For troubleshooting, investigate your workload to see which queries are causing this behavior, and see the [DB instance RAM recommendations](#) document.

Intent: This alarm is used to detect consistent low cache hit ratio in order to prevent a sustained performance decrease in the Aurora instance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: You can set the threshold for buffer cache hit ratio to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

EngineUptime

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor low downtime of the writer DB instance. The writer DB instance can go down due to a reboot, maintenance, upgrade, or failover. When the uptime reaches 0 because of a failover in the cluster, and the cluster has one or more Aurora Replicas, then an Aurora Replica is promoted to the primary writer instance during a failure event. To increase the availability of your DB cluster, consider creating one or more Aurora Replicas in two or more different Availability Zones. For more information check [factors that influence Aurora downtime](#).

Intent: This alarm is used to detect whether the Aurora writer DB instance is in downtime. This can prevent long-running failure in the writer instance that occurs because of a crash or failover.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 60 seconds, and often less than 30 seconds.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

RollbackSegmentHistoryListLength

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a consistent high rollback segment history length of an Aurora instance. A high InnoDB history list length indicates that a large number of

old row versions, queries and database shutdowns have become slower. For more information and troubleshooting, see [the InnoDB history list length increased significantly](#) documentation.

Intent: This alarm is used to detect consistent high rollback segment history length. This can help you prevent sustained performance degradation and high CPU usage in the Aurora instance. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: 1000000.0

Threshold justification: Setting this threshold to 1 million should give you time to investigate the problem. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

StorageNetworkThroughput

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor high storage network throughput. If storage network throughput passes the total network bandwidth of the [EC2 instance](#), it can lead to high read and write latency, which can cause degraded performance. You can check your EC2 instance type from AWS Console. For troubleshooting, check any changes on write/read latencies and evaluate if you've also hit an alarm on this metric. If that is the case, evaluate your workload pattern during the times that the alarm was triggered. This can help you identify if you can optimize your workload to reduce the total amount of network traffic. If this is not possible, you might need to consider scaling your instance.

Intent: This alarm is used to detect high storage network throughput. Detecting high throughput can prevent network packet drops and degraded performance.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 80%-90% of the total network bandwidth of the EC2 instance type, and then use that result as the threshold value to proactively take action before the network packets are affected. You can also review the criticality and requirements of storage network throughput and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Route 53 Public Data Plane

HealthCheckStatus

Dimensions: HealthCheckId

Alarm description: This alarm helps to detect unhealthy endpoints as per health checkers. To understand the reason for a failure that results in unhealthy status, use the Health Checkers tab in the Route 53 Health Check Console to view the status from each Region as well as the last failure of the health check. The status tab also displays the reason that the endpoint is reported as unhealthy. Refer to [troubleshooting steps](#).

Intent: This alarm uses Route53 health checkers to detect unhealthy endpoints.

Statistic: Average

Recommended threshold: 1.0

Threshold justification: The status of the endpoint is reported as 1 when it's healthy. Everything less than 1 is unhealthy.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Amazon S3

4xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps us report the total number of 4xx error status codes that are made in response to client requests. 403 error codes might indicate an incorrect IAM policy, and 404 error codes might indicate mis-behaving client application, for example. [Enabling S3 server access logging](#) on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code. To understand more about the error code, see [Error Responses](#).

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The recommended threshold is to detect if more than 5% of total requests are getting 4XX errors. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps you detect a high number of server-side errors. These errors indicate that a client made a request that the server couldn't complete. This can help you correlate the issue your application is facing because of S3. For more information to help you

efficiently handle or reduce errors, see [Optimizing performance design patterns](#). Errors might also be caused by an the issue with S3, check [AWS service health dashboard](#) for the status of Amazon S3 in your Region.

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XXError. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

OperationsFailedReplication

Dimensions: SourceBucket, DestinationBucket, RuleId

Alarm description: This alarm helps in understanding a replication failure. This metric tracks the status of new objects replicated using S3 CRR or S3 SRR, and also tracks existing objects replicated using S3 batch replication. See [Replication troubleshooting](#) for more details.

Intent: This alarm is used to detect if there is a failed replication operation.

Statistic: Maximum

Recommended threshold: 0.0

Threshold justification: This metric emits a value of 0 for successful operations, and nothing when there are no replication operations carried out for the minute. When the metric emits a value greater than 0, the replication operation is unsuccessful.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

S3ObjectLambda

4xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps us report the total number of 4xx error status code that are made in response to client requests. [Enabling S3 server access logging](#) on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code.

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps to detect high number of server-side errors. These errors indicate that a client made a request that the server couldn't complete. These errors might be

caused by an issue with S3, check [AWS service health dashboard](#) for the status of Amazon S3 in your Region. This can help you correlate the issue your application is facing because of S3. For information to help you efficiently handle or reduce these errors, see [Optimizing performance design patterns](#).

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

LambdaResponse4xx

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps you detect and diagnose failures (500s) in calls to S3 Object Lambda. These errors can be caused by errors or misconfigurations in the Lambda function responsible for responding to your requests. Investigating the CloudWatch Log Streams of the Lambda function associated with the Object Lambda Access Point can help you pinpoint the issue's origin based on the response from S3 Object Lambda.

Intent: This alarm is used to detect 4xx client errors for WriteGetObjectResponse calls.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed.

However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SNS

NumberOfMessagesPublished

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages published is too low. For troubleshooting, check why the publishers are sending less traffic.

Intent: This alarm helps you proactively monitor and detect significant drops in notification publishing. This helps you identify potential issues with your application or business processes, so that you can take appropriate actions to maintain the expected flow of notifications. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages published should be in line with the expected number of published messages for your application. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsDelivered

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages delivered is too low. This could be because of unintentional unsubscribing of an endpoint, or because of an SNS event that causes messages to experience delay.

Intent: This alarm helps you detect a drop in the volume of messages delivered. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages delivered should be in line with the expected number of messages produced and the number of consumers. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsFailed

Dimensions: TopicName

Alarm description: This alarm can detect when the number of failed SNS messages is too high. To troubleshoot failed notifications, enable logging to CloudWatch Logs. Checking the logs can help you find which subscribers are failing, as well as the status codes they are returning.

Intent: This alarm helps you proactively find issues with the delivery of notifications and take appropriate actions to address them.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the impact of failed notifications. Review the SLAs provided to your end users, fault

tolerance and criticality of notifications and analyze historical data, and then select a threshold accordingly. The number of notifications failed should be 0 for topics that have only SQS, Lambda or Firehose subscriptions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidAttributes

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid attributes or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid attributes are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid attributes are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidMessageBody

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid message bodies, or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid message bodies are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid message bodies are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsRedrivenToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor the number of messages that are moved to a dead-letter queue.

Intent: The alarm is used to detect messages that moved to a dead-letter queue. We recommend that you create this alarm when SNS is coupled with SQS, Lambda or Firehose.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: In a healthy system of any subscriber type, messages should not be moved to the dead-letter queue. We recommend that you be notified if any messages land in the queue, so that you can identify and address the root cause, and potentially redrive the messages in the dead-letter queue to prevent data loss.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFailedToRedriveToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor messages that couldn't be moved to a dead-letter queue. Check whether your dead-letter queue exists and that it's configured correctly. Also, verify that SNS has permissions to access the dead-letter queue. Refer to the [dead-letter queue documentation](#) to learn more.

Intent: The alarm is used to detect messages that couldn't be moved to a dead-letter queue.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: It's almost always a mistake if messages can't be moved to the dead-letter queue. The recommendation for the threshold is 0, meaning all messages that fail processing must be able to reach the dead-letter queue when the queue has been configured.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSMonthToDateSpentUSD

Dimensions: TopicName

Alarm description: The alarm helps to monitor if you have a sufficient quota in your account for SNS to be able to deliver messages. If you reach your quota, SNS won't be able to deliver SMS messages. For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see [Setting SMS messaging preferences](#).

Intent: This alarm is used to detect if you have a sufficient quota in your account for your SMS messages to be delivered successfully.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold in accordance with the quota (Account spend limit) for the account. Choose a threshold which informs you early enough that you are reaching your quota limit so that you have time to request an increase.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSSuccessRate

Dimensions: TopicName

Alarm description: This alarm helps to monitor the rate of failing SMS message deliveries. You can set up [Cloudwatch Logs](#) to understand the nature of the failure and take action based on that.

Intent: This alarm is used to detect failing SMS message deliveries.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold for the alarm in line with your tolerance for failing SMS message deliveries.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SQS

ApproximateAgeOfOldestMessage

Dimensions: QueueName

Alarm description: This alarm watches the age of the oldest message in the queue. You can use this alarm to monitor if your consumers are processing SQS messages at the desired speed. Consider increasing the consumer count or consumer throughput to reduce message age. This metric can be used in combination with `ApproximateNumberOfMessagesVisible` to determine how big the queue backlog is and how quickly messages are being processed. To prevent messages from being deleted before processed, consider configuring the dead-letter queue to sideline potential poison pill messages.

Intent: This alarm is used to detect whether the age of the oldest message in the QueueName queue is too high. High age can be an indication that messages are not processed quickly enough or that there are some poison-pill messages that are stuck in the queue and can't be processed.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected message processing time. You can use historical data to calculate the average message processing time, and then set the threshold to 50% higher than the maximum expected SQS message processing time by queue consumers.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesNotVisible

Dimensions: QueueName

Alarm description: This alarm helps to detect a high number of in-flight messages with respect to QueueName. For troubleshooting, check [message backlog decreasing](#).

Intent: This alarm is used to detect a high number of in-flight messages in the queue. If consumers do not delete messages within the visibility timeout period, when the queue is polled, messages reappear in the queue. For FIFO queues, there can be a maximum of 20,000 in-flight messages. If you reach this quota, SQS returns no error messages. A FIFO queue looks through the first 20k messages to determine available message groups. This means that if you have a backlog of messages in a single message group, you cannot consume messages from other message groups that were sent to the queue at a later time until you successfully consume the messages from the backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected number of messages in flight. You can use historical data to calculate the maximum expected number of messages in flight and set the threshold to 50% over this value. If consumers of the queue are processing but not deleting messages from the queue, this number will suddenly increase.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesVisible

Dimensions: QueueName

Alarm description: This alarm watches for the message queue backlog to be bigger than expected, indicating that consumers are too slow or there are not enough consumers. Consider increasing the consumer count or speeding up consumers, if this alarm goes into ALARM state.

Intent: This alarm is used to detect whether the message count of the active queue is too high and consumers are slow to process the messages or there are not enough consumers to process them.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: An unexpectedly high number of messages visible indicates that messages are not being processed by a consumer at the expected rate. You should consider historical data when you set this threshold.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

NumberOfMessagesSent

Dimensions: QueueName

Alarm description: This alarm helps to detect if there are no messages being sent from a producer with respect to QueueName. For troubleshooting, check the reason that the producer is not sending messages.

Intent: This alarm is used to detect when a producer stops sending messages.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the number of messages sent is 0, the producer is not sending any messages. If this queue has a low TPS, increase the number of EvaluationPeriods accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

AWS VPN

TunnelState

Dimensions: VpnId

Alarm description: This alarm helps you understand if the state of one or more tunnels is DOWN. For troubleshooting, see [VPN tunnel troubleshooting](#).

Intent: This alarm is used to detect if at least one tunnel is in the DOWN state for this VPN, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that at least one tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

TunnelState

Dimensions: TunnelIpAddress

Alarm description: This alarm helps you understand if the state of this tunnel is DOWN. For troubleshooting, see [VPN tunnel troubleshooting](#).

Intent: This alarm is used to detect if the tunnel is in the DOWN state, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that the tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Alarming on metrics

The steps in the following sections explain how to create CloudWatch alarms on metrics.

Create a CloudWatch alarm based on a static threshold

You choose a CloudWatch metric for the alarm to watch, and the threshold for that metric. The alarm goes to ALARM state when the metric breaches the threshold for a specified number of evaluation periods.

If you are creating an alarm in an account set up as a monitoring account in CloudWatch cross-account observability, you can set up the alarm to watch a metric in a source account linked to this monitoring account. For more information, see [CloudWatch cross-account observability](#).

To create an alarm based on a single metric

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
 - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
 - In the search box, enter the name of a metric, account ID, account label, dimension, or resource ID. Then, choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.
6. Choose the **Graphed metrics** tab.
 - a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
 - b. Under **Period**, choose the evaluation period for the alarm. When evaluating the alarm, each period is aggregated into one data point.

You can also choose whether the y-axis legend appears on the left or right while you're creating the alarm. This preference is used only while you're creating the alarm.

- c. Choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.

7. Under **Conditions**, specify the following:

- a. For **Whenever *metric* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
- b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

- c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
- d. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples](#).

8. Choose **Next**.

9. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

In CloudWatch cross-account observability, you can choose to have notifications sent to multiple AWS accounts. For example, to both the monitoring account and the source account.

To have the alarm not send notifications, choose **Remove**.

10. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about

Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

 **Note**

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

11. When finished, choose **Next**.
12. Enter a name and description for the alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources. Then choose **Next**.
13. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see [Add or remove an alarm widget from a CloudWatch dashboard](#).

Create a CloudWatch alarm based on a metric math expression

To create an alarm based on a metric math expression, choose one or more CloudWatch metrics to use in the expression. Then, specify the expression, threshold, and evaluation periods.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To create an alarm that's based on a metric math expression

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**, and then perform one of the following actions:

- Select a namespace from the **AWS namespaces** dropdown or **Custom namespaces** dropdown. After you select a namespace, you continue choosing options until a list of metrics appears, where you select the checkbox next to the correct metric.
 - Use the search box to find a metric, account ID, dimension, or resource ID. After you enter the metric, dimension, or resource ID, you continue choosing options until a list of metrics appears, where you select the check box next to the correct metric.
5. (Optional) If you want to add another metric to a metric math expression, you can use the search box to find a specific metric. You can add as many as 10 metrics to a metric math expression.
 6. Select the **Graphed metrics** tab. For each of the metrics that you previously added, perform the following actions:
 - a. Under the **Statistic** column, select the dropdown menu. In the dropdown menu, choose one of the predefined statistics or percentiles. Use the search box in the dropdown menu to specify a custom percentile.
 - b. Under the **Period** column, select the dropdown menu. In the dropdown menu, choose one of the predefined evaluation periods.

While you're creating your alarm, you can specify whether the Y-axis legend appears on the left or right side of your graph.

 **Note**

When CloudWatch evaluates alarms, periods are aggregated into single data points.

7. Choose the **Add math** dropdown, and then select **Start with an empty expression** from the list of predefined metric math expressions.

After you choose **Start with an empty expression**, a math expression box appears where you apply or edit math expressions.

8. In the math expression box, enter your math expression, and then choose **Apply**.

After you choose **Apply**, an **ID** column appears next to the **Label** column.

To use a metric or the result of another metric math expression as part of your current math expression's formula, you use the value that's shown under the **ID** column. To change the value of **ID**, you select the pen-and-paper icon next to the current value. The new value must begin

with a lowercase letter and can include numbers, letters, and the underscore symbol. Changing the value of **ID** to a more significant name can make your alarm graph easier to understand.

For information about the functions that are available for metric math, see [Metric math syntax and functions](#).

9. (Optional) Add more math expressions, using both metrics and the results of other math expressions in the formulas of the new math expressions.
10. When you have the expression to use for the alarm, clear the check boxes to the left of every other expression and every metric on the page. Only the check box next to the expression to use for the alarm should be selected. The expression that you choose for the alarm must produce a single time series and show only one line on the graph. Then choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

11. For **Whenever *expression* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
12. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

13. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
14. Choose **Next**.
15. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

16. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a

Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

Note

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

17. When finished, choose **Next**.
18. Enter a name and description for the alarm. Then choose **Next**.

The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

19. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see [Add or remove an alarm widget from a CloudWatch dashboard](#).

Create a CloudWatch alarm based on a Metrics Insights query

You can create an alarm on any Metrics Insights query that returns a single time series. This can be especially useful to create dynamic alarms that watch aggregated metrics across a fleet of your infrastructure or applications. Create the alarm once, and it adjusts as resources are added to or removed from the fleet. For example, you can create an alarm that watches the CPU utilization of all your instances, and the alarm dynamically adjusts as you add or remove instances.

For complete instructions, see [Create alarms on Metrics Insights queries](#).

Create an alarm based on a connected data source

You can create alarms that watch metrics from data sources that aren't in CloudWatch. For more information about creating connections to these other data sources, see [Query metrics from other data sources](#).

To create an alarm on metrics from a data source that you have connected to

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics, All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.
5. The query builder prompts you for the information necessary for the query to retrieve the metrics to use for the alarm. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus and Prometheus data sources, a PromQL query editor box with a query helper appears.
6. When you have finished constructing the query, choose **Graph query**.
7. If the sample graph looks the way that you expect, choose **Create alarm**.
8. The **Specify metric and conditions** page appears. If the query you are using produces more than one time series, you see a warning banner at the top of the page. If you do, select a function to use to aggregate the time series in **Aggregation function**.
9. (Optional) Add a **Label** for the alarm.
10. For **Whenever *your-metric-name* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal**, or **Lower**. Then for **than . . .**, specify a number for your threshold value.
11. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see [Evaluating an alarm](#).

12. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
13. Choose **Next**.


14. For **Notification**, specify an Amazon SNS topic to notify when your alarm transitions to the ALARM, OK, or INSUFFICIENT_DATA state.
 - a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

 **Note**

We recommend that you set the alarm to take actions when it goes into **Insufficient data** state in addition to when it goes into **Alarm** state. This is because many issues with the Lambda function that connects to the data source can cause the alarm to transition to **Insufficient data**.

- b. (Optional) To not send Amazon SNS notifications, choose **Remove**.
15. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

 **Note**

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

16. Choose **Next**.
17. Under **Name and description**, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

Tip

The alarm name must contain only UTF-8 characters. It can't contain ASCII control characters.

18. Under **Preview and create**, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Details about alarms for connected data sources

- When CloudWatch evaluates an alarm, it does so every minute, even if the period for the alarm is longer than one minute. For the alarm to work, the Lambda function must be able to return a list of timestamps starting on any minute, not only on multiples of the period length. These timestamps must be spaced one period length apart.

Therefore, if the data source queried by the Lambda can only return timestamps that are multiples of the period length, the function should "re-sample" the fetched data to match the timestamps expected by the `GetMetricData` request.

For example, an alarm with a five-minute period is evaluated every minute using five-minute windows that shift by one minute each time. In this case:

- For the alarm evaluation at 12:15:00, CloudWatch expects data points with timestamps of 12:00:00, 12:05:00, and 12:10:00.
- Then for the alarm evaluation at 12:16:00, CloudWatch expects data points with timestamps of 12:01:00, 12:06:00, and 12:11:00.
- When CloudWatch evaluates an alarm, any data points returned by the Lambda function that don't align with the expected timestamps are dropped, and the alarm is evaluated using the remaining expected data points. For example, when the alarm is evaluated at 12:15:00 it expects data with timestamps of 12:00:00, 12:05:00, and 12:10:00. If it receives data with timestamps of 12:00:00, 12:05:00, 12:06:00, and 12:10:00, the data from 12:06:00 is dropped and CloudWatch evaluates the alarm using the other timestamps.

Then for the next evaluation at 12:16:00, it expects data with timestamps of 12:01:00, 12:06:00, and 12:11:00. If it only has the data with timestamps of 12:00:00, 12:05:00, and 12:10:00, all of these data points are ignored at 12:16:00 and the alarm transitions into the

state according to how you specified the alarm to treat missing data. For more information, see [Evaluating an alarm](#).

- We recommend that you create these alarms to take actions when they transition to the `INSUFFICIENT_DATA` state, because several Lambda function failure use cases will transition the alarm to `INSUFFICIENT_DATA` regardless of the way that you set the alarm to treat missing data.
- If the Lambda function returns an error or returns partial data:
 - If there is a permission problem with calling the Lambda function, the alarm begins having missing data transitions according to how you specified the alarm to treat missing data when you created it.
 - If the Lambda function returns `'StatusCode' = 'PartialData'`, then alarm evaluation fails, and the alarm transitions to `INSUFFICIENT_DATA` after three attempts. This takes about three minutes.
 - Any other error coming from the Lambda function causes the alarm to transition to `INSUFFICIENT_DATA`.
- If the metric requested by the Lambda function has some delay so that the last data point is always missing, you should use a workaround. You can create an M out of N alarm or increase the evaluation period of the alarm. For more information about M out of N alarms, see [Evaluating an alarm](#).

Create a CloudWatch alarm based on anomaly detection

You can create an alarm based on CloudWatch anomaly detection, which analyzes past metric data and creates a model of expected values. The expected values take into account the typical hourly, daily, and weekly patterns in the metric.

You set a value for the anomaly detection threshold, and CloudWatch uses this threshold with the model to determine the "normal" range of values for the metric. A higher value for the threshold produces a thicker band of "normal" values.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or either above or below the band.

You also can create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands.

In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

For more information, see [Using CloudWatch anomaly detection](#).

Note

If you're already using anomaly detection for visualization purposes on a metric in the Metrics console and you create an anomaly detection alarm on that same metric, then the threshold that you set for the alarm doesn't change the threshold that you already set for visualization. For more information, see [Creating a graph](#).

To create an alarm that's based on anomaly detection

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
 - Choose the service namespace that contains your metric, and then continue choosing options as they appear to narrow down your options. When a list of metrics appears, select the check box that's next to your metric.
 - In the search box, enter the name of a metric, dimension, or resource ID. Select one of the results, and then continue choosing options as they appear until a list of metrics appears. Select the check box that's next to your metric.
6. Choose **Graphed metric**.
 - a. (Optional) For *Statistic*, choose the dropdown, and then select one of the predefined statistics or percentiles. You can use the search box in the dropdown to specify a custom percentile, such as **p95.45**.
 - b. (Optional) For *Period*, choose the dropdown, and then select one of the predefined evaluation periods.

Note

When CloudWatch evaluates your alarm, it aggregates the period into a single datapoint. For an anomaly detection alarm, the evaluation period must be one minute or longer.

7. Choose **Next**.
8. Under **Conditions**, specify the following:
 - a. Choose **Anomaly detection**.

If the model for this metric and statistic already exists, CloudWatch displays a preview of the anomaly detection band in the graph at the top of the screen. After you create your alarm, it can take up to 15 minutes for the actual anomaly detection band to appear in the graph. Before that, the band that you see is an approximation of the anomaly detection band.

Tip

To see the graph at the top of the screen in a longer time frame, choose **Edit** at the top-right of the screen.

If the model for this metric and statistic doesn't already exist, CloudWatch generates the anomaly detection band after you finish creating your alarm. For new models, it can take up to 3 hours for the actual anomaly detection band to appear in your graph. It can take up to two weeks for the new model to train, so the anomaly detection band shows more accurate expected values.

- b. For **Whenever *metric* is**, specify when to trigger the alarm. For example, when the metric is greater than, lower than, or outside the band (in either direction).
- c. For **Anomaly detection threshold**, choose the number to use for the anomaly detection threshold. A higher number creates a thicker band of "normal" values that is more tolerant of metric changes. A lower number creates a thinner band that will go to ALARM state with smaller metric deviations. The number does not have to be a whole number.
- d. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values

here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see [Evaluating an alarm](#).

- e. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
 - f. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **Ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples](#).
9. Choose **Next**.
 10. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.


To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

Choose **Remove** if you don't want the alarm to send notifications.

11. You can set up the alarm to perform EC2 actions or invoke a Lambda function when it changes state, or to create a Systems Manager OpsItem or incident when it goes into ALARM state. To do this, choose the appropriate button and then choose the alarm state and action to perform.

If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

 **Note**

To create an alarm that performs an AWS Systems Manager Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

12. Choose **Next**.

13. Under **Name and description**, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

 **Tip**

The alarm name must contain only UTF-8 characters, and can't contain ASCII control characters

14. Under **Preview and create**, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Modifying an anomaly detection model

After you create an alarm, you can adjust the anomaly detection model. You can exclude certain time periods from being used in the model creation. It is critical that you exclude unusual events such as system outages, deployments, and holidays from the training data. You can also specify whether to adjust the model for Daylight Savings Time changes.

To adjust the anomaly detection model for an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose the name of the alarm. If necessary, use the search box to find the alarm.
4. Choose **Analyse, In metrics**.
5. In the **Details** column, choose **ANOMALY_DETECTION_BAND, Edit anomaly detection model**.
6. To exclude a time period from being used to produce the model, choose the calendar icon by **End date**. Then, select or enter the days and times to exclude from training and choose **Apply**.
7. If the metric is sensitive to Daylight Savings Time changes, select the appropriate time zone in the **Metric timezone** box.
8. Choose **Update**.

Deleting an anomaly detection model

Using anomaly detection for an alarm accrues charges. As a best practice, if your alarm no longer needs an anomaly detection model, delete the alarm first and the model second. When anomaly detection alarms are evaluated, any missing anomaly detectors are created on your behalf. If you delete the model without deleting the alarm, the alarm automatically recreates the model.

To delete an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose the name of the alarm.
4. Choose **Actions, Delete**.
5. In the confirmation box, choose **Delete**.

To delete an anomaly detection model that was used for an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
3. Choose **Browse**, and then select the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.
 - (Optional) If you're using the original interface, choose **All metrics**, and then choose the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.
4. Choose **Graphed metrics**.
5. In the **Graphed metrics** tab, choose the name of the anomaly detection model that you want to remove, and choose **Delete anomaly detection model**.
 - (Optional) If you're using the original interface, choose **Edit model**. You're directed to a new screen. On the new screen, choose **Delete model**, and then choose **Delete**.

Alarming on logs

The steps in the following sections explain how to create CloudWatch alarms on logs.

Create a CloudWatch alarm based on a log group-metric filter

The procedure in this section describes how to create an alarm based on a log group-metric filter. With metric filters, you can look for terms and patterns in log data as the data is sent to CloudWatch. For more information, see [Create metrics from log events using filters](#) in the *Amazon CloudWatch Logs User Guide*. Before you create an alarm based on a log group-metric filter, you must complete the following actions:

- Create a log group. For more information, see [Working with log groups and log streams](#) in the *Amazon CloudWatch Logs User Guide*.
- Create a metric filter. For more information, see [Create a metric filter for a log group](#) in the *Amazon CloudWatch Logs User Guide*.


To create an alarm based on a log group-metric filter

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Choose the log group that includes your metric filter.
4. Choose **Metric filters**.
5. In the metric filters tab, select the box for the metric filter that you want to base your alarm on.
6. Choose **Create alarm**.
7. (Optional) Under **Metric**, edit **Metric name**, **Statistic**, and **Period**.
8. Under **Conditions**, specify the following:
 - a. For **Threshold type**, choose **Static** or **Anomaly detection**.
 - b. For **Whenever *your-metric-name* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal**, or **Lower**.
 - c. For **than . . .**, specify a number for your threshold value.
9. Choose **Additional configuration**.
 - a. For **Data points to alarm**, specify how many data points trigger your alarm to go into the ALARM state. If you specify matching values, your alarm goes into the ALARM state if that many consecutive periods are breaching. To create an M-out-of-N alarm, specify a number

for the first value that's lower than the number you specify for the second value. For more information, see [Using Amazon CloudWatch alarms](#).

- b. For **Missing data treatment**, select an option to specify how to treat missing data when your alarm is evaluated.
10. Choose **Next**.
 11. For **Notification**, specify an Amazon SNS topic to notify when your alarm is in the ALARM, OK, or INSUFFICIENT_DATA state.
 - a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.
 - b. (Optional) To not send notifications, choose **Remove**.
 12. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

 **Note**

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

13. Choose **Next**.
14. For **Name and description**, enter a name and description for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
15. For **Preview and create**, check that your configuration is correct, and choose **Create alarm**.

Combining alarms

With CloudWatch, you can combine several alarms into one *composite alarm* to create a summarized, aggregated health indicator over a whole application or group of resources. Composite alarms are alarms that determine their state by monitoring the states of other alarms. You define rules to combine the status of those monitored alarms using Boolean logic.

You can use composite alarms to reduce alarm noise by taking actions only at an aggregated level. For example, you can create a composite alarm to send a notification to your web server team if any alarm related to your web server triggers. When any of those alarms goes into the ALARM state, the composite alarm goes itself in the ALARM state and sends a notification to your team. If other alarms related to your web server also go into the ALARM state, your team does not get overloaded with new notifications since the composite alarm has already notified them about the existing situation.

You can also use composite alarms to create complex alarming conditions and take actions only when many different conditions are met. For example, you can create a composite alarm that combines a CPU alarm and a memory alarm, and would only notify your team if both the CPU and the memory alarms have triggered.

Using composite alarms

When you use composite alarms, you have two options:

- Configure the actions you want to take only at the composite alarm level, and create the underlying monitored alarms without actions
- Configure a different set of actions at the composite alarm level. For example, the composite alarm actions could engage a different team in case of a widespread issue.

Composite alarms can take only the following actions:

- Notify Amazon SNS topics
- Invoke Lambda functions
- Create OpsItems in Systems Manager Ops Center
- Create incidents in Systems Manager Incident Manager

Note

All the underlying alarms in your composite alarm must be in the same account and the same Region as your composite alarm. However, if you set up a composite alarm in a CloudWatch cross-account observability monitoring account, the underlying alarms can watch metrics in different source accounts and in the monitoring account itself. For more information, see [CloudWatch cross-account observability](#).

A single composite alarm can monitor 100 underlying alarms, and 150 composite alarms can monitor a single underlying alarm.

Rule expressions

All composite alarms contain rule expressions. Rule expressions tell composite alarms which other alarms to monitor and determine their states from. Rule expressions can refer to metric alarms and composite alarms. When you reference an alarm in a rule expression, you designate a function to the alarm that determines which of the following three states the alarm will be in:

- ALARM

ALARM ("alarm-name or alarm-ARN") is TRUE if the alarm is in ALARM state.

- OK

OK ("alarm-name or alarm-ARN") is TRUE if the alarm is in OK state.

- INSUFFICIENT_DATA

INSUFFICIENT_DATA ("alarm-name or alarm-ARN") is TRUE if the named alarm is in INSUFFICIENT_DATA state.

Note

TRUE always evaluates to TRUE, and FALSE always evaluates to FALSE.

Example expressions

The request parameter `AlarmRule` supports the use of the logical operators AND, OR, and NOT, so you can combine multiple functions into a single expressions. The following example expressions show how you can configure the underlying alarms in your composite alarm:

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)`

The expression specifies that the composite alarm goes into ALARM only if `CPUUtilizationTooHigh` and `DiskReadOpsTooHigh` are in ALARM.

- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)`

The expression specifies that the composite alarm goes into ALARM if `CPUUtilizationTooHigh` is in ALARM and `DeploymentInProgress` is not in ALARM. This is an example of a composite alarm that reduces alarm noise during a deployment window.

- `(ALARM(CPUUtilizationTooHigh) OR ALARM(DiskReadOpsTooHigh)) AND OK(NetworkOutTooHigh)`

The expression specifies that the composite alarm goes into ALARM if `(ALARM(CPUUtilizationTooHigh) or (DiskReadOpsTooHigh))` is in ALARM and `(NetworkOutTooHigh)` is in OK. This is an example of a composite alarm that reduces alarm noise by not sending you notifications when either of the underlying alarms aren't in ALARM while a network issue is occurring.

Topics

- [Create a composite alarm](#)
- [Suppressing composite alarm actions](#)


Create a composite alarm

The steps in this section explain how to use the CloudWatch console to create a composite alarm. You can also use the API or AWS CLI to create a composite alarm. For more information, see [PutCompositeAlarm](#) or [put-composite-alarm](#)

To create a composite alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.

3. From the list of alarms, select the check box next to each of the existing alarms that you want to reference in your rule expression, and then choose **Create composite alarm**.
4. Under **Specify composite alarm conditions**, specify the rule expression for your new composite alarm.

 **Note**

Automatically, the alarms that you selected from the list of alarms are listed in the **Conditions** box. By default, the ALARM function has been designated to each of your alarms, and each of your alarms is joined by the logical operator OR.

You can use the following substeps to modify your rule expression:

- a. You can change the required state for each of your alarms from ALARM to OK or INSUFFICIENT_DATA.
- b. You can change the logical operator in your rule expression from OR to AND or NOT, and you can add parentheses to group your functions.
- c. You can include other alarms in your rule expression or delete alarms from your rule expression.

Example: Rule expression with conditions

```
(ALARM("CPUUtilizationTooHigh") OR  
ALARM("DiskReadOpsTooHigh")) AND  
OK("NetworkOutTooHigh")
```

In the example rule expression where the composite alarm goes into ALARM when ALARM ("CPUUtilizationTooHigh" or ALARM("DiskReadOpsTooHigh")) is in ALARM at the same time as OK("NetworkOutTooHigh") is in OK.

5. When finished, choose **Next**.
6. Under **Configure actions**, you can choose from the following:

For **Notification**

- **Select an existing SNS topic, Create a new SNS topic, or Use a topic ARN** to define the SNS topic that will receive the notification.

- **Add notification**, so your alarm can send multiple notifications for the same alarm state or different alarm states.
- **Remove** to stop your alarm from sending notifications or taking actions.

(Optional) To have the alarm invoke a Lambda function when it changes state, choose **Add Lambda action**. Then specify the function name or ARN, and optionally choose a specific version of the function.

For *Systems Manager action*

- **Add Systems Manager action**, so your alarm can perform an SSM action when it goes into ALARM.

To learn more about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) in the *AWS Systems Manager User Guide* and [Incident creation](#) in the *Incident Manager User Guide*. To create an alarm that performs an SSM Incident Manager action, you must have the correct permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#) in the *Incident Manager User Guide*.

7. When finished, choose **Next**.
8. Under **Add name and description**, enter an alarm name and *optional* description for your new composite alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
9. When finished, choose **Next**.
10. Under **Preview and create**, confirm your information, and then choose **Create composite alarm**.

Note

You can create a cycle of composite alarms, where one composite alarm and another composite alarm depend on each other. If you find yourself in this scenario, your composite alarms stop being evaluated, and you can't delete your composite alarms because they're dependent on each other. The easiest way to break the cycle of

dependency between your composite alarms is to change the function `AlarmRule` in one of your composite alarms to `False`.

Suppressing composite alarm actions

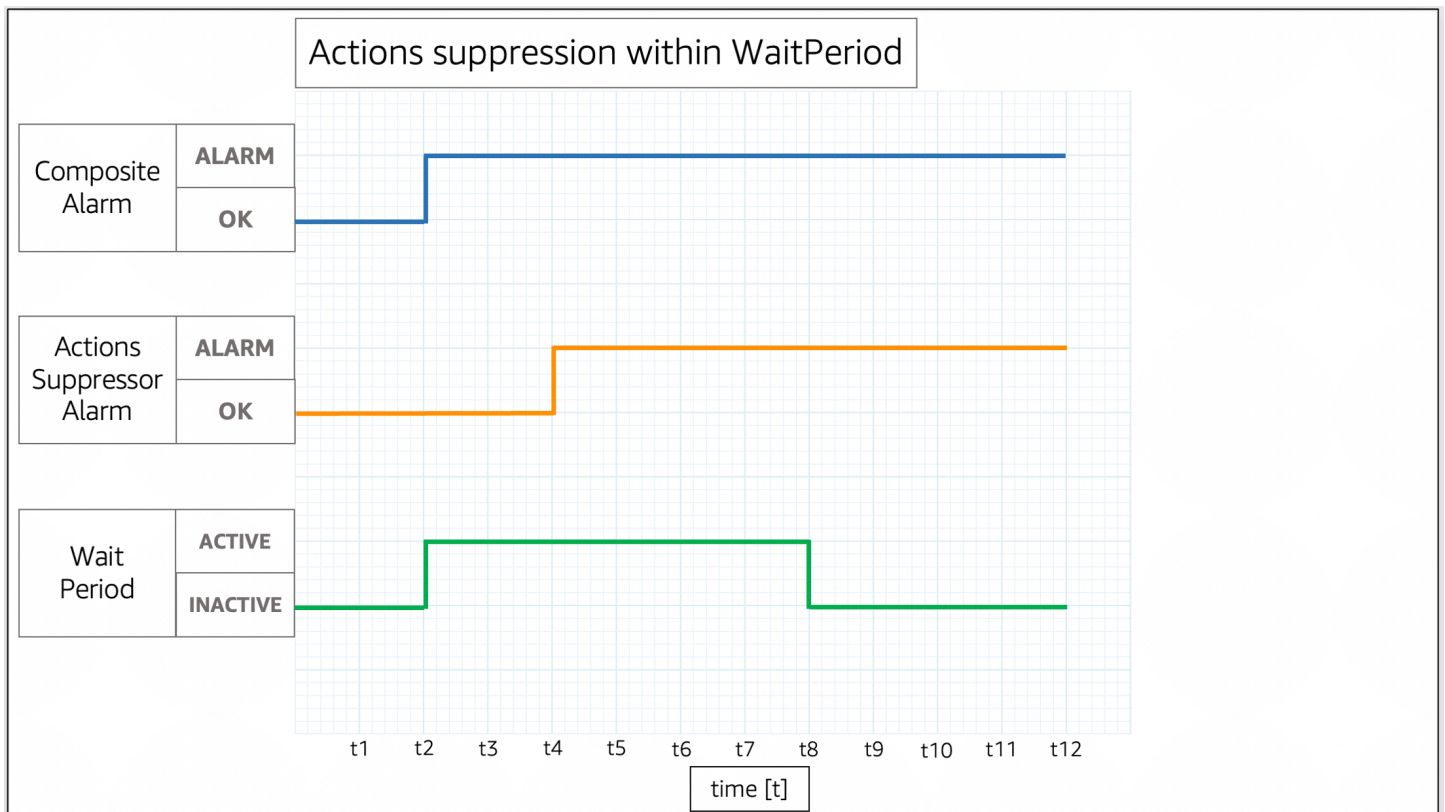
Because composite alarms allow you to get an aggregated view of your health across multiple alarms, there are common situations where it is expected for those alarms to trigger. For example, during a maintenance window of your application or when you investigate an ongoing incident. In such situations, you may want to suppress the actions of your composite alarms, to prevent unwanted notifications or the creation of new incident tickets.

With composite alarm action suppression, you define alarms as suppressor alarms. Suppressor alarms prevent composite alarms from taking actions. For example, you can specify a suppressor alarm that represents the status of a supporting resource. If the supporting resource is down, the suppressor alarm prevents the composite alarm from sending notifications. Composite alarm action suppression helps you reduce alarm noise, so you spend less time managing your alarms and more time focusing on your operations.

You specify suppressor alarms when you configure composite alarms. Any alarm can function as a suppressor alarm. When a suppressor alarm changes states from OK to ALARM, its composite alarm stops taking actions. When a suppressor alarm changes states from ALARM to OK, its composite alarm resumes taking actions.

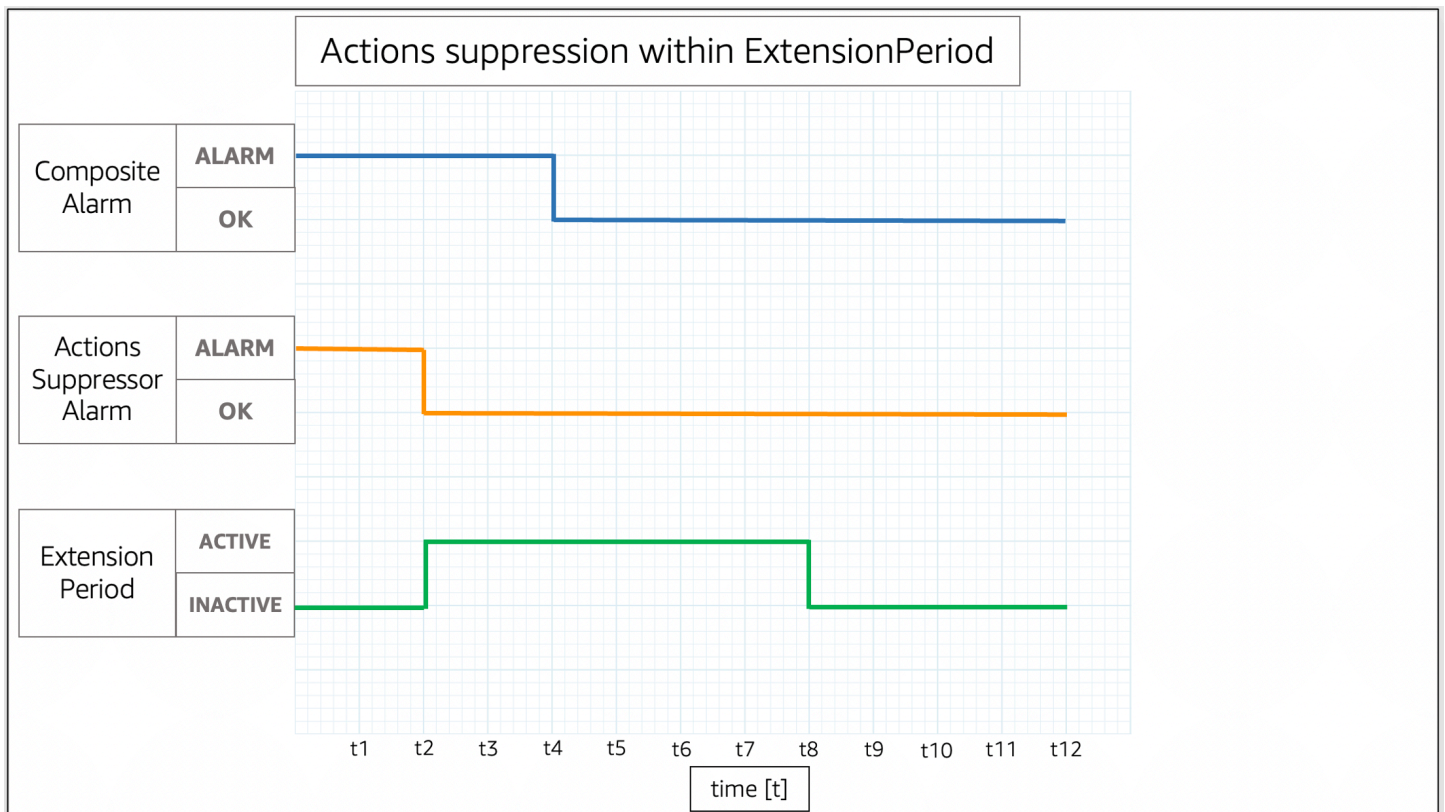
WaitPeriod and ExtensionPeriod

When you specify a suppressor alarm, you set the parameters `WaitPeriod` and `ExtensionPeriod`. These parameters prevent composite alarms from taking actions unexpectedly while suppressor alarms change states. Use `WaitPeriod` to compensate for any delays that can occur when a suppressor alarm changes from OK to ALARM. For example, if a suppressor alarm changes from OK to ALARM within 60 seconds, set `WaitPeriod` to 60 seconds.



In the image, the composite alarm changes from OK to ALARM at t2. A WaitPeriod starts at t2 and ends at t8. This gives the suppressor alarm time to change states from OK to ALARM at t4 before it suppresses the composite alarm's actions when the WaitPeriod expires at t8.

Use ExtensionPeriod to compensate for any delays that can occur when a composite alarm changes to OK following a suppressor alarm changing to OK. For example, if a composite alarm changes to OK within 60 seconds of a suppressor alarm changing to OK, set ExtensionPeriod to 60 seconds.



In the image, the suppressor alarm changes from ALARM to OK at t2. An ExtensionPeriod starts at t2 and ends at t8. This gives the composite alarm time to change from ALARM to OK before the ExtensionPeriod expires at t8.

Composite alarms don't take actions when WaitPeriod and ExtensionPeriod become active. Composite alarms take actions that are based on their current states when ExtensionPeriod and WaitPeriod become inactive. We recommend that you set the value for each parameter to 60 seconds, as CloudWatch evaluates metric alarms every minute. You can set the parameters to any integer in seconds.

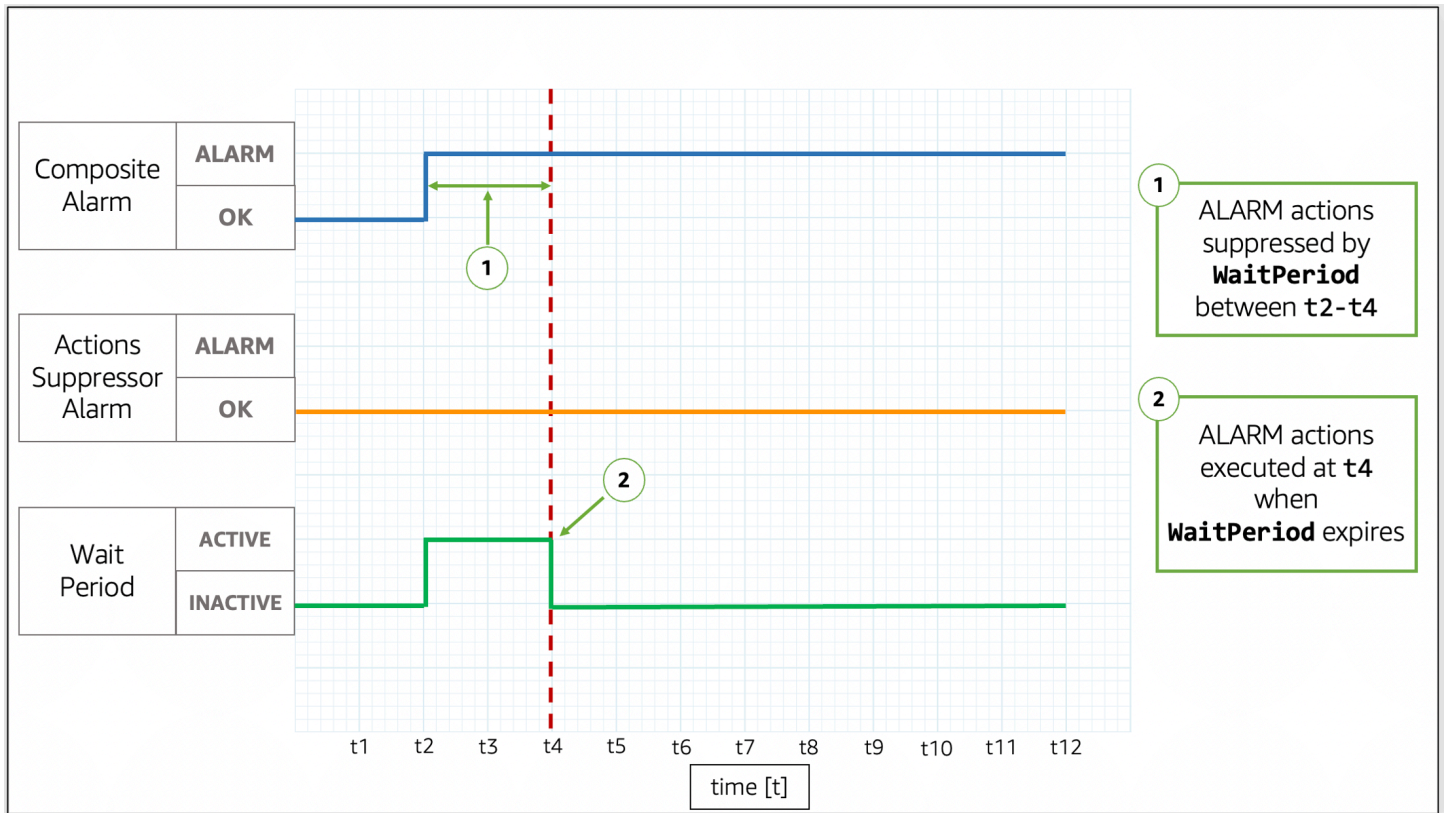
The following examples describe in more detail how WaitPeriod and ExtensionPeriod prevent composite alarms from taking actions unexpectedly.

Note

In the following examples, WaitPeriod is configured as 2 time units, and ExtensionPeriod is configured as 3 time units.

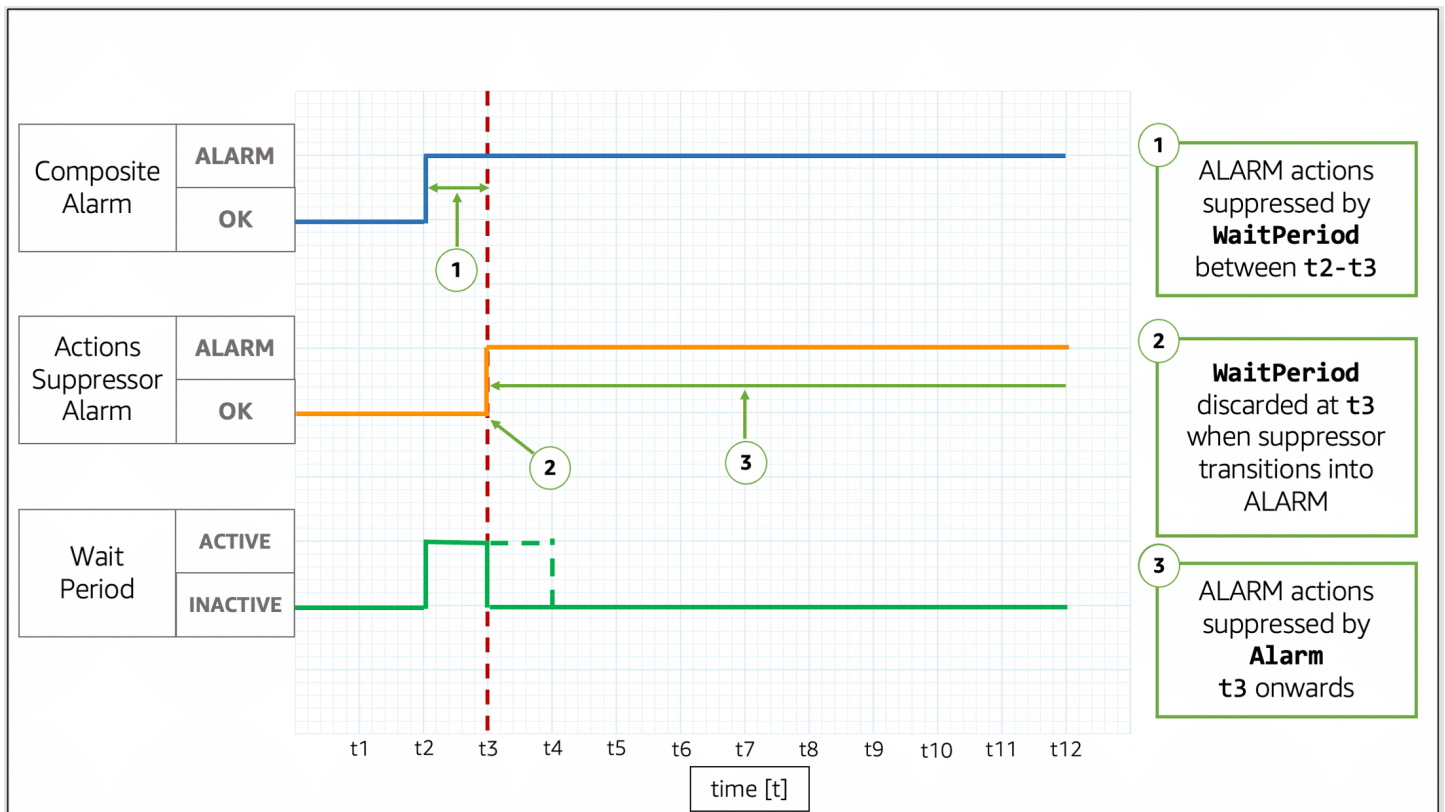
Examples

Example 1: Actions are not suppressed after WaitPeriod



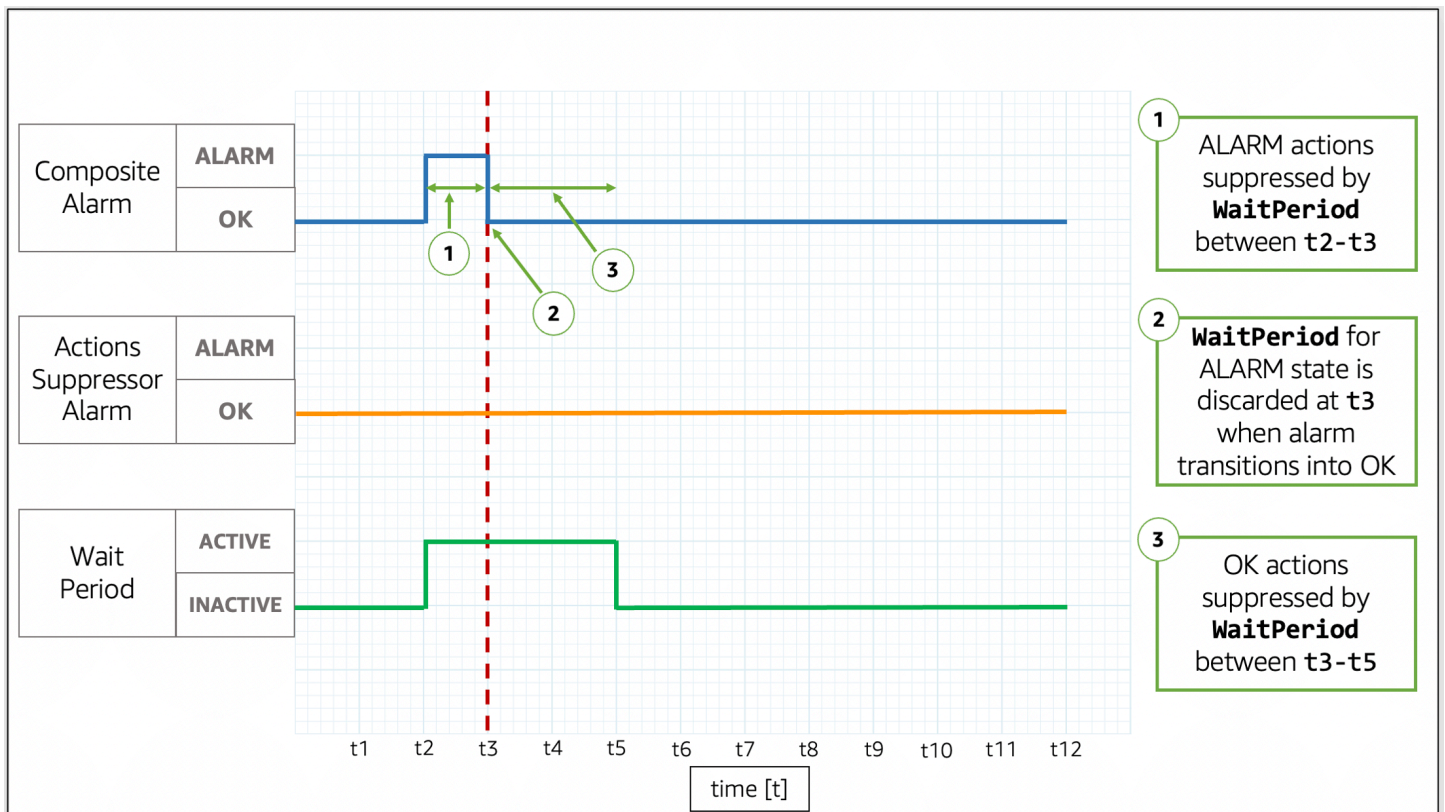
In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4, so it can prevent the composite alarm from taking actions. After the `WaitPeriod` expires at t4, the composite alarm takes its actions because the suppressor alarm is still in OK.

Example 2: Actions are suppressed by alarm before WaitPeriod expires



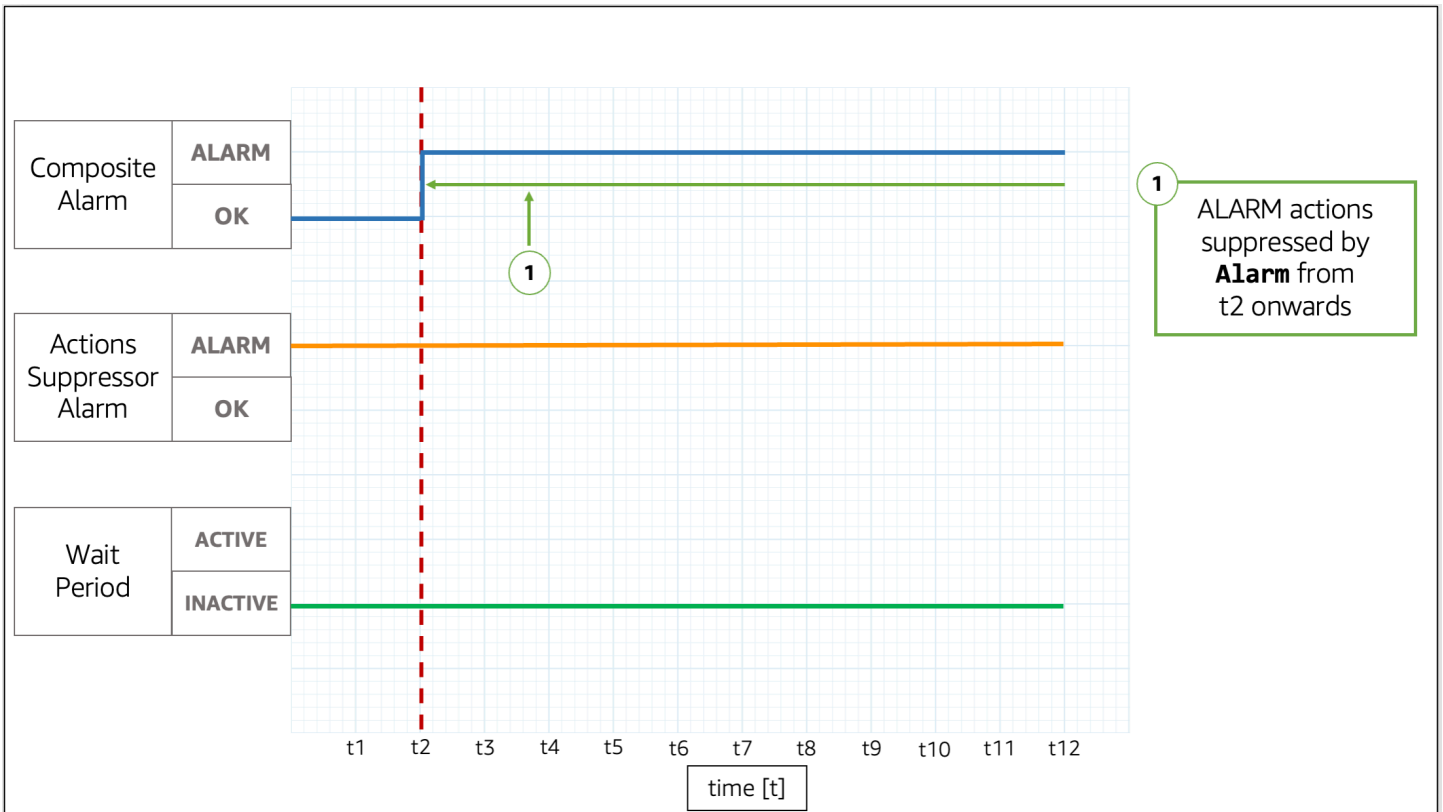
In the image, the composite alarm changes states from OK to ALARM at t_2 . A `WaitPeriod` starts at t_2 and ends at t_4 . This gives the suppressor alarm time to change states from OK to ALARM at t_3 . Because the suppressor alarm changes states from OK to ALARM at t_3 , the `WaitPeriod` that started at t_2 is discarded, and the suppressor alarm now stops the composite alarm from taking actions.

Example 3: State transition when actions are suppressed by `WaitPeriod`



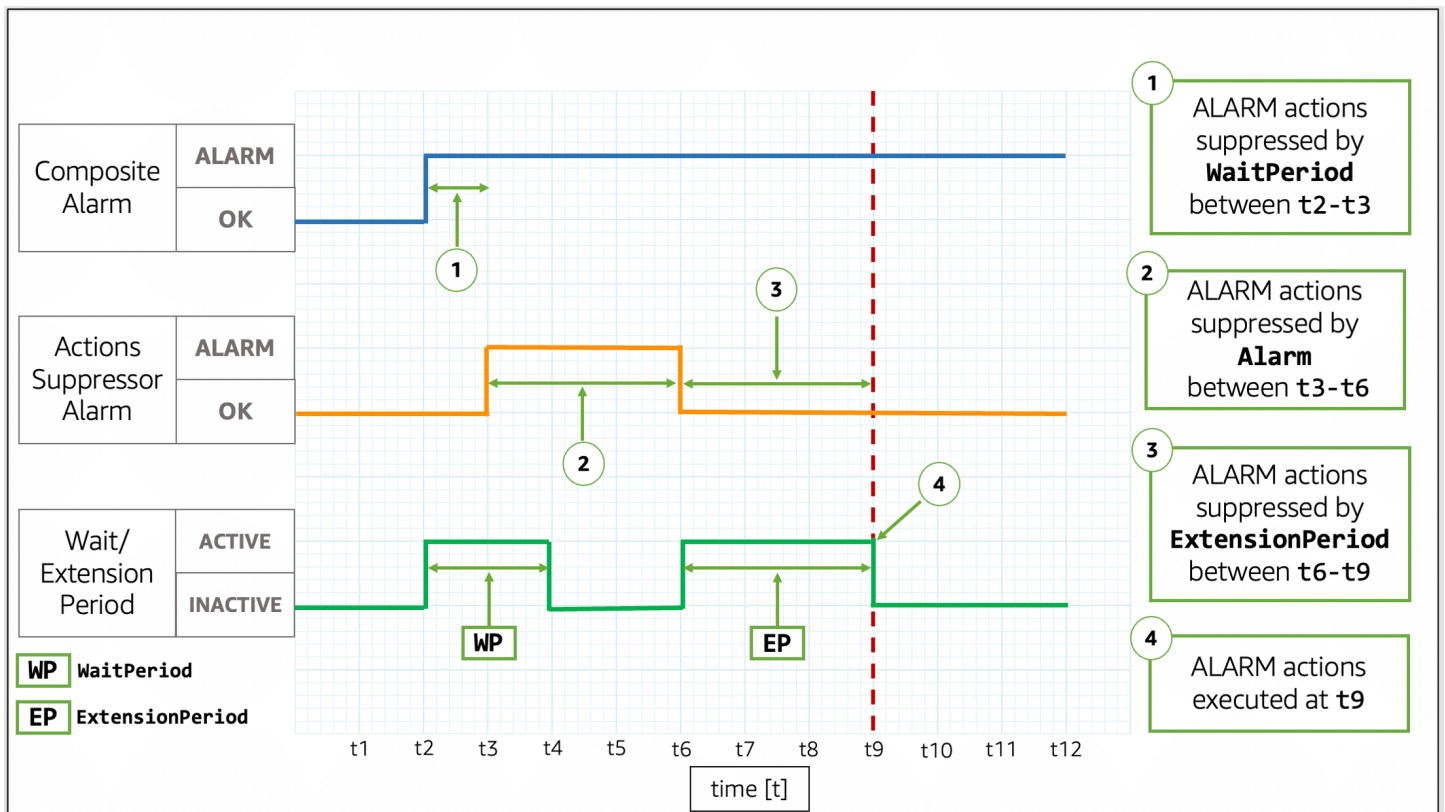
In the image, the composite alarm changes states from OK to ALARM at t_2 . A **WaitPeriod** starts at t_2 and ends at t_4 . This gives the suppressor alarm time to change states. The composite alarm changes back to OK at t_3 , so the **WaitPeriod** that started at t_2 is discarded. A new **WaitPeriod** starts at t_3 and ends at t_5 . After the new **WaitPeriod** expires at t_5 , the composite alarm takes its actions.

Example 4: State transition when actions are suppressed by alarm



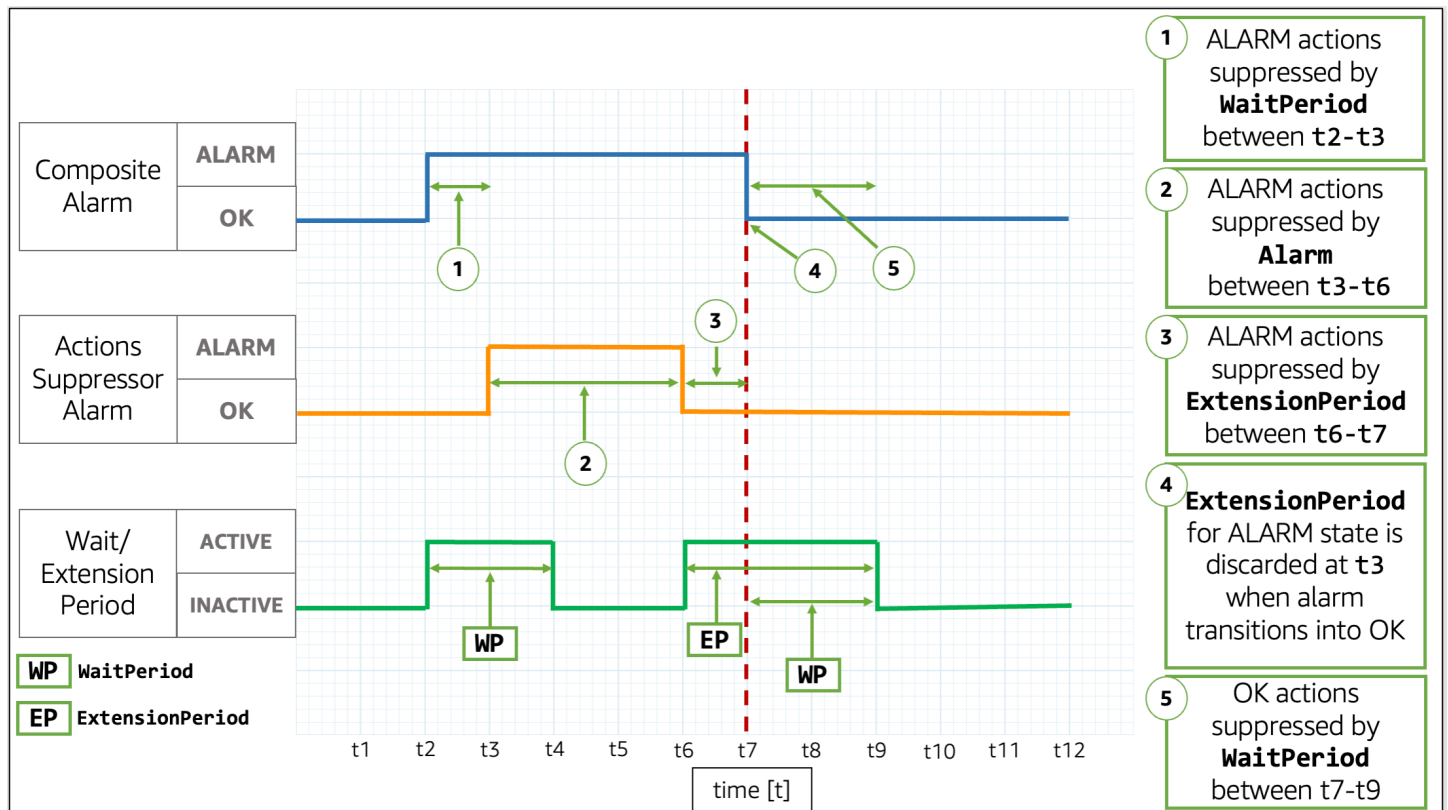
In the image, the composite alarm changes states from OK to ALARM at t2. The suppressor alarm is already in ALARM. The suppressor alarm stops the composite alarm from taking actions.

Example 5: Actions are not suppressed after ExtensionPeriod



In the image, the composite alarm changes states from OK to ALARM at t_2 . A WaitPeriod starts at t_2 and ends at t_4 . This gives the suppressor alarm time to change states from OK to ALARM at t_3 before it suppresses the composite alarm's actions until t_6 . Because the suppressor alarm changes states from OK to ALARM at t_3 , the WaitPeriod that started at t_2 is discarded. At t_6 , the suppressor alarm changes to OK. An ExtensionPeriod starts at t_6 and ends at t_9 . After the ExtensionPeriod expires, the composite alarm takes its actions.

Example 6: State transition when actions are suppressed by ExtensionPeriod



In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from OK to ALARM at t3 before it suppresses the composite alarm's actions until t6. Because the suppressor alarm changes states from OK to ALARM at t3, the `WaitPeriod` that started at t2 is discarded. At t6, the suppressor alarm changes back to OK. An `ExtensionPeriod` starts at t6 and ends at t9. When the composite alarm changes back to OK at t7, the `ExtensionPeriod` is discarded, and a new `WaitPeriod` starts at t7 and ends at t9.

Tip

If you replace the action suppressor alarm, any active `WaitPeriod` or `ExtensionPeriod` is discarded.

Acting on alarm changes

CloudWatch can notify users on two types of alarm changes: when an alarm changes state, and when the configuration of an alarm gets updated.

When an alarm evaluates, it might change from one state to another, such as ALARM, OK or INSUFFICIENT_DATA. These alarm state changes can signal a possible incident, a return to normal, or a metric being unavailable. In such cases, you might want to engage or notify users using either of the following options:

- You can configure the alarm to send a notification to a SNS topic as part of the alarm's actions. An SNS topic can then be configured for application-to-application (A2A) messaging as well as application-to-person (A2P) notifications, including channels such as email notifications and SMS. All the destinations that you define for your SNS topic receive the alarm notification. For more information, see [Amazon SNS event destinations](#).
- You can configure notifications for alarm state change events. AWS User Notifications offers a native way to configure such notifications and is the recommended approach.

Additionally, CloudWatch sends events to Amazon EventBridge whenever alarms change state, and when alarms are created, deleted, or updated. You can write EventBridge rules to take actions or be notified when EventBridge receives these events.

Topics

- [Notifying users on alarm changes](#)
- [Alarm events and EventBridge](#)

Notifying users on alarm changes

This section explains how you can use AWS User Notifications or Amazon Simple Notification Service to have users be notified of alarm changes.

Setting up AWS User Notifications

You can use [AWS User Notifications](#) to set up delivery channels to get notified about CloudWatch alarm state change and configuration change events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, [AWS Chatbot](#) chat notifications, or [AWS Console Mobile Application push notifications](#). You can also see notifications in the at [Console Notifications Center](#). User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

The notification configurations you create with AWS User Notifications do not count towards the limit on the number of actions you can configure per target alarm state. As AWS User Notifications matches the events emitted to Amazon EventBridge, it sends notifications for all the alarms in your account and selected Regions, unless you specify an advanced filter to allowlist or denylist specific alarms or patterns.

The following example of an advanced filter matches an alarm state change from OK to ALARM on the alarm named `ServerCpuTooHigh`.

```
{
  "detail": {
    "alarmName": ["ServerCpuTooHigh"],
    "previousState": { "value": ["OK"] },
    "state": { "value": ["ALARM"] }
  }
}
```

You can use any of the properties published by an alarm in EventBridge events to create a filter. For more information, see [Alarm events and EventBridge](#).

Setting up Amazon SNS notifications

You can use Amazon Simple Notification Service to send both application-to-application (A2A) messaging and application-to-person (A2P) messaging, including mobile text messaging (SMS) and email messages. For more information, see [Amazon SNS event destinations](#).

For every state that an alarm can take, you can configure the alarm to send a message to an SNS topic. Every Amazon SNS topic you configure for a state on a given alarm counts towards the limit on the number of actions you can configure for that alarm and state. You can send messages to the same Amazon SNS topic from any alarms in your account, and use the same Amazon SNS topic for both application (A2A) and person (A2P) consumers. Because this configuration is done at the alarm level, only the alarms you have configured send messages to the selected Amazon SNS topic.

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic. For an example, see [Setting up an Amazon SNS topic using the AWS Management Console](#). Or for more information, see [Getting started with Amazon SNS](#).

Alternatively, if you plan to use the AWS Management Console to create your CloudWatch alarm, you can skip this procedure because you can create the topic when you create the alarm.

When you create a CloudWatch alarm, you can add actions for any target state the alarm enters. Add an Amazon SNS notification for the state you want to be notified about, and select the Amazon SNS topic you created in the previous step to send an email notification when the alarm enters the selected state.

Note

When you create an Amazon SNS topic, you choose to make it a *standard topic* or a *FIFO topic*. CloudWatch guarantees the publication of all alarm notifications to both types of topics. However, even if you use a FIFO topic, in rare cases CloudWatch sends the notifications to the topic out of order. If you use a FIFO topic, the alarm sets the message group ID of the alarm notifications to be a hash of the ARN of the alarm.

Preventing confused deputy security issues

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#), [aws:SourceAccount](#), [aws:SourceOrgID](#), and [aws:SourceOrgPaths](#) global condition context keys in resource policies to limit the permissions that Amazon SNS gives another service to the resource. Use `aws:SourceArn` to associate only one resource with cross-service access. Use `aws:SourceAccount` to let any resource in that account be associated with the cross-service use. Use `aws:SourceOrgID` to allow any resource from any account within an organization be associated with the cross-service use. Use `aws:SourceOrgPaths` to associate any resource from accounts within an AWS Organizations path with the cross-service use. For more information about using and understanding paths, see [Understand the AWS Organizations entity path](#).

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn`

global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:service:*:123456789012:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both `aws:SourceAccount` and `aws:SourceArn` to limit permissions.

To protect against the confused deputy problem at scale, use the `aws:SourceOrgID` or `aws:SourceOrgPaths` global condition context key with the organization ID or organization path of the resource in your resource-based policies. Policies that include the `aws:SourceOrgID` or `aws:SourceOrgPaths` key will automatically include the correct accounts and you don't have to manually update the policies when you add, remove, or move accounts in your organization.

The value of `aws:SourceArn` must be the ARN of the alarm that is sending notifications.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in to prevent the confused deputy problem.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "cloudwatch.amazonaws.com"
    },
    "Action": "SNS:Publish",
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "111122223333"
      }
    }
  }]
}
```

If an alarm ARN includes any non-ASCII characters, use only the `aws:SourceAccount` global condition key to limit the permissions.

Setting up an Amazon SNS topic using the AWS Management Console

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic.

To create an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. On the Amazon SNS dashboard, under **Common actions**, choose **Create Topic**.
3. In the **Create new topic** dialog box, for **Topic name**, enter a name for the topic (for example, **my-topic**).
4. Choose **Create topic**.
5. Copy the **Topic ARN** for the next task (for example, `arn:aws:sns:us-east-1:111122223333:my-topic`).

To subscribe to an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Subscriptions**, **Create subscription**.
3. In the **Create subscription** dialog box, for **Topic ARN**, paste the topic ARN that you created in the previous task.
4. For **Protocol**, choose **Email**.
5. For **Endpoint**, enter an email address that you can use to receive the notification, and then choose **Create subscription**.
6. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon SNS.

To publish a test message to an SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic and choose **Publish to topic**.
4. In the **Publish a message** page, for **Subject**, enter a subject line for your message, and for **Message**, enter a brief message.
5. Choose **Publish Message**.
6. Check your email to confirm that you received the message.

Setting up an SNS topic using the AWS CLI

First, you create an SNS topic, and then you publish a message directly to the topic to test that you have properly configured it.

To set up an SNS topic

1. Create the topic using the [create-topic](#) command as follows.

```
aws sns create-topic --name my-topic
```

Amazon SNS returns a topic ARN with the following format:

```
{
  "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"
}
```

2. Subscribe your email address to the topic using the [subscribe](#) command. If the subscription request succeeds, you receive a confirmation email message.

```
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --
protocol email --notification-endpoint my-email-address
```

Amazon SNS returns the following:

```
{
  "SubscriptionArn": "pending confirmation"
}
```

3. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon Simple Notification Service.

4. Check the subscription using the [list-subscriptions-by-topic](#) command.

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS returns the following:

```
{
  "Subscriptions": [
    {
      "Owner": "111122223333",
      "Endpoint": "me@mycompany.com",
      "Protocol": "email",
      "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
      "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
  ]
}
```

5. (Optional) Publish a test message to the topic using the [publish](#) command.

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS returns the following.

```
{
  "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. Check your email to confirm that you received the message.

Alarm events and EventBridge

CloudWatch sends events to Amazon EventBridge whenever a CloudWatch alarm is created, updated, deleted, or changes alarm state. You can use EventBridge and these events to write rules that take actions, such as notifying you, when an alarm changes state. For more information, see [What is Amazon EventBridge?](#)

CloudWatch guarantees the delivery of alarm state change events to EventBridge.

Sample events from CloudWatch

This section includes example events from CloudWatch.

State change for a single-metric alarm

```

{
  "version": "0",
  "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2019-10-02T17:04:40Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
  ],
  "detail": {
    "alarmName": "ServerCpuTooHigh",
    "configuration": {
      "description": "Goes into alarm when server CPU utilization is too high!",
      "metrics": [
        {
          "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",
          "metricStat": {
            "metric": {
              "dimensions": {
                "InstanceId": "i-12345678901234567"
              },
              "name": "CPUUtilization",
              "namespace": "AWS/EC2"
            },
            "period": 300,
            "stat": "Average"
          },
          "returnData": true
        }
      ]
    },
    "previousState": {
      "reason": "Threshold Crossed: 1 out of the last 1 datapoints [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0) (minimum 1 datapoint for ALARM -> OK transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":\"2019-10-01T13:56:40.985+0000\",\"startDate\":\"2019-10-01T13:46:00.000+0000\",\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[0.0666851903306472],\"threshold\":50.0}",
      "timestamp": "2019-10-01T13:56:40.987+0000",
      "value": "OK"
    }
  }
}

```

```

    },
    "state": {
      "reason": "Threshold Crossed: 1 out of the last 1 datapoints
[99.50160229693434 (02/10/19 16:59:00)] was greater than the threshold (50.0) (minimum
1 datapoint for OK -> ALARM transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\\\"2019-10-02T17:04:40.985+0000\\\",\\\"startDate\\\":\\\"2019-10-02T16:59:00.000+0000\\\",
\\\"statistic\\\":\\\"Average\\\",\\\"period\\\":300,\\\"recentDatapoints\\\":[99.50160229693434],
\\\"threshold\\\":50.0}\",
      "timestamp": "2019-10-02T17:04:40.989+0000",
      "value": "ALARM"
    }
  }
}

```

State change for a metric math alarm

```

{
  "version": "0",
  "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2019-10-02T17:20:48Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
  ],
  "detail": {
    "alarmName": "TotalNetworkTrafficTooHigh",
    "configuration": {
      "description": "Goes into alarm if total network traffic exceeds 10Kb",
      "metrics": [
        {
          "expression": "SUM(METRICS())",
          "id": "e1",
          "label": "Total Network Traffic",
          "returnData": true
        },
        {
          "id": "m1",
          "metricStat": {
            "metric": {

```

```

        "dimensions": {
            "InstanceId": "i-12345678901234567"
        },
        "name": "NetworkIn",
        "namespace": "AWS/EC2"
    },
    "period": 300,
    "stat": "Maximum"
},
"returnData": false
},
{
    "id": "m2",
    "metricStat": {
        "metric": {
            "dimensions": {
                "InstanceId": "i-12345678901234567"
            },
            "name": "NetworkOut",
            "namespace": "AWS/EC2"
        },
        "period": 300,
        "stat": "Maximum"
    },
    "returnData": false
}
]
},
"previousState": {
    "reason": "Unchecked: Initial alarm creation",
    "timestamp": "2019-10-02T17:20:03.642+0000",
    "value": "INSUFFICIENT_DATA"
},
"state": {
    "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0
(02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint for
OK -> ALARM transition).",
    "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:20:48.551+0000\",\"startDate\":\"2019-10-02T17:10:00.000+0000\",
\"period\":300,\"recentDatapoints\":[45628.0],\"threshold\":10000.0}",
    "timestamp": "2019-10-02T17:20:48.554+0000",
    "value": "ALARM"
}
}

```

```
}

```

State change for an anomaly detection alarm

```
{
  "version": "0",
  "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2019-10-03T16:00:04Z",
  "region": "us-east-1",
  "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU Utilization
Anomaly"],
  "detail": {
    "alarmName": "EC2 CPU Utilization Anomaly",
    "state": {
      "value": "ALARM",
      "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0
(03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or
greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK ->
ALARM transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T16:00:04.650+0000\",\"startDate\":\"2019-10-03T15:58:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.0],\"recentLowerThresholds\":
[0.020599444741798756],\"recentUpperThresholds\":[0.3006915352732461]}",
      "timestamp": "2019-10-03T16:00:04.653+0000"
    },
    "previousState": {
      "value": "OK",
      "reason": "Thresholds Crossed: 1 out of the last 1 datapoints
[0.1666666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds
[0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803]
(minimum 1 datapoint for ALARM -> OK transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T15:59:04.670+0000\",\"startDate\":\"2019-10-03T15:57:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.1666666666664241],\"recentLowerThresholds\":
[0.0206719426210418],\"recentUpperThresholds\":[0.30076870222143803]}",
      "timestamp": "2019-10-03T15:59:04.672+0000"
    },
    "configuration": {
      "description": "Goes into alarm if CPU Utilization is out of band",
      "metrics": [{

```



```

        "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.FirstChild transitioned to ALARM
at Friday 22 July, 2022 15:57:45 UTC",
        "reasonData": "{\"triggeringAlarms\": [{\"arn\": \"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\", \"state\": {\"value\": \"ALARM\", \"timestamp
\": \"2022-07-22T15:57:45.394+0000\"}}]}",
        "timestamp": "2022-07-22T15:57:45.394+0000"
    },
    "previousState": {
        "value": "OK",
        "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.Main was created and its alarm
rule evaluates to OK",
        "reasonData": "{\"triggeringAlarms\": [{\"arn\": \"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:TotalNetworkTrafficTooHigh\", \"state\": {\"value\": \"OK\",
\"timestamp\": \"2022-07-14T16:28:57.770+0000\"}}, {\"arn\": \"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\", \"state\": {\"value\": \"OK\", \"timestamp\":
\"2022-07-14T16:28:54.191+0000\"}}]}",
        "timestamp": "2022-07-22T15:56:14.552+0000"
    },
    "configuration": {
        "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
        "actionsSuppressor": "ServiceMaintenanceAlarm",
        "actionsSuppressorWaitPeriod": 120,
        "actionsSuppressorExtensionPeriod": 180
    }
}
}

```

Creation of a composite alarm

```

{
    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ]
}

```

```

    ],
    "detail": {
      "alarmName": "ServiceAggregatedAlarm",
      "operation": "create",
      "state": {
        "value": "INSUFFICIENT_DATA",
        "timestamp": "2022-03-03T17:06:22.289+0000"
      },
      "configuration": {
        "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
        "alarmName": "ServiceAggregatedAlarm",
        "description": "Aggregated monitor for instance",
        "actionsEnabled": true,
        "timestamp": "2022-03-03T17:06:22.289+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
      }
    }
  }
}

```

Creation of a composite alarm with a suppressor alarm

```

{
  "version": "0",
  "id": "454773e1-09f7-945b-aa2c-590af1c3f8e0",
  "detail-type": "CloudWatch Alarm Configuration Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2022-07-14T13:59:46Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
  ],
  "detail": {
    "alarmName": "ServiceAggregatedAlarm",
    "operation": "create",
    "state": {
      "value": "INSUFFICIENT_DATA",
      "timestamp": "2022-07-14T13:59:46.425+0000"
    },
    "configuration": {

```

```

    "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
    "actionsSuppressor": "ServiceMaintenanceAlarm",
    "actionsSuppressorWaitPeriod": 120,
    "actionsSuppressorExtensionPeriod": 180,
    "alarmName": "ServiceAggregatedAlarm",
    "actionsEnabled": true,
    "timestamp": "2022-07-14T13:59:46.425+0000",
    "okActions": [],
    "alarmActions": [],
    "insufficientDataActions": []
  }
}
}

```

Update of a metric alarm

```

{
  "version": "0",
  "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
  "detail-type": "CloudWatch Alarm Configuration Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2022-03-03T17:06:34Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
  ],
  "detail": {
    "alarmName": "ServerCpuTooHigh",
    "operation": "update",
    "state": {
      "value": "INSUFFICIENT_DATA",
      "timestamp": "2022-03-03T17:06:13.757+0000"
    },
    "configuration": {
      "evaluationPeriods": 1,
      "threshold": 80,
      "comparisonOperator": "GreaterThanThreshold",
      "treatMissingData": "ignore",
      "metrics": [
        {

```

```
        "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
        "metricStat": {
            "metric": {
                "namespace": "AWS/EC2",
                "name": "CPUUtilization",
                "dimensions": {
                    "InstanceId": "i-12345678901234567"
                }
            },
            "period": 300,
            "stat": "Average"
        },
        "returnData": true
    }
],
"alarmName": "ServerCpuTooHigh",
"description": "Goes into alarm when server CPU utilization is too high!",
"actionsEnabled": true,
"timestamp": "2022-03-03T17:06:34.267+0000",
"okActions": [],
"alarmActions": [],
"insufficientDataActions": []
},
"previousConfiguration": {
    "evaluationPeriods": 1,
    "threshold": 70,
    "comparisonOperator": "GreaterThanThreshold",
    "treatMissingData": "ignore",
    "metrics": [
        {
            "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
            "metricStat": {
                "metric": {
                    "namespace": "AWS/EC2",
                    "name": "CPUUtilization",
                    "dimensions": {
                        "InstanceId": "i-12345678901234567"
                    }
                },
                "period": 300,
                "stat": "Average"
            },
            "returnData": true
        }
    ]
}
```

```

    ],
    "alarmName": "ServerCpuTooHigh",
    "description": "Goes into alarm when server CPU utilization is too high!",
    "actionsEnabled": true,
    "timestamp": "2022-03-03T17:06:13.757+0000",
    "okActions": [],
    "alarmActions": [],
    "insufficientDataActions": []
  }
}
}

```

Update of a composite alarm with a suppressor alarm

```

{
  "version": "0",
  "id": "4c6f4177-6bd5-c0ca-9f05-b4151c54568b",
  "detail-type": "CloudWatch Alarm Configuration Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2022-07-14T13:59:56Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
  ],
  "detail": {
    "alarmName": "ServiceAggregatedAlarm",
    "operation": "update",
    "state": {
      "actionsSuppressedBy": "WaitPeriod",
      "value": "ALARM",
      "timestamp": "2022-07-14T13:59:46.425+0000"
    },
    "configuration": {
      "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
      "actionsSuppressor": "ServiceMaintenanceAlarm",
      "actionsSuppressorWaitPeriod": 120,
      "actionsSuppressorExtensionPeriod": 360,
      "alarmName": "ServiceAggregatedAlarm",
      "actionsEnabled": true,
      "timestamp": "2022-07-14T13:59:56.290+0000",
      "okActions": [],

```

```

        "alarmActions": [],
        "insufficientDataActions": []
    },
    "previousConfiguration": {
        "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
        "actionsSuppressor": "ServiceMaintenanceAlarm",
        "actionsSuppressorWaitPeriod": 120,
        "actionsSuppressorExtensionPeriod": 180,
        "alarmName": "ServiceAggregatedAlarm",
        "actionsEnabled": true,
        "timestamp": "2022-07-14T13:59:46.425+0000",
        "okActions": [],
        "alarmActions": [],
        "insufficientDataActions": []
    }
}
}
}

```

Deletion of a metric math alarm

```

{
    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:07:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "operation": "delete",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:17.672+0000"
        }
    },
    "configuration": {
        "evaluationPeriods": 1,
        "threshold": 10000,
    }
}

```

```
"comparisonOperator": "GreaterThanThreshold",
"treatMissingData": "ignore",
"metrics": [{
  "id": "m1",
  "metricStat": {
    "metric": {
      "namespace": "AWS/EC2",
      "name": "NetworkIn",
      "dimensions": {
        "InstanceId": "i-12345678901234567"
      }
    },
    "period": 300,
    "stat": "Maximum"
  },
  "returnData": false
},
{
  "id": "m2",
  "metricStat": {
    "metric": {
      "namespace": "AWS/EC2",
      "name": "NetworkOut",
      "dimensions": {
        "InstanceId": "i-12345678901234567"
      }
    },
    "period": 300,
    "stat": "Maximum"
  },
  "returnData": false
},
{
  "id": "e1",
  "expression": "SUM(METRICS())",
  "label": "Total Network Traffic",
  "returnData": true
}
],
"alarmName": "TotalNetworkTrafficTooHigh",
"description": "Goes into alarm if total network traffic exceeds 10Kb",
"actionsEnabled": true,
"timestamp": "2022-03-03T17:06:17.672+0000",
"okActions": [],
```

```

        "alarmActions": [],
        "insufficientDataActions": []
    }
}

```

Deletion of a composite alarm with a suppressor alarm

```

{
  "version": "0",
  "id": "e34592a1-46c0-b316-f614-1b17a87be9dc",
  "detail-type": "CloudWatch Alarm Configuration Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2022-07-14T14:00:01Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
  ],
  "detail": {
    "alarmName": "ServiceAggregatedAlarm",
    "operation": "delete",
    "state": {
      "actionsSuppressedBy": "WaitPeriod",
      "value": "ALARM",
      "timestamp": "2022-07-14T13:59:46.425+0000"
    },
    "configuration": {
      "alarmRule": "ALARM(ServerCpuTooHigh) OR
ALARM(TotalNetworkTrafficTooHigh)",
      "actionsSuppressor": "ServiceMaintenanceAlarm",
      "actionsSuppressorWaitPeriod": 120,
      "actionsSuppressorExtensionPeriod": 360,
      "alarmName": "ServiceAggregatedAlarm",
      "actionsEnabled": true,
      "timestamp": "2022-07-14T13:59:56.290+0000",
      "okActions": [],
      "alarmActions": [],
      "insufficientDataActions": []
    }
  }
}

```


Managing alarms

Edit or delete a CloudWatch alarm

You can edit or delete an existing alarm.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action, Copy**.

To edit an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose the name of the alarm.
4. To add or remove tags, choose the **Tags** tab and then choose **Manage tags**.
5. To edit other parts of the alarm, choose **Actions, Edit**.

The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.

6. To change the metric, choose **Edit**, choose the **All metrics** tab, and do one of the following:
 - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
 - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.

Choose **Select metric**.

7. To change other aspects of the alarm, choose the appropriate options. To change how many data points must be breaching for the alarm to go into ALARM state or to change how missing data is treated, choose **Additional configuration**.
8. Choose **Next**.
9. Under **Notification, Auto Scaling action**, and **EC2 action**, optionally edit the actions taken when the alarm is triggered. Then choose **Next**.
10. Optionally change the alarm description.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action, Copy**.

11. Choose **Next**.
12. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Update alarm**.

To update an email notification list that was created using the Amazon SNS console

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. In the navigation pane, choose **Topics** and then select the ARN for your notification list (topic).
3. Do one of the following:
 - To add an email address, choose **Create subscription**. For **Protocol**, choose **Email**. For **Endpoint**, enter the email address of the new recipient. Choose **Create subscription**.
 - To remove an email address, choose the **Subscription ID**. Choose **Other subscription actions, Delete subscriptions**.
4. Choose **Publish to topic**.

To delete an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**.
3. Select the check box to the left of the name of the alarm, and choose **Actions, Delete**.
4. Choose **Delete**.

Hide Auto Scaling alarms

When you view your alarms in the AWS Management Console, you can hide the alarms related to both Amazon EC2 Auto Scaling and Application Auto Scaling. This feature is available only in the AWS Management Console.

To temporarily hide Auto Scaling alarms

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Alarms, All alarms**, and select **Hide Auto Scaling alarms**.

Alarm use cases and examples

The following sections provides examples and tutorials for alarms for common use cases.

Create a billing alarm to monitor your estimated AWS charges

You can monitor your estimated AWS charges by using Amazon CloudWatch. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and represents worldwide charges. This data includes the estimated charges for every service in AWS that you use, in addition to the estimated overall total of your AWS charges.

The alarm triggers when your account billing exceeds the threshold you specify. It triggers only when the current billing exceeds the threshold. It doesn't use projections based on your usage so far in the month.

If you create a billing alarm at a time when your charges have already exceeded the threshold, the alarm goes to the ALARM state immediately.

Note

For information about analyzing CloudWatch charges that you have already been billed for, see [CloudWatch billing and cost](#).

Tasks

- [Enabling billing alerts](#)
- [Create a billing alarm](#)
- [Deleting a billing alarm](#)

Enabling billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated AWS charges and create an alarm using billing metric data. After

you enable billing alerts, you can't disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using account root user credentials or as an IAM user that has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see [AWS Partner Network](#).

To enable the monitoring of estimated charges

1. Open the AWS Billing console at <https://console.aws.amazon.com/billing/>.
2. In the navigation pane, choose **Billing Preferences**.
3. By **Alert preferences** choose **Edit**.
4. Choose **Receive CloudWatch Billing Alerts**.
5. Choose **Save preferences**.

Create a billing alarm

Important

Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing). For more information, see [Enabling billing alerts](#).

In this procedure, you create an alarm that sends a notification when your estimated charges for AWS exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In **Browse**, choose **Billing**, and then choose **Total Estimated Charge**.

Note

If you don't see the **Billing/Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see [Enabling billing alerts](#).

5. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.
6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
10. For **than . . .**, define the value that you want to cause your alarm to trigger. For example, **200 USD**.

The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see [What is AWS Billing?](#)

Note

After you define a threshold value, the preview graph displays your estimated charges for the current month.

11. Choose **Additional Configuration** and do the following:
 - For **Datapoints to alarm**, specify **1 out of 1**.
 - For **Missing data treatment**, choose **Treat missing data as missing**.
12. Choose **Next**.

13. Under **Notification**, ensure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the ALARM state. The Amazon SNS topic can include your email address so that you receive email when the billing amount crosses the threshold that you specified.

You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

14. Choose **Next**.
15. Under **Name and description**, enter a name for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters.
 - (Optional) Enter a description of your alarm. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
16. Choose **Next**.
17. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

Deleting a billing alarm

You can delete your billing alarm when you no longer need it.

To delete a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms, All alarms**.
4. Select the check box next to the alarm and choose **Actions, Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Create a CPU usage alarm

You can create an CloudWatch alarm that sends a notification using Amazon SNS when the alarm changes state from OK to ALARM.

The alarm changes to the ALARM state when the average CPU use of an EC2 instance exceeds a specified threshold for consecutive specified periods.

Setting up a CPU usage alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. In the **All metrics** tab, choose **EC2 metrics**.
6. Choose a metric category (for example, **Per-Instance Metrics**).
7. Find the row with the instance that you want listed in the **InstanceID** column and **CPUUtilization** in the **Metric Name** column. Select the check box next to this row, and choose **Select metric**.
8. Under **Specify metric and conditions**, for **Statistic** choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95.45**).
9. Choose a period (for example, **5 minutes**).
10. Under **Conditions**, specify the following:
 - a. For **Threshold type**, choose **Static**.
 - b. For **Whenever CPUUtilization is**, specify **Greater**. Under **than...**, specify the threshold that is to trigger the alarm to go to ALARM state if the CPU utilization exceeds this percentage. For example, 70.
 - c. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).

- d. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
 - e. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see [Percentile-based CloudWatch alarms and low data samples](#).
11. Choose **Next**.
 12. Under **Notification**, choose **In alarm** and select an SNS topic to notify when the alarm is in ALARM state

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.
 13. When finished, choose **Next**.
 14. Enter a name and description for the alarm. Then choose **Next**.

The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
 15. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Setting up a CPU usage alarm using the AWS CLI

Use these steps to use the AWS CLI to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Set up an SNS topic. For more information, see [Setting up Amazon SNS notifications](#).
2. Create an alarm using the [put-metric-alarm](#) command as follows.

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic
```



```
Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --
dimensions Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions
arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. Test the alarm by forcing an alarm state change using the [set-alarm-state](#) command.

a. Change the alarm state from INSUFFICIENT_DATA to OK.

```
aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
"initializing" --state-value OK
```

b. Change the alarm state from OK to ALARM.

```
aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
"initializing" --state-value ALARM
```

c. Check that you have received a notification about the alarm.

Create a load balancer latency alarm that sends email

You can set up an Amazon SNS notification and configure an alarm that monitors latency exceeding 100 ms for your Classic Load Balancer.

Setting up a latency alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose **Create alarm**.
4. Under **CloudWatch Metrics by Category**, choose the **ELB Metrics** category.
5. Select the row with the Classic Load Balancer and the **Latency** metric.
6. For the statistic, choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95.45**).
7. For the period, choose **1 Minute**.
8. Choose **Next**.

9. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **myHighCpuAlarm**) and a description of the alarm (for example, **Alarm when Latency exceeds 100s**). Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters

The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

10. Under **Whenever**, for **is**, choose **>** and enter **0.1**. For **for**, enter **3**.
11. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.

For **Percentiles with low samples**, choose **ignore (maintain the alarm state)** so that the alarm evaluates only situations with adequate numbers of data samples.

12. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **myHighCpuAlarm**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent.

13. Choose **Create Alarm**.

Setting up a latency alarm using the AWS CLI

Use these steps to use the AWS CLI to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Set up an SNS topic. For more information, see [Setting up Amazon SNS notifications](#).
2. Create the alarm using the [put-metric-alarm](#) command as follows:

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --
```

```
dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-  
actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. Test the alarm by forcing an alarm state change using the [set-alarm-state](#) command.
 - a. Change the alarm state from INSUFFICIENT_DATA to OK.

```
aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason  
"initializing" --state-value OK
```

- b. Change the alarm state from OK to ALARM.

```
aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason  
"initializing" --state-value ALARM
```

- c. Check that you have received an email notification about the alarm.

Create a storage throughput alarm that sends email

You can set up an SNS notification and configure an alarm that is triggered when Amazon EBS exceeds 100 MB throughput.

Setting up a storage throughput alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All Alarms**.
3. Choose **Create alarm**.
4. Under **EBS Metrics**, choose a metric category.
5. Select the row with the volume and the **VolumeWriteBytes** metric.
6. For the statistic, choose **Average**. For the period, choose **5 Minutes**. Choose **Next**.
7. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **myHighWriteAlarm**) and a description of the alarm (for example, **VolumeWriteBytes exceeds 100,000 KiB/s**). The name must contain only UTF-8 characters, and can't contain

ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

8. Under **Whenever**, for **is**, choose **>** and enter **100000**. For **for**, enter **15** consecutive periods.

A graphical representation of the threshold is shown under **Alarm Preview**.

9. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.
10. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create one.

To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **myHighCpuAlarm**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

11. Choose **Create Alarm**.

Setting up a storage throughput alarm using the AWS CLI

Use these steps to use the AWS CLI to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Create an SNS topic. For more information, see [Setting up Amazon SNS notifications](#).
2. Create the alarm.

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm
when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --
namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --
comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-
volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-
east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-
east-1:111122223333:my-insufficient-data-topic
```

3. Test the alarm by forcing an alarm state change using the [set-alarm-state](#) command.
 - a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason  
"initializing" --state-value OK
```

- b. Change the alarm state from OK to ALARM.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason  
"initializing" --state-value ALARM
```

- c. Change the alarm state from ALARM to INSUFFICIENT_DATA.

```
aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason  
"initializing" --state-value INSUFFICIENT_DATA
```

- d. Check that you have received an email notification about the alarm.

Create an alarm on Performance Insights counter metrics from an AWS database

CloudWatch includes a **DB_PERF_INSIGHTS** metric math function which you can use to bring Performance Insights counter metrics into CloudWatch from Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). **DB_PERF_INSIGHTS** also brings in the DBLoad metric at sub-minute intervals. You can set CloudWatch alarms on these metrics.

For more information about Amazon RDS Performance Insights, see [Monitoring DB load with Performance Insights on Amazon RDS](#).

For more information about Amazon DocumentDB Performance Insights, see [Monitoring with Performance Insights](#).

Anomaly detection is not supported for alarms based on the **DB_PERF_INSIGHTS** function.

Note

High-resolution metrics with sub-minute granularity retrieved by **DB_PERF_INSIGHTS** are only applicable to the **DBLoad** metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see [Monitoring OS metrics with Enhanced Monitoring](#).

You can create a high-resolution alarm using the **DB_PERF_INSIGHTS** function. The maximum evaluation range for a high-resolution alarm is three hours. You can use the

CloudWatch console to graph metrics retrieved with the **DB_PERF_INSIGHTS** function for any time range.

To create an alarm that's based on Performance Insights metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Choose the **Add math** dropdown, and then select **Database Performance Metrics, DB_PERF_INSIGHTS** from the list.

After you choose **DB_PERF_INSIGHTS**, a math expression box appears where you apply or edit math expressions.

6. In the math expression box, enter your **DB_PERF_INSIGHTS** math expression, and then choose **Apply**.

For example, `DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNORSTUVWXY1', 'os.cpuUtilization.user.avg')`

Important

When you use the **DB_PERF_INSIGHTS** math expression, you must specify the Unique Database Resource ID of the database. This is different than the database identifier. To find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is displayed in the **Configuration** section.

For information about the **DB_PERF_INSIGHTS** function and other functions that are available for metric math, see [Metric math syntax and functions](#).

7. Choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

8. For **Whenever *expression* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
9. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see [Evaluating an alarm](#).


10. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see [Configuring how CloudWatch alarms treat missing data](#).
11. Choose **Next**.
12. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

To have the alarm not send notifications, choose **Remove**.

13. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see [Configuring CloudWatch to create OpsItems from alarms](#) and [Incident creation](#).

 **Note**

To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see [Identity-based policy examples for AWS Systems Manager Incident Manager](#).

14. When finished, choose **Next**.
15. Enter a name and description for the alarm. Then choose **Next**.

The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

16. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Create alarms to stop, terminate, reboot, or recover an EC2 instance

Using Amazon CloudWatch alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your EC2 instances. You can use the stop or terminate actions to help you save money when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances or recover them onto new hardware if a system impairment occurs.

There are a number of scenarios in which you might want to automatically stop or terminate your instance. For example, you might have instances dedicated to batch payroll processing jobs or scientific computing tasks that run for a period of time and then complete their work. Rather than letting those instances sit idle (and accrue charges), you can stop or terminate them, which helps you to save money. The main difference between using the stop and the terminate alarm actions is that you can easily restart a stopped instance if you need to run it again later. You can also keep the same instance ID and root volume. However, you cannot restart a terminated instance. Instead, you must launch a new instance.

You can add the stop, terminate, or reboot, actions to any alarm that is set on an Amazon EC2 per-instance metric, including basic and detailed monitoring metrics provided by Amazon CloudWatch (in the `AWS/EC2` namespace), in addition to any custom metrics that include the `"InstanceId="` dimension, as long as the `InstanceId` value refers to a valid running Amazon EC2 instance. You can also add the recover action to alarms that is set on any Amazon EC2 per-instance metric except for `StatusCheckFailed_Instance`.

Important

Alarms configured on Amazon EC2 metrics can temporarily enter the `INSUFFICIENT_DATA` state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions,

we recommend that you configure those alarms to treat missing data as missing, and to have these alarms trigger only when in the ALARM state.

For more information about how you can configure CloudWatch to act on missing metrics that have alarms set on them, see [Configuring how CloudWatch alarms treat missing data](#).

To set up a CloudWatch alarm action that can reboot, stop, or terminate an instance, you must use a service-linked IAM role, *AWSServiceRoleForCloudWatchEvents*. The *AWSServiceRoleForCloudWatchEvents* IAM role enables AWS to perform alarm actions on your behalf.

To create the service-linked role for CloudWatch Events, use the following command:

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

Console support

You can create alarms using the CloudWatch console or the Amazon EC2 console. The procedures in this documentation use the CloudWatch console. For procedures that use the Amazon EC2 console, see [Create Alarms That Stop, Terminate, Reboot, or Recover an Instance](#) in the *Amazon EC2 User Guide*.

Permissions

If you are using an AWS Identity and Access Management (IAM) account to create or modify an alarm that performs EC2 actions or Systems Manager OpsItem actions, you must have the `iam:CreateServiceLinkedRole` permission.

Contents

- [Adding stop actions to Amazon CloudWatch alarms](#)
- [Adding terminate actions to Amazon CloudWatch alarms](#)
- [Adding reboot actions to Amazon CloudWatch alarms](#)
- [Adding recover actions to Amazon CloudWatch alarms](#)
- [Viewing the history of triggered alarms and actions](#)

Adding stop actions to Amazon CloudWatch alarms

You can create an alarm that stops an Amazon EC2 instance when a certain threshold has been met. For example, you may run development or test instances and occasionally forget to shut them off. You can create an alarm that is triggered when the average CPU utilization percentage has been lower than 10 percent for 24 hours, signaling that it is idle and no longer in use. You can adjust the threshold, duration, and period to suit your needs, plus you can add an SNS notification, so that you will receive an email when the alarm is triggered.

Amazon EC2 instances that use an Amazon Elastic Block Store volume as the root device can be stopped or terminated, whereas instances that use the instance store as the root device can only be terminated.

To create an alarm to stop an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. For **AWS namespaces**, choose **EC2**.
6. Do the following:
 - a. Choose **Per-Instance Metrics**.
 - b. Select the check box in the row with the correct instance and the **CPUUtilization** metric.
 - c. Choose the **Graphed metrics** tab.
 - d. For the statistic, choose **Average**.
 - e. Choose a period (for example, **1 Hour**).
 - f. Choose **Select metric**.
7. For the **Define Alarm** step, do the following:
 - a. Under **Conditions**, choose **Static**.
 - b. Under **Whenever CPUUtilization is**, choose **Lower**.
 - c. For **than**, type **10**.
 - d. Choose **Next**.
 - e. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, `Stop_EC2_Instance`). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- f. Choose **Add EC2 Action**.
- g. For **Alarm state trigger**, choose **In alarm**. For **Take the following action**, choose **Stop this instance**.
- h. Choose **Next**.
- i. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
- j. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Adding terminate actions to Amazon CloudWatch alarms

You can create an alarm that terminates an EC2 instance automatically when a certain threshold has been met (as long as termination protection is not enabled for the instance). For example, you might want to terminate an instance when it has completed its work, and you don't need the instance again. If you might want to use the instance later, you should stop the instance instead of terminating it. For information about enabling and disabling termination protection for an instance, see [Enabling Termination Protection for an Instance](#) in the *Amazon EC2 User Guide*.

To create an alarm to terminate an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the **CPUUtilization** metric.
 - c. For the statistic, choose **Average**.
 - d. Choose a period (for example, **1 Hour**).
 - e. Choose **Next**.

4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Terminate EC2 instance) and a description of the alarm (for example, Terminate EC2 instance when CPU is idle for too long). Alarm names must contain only ASCII characters.
 - b. Under **Whenever**, for **is**, choose **<** and type **10**. For **for**, type **24** consecutive periods.

A graphical representation of the threshold is shown under **Alarm Preview**.

- c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Terminate_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- d. Choose **EC2 Action**.
 - e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Terminate this instance**.
 - f. Choose **Create Alarm**.

Adding reboot actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically reboots the instance. The reboot alarm action is recommended for Instance Health Check failures (as opposed to the recover alarm action, which is suited for System Health Check failures). An instance reboot is equivalent to an operating system reboot. In most cases, it takes only a few minutes to reboot your instance. When you reboot an instance, it remains on the same physical host, so your instance keeps its public DNS name, private IP address, and any data on its instance store volumes.

Rebooting an instance doesn't start a new instance billing hour, unlike stopping and restarting your instance. For more information about rebooting an instance, see [Reboot Your Instance](#) in the *Amazon EC2 User Guide*.

⚠ Important

To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set reboot alarms to three evaluation periods of one minute each.

To create an alarm to reboot an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the **StatusCheckFailed_Instance** metric.
 - c. For the statistic, choose **Minimum**.
 - d. Choose a period (for example, **1 Minute**).
 - e. Choose **Next**.
4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Reboot EC2 instance) and a description of the alarm (for example, Reboot EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
 - b. Under **Whenever**, for **is**, choose **>** and type **0**. For **for**, type **3** consecutive periods.

A graphical representation of the threshold is shown under **Alarm Preview**.

- c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Reboot_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- d. Choose **EC2 Action**.

- e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Reboot this instance**.
- f. Choose **Create Alarm**.

Adding recover actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically recovers the instance if it becomes impaired due to an underlying hardware failure or a problem that requires AWS involvement to repair. Terminated instances cannot be recovered. A recovered instance is identical to the original instance, including the instance ID, private IP addresses, Elastic IP addresses, and all instance metadata.

When the `StatusCheckFailed_System` alarm is triggered, and the recover action is initiated, you will be notified by the Amazon SNS topic that you chose when you created the alarm and associated the recover action. During instance recovery, the instance is migrated during an instance reboot, and any data that is in-memory is lost. When the process is complete, information is published to the SNS topic you've configured for the alarm. Anyone who is subscribed to this SNS topic will receive an email notification that includes the status of the recovery attempt and any further instructions. You will notice an instance reboot on the recovered instance.

The recover action can be used only with `StatusCheckFailed_System`, not with `StatusCheckFailed_Instance`.

Examples of problems that cause system status checks to fail include:

- Loss of network connectivity
- Loss of system power
- Software issues on the physical host
- Hardware issues on the physical host that impact network reachability

The recover action is supported only on some instance types. For more information about supported instance types and other requirements, see [Recover your instance](#) and [Requirements](#).

Important

To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set

recover alarms to two evaluation periods of one minute each and reboot alarms to three evaluation periods of one minute each.

To create an alarm to recover an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms, Create Alarm**.
3. For the **Select Metric** step, do the following:
 - a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
 - b. Select the row with the instance and the **StatusCheckFailed_System** metric.
 - c. For the statistic, choose **Minimum**.
 - d. Choose a period (for example, **1 Minute**).
4. For the **Define Alarm** step, do the following:
 - a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Recover EC2 instance) and a description of the alarm (for example, Recover EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
 - b. Under **Whenever**, for **is**, choose **>** and type **0**. For **for**, type **2** consecutive periods.
 - c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

Important

To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set recover alarms to two evaluation periods of one minute each.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Recover_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

- d. Choose **EC2 Action**.
- e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Recover this instance**.
- f. Choose **Create Alarm**.

Viewing the history of triggered alarms and actions

You can view alarm and action history in the Amazon CloudWatch console. Amazon CloudWatch keeps the last 30 days of alarm and action history.

To view the history of triggered alarms and actions

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms** and select an alarm.
3. To view the most recent state transition along with the time and metric values, choose **Details**.
4. To view the most recent history entries, choose **History**.

Alarms and tagging

Tags are key-value pairs that can help you organize and categorize your resources. You can also use them to scope user permissions by granting a user permission to access or change only resources with certain tag values. For more general information about tagging resources, see [Tagging your AWS resources](#)

The following list explains some details about how tagging works with CloudWatch alarms.

- To be able to set or update tags for a CloudWatch resource, you must be signed on to an account that has the `cloudwatch:TagResource` permission. For example, to create an alarm and set tags for it, you must have the `cloudwatch:TagResource` permission in addition to the `cloudwatch:PutMetricAlarm` permission. We recommend that you make sure anyone in your organization who will create or update CloudWatch resources has the `cloudwatch:TagResource` permission.
- Tags can be used for tag-based authorization control. For example, IAM user or role permissions can include conditions to limit CloudWatch calls to specific resources based on their tags. However, keep in mind the following
 - Tags with names that start with `aws:` can't be used for tag-based authorization control.

- Composite alarms do not support tag-based authorization control.

Application monitoring (APM)

CloudWatch includes the following observability solutions that you can use to help you monitor your own applications and services.

Topics

- [Application Signals](#)
- [Service level objectives \(SLOs\)](#)
- [Synthetic monitoring \(canaries\)](#)
- [CloudWatch RUM](#)
- [Perform launches and A/B experiments with CloudWatch Evidently](#)

Application Signals

Use CloudWatch Application Signals to automatically instrument your applications on AWS so that you can monitor current application health and track long-term application performance against your business objectives. Application Signals provides you with a unified, application-centric view of your applications, services, and dependencies, and helps you monitor and triage application health.

- Enable Application Signals to automatically collect metrics and traces from your applications, and display key metrics such as call volume, availability, latency, faults, and errors. Quickly see and triage current operational health, and whether your applications are meeting their longer-term performance goals, without writing custom code or creating dashboards.
- Create and monitor [service-level objectives \(SLOs\)](#) with Application Signals. Easily create and track status of SLOs related to CloudWatch metrics, including the new standard application metrics that Application Signals collects. See and track the [service level indicator \(SLI\)](#) status of your application services within a services list and topology map. Create alarms to track your SLOs, and track the new standard application metrics that Application Signals collects.
- See a map of your application topology that Application Signals automatically discovers, that gives you a visual representation of your applications, dependencies, and their connectivity.
- Application Signals works with [CloudWatch RUM](#), [CloudWatch Synthetics canaries](#), [AWS Service Catalog AppRegistry](#), and Amazon EC2 Auto Scaling to display your client pages, Synthetics canaries, and application names within dashboards and maps.

Use Application Signals for daily application monitoring

Use Application Signals within the CloudWatch console, as part of daily application monitoring:

1. If you have created service level objectives (SLOs) for your services, start with the [Service Level Objectives \(SLO\)](#) page. This gives you an immediate view of the health of your most critical services and operations. Choose the service or operation name for an SLO to open the [Service detail](#) page and see detailed service information as you troubleshoot issues.
2. Open the [Services](#) page to see a summary of all your services, and quickly see services with the highest fault rate or latency. If you have created SLOs, look at the Services table to see which services have unhealthy service level indicators (SLIs). If a particular service is in an unhealthy state, select the service to open the [Service detail](#) page and see service operations, dependencies, Synthetics canaries, and client requests. Select a point in a graph to see correlated traces so that you can troubleshoot and identify the root cause of operational issues.
3. If new services have been deployed or dependencies have changed, open the [Service Map](#) to inspect your application topology. See a map of your applications that shows the relationship between clients, Synthetics canaries, services, and dependencies. Quickly see SLI health, view key metrics such as call volume, fault rate, and latency, and drill down to see more detailed information in the [Service detail](#) page.

Using Application Signals incurs charges. For information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Note

It is not necessary to enable Application Signals to use CloudWatch Synthetics, CloudWatch RUM, or CloudWatch Evidently. However, Synthetics and CloudWatch RUM work with Application Signals to provide benefits when you use these features together.

Supported languages and architectures

Application Signals supports Java applications and Python applications.

Application Signals is supported and tested on Amazon EKS, Amazon ECS, and Amazon EC2. On Amazon EKS clusters, it automatically discovers the names of your services and clusters. On other architectures, you must supply the names of services and environments when you enable those services for Application Signals.

The instructions for enabling Application Signals on Amazon EC2 should work on any architecture that supports the CloudWatch agent and AWS Distro for OpenTelemetry. However, the instructions have not been tested on architectures other than Amazon ECS and Amazon EC2.

Supported Regions

Application Signals is supported in every commercial Region except for Canada West (Calgary).

Topics

- [Permissions required for Application Signals](#)
- [Enable Application Signals](#)
- [Monitor the operational health of your applications with Application Signals](#)
- [Standard application metrics collected](#)

Permissions required for Application Signals

This section explains the permissions necessary for you to enable, manage, and operate Application Signals.

Permissions to enable and manage Application Signals

To manage Application Signals, you must be signed on with the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
      "Effect": "Allow",
      "Action": "application-signals:*",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": "*"
    }
  ],
  {
```

```

    "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
},
{
    "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
    "Effect": "Allow",
    "Action": [
        "logs:StartQuery",
        "logs:DescribeMetricFilters"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
},
{
    "Sid": "CloudWatchApplicationSignalsLogsPermissions",
    "Effect": "Allow",
    "Action": [
        "logs:GetQueryResults",
        "logs:StopQuery"
    ],
    "Resource": "*"
},
{
    "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
    "Effect": "Allow",
    "Action": [
        "synthetics:DescribeCanaries",
        "synthetics:DescribeCanariesLastRun",
        "synthetics:GetCanaryRuns"
    ],
    "Resource": "*"
},
{
    "Sid": "CloudWatchApplicationSignalsRumPermissions",
    "Effect": "Allow",
    "Action": [
        "rum:BatchCreateRumMetricDefinitions",
        "rum:BatchDeleteRumMetricDefinitions",
        "rum:BatchGetRumMetricDefinitions",
        "rum:GetAppMonitor",

```

```

        "rum:GetAppMonitorData",
        "rum:ListAppMonitors",
        "rum:PutRumMetricsDestination",
        "rum:UpdateRumMetricDefinition"
    ],
    "Resource": "*"
},
{
    "Sid": "CloudWatchApplicationSignalsXrayPermissions",
    "Effect": "Allow",
    "Action": [
        "xray:GetTraceSummaries"
    ],
    "Resource": "*"
},
{
    "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
    "Effect": "Allow",
    "Action": "cloudwatch:PutMetricAlarm",
    "Resource": [
        "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
        "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
        "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
    ]
},
{
    "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
        }
    }
},
{
    "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
    "Effect": "Allow",
    "Action": "iam:GetRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
},

```

```

{
  "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
  "Effect": "Allow",
  "Action": [
    "sns:CreateTopic",
    "sns:Subscribe"
  ],
  "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
},
{
  "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
  "Effect": "Allow",
  "Action": "sns:ListTopics",
  "Resource": "*"
}
]
}

```

To enable Application Signals on Amazon EC2, Kubernetes, or custom architectures, see [Enable Application Signals on other platforms with a custom setup](#). To enable and manage Application Signals on Amazon EKS using the [Amazon CloudWatch Observability EKS add-on](#), you need the following permissions.

Important

These permissions include `iam:PassRole` with Resource `"*"` and `eks:CreateAddon` with Resource `"*"`. These are powerful permissions and you should use caution in granting them.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsEksAddonManagementPermissions",
      "Effect": "Allow",
      "Action": [
        "eks:AccessKubernetesApi",
        "eks:CreateAddon",
        "eks:DescribeAddon",
        "eks:DescribeAddonConfiguration",

```

```

        "eks:DescribeAddonVersions",
        "eks:DescribeCluster",
        "eks:DescribeUpdate",
        "eks:ListAddons",
        "eks:ListClusters",
        "eks:ListUpdates",
        "iam:ListRoles",
        "iam:PassRole"
    ],
    "Resource": "*"
},
{
  "Sid":
  "CloudWatchApplicationSignalsEksCloudWatchObservabilityAddonManagementPermissions",
  "Effect": "Allow",
  "Action": [
    "eks:DeleteAddon",
    "eks:UpdateAddon"
  ],
  "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
}
]
}

```

The Application Signals dashboard shows the AWS Service Catalog AppRegistry applications that your SLOs are associated with. To see these applications in the SLO pages, you must have the following permissions:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
      "Effect": "Allow",
      "Action": "tag:GetResources",
      "Resource": "*"
    }
  ]
}

```


Operating Application Signals

Service operators who are using Application Signals to monitor services and SLOs must be signed on to an account with the following read only permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
      "Effect": "Allow",
      "Action": [
        "application-signals:BatchGet*",
        "application-signals:Get*",
        "application-signals:List*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
      "Effect": "Allow",
      "Action": "iam:GetRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
      "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:StartQuery",
        "logs:DescribeMetricFilters"
      ],
      "Resource": "arn:aws:logs::*:log-group:/aws/application-signals/data:*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsLogsPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:GetQueryResults",
        "logs:StopQuery"
      ],
      "Resource": "*"
    }
  ],
}
```

```

    {
      "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
      "Effect": "Allow",
      "Action": [
        "synthetics:DescribeCanaries",
        "synthetics:DescribeCanariesLastRun",
        "synthetics:GetCanaryRuns"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
      "Effect": "Allow",
      "Action": [
        "rum:BatchGetRumMetricDefinitions",
        "rum:GetAppMonitor",
        "rum:GetAppMonitorData",
        "rum:ListAppMonitors"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
      "Effect": "Allow",
      "Action": [
        "xray:GetTraceSummaries"
      ],

```

```

        "Resource": "*"
    }
]
}

```

To see which AWS Service Catalog AppRegistry Applications that your SLOs are associated within the Application Signals dashboard, you must have the following permissions:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
      "Effect": "Allow",
      "Action": "tag:GetResources",
      "Resource": "*"
    }
  ]
}

```

To check if Application Signals on Amazon EKS using the [Amazon CloudWatch Observability EKS add-on](#) is enabled, you need to have the following permissions:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsEksReadPermissions",
      "Effect": "Allow",
      "Action": [
        "eks:ListAddons",
        "eks:ListClusters"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsEksDescribeAddonReadPermissions",
      "Effect": "Allow",
      "Action": [
        "eks:DescribeAddon"
      ],
      "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
    }
  ]
}

```

```
}  
  ]  
}
```

Enable Application Signals

The topics in this section explain how to enable CloudWatch Application Signals in your environment. Application Signals is supported on Amazon EKS clusters with a setup workflow using the console. It's also supported on other platforms, including Amazon EC2, with a custom setup process.

Topics

- [Application Signals supported systems](#)
- [OpenTelemetry compatibility considerations](#)
- [Enable Application Signals on Amazon EKS clusters](#)
- [Enable Application Signals on Amazon EC2, Amazon ECS, or Kubernetes](#)
- [Update to required versions of agents or Amazon EKS add-on](#)
- [Troubleshooting your Application Signals installation](#)
- [Configuring Application Signals](#)

Application Signals supported systems

Application Signals is supported and tested on Amazon EKS, native Kubernetes, Amazon ECS, and Amazon EC2. The instructions for enabling Application Signals on Amazon EC2 should work on any platform that supports the CloudWatch agent and AWS Distro for OpenTelemetry, but the instructions have not been tested on other platforms.

Java compatibility

Application Signals supports Java applications, and supports the same Java libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see [Supported libraries, frameworks, application servers, and JVMs](#).

JVM versions 8, 11, and 17 are supported.

Python compatibility

Application Signals supports the same libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see **Supported packages** at [opentelemetry-python-contrib](#).

Python versions 3.8 and later are supported.

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant](#).
- For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation](#).
 - Use the `--noreload` flag to prevent automatic reloading.
 - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

OpenTelemetry compatibility considerations

To onboard your applications with CloudWatch Application Signals, we recommend that you completely remove any existing application performance monitoring solution from your application beforehand. This includes removing any instrumentation code and configurations.

Even though Application Signals uses OpenTelemetry instrumentation, it is not guaranteed to be compatible with your existing OpenTelemetry instrumentation or configuration. In a best-case scenario, you might be able to keep some of your OpenTelemetry functionality, such as custom metrics. However, be sure to read the following sections for details.

Considerations if you already use OpenTelemetry

If you are already using OpenTelemetry with your application, the rest of this section contains important information to achieve compatibility with Application Signals.

- Before you enable your application for Application Signals, you must remove the injection of any other auto-instrumentation agents based on OpenTelemetry from your application. This helps

avoid configuration conflicts. You can continue to use manual instrumentation using compatible OpenTelemetry APIs along with Application Signals.

- If you are using manual instrumentation to generate custom spans or metrics from your application, then depending on the complexity of the instrumentation, enabling Application Signals might cause them to stop generating data or have other undesirable behavior. You might be able to use some of the available configurations in OpenTelemetry (except for the ones mentioned in the table later in this section) to retain the desired behavior of your existing metrics or spans. For more information about these configurations, see [SDK Configuration](#) in the OpenTelemetry documentation.

For example, by using the `OTEL_EXPORTER_OTLP_METRICS_ENDPOINT` configuration and a self-managed OpenTelemetry Collector instance, you might be able to continue sending your custom metrics to your desired destination.

- Some environment variables or system properties must not be used with Application Signals, while you can use others as long as you follow the guidance in the table. See the following table for details.

| Environment variable | Recommendation with Application Signals |
|---|--|
| General environment variables | |
| <code>OTEL_SDK_DISABLED</code> | Must not be set to <code>true</code> . |
| <code>OTEL_TRACES_EXPORTER</code> | Must be set to <code>otlp</code> . |
| <code>OTEL_LOGS_EXPORTER</code> | Set to <code>none</code> to disable other logs exporters. |
| <code>OTEL_EXPORTER_OTLP_ENDPOINT</code> | Must not be used. |
| <code>OTEL_EXPORTER_OTLP_TRACES_ENDPOINT</code> | Must not be used. |
| <code>OTEL_ATTRIBUTE_COUNT_LIMIT</code> | If set, must be set high enough to include approximately 10 more span attributes that are added by CloudWatch Application Signals. |
| <code>OTEL_EXPORTER_OTLP_PROTOCOL</code> | Must be set to <code>http/protobuf</code> |

| Environment variable | Recommendation with Application Signals |
|--|--|
| OTEL_PROPAGATORS | If set, must include <code>xray</code> for tracing. |
| OTEL_TRACES_SAMPLER | If set, must be <code>xray</code> to use X-Ray centralized sampling.

To use local sampling, set this to <code>parentbased_traceidratio</code> and specify the sampling rate in <code>OTEL_TRACES_SAMPLER_ARG</code> . |
| OTEL_TRACES_SAMPLER_ARG | If you are using the default of X-Ray centralized trace sample, this variable must not be used.

If you are using local sampling instead, set the sampling rate in this variable. For example, <code>0.05</code> for a 5% sampling rate. |
| Java-specific environment variables | |
| OTEL_JAVA_ENABLED_RESOURCE_PROVIDERS | If set, must include AWS resource detectors. |
| Python-specific environment variables | |
| OTEL_PYTHON_CONFIGURATOR | If used, must be set to <code>aws_configurator</code> |
| OTEL_PYTHON_DISTRO | If used, must be set to <code>aws_distro</code> |

Enable Application Signals on Amazon EKS clusters

CloudWatch Application Signals is supported for Java and Python applications running in Amazon EKS clusters. To enable Application Signals for applications in an Amazon EKS cluster, you have two options:

- To enable Application Signals for your applications on an existing Amazon EKS cluster, use the steps in [Enable Application Signals on an Amazon EKS cluster with your services](#).

- To try out Application Signals in a non-production environment with a sample application, use the instructions in [Enable Application Signals on a new Amazon EKS cluster with a sample app](#). This workflow uses scripts provided by AWS to create a new Amazon EKS cluster and install a sample application enabled for Application Signals. This lets you see and test the end-to-end functionality of Application Signals.

Topics

- [Enable Application Signals on an Amazon EKS cluster with your services](#)
- [Enable Application Signals on a new Amazon EKS cluster with a sample app](#)

Enable Application Signals on an Amazon EKS cluster with your services

To enable CloudWatch Application Signals on your applications on an existing Amazon EKS cluster, use the instructions in this section.

Important

If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see [OpenTelemetry compatibility considerations](#) before you enable Application Signals.

To enable Application Signals for your applications on an existing Amazon EKS cluster

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Services**.
3. If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. To do so, do the following. You need to do this only once for your account.
 - a. Choose **Start discovering your Services**.
 - b. Select the check box and choose **Start discovering Services**.

Completing this step for the first time in your account creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see [Service-linked role permissions for CloudWatch Application Signals](#).

4. Choose **Enable Application Signals**.
5. For **Specify platform**, choose **EKS**.
6. For **Select an EKS cluster**, select the cluster where you want to enable Application Signals.
7. If this cluster does not already have the Amazon CloudWatch Observability EKS add-on enabled, you are prompted to enable it. If this is the case, do the following:
 - a. Choose **Add CloudWatch Observability EKS add-on**. The Amazon EKS console appears.
 - b. Select the check box for **Amazon CloudWatch Observability** and choose **Next**.

The CloudWatch Observability EKS add-on enables both Application Signals and CloudWatch Container Insights with enhanced observability for Amazon EKS. For more information about Container Insights, see [Container Insights](#).

- c. Select the most recent version of the add-on to install.
- d. Select an IAM role to use for the add-on. If you choose **Inherit from node**, attach the correct permissions to the IAM role used by your worker nodes. Replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \  
--role-name my-worker-node-role \  
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \  
--policy-arn arn:aws:iam::aws:policy/AWSXRayWriteOnlyAccess
```

- e. If you want to create a service role to use the add-on, see [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart](#).
- f. Choose **Next**, confirm the information on the screen, and choose **Create**.

- g. In the next screen, choose **Enable CloudWatch Application Signals** to return to the CloudWatch console and finish the process.
8. There are two options for enabling your applications for Application Signals. For consistency, we recommend that you choose one option per cluster.
 - The **Console** option is simpler. Using this method causes your pods to immediately restart.
 - The **Annotate Manifest File** method gives you more control of when your pods restart, and can also help you manage your monitoring in a more decentralized way if you don't want to centralize it.

Console

The **Console** option uses the advanced configuration of the Amazon CloudWatch Observability EKS add-on to setup Application Signals for your services. For more information about the add-on, see [\(Optional\) Additional configuration](#).

If you don't see a list of workloads and namespaces, ensure you have the right permissions to view them for this cluster. For more information, see [Required permissions](#).

You can monitor single workloads or entire namespaces.

To monitor a single workload:

1. Select the check box by the workload that you want to monitor.
2. Select the language of the workload. For Python applications, ensure your application follows the required prerequisites before continuing. For more information, see [Python application doesn't start after Application Signals is enabled](#).
3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject AWS Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To monitor an entire namespace:

1. Select the check box by the namespace that you want to monitor.
2. Select the language of the workload. This applies it to all workloads in this namespace, whether they are currently deployed or will be deployed in the future. For Python applications, ensure your application follows the required prerequisites before

continuing. For more information, see [Python application doesn't start after Application Signals is enabled](#).

3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject AWS Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

Annotate manifest file

In the CloudWatch console, the **Monitor Services** section explains that you must add an annotation to a manifest YAML in the cluster. Adding this annotation auto-instruments the application to send metrics, traces, and logs to Application Signals.

You have two options for the annotation:

- **Annotate Workload** auto-instruments a single workload in the cluster.
- **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps:

- To annotate a single workload:
 1. Choose **Annotate Workload**.
 2. Paste one of the following lines into the PodTemplate section of the workload manifest file.
 - **For Java workloads:** `annotations: instrumentation.opentelemetry.io/inject-java: "true"`
 - **For Python workloads:** `annotations: instrumentation.opentelemetry.io/inject-python: "true"`

For Python applications, there are additional required configurations. For more information, see [Python application doesn't start after Application Signals is enabled](#).

3. In your terminal, enter `kubectl apply -f your_deployment_yaml` to apply the change.

- To annotate all workloads in a namespace:
 1. Choose **Annotate Namespace**.
 2. Paste one of the following lines into the metadata section of the namespace manifest file. If the namespace includes both Java and Python workloads, paste both of these lines into the namespace manifest file.

- **If there are Java workloads in the namespace:** annotations:
`instrumentation.opentelemetry.io/inject-java: "true"`
- **If there are Python workloads in the namespace:** annotations:
`instrumentation.opentelemetry.io/inject-python: "true"`

For Python applications, there are additional required configurations. For more information, see [Python application doesn't start after Application Signals is enabled](#).

3. In your terminal, enter `kubectl apply -f your_namespace_yaml` to apply the change.
 4. In your terminal, enter a command to restart all pods in the namespace. An example command to restart deployment workloads is `kubectl rollout restart deployment -n namespace_name`
9. Choose **View Services when done**. This takes you to the Application Signals Services view, where you can see the data that Application Signals is collecting. It might take a few minutes for data to appear.

To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

For more information about the **Services** view, see [Monitor the operational health of your applications with Application Signals](#).

Note

We've identified some considerations that you should keep in mind when enabling Python applications for Application Signals. For more information, see [Python application doesn't start after Application Signals is enabled](#).

Enable Application Signals on a new Amazon EKS cluster with a sample app

To try out CloudWatch Application Signals on a sample app before you instrument your own applications with it, follow the instructions in this section. These instructions use scripts to help you create an Amazon EKS cluster, install a sample application, and instrument the sample application to work with Application Signals.

The sample application is a Spring “Pet Clinic” application that is composed of four microservices. These services run on Amazon EKS on Amazon EC2 and leverage Application Signals enablement scripts to enable the cluster with the Java or Python auto-instrumentation agent.

Requirements

- Currently, Application Signals monitors only Java and Python applications.
- You must have the AWS CLI installed on the instance. We recommend AWS CLI version 2, but version 1 should also work. For more information about installing the AWS CLI, see [Install or update the latest version of the AWS CLI](#).
- The scripts in this section are intended to be run in Linux and macOS environments. For Windows instances, we recommend that you use an AWS Cloud9 environment to run these scripts. For more information about AWS Cloud9, see [What is AWS Cloud9?](#)
- Install a supported version of `kubectl`. You must use a version of `kubectl` within one minor version difference of your Amazon EKS cluster control plane. For example, a 1.26 `kubectl` client works with Kubernetes 1.25, 1.26, and 1.27 clusters. If you already have an Amazon EKS cluster, you might need to configure AWS credentials for `kubectl`. For more information, see [Creating or updating a kubeconfig file for an Amazon EKS cluster](#).
- Install `eksctl`. `eksctl` uses the AWS CLI to interact with AWS, which means it uses the same AWS credentials as the AWS CLI. For more information, see [Installing or updating eksctl](#).
- Install `jq`. `jq` is required to run the Application Signals enablement scripts. For more information, see [Download jq](#).

Step 1: Download the scripts

To download the scripts to set up CloudWatch Application Signals with a sample app, you can download and uncompress the zipped GitHub project file to a local drive, or you can clone the GitHub project.

To clone the project, open a terminal window and enter the following Git command in a given working directory.

```
git clone https://github.com/aws-observability/application-signals-demo.git
```

Step 2: Build and deploy the sample application

To build and push the sample application images, [follow these instructions](#).

Step 3: Deploy and enable Application Signals and the sample application

Be sure that you have completed the requirements listed in [Enable Application Signals on a new Amazon EKS cluster with a sample app](#) before you complete the following steps.

To deploy and enable Application Signals and the sample application

1. Enter the following command. Replace *new-cluster-name* with the name that you want to use for the new cluster. Replace *region-name* with the name of the AWS Region, such as us-west-1.

This command sets up the sample app running in a new Amazon EKS cluster with Application Signals enabled.

```
# this script sets up a new cluster, enables Application Signals, and deploys the
# sample application
cd application-signals-demo/scripts/eks/appsignals/one-step && ./setup.sh new-
cluster-name region-name
```

The setup script takes about 30 minutes to run, and does the following:

- Creates a new Amazon EKS cluster in the specified Region.
- Creates the necessary IAM permissions for Application Signals (arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess and arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy).
- Enables Application Signals by installing the CloudWatch agent and Auto-instrumenting the sample application for CloudWatch metrics and X-Ray traces.
- Deploys the PetClinic Spring sample application in the same Amazon EKS cluster.
- Creates five CloudWatch Synthetics canaries, named pc-add-vist, pc-create-owners, pc-visit-pet, pc-visit-vet, pc-clinic-traffic. These canaries will run at a one-minute frequency to generate synthetic traffic for the sample app and demonstrate how Synthetics canaries appear in Application Signals.

- Creates four service level objectives (SLOs) for the PetClinic application with the following names:
 - **Availability for Searching an Owner**
 - **Latency for Searching an Owner**
 - **Availability for Registering an Owner**
 - **Latency for Registering an Owner**
 - Creates the required IAM role with a custom trust policy granting Application Signals the following permissions:
 - `cloudwatch:PutMetricData`
 - `cloudwatch:GetMetricData`
 - `xray:GetServiceGraph`
 - `logs:StartQuery`
 - `logs:GetQueryResults`
2. (Optional) If you want to review the source code for the PetClinic sample application, you can find them under the root folder.

```
- application-signals-demo
  - spring-petclinic-admin-server
  - spring-petclinic-api-gateway
  - spring-petclinic-config-server
  - spring-petclinic-customers-service
  - spring-petclinic-discovery-server
  - spring-petclinic-vets-service
  - spring-petclinic-visits-service
```

3. To view the deployed PetClinic sample application, run the following command to find the URL:

```
kubectl get ingress
```

Step 4: Monitor the sample application

After completing the steps in the previous section to create the Amazon EKS cluster and deploy the sample application, you can use Application Signals to monitor the application.

Note

For the Application Signals console to start populating, some traffic must reach the sample application. Part of the previous steps created CloudWatch Synthetics canaries that generate traffic to the sample application.

Service health monitoring

After it is enabled, CloudWatch Application Signals automatically discovers and populates a list of services without requiring any additional setup.

To view the list of discovered services and monitor their health

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Services**.
3. To view a service, its operations, and its dependencies, choose the name of one of the services in the list.

This unified, application-centric view helps provide a full perspective of how users are interacting with your service. This can help you triage issues if performance anomalies occur. For complete details about the **Services** view, see [Monitor the operational health of your applications with Application Signals](#).

4. Choose the **Service Operations** tab to see the standard application metrics for that service's operations. The operations are the API operations that the service calls, for example.

Then, to view the graphs for a single operation of that service, choose that operation name.

5. Choose the **Dependencies** tab to see the dependencies that your application has, along with the critical application metrics for each dependency. Dependencies include AWS services and third-party services that your application calls.
6. To view correlated traces from the service details page, choose a data point in one of the three graphs above the table. This populates a new pane with filtered traces from the time period. These traces are sorted and filtered based on the graph that you chose. For example, if you chose the **Latency** graph, the traces are sorted by service response time.
7. In the CloudWatch console navigation pane, choose **SLOs**. You see the SLOs that the script created for the sample application. For more information about SLOs, see [Service level objectives \(SLOs\)](#).

(Optional) Step 5: Cleanup

When you're finished testing Application signals, you can use a script provided by Amazon to clean up and delete the artifacts created in your account for the sample application. To perform the cleanup, enter the following command. Replace *new-cluster-name* with the name of the cluster that you created for the sample app, and replace *region-name* with the name of the AWS Region, such as `us-west-1`.

```
cd application-signals-demo/scripts/eks/appsignals/one-step && ./cleanup.sh new-cluster-name region-name
```

Enable Application Signals on Amazon EC2, Amazon ECS, or Kubernetes

Enable CloudWatch Application Signals on platforms other than Amazon EKS by using the custom setup steps in these sections. On these architectures, you install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself.

On these architectures, Application Signals doesn't autodiscover the names of your services or their clusters or hosts. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Topics

- [Use a custom setup to enable Application Signals on Amazon ECS](#)
- [Enable Application Signals on Amazon EC2](#)
- [Enable Application Signals on Kubernetes platforms](#)

Use a custom setup to enable Application Signals on Amazon ECS

Use these custom setup instructions to onboard your applications on Amazon ECS to CloudWatch Application Signals. You install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself.

Important

Only the `awsipc` network mode is supported. Both the EC2 and Fargate launch types are supported.

On Amazon ECS clusters, Application Signals doesn't autodiscover the names of your services. You must specify your service names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Step 1: Enable Application Signals in your account

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. To do so, do the following. You need to do this only once for your account.

To enable Application Signals for your applications

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Services**.
3. Choose **Start discovering your Services**.
4. Select the check box and choose **Start discovering Services**.

Completing this step for the first time in your account creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see [Service-linked role permissions for CloudWatch Application Signals](#).

Step 2: Create IAM roles

You must create an IAM role. If you already have created this role, you might need to add permissions to it.

- **ECS task role**— Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy**.

For more information about creating IAM roles, see [Creating IAM Roles](#).

Step 3: Prepare CloudWatch agent configuration

First, prepare the agent configuration with Application Signals enabled. To do this, create a local file named `/tmp/ecs-cwagent.json`.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

Then upload this configuration to the SSM Parameter Store. To do this, enter the following command. In the file, replace `$REGION` with your actual Region name.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

Step 4: Instrument your application with the CloudWatch agent

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace `$REGION` with your actual Region name. Replace `$IMAGE` with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": true,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace `$IMAGE` with the latest image from the [AWS Distro for OpenTelemetry Amazon ECR image repository](#).

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
```

```

    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}

```

4. Add the following environment variables to your application container. You must be using version 1.32.2 or later of the AWS Distro for OpenTelemetry auto-instrumentation agent for Java

| Environment variable | Setting to enable Application Signals |
|--------------------------|---|
| OTEL_RESOURCE_ATTRIBUTES | <p>Specify the following information as key-value pairs:</p> <ul style="list-style-type: none"> • <code>service.name</code> sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of <code>UnknownService</code> is used. • <code>deployment.environment</code> sets the environment that the application runs in. This will be displayed as the Hosted In environment of your application in Application Signals dashboards. If you don't specify this, the default of <code>generic:default</code> is used. <p>This attribute key is used only by Application Signals, and is converted into X-Ray</p> |

| Environment variable | Setting to enable Application Signals |
|---|--|
| | <p>trace annotations and CloudWatch metric dimensions.</p> <p>(Optional) To enable trace log correlation, set an additional environment variable <code>aws.log.group.names</code> to be the log group name for your application log. By doing so, the traces from your application can be correlated with the relevant log entries from this specific log group. For this variable, replace <i><code>\$YOUR_APPLICATION_LOG_GROUP</code></i> with the log group name for your application. You'll also need to change the logging configuration in your application. For more information, see Enable trace log correlation.</p> |
| <code>OTEL_AWS_APPLICATION_SIGNALS_ENABLED</code> | Set to <code>true</code> to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals. |
| <code>OTEL_METRICS_EXPORTER</code> | Set to <code>none</code> to disable other metrics exporters. |
| <code>OTEL_LOGS_EXPORTER</code> | Set to <code>none</code> to disable other logs exporters. |
| <code>OTEL_EXPORTER_OTLP_PROTOCOL</code> | Set to <code>http/protobuf</code> to send metrics and traces to Application Signals using HTTP. |
| <code>OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT</code> | Set to <code>http://localhost:4316/v1/metrics</code> to send metrics to the CloudWatch sidecar. |

| Environment variable | Setting to enable Application Signals |
|------------------------------------|---|
| OTEL_EXPORTER_OTLP_TRACES_ENDPOINT | Set to <code>http://localhost:4316/v1/traces</code> to send traces to the CloudWatch sidecar. |
| OTEL_TRACES_SAMPLER | Set this to <code>xray</code> to set X-Ray as the traces sampler. |
| OTEL_PROPAGATORS | Set <code>xray</code> as one of the propagators. |
| JAVA_TOOL_OPTIONS | Set to <code>" -javaagent:\$ <i>AWS_ADOT_JAVA_INSTRUMENTATION_PATH</i> "</code>
Replace <i>AWS_ADOT_JAVA_INSTRUMENTATION_PATH</i> with the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example, <code>/otel-auto-instrumentation/javaagent.jar</code> |

5. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure.

For a Java application, use the following.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
"aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,aws.hostedIn.environment=
$HOST_ENV,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
```

```
    "value": "none"
  },
  {
    "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
    "value": "http/protobuf"
  },
  {
    "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
    "value": "true"
  },
  {
    "name": "JAVA_TOOL_OPTIONS",
    "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
  },
  {
    "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
    "value": "http://localhost:4316/v1/metrics"
  },
  {
    "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
    "value": "http://localhost:4316/v1/traces"
  },
  {
    "name": "OTEL_TRACES_SAMPLER",
    "value": "xray"
  },
  {
    "name": "OTEL_PROPAGATORS",
    "value": "tracecontext,baggage,b3,xray"
  }
],
"mountPoints": [
  {
    "sourceVolume": "opentelemetry-auto-instrumentation",
    "containerPath": "/otel-auto-instrumentation",
    "readOnly": false
  }
]
}
```


Python

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant](#).
- For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation](#).
 - Use the `--noreload` flag to prevent automatic reloading.
 - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

To instrument your Python application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [  
  {  
    "name": "opentelemetry-auto-instrumentation-python"  
  }  
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace `$REGION` with your actual Region name. Replace `$IMAGE` with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

```
{  
  "name": "ecs-cwagent",  
  "image": "$IMAGE",  
  "essential": true,  
  "secrets": [  
    {
```

```
    "name": "CW_CONFIG_CONTENT",
    "valueFrom": "ecs-cwagent"
  }
],
"logConfiguration": {
  "logDriver": "awslogs",
  "options": {
    "awslogs-create-group": "true",
    "awslogs-group": "/ecs/ecs-cwagent",
    "awslogs-region": "$REGION",
    "awslogs-stream-prefix": "ecs"
  }
}
}
```

3. Append a new container `init` to your application's task definition. Replace `$IMAGE` with the latest image from the [AWS Distro for OpenTelemetry Amazon ECR image repository](#).

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "-a",
    "/autoinstrumentation/.",
    "/otel-auto-instrumentation-python"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

4. Add the following environment variables to your application container.

| Environment variable | Setting to enable Application Signals |
|--------------------------|---|
| OTEL_RESOURCE_ATTRIBUTES | <p>Specify the following information as key-value pairs:</p> <ul style="list-style-type: none">• <code>service.name</code> sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of <code>UnknownService</code> is used.• <code>deployment.environment</code> sets the environment that the application runs in. This will be displayed as the Hosted In environment of your application in Application Signals dashboards. If you don't specify this, the default of <code>generic:default</code> is used. <p>This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.</p> <p>(Optional) To enable trace log correlation, set an additional environment variable <code>aws.log.group.names</code> to be the log group name for your application log. By doing so, the traces from your application can be correlated with the relevant log entries from this specific log group. For this variable, replace <i><code>\$YOUR_APPLICATION_LOG_GROUP</code></i> with the log group name for your application. You'll also need to change the logging</p> |

| Environment variable | Setting to enable Application Signals |
|--|---|
| | configuration in your application. For more information, see Enable trace log correlation . |
| OTEL_AWS_APPLICATION_SIGNALS_ENABLED | Set to <code>true</code> to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals. |
| OTEL_METRICS_EXPORTER | Set to <code>none</code> to disable other metrics exporters. |
| OTEL_EXPORTER_OTLP_PROTOCOL | Set to <code>http/protobuf</code> to send metrics and traces to CloudWatch using HTTP. |
| OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT | Set to <code>http://127.0.0.1:4316/v1/metrics</code> to send metrics to the CloudWatch sidecar. |
| OTEL_EXPORTER_OTLP_TRACES_ENDPOINT | Set to <code>http://127.0.0.1:4316/v1/traces</code> to send traces to the CloudWatch sidecar. |
| OTEL_TRACES_SAMPLER | Set this to <code>xray</code> to set X-Ray as the traces sampler. |
| OTEL_PROPAGATORS | Add <code>xray</code> as one of the propagators. |
| OTEL_PYTHON_DISTRO | Set to <code>aws_distro</code> to use the ADOT Python instrumentation. |
| OTEL_PYTHON_CONFIGURATOR | Set to <code>aws_configuration</code> to use the ADOT Python configuration. |

| Environment variable | Setting to enable Application Signals |
|------------------------|--|
| PYTHONPATH | Replace <code>\$APP_PATH</code> with the location of the application's working directory within the container. This is required for the Python interpreter to find your application modules. |
| DJANGO_SETTINGS_MODULE | Required only for Django applications. Set it to the location of your Django application's <code>settings.py</code> file. Replace <code>\$PATH_TO_SETTINGS</code> . |

5. Mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure. In the following example, the value for `OTEL_RESOURCE_ATTRIBUTES` includes an optional `aws.log.group.names` that helps enable trace log correlation. For more information, see the table in the previous step.

For a Python application, use the following.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
  ],
}
```

```

    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
"aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,aws.hostedIn.environment=
$HOST_ENV,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}

```

```
    }  
  ]  
}
```

Step 5: Deploy your application

Create a new revision of your task definition and deploy it to your application cluster. You should see three containers in the newly created task:

- `init`
- `ecs-cwagent`
- `my-app`

Enable Application Signals on Amazon EC2

For applications running on Amazon EC2 or other instance types, you install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

The instructions in this section are for both Java applications and Python applications. The steps have been tested on Amazon EC2 instances, but are also expected to work on other architectures that support AWS Distro for OpenTelemetry.

Requirements

- To get support for Application Signals, you must use the most recent version of both the CloudWatch agent and the AWS Distro for OpenTelemetry agent.
- You must have the AWS CLI installed on the instance. We recommend AWS CLI version 2, but version 1 should also work. For more information about installing the AWS CLI, see [Install or update the latest version of the AWS CLI](#).

⚠ Important

If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see [OpenTelemetry compatibility considerations](#) before you enable Application Signals.

Step 1: Enable Application Signals in your account

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. To do so, do the following. You need to do this only once for your account.

To enable Application Signals for your applications

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Services**.
3. Choose **Start discovering your Services**.
4. Select the check box and choose **Start discovering your Services**.

When you complete this step for the first time in your account, Application Signals creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see [Service-linked role permissions for CloudWatch Application Signals](#).

Step 2: Download and start the CloudWatch agent

To install the CloudWatch agent as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the CloudWatch agent to the instance. If the instance already has the CloudWatch agent installed, you might need to update it. Only versions of the agent released on November 30, 2023 or later support CloudWatch Application Signals.

For information about downloading the CloudWatch agent, see [Download the CloudWatch agent package](#).

2. Before you start the CloudWatch agent, configure it to enable Application Signals. The following example is a CloudWatch agent configuration that enables Application Signals for both metrics and traces on an EC2 host.

We recommend that you place this file at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on Linux systems.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

3. Attach the **CloudWatchAgentServerPolicy** IAM policy to the IAM role of your Amazon EC2 instance. For permissions for on-premises hosts, see [Permissions for on-premises servers](#).
 - a. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. Choose **Roles** and find the role used by your Amazon EC2 instance. Then choose the name of that role.
 - c. In the **Permissions** tab, choose **Add permissions, Attach policies**.

- d. Find **CloudWatchAgentServerPolicy**. Use the search box if needed. Then select the check box for that policy and choose **Add permissions**.
4. Start the CloudWatch agent by entering the following commands. Replace *agent-config-file-path* with the path to the CloudWatch agent configuration file, such as `./amazon-cloudwatch-agent.json`. You must include the `file:` prefix as shown.

```
export CONFIG_FILE_PATH=./amazon-cloudwatch-agent.json
```

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \  
-a fetch-config \  
-m ec2 -s -c file:agent-config-file-path
```

Permissions for on-premises servers

For an on-premises host, you will need to provide AWS authorization to your device.

To set up permissions for an on-premises host

1. Create the IAM user to be used to provide permissions to your on-premises host:
 - a. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. Choose **Users**, **Create User**.
 - c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for AWS that will be used to authenticate your host. Then choose **Next**.
 - d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
 - e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.
 - f. On the **Review and create** page, ensure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.
 - g. Choose **Create user**.
2. Create and retrieve your AWS access key and secret key:
 - a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
 - b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.

- c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
 - d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.
 - e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.
 - f. Choose **Done**.
3. Configure your AWS credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Step 3: Instrument your application and start it

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your Java applications as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the AWS Distro for OpenTelemetry Java auto-instrumentation agent. You can download the latest version by using [this link](#). You can view information about all released versions at [aws-otel-java-instrumentation Releases](#).
2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.
 - For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:
 - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- (Optional) `deployment.environment` sets the environment that the application runs in. This will be displayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
 - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`
 - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
 - If this is an on-premises host, it is set to `generic:default`

This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

- For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`
- For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`
- For the `JAVA_TOOL_OPTIONS` variable, specify the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored.

```
export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
```

For example:

```
export AWS_ADOT_JAVA_INSTRUMENTATION_PATH=./aws-opentelemetry-agent.jar
```

- For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to `none`. This disables other metrics exporters so that only the Application Signals exporter is used.


```
java -jar $MY_JAVA_APP.jar
```

Python

To instrument your Python applications as part of enabling Application Signals on an Amazon EC2 instance

1. Download the latest version of the AWS Distro for OpenTelemetry Python auto-instrumentation agent. Install it by running the following command.

```
pip install aws-opentelemetry-distro
```

You can view information about all released versions at [AWS Distro for OpenTelemetry Python instrumentation](#).

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.
 - a. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:
 - `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.
 - `deployment.environment` sets the environment that the application runs in. This will be displayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
 - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:name-of-Auto-Scaling-group`.
 - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
 - If this is an on-premises host, it is set to `generic:default`

This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

- b. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, specify `http/protobuf` to export telemetry data over HTTP to the CloudWatch agent endpoints listed in the following steps.
 - c. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`
 - d. For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`
 - e. For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to `none`. This disables other metrics exporters so that only the Application Signals exporter is used.
 - f. Set the `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` variable to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
3. Start your application with the environment variables discussed in the previous step. The following is an example of a starting script.
 - Replace `$SVC_NAME` with your application name. This will be displayed as the name of the application, in Application Signals dashboards.
 - Replace `$PYTHON_APP` with the location and name of your application.

```
OTEL_METRICS_EXPORTER=none \  
OTEL_LOGS_EXPORTER=none \  
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \  
OTEL_PYTHON_DISTRO=aws_distro \  
OTEL_PYTHON_CONFIGURATOR=aws_configurator \  
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \  
OTEL_TRACES_SAMPLER=xray \  
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \  

```

```

OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
\
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES=service.name=$SVC_NAME \
opentelemetry-instrument python $MY_PYTHON_APP.py

```

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant](#).
 - For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation](#).
 - Use the `--noreload` flag to prevent automatic reloading.
 - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.
4. (Optional) To enable trace log correlation, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. Then the traces from your application can be correlated with the relevant log entries from this specific log group. To enable trace log correlation, you also need to change the logging configuration in your application. For more information, see [Enable trace log correlation](#).

The following is an example of a starting script that helps enable trace log correlation.

```

OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
\

```



```
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \  
OTEL_RESOURCE_ATTRIBUTES=aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,aws.hosted.in.en\  
\  
java -jar $MY_PYTHON_APP.jar
```

Enable Application Signals on Kubernetes platforms

To enable applications running on Kubernetes systems other than Amazon EKS, follow the instructions in this section. Application Signals supports both Java applications and Python applications.

Requirements

- You have admin permission on the Kubernetes cluster where you are enabling Application Signals.
- You must have the AWS CLI installed on the environment where your Kubernetes cluster is running. For more information about installing the AWS CLI, see [Install or update the latest version of the AWS CLI](#).
- You have `kubectl` and `helm` installed on your local terminal. For more information, see the [kubectl](#) and [Helm](#) documentation.

Step 1: Enable Application Signals in your account

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. To do so, do the following. You need to do this only once for your account.

To enable Application Signals for your applications

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Services**.
3. Choose **Start discovering your Services**.
4. Select the check box and choose **Start discovering your Services**.

When you complete this step for the first time in your account, Application Signals creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see [Service-linked role permissions for CloudWatch Application Signals](#).

Step 2: Install the CloudWatch agent operator in your cluster

Installing the CloudWatch agent operator will install the operator, the CloudWatch agent, and other auto-instrumentation into your cluster. To do so, enter the following command. Replace `$REGION` with your AWS Region. Replace `$YOUR_CLUSTER_NAME` with the name that you want to appear for your cluster in Application Signals dashboards.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm install amazon-cloudwatch-operator aws-observability/amazon-cloudwatch-
observability \
--namespace amazon-cloudwatch --create-namespace \
--set region=$REGION \
--set clusterName=$YOUR_CLUSTER_NAME
```

For more information, see [amazon-cloudwatch-observability](#) on GitHub.

Step 3: Set up AWS credentials for your Kubernetes clusters

Important

If your Kubernetes cluster is hosted on Amazon EC2, you can skip this section and proceed to [Step 4: Add annotations](#).

If your Kubernetes cluster is hosted on-premises, you must use the instructions in this section to add AWS credentials to your Kubernetes environment.

To set up permissions for an on-premises Kubernetes cluster

1. Create the IAM user to be used to provide permissions to your on-premises host:
 - a. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. Choose **Users**, **Create User**.
 - c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for AWS that will be used to authenticate your host. Then choose **Next**.
 - d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
 - e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.
 - f. On the **Review and create** page, ensure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.
 - g. Choose **Create user**.
2. Create and retrieve your AWS access key and secret key:
 - a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
 - b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.
 - c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
 - d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.
 - e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.
 - f. Choose **Done**.
3. Configure your AWS credentials in your on-premises host by entering the following command. Replace **ACCESS_KEY_ID** and **SECRET_ACCESS_ID** with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step. By default, the credential file is saved in `/home/user/.aws/credentials`.

```
$ aws configure --profile AmazonCloudWatchAgent
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

4. Edit the custom resource that the CloudWatch agent installed using the Helm chart to add the newly created AWS credentials secret.

```
kubectl edit amazoncloudwatchagent cloudwatch-agent -n amazon-cloudwatch
```

5. While your file editor is open mount the AWS credentials into the CloudWatch agent container by adding the following configuration to the top of the deployment. Replace the path `/home/user/.aws/credentials` with the location of your local AWS credentials file.

```
apiVersion: cloudwatch.aws.amazon.com/v1alpha1
kind: AmazonCloudWatchAgent
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  volumeMounts:
  - mountPath: /rootfs
    volumeMounts:
    - name: aws-credentials
      mountPath: /root/.aws
      readOnly: true
  volumes:
  - hostPath:
      path: /home/user/.aws/credentials
      name: aws-credentials
---
```

Step 4: Add annotations

The next step is to instrument your application for CloudWatch Application Signals by adding a language-specific [annotation](#) to your Kubernetes [workload](#) or [namespace](#). This annotation auto-instruments your application to send metrics, traces, and logs to Application Signals.

To add the annotations for Application Signals

1. You have two options for the annotation:
 - **Annotate Workload** auto-instruments a single workload in a cluster.
 - **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps.

2. To annotate a single workload, enter one of the following commands. Replace `$WORKLOAD_TYPE` and `$WORKLOAD_NAME` with the actual values for your workload.

- For Java workloads:

```
kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template": {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-java": "true"}}}}}'
```

- For Python workloads:

```
kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template": {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-python": "true"}}}}}'
```

For Python applications, there are additional required configurations. For more information, see [Python application doesn't start after Application Signals is enabled](#).

3. To annotate all workloads in a namespace, enter one of the following commands. Replace `$NAMESPACE` with the name of your namespace.

If the namespace includes both Java and Python workloads, add both annotations to the namespace.

- For Java workloads in the namespace:

```
kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-java=true
```

- For Python workloads in the namespace:

```
kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-python=true
```

For Python applications, there are additional required configurations. For more information, see [Python application doesn't start after Application Signals is enabled](#).

After adding the annotations, restart all pods in the namespace by entering the following command:

```
kubectl rollout restart
```

4. When the previous steps are completed, in the CloudWatch console, choose **Application Signals, Services**. This opens the dashboards where you can see the data that Application Signals collects. It might take a few minutes for data to appear.

For more information about the **Services** view, see [Monitor the operational health of your applications with Application Signals](#).

Update to required versions of agents or Amazon EKS add-on

After August 9, 2024, CloudWatch Application Signals will no longer support older versions of the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent, and the AWS Distro for OpenTelemetry auto-instrumentation agent.

- For the Amazon CloudWatch Observability EKS add-on, versions older than `v1.7.0-eksbuild.1` won't be supported.
- For the CloudWatch agent, versions older than `1.300040.0` won't be supported.
- For the AWS Distro for OpenTelemetry auto-instrumentation agent:
 - For Java, versions older than `1.32.2` won't be supported.
 - For Python, versions older than `0.2.0` won't be supported.

Important

The latest versions of the agents include updates to the Application Signals metric schema. These updates are not backward compatible, and this can result in data issues if incompatible versions are used. To help ensure a seamless transition to the new functionality, do the following:

- If your application is running on Amazon EKS, be sure to restart all your instrumented applications after you update the Amazon CloudWatch Observability add-on.

- For applications running on other platforms, be sure to upgrade **both** the CloudWatch agent and the AWS OpenTelemetry auto-instrumentation agent to the latest versions.

The instructions in the following sections can help you update to a supported version.

Contents

- [Update the Amazon CloudWatch Observability EKS add-on](#)
 - [Use the console](#)
 - [Use the AWS CLI](#)
- [Update the CloudWatch agent and ADOT agent](#)
 - [Update on Amazon ECS](#)
 - [Update on Amazon EC2 or other architectures](#)

Update the Amazon CloudWatch Observability EKS add-on

To the Amazon CloudWatch Observability EKS add-on, you can use the AWS Management Console or the AWS CLI.

Use the console

To upgrade the add-on using the console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the name of the Amazon EKS cluster to update.
3. Choose the **Add-ons** tab, then choose **Amazon CloudWatch Observability**.
4. Choose **Edit**, select the version you want to update to, and then choose **Save changes**.

Be sure to choose `v1.7.0-eksbuild.1` or later.

5. Enter one of the following AWS CLI commands to restart your services.

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

Use the AWS CLI

To upgrade the add-on using the AWS CLI

1. Enter the following command to find the latest version.

```
aws eks describe-addon-versions \  
--addon-name amazon-cloudwatch-observability
```

2. Enter the following command to update the add-on. Replace *\$VERSION* with a version that is v1.7.0-eksbuild.1 or later. Replace *\$AWS_REGION* and *\$CLUSTER* with your Region and cluster name.

```
aws eks update-addon \  
--region $AWS_REGION \  
--cluster-name $CLUSTER \  
--addon-name amazon-cloudwatch-observability \  
--addon-version $VERSION \  
# required only if the advanced configuration is used.  
--configuration-values $JSON_CONFIG
```

Note

If you're using a custom configuration for the add-on, you can find an example of the configuration to use for *\$JSON_CONFIG* in [Enable CloudWatch Application Signals](#).

3. Enter one of the following AWS CLI commands to restart your services.

```
# Restart a deployment  
kubectl rollout restart deployment/name  
# Restart a daemonset  
kubectl rollout restart daemonset/name  
# Restart a statefulset  
kubectl rollout restart statefulset/name
```

Update the CloudWatch agent and ADOT agent

If your services are running on architectures other than Amazon EKS, you will need to upgrade both the CloudWatch agent and the ADOT auto-instrumentation agent to use the latest Application Signals features.

Update on Amazon ECS

To upgrade your agents for services running on Amazon ECS

1. Create a new task definition revision. For more information, see [Updating a task definition using the console](#).
2. Replace the \$IMAGE of the ecs-cwagent container with the latest image tag from [cloudwatch-agent](#) on Amazon ECR.

If you upgrade to a fixed version, be sure to use a version equal to or later than 1.300040.0.

3. Replace the \$IMAGE of the init container with the latest image tag from the following locations:
 - For Java, use [aws-observability/adot-autoinstrumentation-java](#)

If you upgrade to a fixed version, be sure to use a version equal to or later than 1.32.2.
 - For Python, use [aws-observability/adot-autoinstrumentation-python](#)

If you upgrade to a fixed version, be sure to use a version equal to or later than 0.2.0.
4. Update the Application Signals environment variables in your app container by following the instructions at [Step 4: Instrument your application with the CloudWatch agent](#).
5. Deploy your service with the new task definition.

Update on Amazon EC2 or other architectures

To upgrade your agents for services running on Amazon EC2 or other architectures

1. Follow the instructions at [Download the CloudWatch agent package](#) and upgrade the CloudWatch agent to the latest version. Be sure to select version 1.300040.0 or later.
2. Download the latest version of the AWS Distro for OpenTelemetry auto-instrumentation agent from one of the following locations:

- For Java, use [aws-otel-java-instrumentation](#)

If you upgrade to a fixed version, be sure to choose 1.32.2 or later.

- For Python, use [aws-otel-python-instrumentation](#)

If you upgrade to a fixed version, be sure to choose 0.2.0 or later.

3. Apply the updated Application Signals environment variables to your application, then start your application. For more information, see [Step 3: Instrument your application and start it.](#)

Troubleshooting your Application Signals installation

This section contains troubleshooting tips for CloudWatch Application Signals.

Topics

- [Application doesn't start after Application Signals is enabled](#)
- [Python application doesn't start after Application Signals is enabled](#)
- [No application data in Application Signals dashboard](#)
- [Service metrics or dependency metrics have Unknown values](#)
- [Handling a ConfigurationConflict when managing the Amazon CloudWatch Observability EKS add-on](#)
- [I want to filter out unnecessary metrics and traces](#)
- [What does InternalOperation mean?](#)
- [Can I disable FluentBit?](#)
- [Can I filter container logs before exporting to the CloudWatch Logs?](#)

Application doesn't start after Application Signals is enabled

If your application on an Amazon EKS cluster doesn't start after you enable Application Signals on the cluster, check for the following:

- Check if the application has been instrumented by another monitoring solution. Application Signals might not support co-existing with other instrumentation solutions.
- Confirm that your the application meets the compatibility requirements to use Application Signals. For more information, see [Application Signals supported systems](#).
- If your application failed to pull the Application Signals artifacts such as the AWS Distro for OpenTelemetry Java or Python agent and CloudWatch agent images, it could be a network issue.

To mitigate the issue, remove the annotation `instrumentation.opentelemetry.io/inject-java: "true"` or `instrumentation.opentelemetry.io/inject-python: "true"`

from your application deployment manifest, and re-deploy your application. Then check if the application is working.

Python application doesn't start after Application Signals is enabled

It is a known issue in OpenTelemetry auto-instrumentation that a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, ensure that you set the PYTHONPATH environment variable to the location of your application's working directory. For more information about this issue, see [Python autoinstrumentation setting of PYTHONPATH is not compliant with Python's module resolution behavior, breaking Django applications](#).

For Django applications, there are additional required configurations, which are outlined in the [OpenTelemetry Python documentation](#).

- Use the `--noreload` flag to prevent automatic reloading.
- Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

No application data in Application Signals dashboard

If metrics or traces are missing in the Application Signals dashboards, the following might be causes. Investigate these causes only if you have waited 15 minutes for Application Signals to collect and display data since your last update.

- Make sure that your library and framework you are using is supported by the ADOT Java agent. For more information, see [Libraries / Frameworks](#).
- Make sure that the CloudWatch agent is running. First check the status of the CloudWatch agent pods and make sure they are all in Running status.

```
kubectl -n amazon-cloudwatch get pods.
```

Add the following to the CloudWatch agent configuration file to enable debugging logs, and then restart the agent.

```
"agent": {
```

```
"region": "${REGION}",  
"debug": true  
,
```

Then check for errors in the CloudWatch agent pods.

- Check for configuration issues with the CloudWatch agent. Confirm that the following is still in the CloudWatch agent configuration file and the agent has been restarted since it was added.

```
"agent": {  
  "region": "${REGION}",  
  "debug": true  
},
```

Then check the OpenTelemetry debugging logs for error messages such as `ERROR io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export . . .`. These messages might indicate the problem.

If that doesn't solve the issue, dump and check the environment variables with names that start with `OTEL_` by describing the pod with the `kubectl describe pod` command.

- To enable the OpenTelemetry Python debug logging, set the environment variable `OTEL_PYTHON_LOG_LEVEL` to `debug` and redeploy the application.
- Check for wrong or insufficient permissions for exporting data from the CloudWatch agent. If you see `Access Denied` messages in the CloudWatch agent logs, this might be the issue. It is possible that the permissions applied when you installed the CloudWatch agent were later changed or revoked.
- Check for an AWS Distro for OpenTelemetry (ADOT) issue when generating telemetry data.

Make sure that the instrumentation annotations `instrumentation.opentelemetry.io/inject-java` and `sidecar.opentelemetry.io/inject-java` are applied to the application deployment and the value is `true`. Without these, the application pods will not be instrumented even if the ADOT addon is installed correctly.

Next, check if the `init` container is applied on the application and the `Ready` state is `True`. If the `init` container is not ready, see the status for the reason.

If the issue persists, enable debug logging on the OpenTelemetry Java SDK by setting the environment variable `OTEL_JAVAAGENT_DEBUG` to `true` and redeploying the application. Then look for messages that start with `ERROR io.telemetry`.

- The metric/span exporter might be dropping data. To find out, check the application log for messages that include `Failed to export...`
- The CloudWatch agent might be getting throttled when sending metrics or spans to Application Signals. Check for messages indicating throttling in the CloudWatch agent logs.
- Make sure that you've enabled the service discovery setup. You need to do this only once in your Region.

To confirm this, in the CloudWatch console choose **Application Signals, Services**. If Step 1 is not marked **Complete**, choose **Start discovering your services**. Data should start flowing in within five minutes.

Service metrics or dependency metrics have Unknown values

If you see **UnknownService**, **UnknownOperation**, **UnknownRemoteService**, or **UnknownRemoteOperation** for a dependency name or operation in the Application Signals dashboards, check whether the occurrence of data points for the unknown remote service and unknown remote operation are coinciding with their deployments.

- **UnknownService** means that the name of an instrumented application is unknown. If the `OTEL_SERVICE_NAME` environment variable is undefined and `service.name` isn't specified in `OTEL_RESOURCE_ATTRIBUTES`, the service name is set to `UnknownService`. To fix this, specify the service name in `OTEL_SERVICE_NAME` or `OTEL_RESOURCE_ATTRIBUTES`.
- **UnknownOperation** means that the name of an invoked operation is unknown. This occurs when Application Signals is unable to discover an operation name which invokes the remote call, or when the extracted operation name contains high cardinality values.
- **UnknownRemoteService** means that the name of the destination service is unknown. This occurs when the system is unable to extract the destination service name that the remote call accesses.

One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.service` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteService`. For more information about customizations in the CloudWatch agent, see [Enable CloudWatch Application Signals](#).

- **UnknownRemoteOperation** means that the name of the destination operation is unknown. This occurs when the system is unable to extract the destination operation name that the remote call accesses.

One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.operation` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteOperation`. For more information about customizations in the CloudWatch agent, see [Enable CloudWatch Application Signals](#).

Handling a ConfigurationConflict when managing the Amazon CloudWatch Observability EKS add-on

When you install or update the Amazon CloudWatch Observability EKS add-on, if you notice a failure caused by a `Health Issue` of type `ConfigurationConflict` with a description that starts with `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`, it is likely because you already have the CloudWatch agent and its associated components such as the `ServiceAccount`, the `ClusterRole` and the `ClusterRoleBinding` installed on the cluster. When the add-on tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the Amazon CloudWatch Observability EKS add-on when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see [Deleting the CloudWatch agent and Fluent Bit for Container Insights](#) for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the AWS CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

I want to filter out unnecessary metrics and traces

If Application Signals is collecting traces and metrics that you don't want, see [Manage high-cardinality operations](#) for information about configuring the CloudWatch agent with custom rules to reduce cardinality.

For information about customizing trace sampling rules, see [Configure sampling rules](#) in the X-Ray documentation.

What does InternalOperation mean?

An InternalOperation is an operation that is triggered by the application internally rather than by an external invocation. Seeing InternalOperation is expected, healthy behavior.

Some typical examples where you would see InternalOperation include the following:

- **Preloading on start**– Your application performs an operation named `loadDatafromDB` which reads metadata from a database during the warm up phase. Instead of observing `loadDatafromDB` as a service operation, you'll see it categorized as an InternalOperation.
- **Async execution in the background**– Your application subscribes to an event queue, and processes streaming data accordingly whenever there's an update. Each triggered operation will be under InternalOperation as a service operation.
- **Retrieving host information from a service registry**– Your application talks to a service registry for service discovery. All interactions with the discovery system are classified as an InternalOperation.

Can I disable FluentBit?

You can disable FluentBit by configuring the Amazon CloudWatch Observability EKS add-on. For more information, see [\(Optional\) Additional configuration](#).

Can I filter container logs before exporting to the CloudWatch Logs?

No, filtering container logs is not yet supported.

Configuring Application Signals

This section contains information about configuring CloudWatch Application Signals.

Trace sampling rate

By default, when you enable Application Signals X-Ray centralized sampling is enabled using the default sampling rate settings of `reservoir=1/s` and `fixed_rate=5%`. The environment variables for the AWS Distro for OpenTelemetry (ADOT) SDK agent as set as follows.

| Environment variable | Value | Note |
|-------------------------|---|----------------------------------|
| OTEL_TRACES_SAMPLER | xray | |
| OTEL_TRACES_SAMPLER_ARG | endpoint=http://cloudwatch-agent.amazon-cloudwatch:2000 | Endpoint of the CloudWatch agent |

For information about changing the sampling configuration, see the following:

- To change X-Ray sampling, see [Configure sampling rules](#)
- To change ADOT sampling, see [Configuring the OpenTelemetry Collector for X-Ray remote sampling](#)

If you want to disable X-Ray centralized sampling and use local sampling instead, set the following values for the ADOT SDK Java agent as below. The following example sets the sampling rate at 5%.

| Environment variable | Value |
|-------------------------|--------------------------|
| OTEL_TRACES_SAMPLER | parentbased_traceidratio |
| OTEL_TRACES_SAMPLER_ARG | 0.05 |

For information about more advanced sampling settings, see [OTEL_TRACES_SAMPLER](#).

Enable trace log correlation

You can enable *trace log correlation* in Application Signals. This automatically injects trace IDs and span IDs into the relevant application logs. Then, when you open a trace detail page in the Application Signals console, the relevant log entries (if any) that correlate with the current trace automatically appear at the bottom of the page.

For example, suppose you notice a spike in a latency graph. You can choose the point on the graph to load the diagnostics information for that point in time. You then choose the relevant trace to get more information. When you view the trace information, you can scroll down to see the logs associated with the trace. These logs might reveal patterns or error codes associated with the issues causing the latency spike.

To achieve trace log correlation, Application Signals relies on the [Logger MDC auto-instrumentation](#) for Java and the [OpenTelemetry Logging Instrumentation](#) for Python. Both are provided by OpenTelemetry community. Application Signals uses this to inject trace contexts such as trace ID and span ID into application logs. To enable this, you must manually change your logging configuration to enable the auto-instrumentation.

Depending on the architecture that your application runs on, you might have to also set an environment variable to enable trace log correlation, in addition to following the steps in this section.

- On Amazon EKS, no further steps are needed.
- On Amazon ECS, see steps 4 and 5 in the procedure in [Step 4: Instrument your application with the CloudWatch agent](#).
- On Amazon EC2, see the step 4 in the procedure in [Step 3: Instrument your application and start it](#).

After you enable trace log correlation,

Trace log correlation setup examples

This section contains examples of setting up trace log correlation in several environments.

Spring Boot for Java

Suppose you have a Spring Boot application in a folder called `custom-app`. The application configuration is usually a YAML file named `custom-app/src/main/resources/application.yml` that might look like this:

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}
```

...

To enable trace log correlation, add the following logging configuration.

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...

logging:
  pattern:
    level: trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags}
    %5p
```

Logback for Java

In the logging configuration (such as logback.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into pattern of Encoder. For example, the following configuration prepends the trace context before the log message.

```
<appender name="FILE" class="ch.qos.logback.core.FileAppender">
  <file>app.log</file>
  <append>true</append>
  <encoder>
    <pattern>trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n</pattern>
  </encoder>
</appender>
```

For more information about encoders in Logback, see [Encoders](#) in the Logback documentation.

Log4j2 for Java

In the logging configuration (such as log4j2.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into PatternLayout. For example, the following configuration prepends the trace context before the log message.

```
<Appenders>
  <File name="FILE" fileName="app.log">
    <PatternLayout pattern="trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n"/>
  </File>
</Appenders>
```

For more information about pattern layouts in Log4j2, see [Pattern Layout](#) in the Log4j2 documentation.

Log4j for Java

In the logging configuration (such as log4j.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `PatternLayout`. For example, the following configuration prepends the trace context before the log message.

```
<appender name="FILE" class="org.apache.log4j.FileAppender">;
  <param name="File" value="app.log"/>;
  <param name="Append" value="true"/>;
  <layout class="org.apache.log4j.PatternLayout">;
    <param name="ConversionPattern" value="trace_id=%mdc{trace_id} span_id=
%mdc{span_id} trace_flags=%mdc{trace_flags} %5p - %m%n"/>;
  </layout>;
</appender>;
```

For more information about pattern layouts in Log4j, see [Class Pattern Layout](#) in the Log4j documentation.

Python

Set the environment variable `OTEL_PYTHON_LOG_CORRELATION` to `true` while running your application. For more information, see [Enable trace context injection](#) in the Python OpenTelemetry documentation.

Manage high-cardinality operations

Application Signals includes settings in the CloudWatch agent that you can use to manage the cardinality of your operations and manage the metric exportation to optimize costs. By default, the metric limiting function becomes active when the number of distinct operations for a service

over time exceeds the default threshold of 500. You can tune the behavior by adjusting the configuration settings.

Determine if metric limiting is activated

You can use the following methods to find if the default metric limiting is happening. If it is, you should consider optimizing the cardinality control by following the steps in the next section.

- In the CloudWatch console, choose **Application Signals, Services**. If you see an **Operation** named **AllOtherOperations** or a **RemoteOperation** named **AllOtherRemoteOperations**, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherOperations` for their `Operation` dimension, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherRemoteOperations` for their `RemoteOperation` dimension, then metric limiting is happening.

Optimize cardinality control

To optimize your cardinality control, you can do the following:

- Create custom rules to aggregate operations.
- Configure your metric limiting policy.

Create custom rules to aggregate operations

High-cardinality operations can sometimes be caused by inappropriate unique values extracted from the context. For example, sending out HTTP/S requests that include user IDs or session IDs in the path can lead to hundreds of disparate operations. To resolve such issues, we recommend that you configure the CloudWatch agent with customization rules to rewrite these operations.

In cases where there is a surge in generating numerous different metrics through individual `RemoteOperation` calls, such as `PUT /api/customer/owners/123`, `PUT /api/customer/owners/456`, and similar requests, we recommend that you consolidate these operations into a single `RemoteOperation`. One approach is to standardize all `RemoteOperation` calls that start with `PUT /api/customer/owners/` to a uniform format, specifically `PUT /api/customer/owners/{ownerId}`. The following example illustrates this. For information about other customization rules, see [Enable CloudWatch Application Signals](#).

```

{
  "logs":{
    "metrics_collected":{
      "application_signals":{
        "rules":[
          {
            "selectors":[
              {
                "dimension":"RemoteOperation",
                "match":"PUT /api/customer/owners/*"
              }
            ],
            "replacements":[
              {
                "target_dimension":"RemoteOperation",
                "value":"PUT /api/customer/owners/{ownerId}"
              }
            ],
            "action":"replace"
          }
        ]
      }
    }
  }
}

```

In other cases, high-cardinality metrics might have been aggregated to `AllOtherRemoteOperations`, and it might be unclear what specific metrics are included. The CloudWatch agent is able to log the dropped operations. To identify dropped operations, use the configuration in the following example to activate logging until the problem resurfaces. Then inspect the CloudWatch agent logs (accessible by container stdout or EC2 log files) and search for the keyword `drop metric data`.

```

{
  "agent": {
    "config": {
      "agent": {
        "debug": true
      },
      "traces": {
        "traces_collected": {
          "application_signals": {

```

```
    }
  }
},
"logs": {
  "metrics_collected": {
    "application_signals": {
      "limiter": {
        "log_dropped_metrics": true
      }
    }
  }
}
}
```

Create your metric limiting policy

If the default metric limiting configuration doesn't address the cardinality for your service, you can customize the metric limiter configuration. To do this, add a `limiter` section under the `logs/metrics_collected/application_signals` section in the CloudWatch Agent configuration file.

The following example lowers the threshold of metric limiting from 500 distinct metrics to 100.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "limiter": {
          "drop_threshold": 100
        }
      }
    }
  }
}
```

Monitor the operational health of your applications with Application Signals

Use Application Signals within the [CloudWatch console](#) to monitor and troubleshoot the operational health of your applications:

- **Monitor your application services** — As part of daily operational monitoring, use the [Services](#) page to see a summary of all your services. See services with the highest fault rate or latency, and see which services have unhealthy [service level indicators \(SLIs\)](#). Select a service to open the [Service detail](#) page and see detailed metrics, service operations, Synthetics canaries, and client requests. This can help you troubleshoot and identify the root cause of operational issues.
- **Inspect your application topology** — Use the [Service Map](#) to understand and monitor your application topology over time, including the relationships between clients, Synthetics canaries, services, and dependencies. Instantly see service level indicator (SLI) health and view key metrics such as call volume, fault rate, and latency. Drill down to see more detailed information in the [Service detail](#) page.

Explore an [example scenario](#) that demonstrates how these pages can be used to quickly troubleshoot an operational service health issue, from initial detection to identifying root cause.

How Application Signals enables operational health monitoring

After you [enable your application](#) for Application Signals, your application services, APIs, and their dependencies are automatically discovered and displayed in the **Services**, **Service detail**, and **Service Map** pages. Application Signals collects information from multiple sources to enable service discovery and operational health monitoring:

- [AWS Distro for OpenTelemetry \(ADOT\)](#) — As part of enabling Application Signals, OpenTelemetry Java and Python auto-instrumentation libraries are configured to emit metrics and traces that are collected by the CloudWatch agent. The metrics and traces are used to enable discovery of services, operations, dependencies, and other service information.
- [Service-level objectives \(SLOs\)](#) — After you create service level objectives for your services, the Services, Service detail, and Service Map pages display service level indicator (SLI) health. SLIs can monitor latency, availability, and other operational metrics.
- [CloudWatch Synthetics canaries](#) — When you configure X-Ray tracing on your canaries, calls to your services from your canary scripts are associated with your service and displayed within the Service detail page.
- [CloudWatch Real user monitoring \(RUM\)](#) — When X-Ray tracing is enabled on your CloudWatch RUM web client, requests to your services are automatically associated and displayed within the service detail page.
- [AWS Service Catalog AppRegistry](#) — Application Signals auto-discovers AWS resources within your account and allows you to group them into logical applications created in AppRegistry. The

application name displayed in the Services page is based on the underlying compute resource that your services are running on.

Note

Application Signals displays your services and operations based on metrics and traces emitted within the current time filter that you chose. (By default, this is the past three hours.) If there is no activity within the current time filter for a service, operation, dependency, Synthetics canary, or client page, it won't be displayed.

Currently, up to 1,000 services can be displayed. Discovery of your services and service topology might be delayed up to 10 minutes. Evaluation of your service level indicator (SLI) health might be delayed up to 15 minutes.

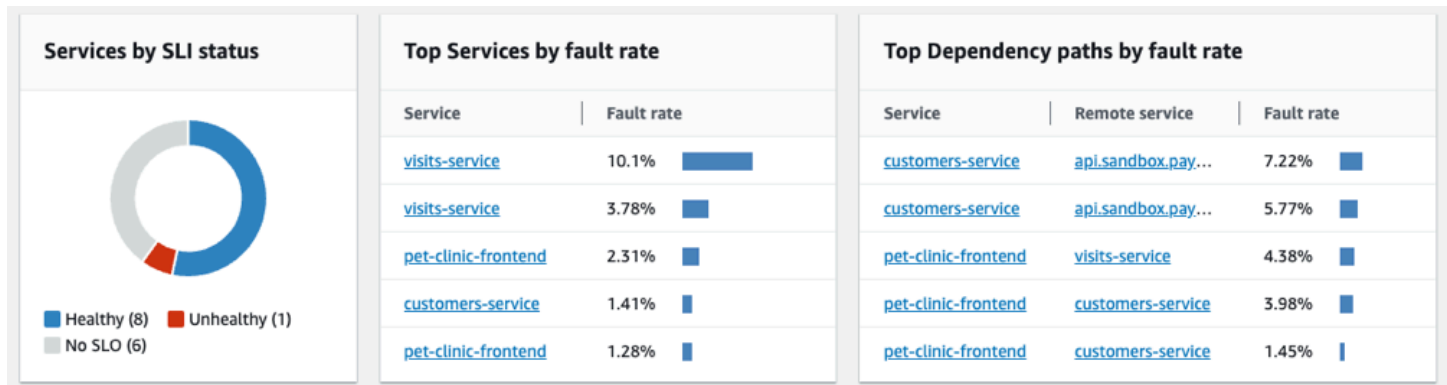
View overall service activity and operational health with the Services page

Use the Services page to see a list of your services that are [enabled for Application Signals](#). You can also view operational metrics and quickly see which services have unhealthy service level indicators (SLIs). Drill down to look for performance anomalies as you identify the root cause of operational issues. To view this page, open the [CloudWatch console](#) and choose **Services** under the **Application Signals** section in the left navigation pane.

Explore operational health metrics for your services

The top of the Services page includes an overall service operational health graph and several tables displaying top services and service dependencies by fault rate. The Services graph on the left displays a breakdown of the number of services that have healthy or unhealthy service level indicators (SLIs) during the current page-level time filter. SLIs can monitor latency, availability, and other operational metrics.

The two tables next to the graph display a list of top services by fault rate. Choose any service name in either table to open a [service detail page](#) and see detailed service operation details. Choose a dependency path to open the detail page and see service dependency details. Both tables display information for up to the past three hours, even if a longer time period filter is chosen at the top right of the page.



Monitor operational health with the Services table

The Services table displays a list of your services that have been enabled for Application Signals. Choose **Enable Application Signals** to open a setup page and start configuring your services. For more information, see [Enable Application Signals](#).

Filter the Services table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.

Services (8) [Info](#) Refresh Create SLO Enable Application Signals

Filter services and resources by text, property or value < 1 > Settings

Name	SLI Status	Application	Hosted in
customers-service	2 Healthy	-	Environment gamma/pet-clinic
customers-service	9 Healthy	Petclinic	Cluster petclinic-sampleApp > Namespace default > Workload customers-service
pet-clinic-frontend	Create SLO	-	Environment gamma/pet-clinic

Choose the name of any service in the table to view a [service detail page](#) containing service-level metrics, operations, and additional details. If you have associated the service's underlying compute resource with an application in AppRegistry or the Applications card on the AWS Management Console home page, choose the application name to display the application details in the [myApplications](#) console page. For services hosted in Amazon EKS, choose any link within the **Hosted in** column to view Cluster, Namespace, or Workload within CloudWatch Container Insights. For services running on Amazon ECS or Amazon EC2, the Environment value is shown.

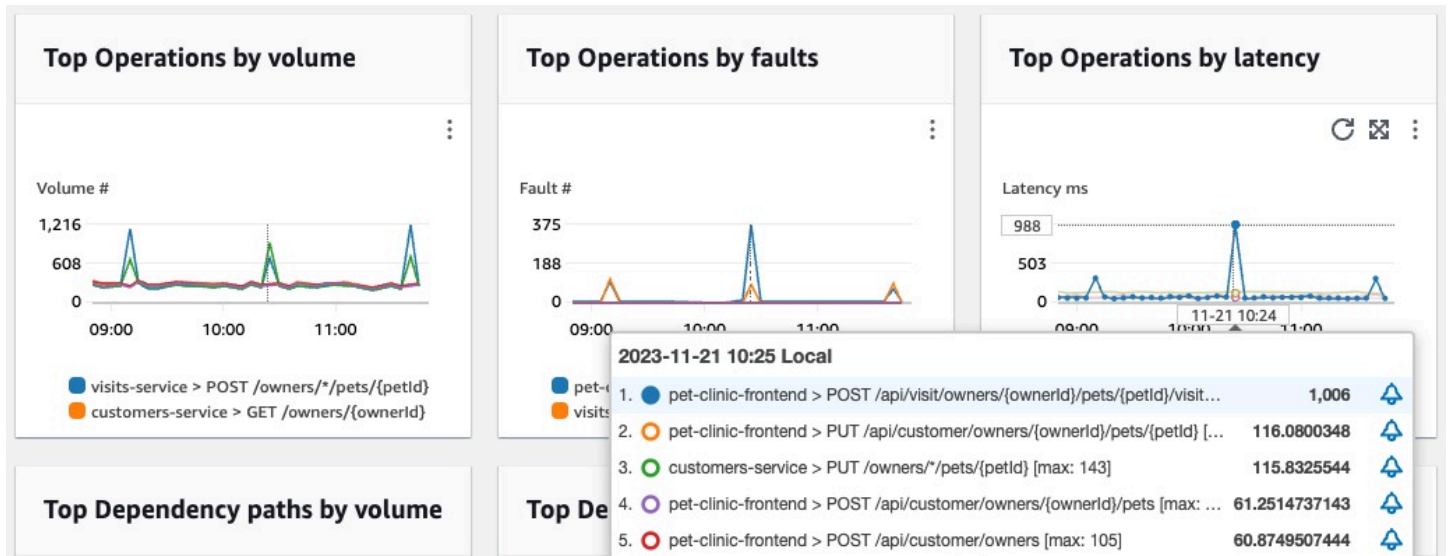
[Service level indicator \(SLI\)](#) status is displayed for each service in the table. Choose the SLI status for a service to display a pop-up containing a link to any unhealthy SLIs, and a link to see all SLOs for the service.

<input type="radio"/>	visits-service	⊗ 1/1 Unhealthy	Service health × 1/1 SLIs are unhealthy ⊗ Availability of Scheduling a Visit <hr/> View all SLO on service
<input type="radio"/>	customers-service	✔ 1 Healthy	
<input type="radio"/>	vets-service	<input type="button" value="Create SLO"/>	

If no SLOs have been created for a service, choose the **Create SLO** button within the **SLI Status** column. To create additional SLOs for any service, select the option button next to the service name, and then choose **Create SLO** at the top-right of the table. When you create SLOs, you can see at a glance which of your services and operations are performing well and which are unhealthy. See [service level objectives \(SLOs\)](#) for more information.

View top operation and dependency metrics

Below the Services table, you can view top operations and dependencies across all services by call volume, faults, and latency. This set of graphs gives you critical information about which operations or dependencies may be unhealthy across all services. Choose any point in a graph to see a pop-up containing more detailed series information. Hover over the series descriptions at the bottom of a graph to see a pop-up containing detailed metrics for a specific operation or dependency path. Select the context menu button at the top-right corner of a graph to see additional options, including viewing CloudWatch metrics or logs pages.



View detailed service activity and operational health with the service detail page

When you instrument your application, [Amazon CloudWatch Application Signals](#) maps all of the services that your application discovers. Use the service detail page to see an overview of your services, operations, dependencies, canaries, and client requests for a single service. To view the service detail page, do the following:

- Open the [CloudWatch console](#).
- Choose **Services** under the **Application Signals** section in the left navigation pane.
- Choose the name of any service from the **Services**, **Top services**, or dependency tables.

The service detail page is organized into the following tabs:

- [Overview](#) — Use this tab to see an overview of a single service, including the number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, top operations and dependencies. These metrics include time series data on latency, faults, and errors across all service operations for that service.
- [Service operations](#) — Use this tab to see a list of the operations that your service calls and interactive graphs with key metrics that measure the health of each operation. You can select a data point in a graph to obtain information about traces, logs, or metrics associated with that data point.
- [Dependencies](#) — Use this tab to see a list of dependencies that your service calls, and a list of metrics for those dependencies.
- [Synthetics canaries](#) — Use this tab to see a list of synthetics canaries that simulate user calls to your service, and key performance metrics for how those canaries.
- [Client pages](#) — Use this tab to see a list of client pages that call your service, and metrics that measure the quality of client interactions with your application.

View your service overview

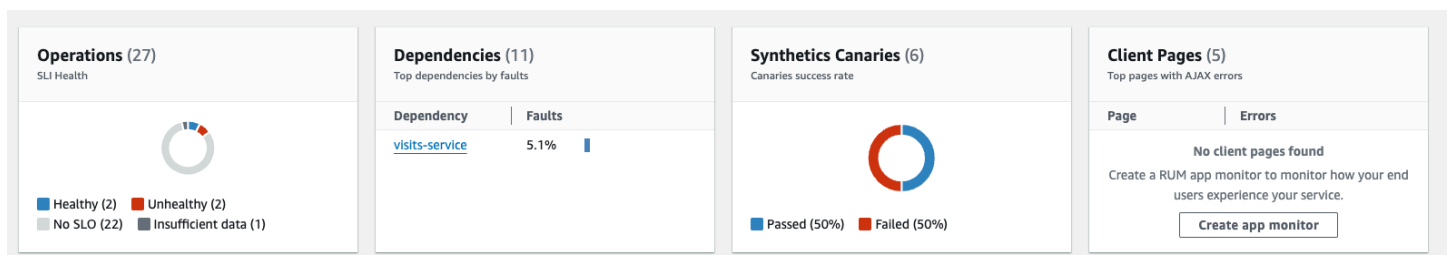
Use the service overview page to view a high-level summary of metrics for all service operations in a single location. Check the performance of all the operations, dependencies, client pages and synthetics canaries that interact with your application. Use this information to help you determine where to focus efforts to identify issues, troubleshoot errors, and find opportunities for optimization.

Choose any link in **Service Details** to view information that is related to a specific service. For example, for services hosted in Amazon EKS, the service details page shows **Cluster**, **Namespace**, and **Workload** information. For services hosted in Amazon ECS or Amazon EC2, the service details page shows the **Environment** value.

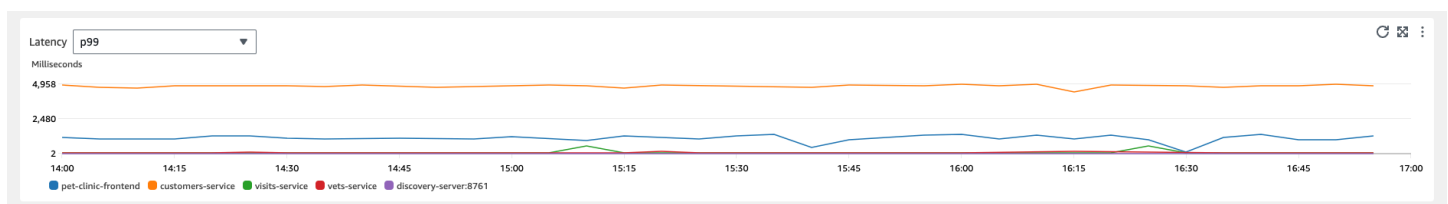
Under **Services**, the **Overview** tab displays a summary of the following:

- **Operations** – Use this tab to see the health of your service operations. The health status is determined by service level indicators (SLI) that are defined as a part of a [service level objective \(SLO\)](#).
- **Dependencies** – Use this tab to see top dependencies of the services called by your application, listed by fault rate.
- **Synthetics canaries** – Use this tab to see the result of simulated calls to endpoints or APIs associated with your service, and the number of failed canaries.
- **Client pages** – Use this tab to see top pages called by clients that have asynchronous JavaScript and XML (AJAX) errors.

The following illustration shows an overview of your services:

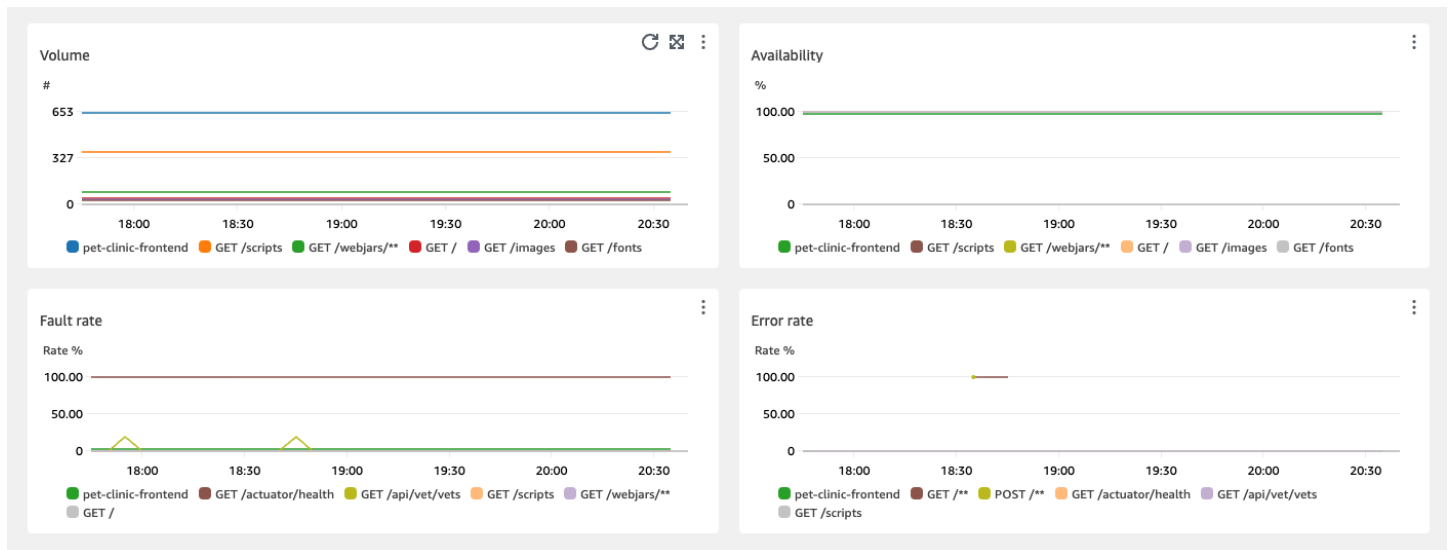


The **Overview** tab also displays a graph of dependencies with the highest latency across all services. Use the **p99**, **p90** and **p50** latency metrics to quickly assess which dependencies are contributing to your total service latency, as follows:



For example, the previous graph shows that 99% of the requests made to the **customer-service** dependency were completed in approximately 4,950 milliseconds. The other dependencies took less time.

Graphs displaying the top four service operations by latency show the volume of requests, availability, fault rate, and error rate for those services, as shown in the following image:



View your service operations

When you instrument your application, [Application Signals](#) discovers all of the service operations that your application calls. Use the **Service operations** tab to see a table that contains the service operations and a set of metrics that measure the performance of a selected operation. These metrics include SLI status, number of dependencies, latency, volume, faults, errors, and availability, as shown in the following image:

Name	SLI Status	Dependencies	Latency p99	Latency p90	Latency p50	Volume	Faults	Errors	Availability
POST /api/visit/owners/{ownerId}/pets/{petId}/visits	2 Healthy	1	517.9 ms	357.4 ms	8.3 ms	12.4K	10.6% (1316)	0% (0)	89.4%
POST /api/customer/owners	2 Healthy	1	9.4K ms	7.4K ms	3.3K ms	2.8K	0% (0)	0% (0)	100%
GET /api/customer/owners/{ownerId}/pets/{petId}	2 Healthy	1	8.3 ms	3.7 ms	2.8 ms	180	0% (0)	0% (0)	100%
GET /	2 Healthy	-	1 ms	0.8 ms	0.7 ms	1.5K	0% (0)	0% (0)	100%
PUT /api/customer/owners/{ownerId}/pets/{petId}	Create SLO	1	341.4 ms	121.2 ms	98.6 ms	180	0% (0)	0% (0)	100%

Filter the table to make it easier to find a service operation by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.

Choose the SLI status for an operation to display a popup containing a link to any unhealthy SLI, and a link to see all SLOs for the operation, as shown in the following table:

Name	SLI Status	Dependencies	Latency p99
<input checked="" type="radio"/> GET /api/customer/owners/{ownerId}/pets/{petId}	⊗ 1/2 Unhealthy		
<input type="radio"/> POST /api/visit/owners/{ownerId}/pets/{petId}/visits	✔ 2 Healthy		
<input type="radio"/> POST /api/customer/owners	✔ 2 Healthy		
<input type="radio"/> PUT /api/customer/owners/{ownerId}/pets/{petId}	✔ 2 Healthy		

Operation health ✕

1/2 SLIs are unhealthy

⊗ [Availability of Adding a Pet](#)

[View all SLO on operation](#)

The service operations table lists the SLI status, the number of healthy or unhealthy SLIs, and the total number of SLOs for each operation.

Use SLIs to monitor latency, availability, and other operational metrics that measure the operational health of a service. Use an SLO to check the performance and health status of your services and operations.

To create an SLO, do the following:

- If an operation does not have an SLO, choose the **Create SLO** button within the **SLI Status** column.
- If an operation already has an SLO, do the following:
 - Select the radio button next to the operation name.
 - Choose **Create SLO** from the **Actions** down arrow at the top right of the table.

For more information, see [service level objectives \(SLOs\)](#).

The **Dependencies** column shows the number of dependencies this operation calls. Choose this number to open the **Dependencies** tab filtered to the selected operation.

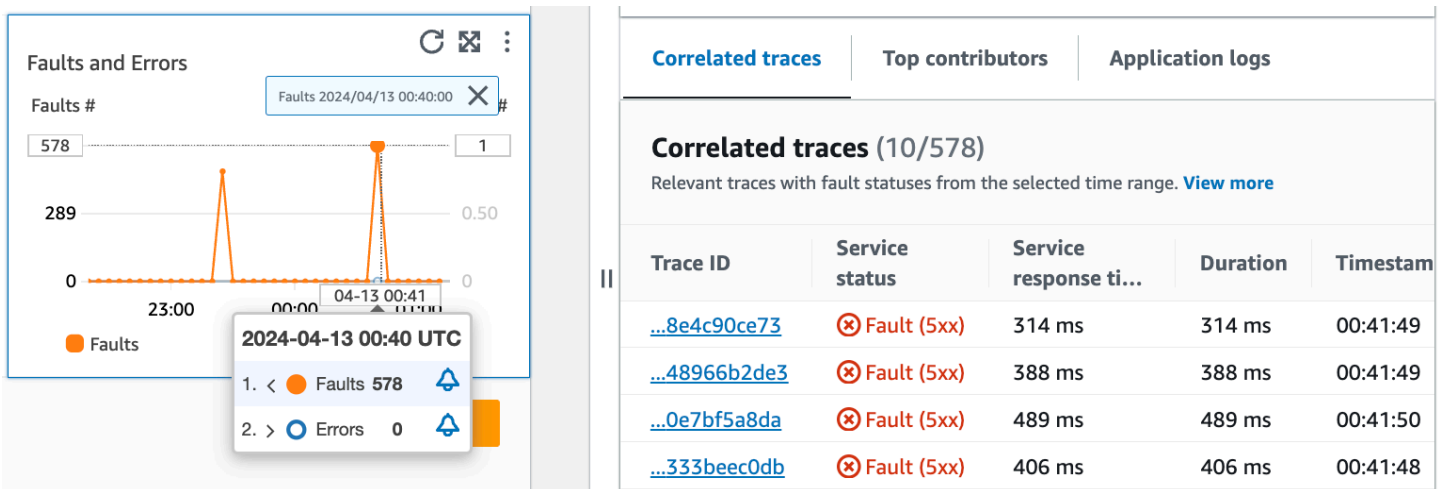
View service operations metrics, correlated traces, and application logs

Application Signals correlates service operation metrics with AWS X-Ray traces, CloudWatch [Container Insights](#), and application logs. Use these metrics to troubleshoot operational health issues. To view metrics as graphical information, do the following:

1. Select a service operation in the **Service operations** table to see a set of graphs for the selected operation above the table with metrics for **Volume and Availability**, **Latency**, and **Faults and Errors**.
2. Hover over a point in a graph to view more information.

3. Select a point to open a diagnostic pane that shows correlated traces, metrics, and application logs for the selected point in the graph.

The following image shows the tooltip that appears after hovering over a point in the graph, and the diagnostic pane which appears after clicking on a point. The tooltip contains information about the associated data point in the **Faults and Errors** graph. The pane contains **Correlated traces**, **Top contributors**, and **Application logs** associated with the selected point.



Correlated traces

Look at related traces to understand an underlying issue with a trace. You can check to see if correlated traces or any service nodes associated with them behave similarly. To examine correlated traces, choose a **Trace ID** from the **Correlated traces** table to open the [X-Ray trace details](#) page for the chosen trace. The trace details page contains a map of service nodes that are associated with the selected trace and a timeline of trace segments.

Top contributors

View the top contributors to find main input sources to a metric. Group contributors by different components to look for similarities within the group and understand how trace behavior differs between them.

The **Top contributors** tab gives metrics for **Call volume**, **Availability**, **Avg latency**, **Errors**, and **Faults** for each group. The following example image shows top contributors to a suite of metrics for an application deployed on an Amazon EKS platform:

Correlated traces	Top contributors	Application logs			
Top contributors (2/2) View ▼					
Top metric statuses powered by Logs Insights. View in Log Insights .					
Top 10 Nodes ▼ by faults					
Name	Call volume	Avail...	Avg latency	Errors	Faults
<input checked="" type="radio"/> i-0cb188a83...	1k	66.1 %	199.2 ms	0	378
<input type="radio"/> i-0ec1f65e4...	1k	66.4 %	188.3 ms	0	361

The top contributors contains the following metrics:

- **Call volume** - Use the call volume to understand the number of requests per time interval for a group.
- **Availability** - Use availability to see what percentage of time that no faults were detected for a group.
- **Avg latency** - Use latency to check the average time that requests ran for a group over a time interval that depends on how long ago the requests that you are investigating were made. Requests that were made less than 15 days prior are evaluated over 1 minute intervals. Requests that were made between 15 and 30 days prior, inclusive, are evaluated over 5 minute intervals. For example, if you are investigating requests that caused a fault 15 days ago, the call volume metric is equal to the number of requests per 5 minute interval.
- **Errors** - The number of errors per group measured over a time interval.
- **Faults** - The number of faults per group over a time interval.

Top contributors using Amazon EKS or Kubernetes

Use information about the top contributors for applications deployed on Amazon EKS or Kubernetes to see operational health metrics grouped by **Node**, **Pod** and **PodTemplateHash**. The following definitions apply:

- A **pod** is a group of one or more Docker containers that share storage and resources. A pod is the smallest unit that can be deployed on a Kubernetes platform. Group by pods to check if errors are related to pod-specific limitations.
- A **node** is a server that runs pods. Group by nodes to check if errors are related to node-specific limitations.
- A **pod template hash** is used to find a particular version of a deployment. Group by pod template hash to check if errors are related to a particular deployment.

Top contributors using Amazon EC2

Use information about the top contributors for applications deployed on Amazon EKS to see operational health metrics grouped by instance ID, and auto scaling group. The following definitions apply:

- An **Instance ID** is a unique identifier for the Amazon EC2 instance that your service runs. Group by instance ID to check if errors are related to a specific Amazon EC2 instance.
- An [auto scaling group](#) is a collection of Amazon EC2 instances that allow you to scale up or down the resources you need to serve your application requests. Group by auto scaling group if you want to check if errors are limited in scope to the instances inside the group.

Top contributors using a custom platform

Use information about the top contributors for applications deployed using [custom instrumentation](#) to see operational health metrics grouped by **Host name**. The following definitions apply:

- A host name identifies a device such as an endpoint or Amazon EC2 instance that is connected to a network. Group by host name to check if your errors are related to a specific physical or virtual device.

View top contributors in Log Insights and Container Insights

View and modify the automatic query that generated metrics for your top contributors in [Log Insights](#). View infrastructure performance metrics by specific groups such as pods or nodes in [Container Insights](#). You can sort clusters, nodes or workloads by resource consumption and quickly identify anomalies or and mitigate risks pro-actively before end user experience is impacted. An image showing how to select these options follows:

Correlated traces
Top contributors
Application logs

Top contributors (2/2)

Top metric statuses powered by Logs Insights. View in [Log Insights](#)

View ▲

View in Container Insights ↗

View in Log Insights ↗

Top 10 Nodes ▼ by faults

	Name	Call volume	Avail...	Avg latency	Errors	Faults
<input checked="" type="radio"/>	i-0cb188a83...	1k	66.1 %	199.2 ms	0	378
<input type="radio"/>	i-0ec1f65e4...	1k	66.4 %	188.3 ms	0	361

In **Container Insights**, you can view metrics for your Amazon EKS or Amazon ECS container that are specific to the grouping of your top contributors. For example, if you grouped by pod for an EKS container to generate top contributors, container insights will show metrics and statistics filtered for your pod.

In **Log Insights**, you can modify the query that generated the metrics under **Top contributors** using the following steps:

1. Select **View in Log Insights**. The **Logs Insights** page that opens contains an query that is automatically generated and contains the following information:
 - The log cluster group name.
 - The operation that you were investigating with CloudWatch.
 - The aggregate of the operational health metric interacted with on the graph.

The log results are automatically filtered to show data from the last five minutes before you selected the data point on the service graph.

2. To edit the query, replace the generated text with your changes. You can also use the **Query generator** to help you generate a new query, or update the existing query.

Application logs

Use the query in the **Application logs** tab to generate logged information for your current log group, service and insert a timestamp. A log group is a group of log streams that you can define when you configure your application.

Use a log group to organize logs with similar characteristics including the following:

- Capture logs from a specific organization, source or function.
- Capture logs that are accessed by a particular user.
- Capture logs for a specific time period.

Use these log streams to track specific groups or time frames. You can also set up monitoring rules, alarms and notifications for these log groups. For more information about log groups, see [Working with log groups and log streams](#).

The application logs query returns the logs, recurring text patterns and graphical visualizations for your log groups.

To run the query, select **Run query in Logs Insights** to either run the automatically generated query or modify the query. To edit the query, replace the automatically generated text with your changes. You can also use the **Query generator** to help you generate a new query or update the existing query.

The following image shows the sample query that is automatically generated based on the selected point in the service operations graph:

Correlated traces | **Top contributors** | **Application logs**

Application logs

View application logs for this plot-point in Logs Insights.


Application Signals has identified the log group and query.

Log group

```
/aws/containerinsights/petclinic-sampleApp/application
```

Query

```
1 | fields @timestamp, @logStream, @message
2 | parse kubernetes.pod_name /(?<service_name>.*?)-[^\s]-
3 | filter kubernetes.namespace_name = "default"
4 | filter service_name = "visits-service"
5 | display @timestamp, @logStream, @message
6 | sort @timestamp desc
7 | limit 50
```

[Run query in Logs Insights](#) 

In the preceding image, CloudWatch has automatically detected the log group that is associated with your selected point, and included it in a generated query.

View your service dependencies

Choose the **Dependencies** tab to display the **Dependencies** table and a set of metrics for the dependencies of all service operations or a single operation. The table contains a list of dependencies discovered by Application Signals, including metrics for latency, call volume, fault rate, error rate, and availability.

At the top of the page, choose an operation from the down arrow list to view its dependencies, or choose **All** to see dependencies for all operations.

Filter the table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter. Select **Group by Dependency** at the top right of the table to group dependencies by service and operation name. When grouping is turned on, expand or collapse a group of dependencies with the **+** icon next to the dependency name.

Dependencies (10) Info Group by Dependency

Dependency	Remote Operation	Target	Latency p99	Latency p90	Latency p50	Volume	Fault rate	Error rate	Availability
<input checked="" type="radio"/> visits-service	POST /owners	-	1.6K ms	324.3 ms	41.8 ms	3.6K	5.1% (183)	3.8% (136)	94.9% (94.92)
<input type="radio"/> customers-service	POST /owners	-	233.6 ms	91.9 ms	42 ms	1.6K	1.9% (30)	0.1% (1)	98.1% (98.09)
<input type="radio"/> customers-service	GET /owners	-	99.5 ms	33.4 ms	3.1 ms	5.1K	0.3% (13)	9.3% (474)	99.7% (99.74)
<input type="radio"/> customers-service	/owners	-	23.2 ms	16.6 ms	9.5 ms	311	0% (0)	0% (0)	100% (100)

The **Dependency** column displays the dependency service name, while the **Remote Operation** column displays the service operation name. When calling AWS services, the **Target** column displays the AWS resource, such as DynamoDB table or Amazon SNS queue.

To select a dependency, select the option next to a dependency in the **Dependencies** table. This shows a set of graphs that display detailed metrics for call volume, availability, faults, and errors. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated traces for the selected point in the graph. Choose a trace ID from the **Correlated traces** table to open the [X-Ray Trace details](#) page for the selected trace.

Traces that are correlated to a dependency, service operation and target per plot-point in the graphs.

Faults
Time-range of traces: 2023-11-22 09:45:00 (UTC-08:00) - 5 minutes

Correlated traces (10/31)
Relevant traces with fault statuses from the selected time range. [View more](#)

Trace ID	Remote service status	Remote service response time	Durati...	Timest...
...40092ffe4d	⊗ Fault (5xx)	269 ms	278 ms	09:49:2
...f3687de6fe	⊗ Fault (5xx)	485 ms	1K ms	09:48:5

View your Synthetics canaries

Choose the **Synthetics Canaries** tab to display the **Synthetics Canaries** table, and a set of metrics for each canary in the table. The table includes metrics for success percentage, average duration, runs, and failure rate. Only canaries that are [enabled for AWS X-Ray tracing](#) are displayed.

Use the filter text box in the synthetics canaries table to find the canary that you are interested in. Each filter that you create appears below the filter text box. Choose **Clear filters** at any time to remove the table filter.

Name	Success Percent	Average Duration	Runs	Failure Rate
<input checked="" type="radio"/> pc-visit-pet	0%	34.6K ms	180	100% (180)
<input type="radio"/> pc-add-visit	0%	34.5K ms	180	100% (180)
<input type="radio"/> pc-visit-valid	0%	7.4K ms	180	100% (180)

Select the radio button next to the name of the canary to see a set of tabs containing graphs detailed metrics including success percentage, errors and duration. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows canary runs that correlate to the selected point. Select a canary run and choose the **Run time** to see artifacts for your selected canary run including logs, HTTP Archive (HAR) files, screenshots, and suggested steps to help you troubleshoot problems. Choose **Larn more** to open the [CloudWatch Synthetics Canaries](#) page next to **Canary runs**.



View your client pages

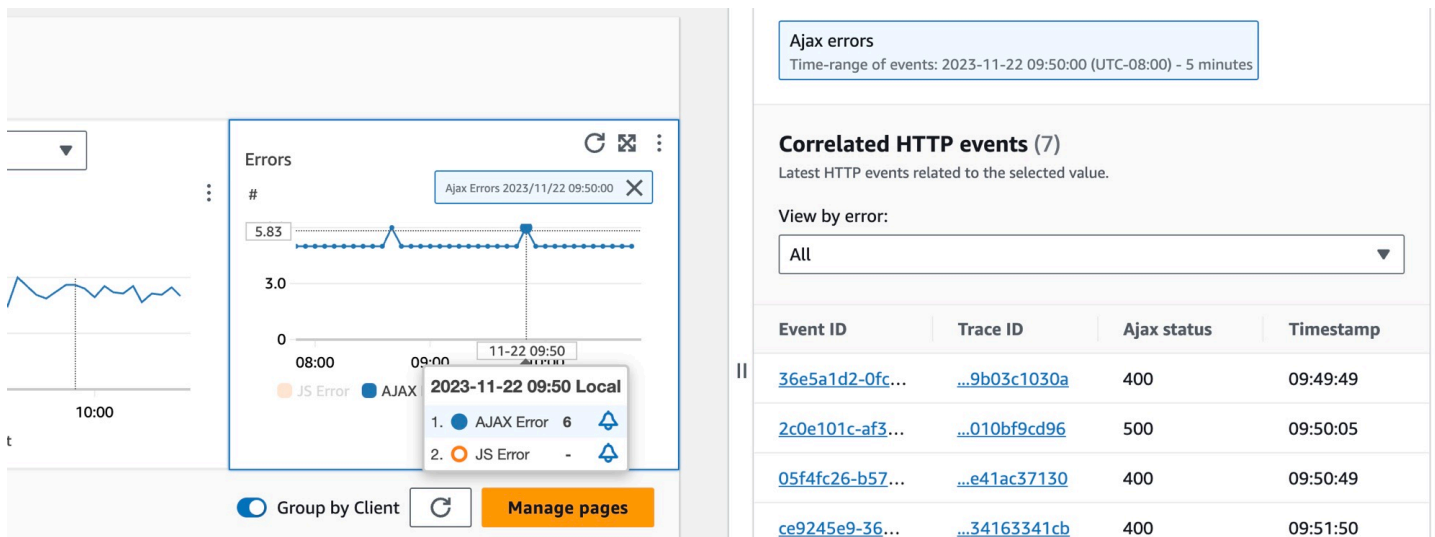
Choose the **Client pages** tab to display a list of client web pages that call your service. Use the set of metrics for the selected client page to measure the quality of your client's experience when interacting with a service or application. These metrics include page loads, web vitals, and errors.

To display your client pages in the table, you must [configure your CloudWatch RUM web client for X-Ray tracing](#) and turn on Application Signals metrics for your client pages. Choose **Manage pages** to select which pages are enabled for Application Signals metrics.

Use the filter text box to find the client page or application monitor that you are interested in below the filter text box. Choose **Clear filters** to remove the table filter. Select **Group by Client** to group client pages by client. When grouped, choose the **+** icon next to a client name to expand the row and see all pages for that client.

Client	Page	Page Loads	Largest Contentful Paint	First Input Delay	Cumulative layout shift	JS errors	Ajax errors
● pulse-rum-pet-clinic-iad	All	377	899.2 ms	1.4 ms	-	-	46
○	/owners/3/pets/4/visits	36	1K ms	1.6 ms	-	-	1
○	/owners/details/1	45	801.2 ms	-	-	-	-
○	/vets	180	-	-	-	-	-

To select a client page, select the option next to a client page in the **Client pages** table. You will see a set of graphs that display detailed metrics. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated performance navigation events for the selected point in the graph. Choose an event ID from the list of navigation events to open the [CloudWatch RUM Page view](#) for the chosen event.



Note

To see AJAX errors within your client pages, use the [CloudWatch RUM web client](#) version 1.15 or newer.

Currently, up to 100 operations, canaries, and client pages, and up to 250 dependencies, can be shown per service.

View your application topology and monitor operational health with the CloudWatch service map

Note

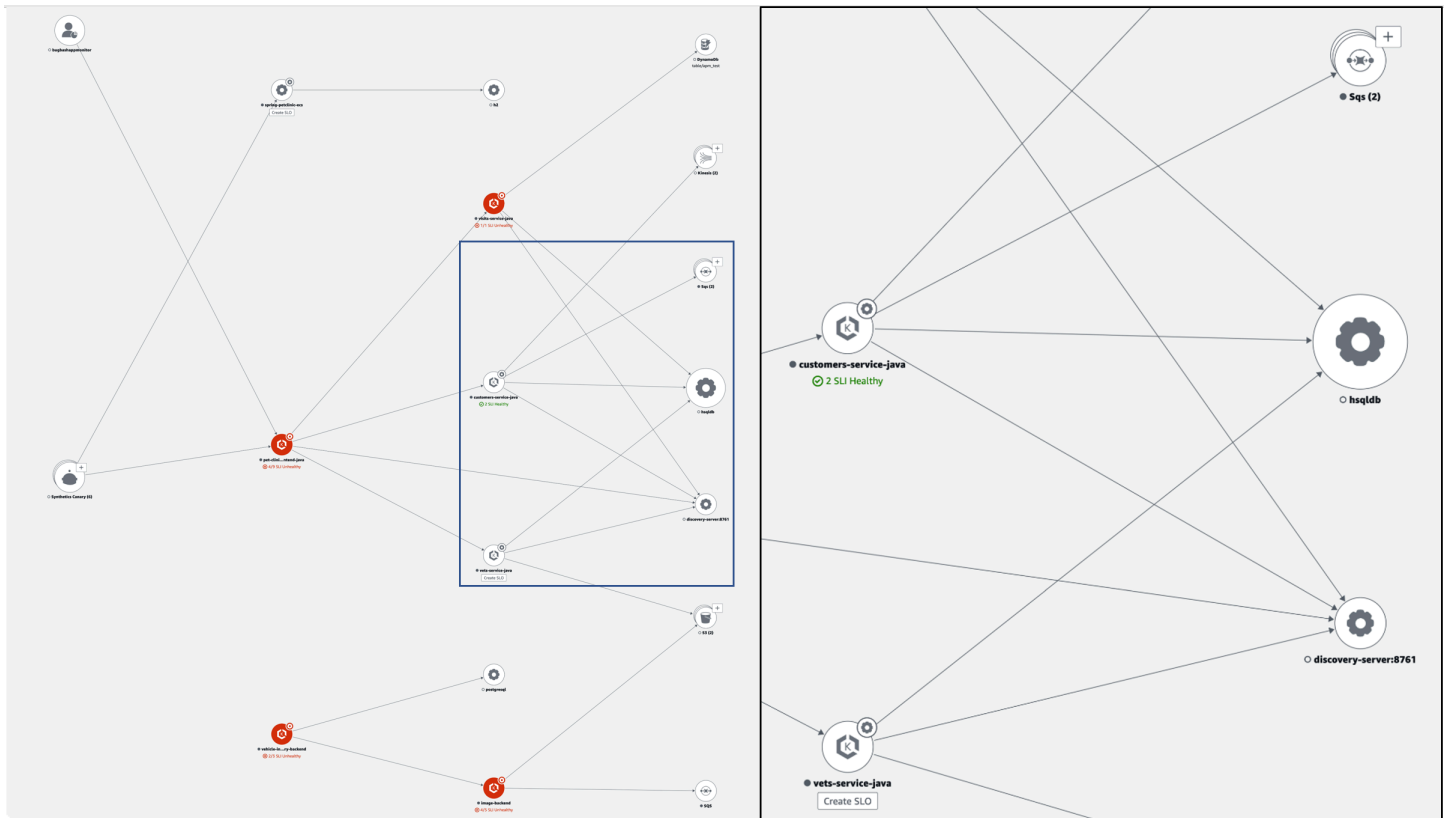
The CloudWatch service map replaces the ServiceLens map. To see a map of your application based on AWS X-Ray traces, open the [X-Ray Trace Map](#). Choose **Trace Map** under the **X-Ray** section in the left navigation pane of the CloudWatch console.

Use the service map to view the topology of your application clients, synthetics canaries, services and dependencies, and monitor operational health. To view the service map, open the [CloudWatch console](#) and choose **Service Map** under the **Application Signals** section in the left navigation pane.

After you [enable your application for Application Signals](#), use the service map to make it easier to monitor your application's operational health:

- View connections between client, canary, service, and dependency nodes to help you understand your application topology and execution flow. This is especially helpful if your service operators are not your development team.
- See which services are meeting or not meeting your [service level objectives \(SLOs\)](#). When a service is not meeting your SLOs, you can quickly identify whether a downstream service or dependency might be contributing to the issue or impacting multiple upstream services.
- Select an individual client, synthetics canary, service, or dependency node to see related metrics. The [Service details](#) page shows more detailed information about operations, dependencies, synthetics canaries, and client pages.
- Filter and zoom the service map to make it easier to focus on a part of your application topology, or see the entire map. Create a filter by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the filter.

The following example service map shows services with edges connecting them to components that they interact with. If an SLO is defined, the service map also shows health status.

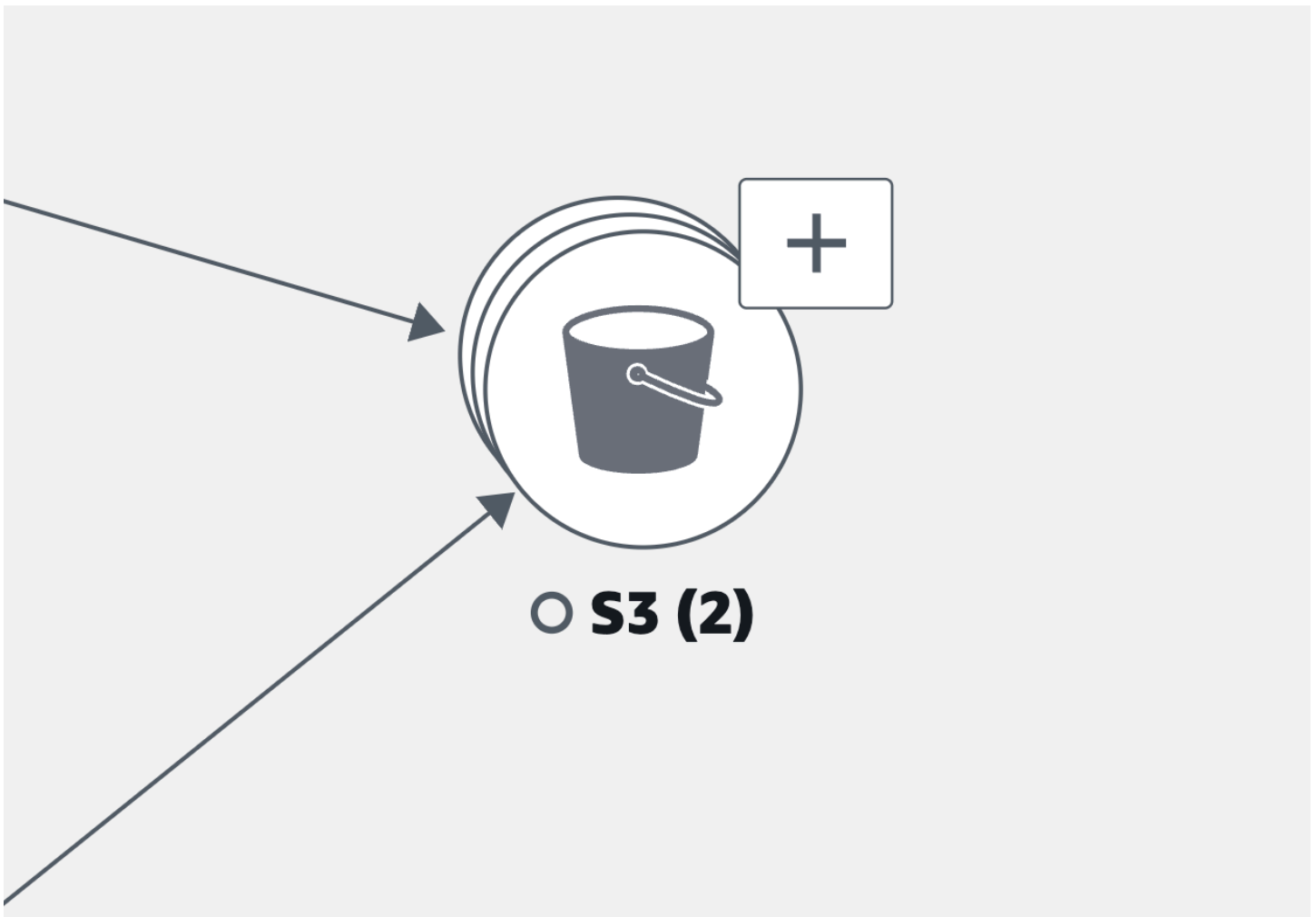


Explore the service map

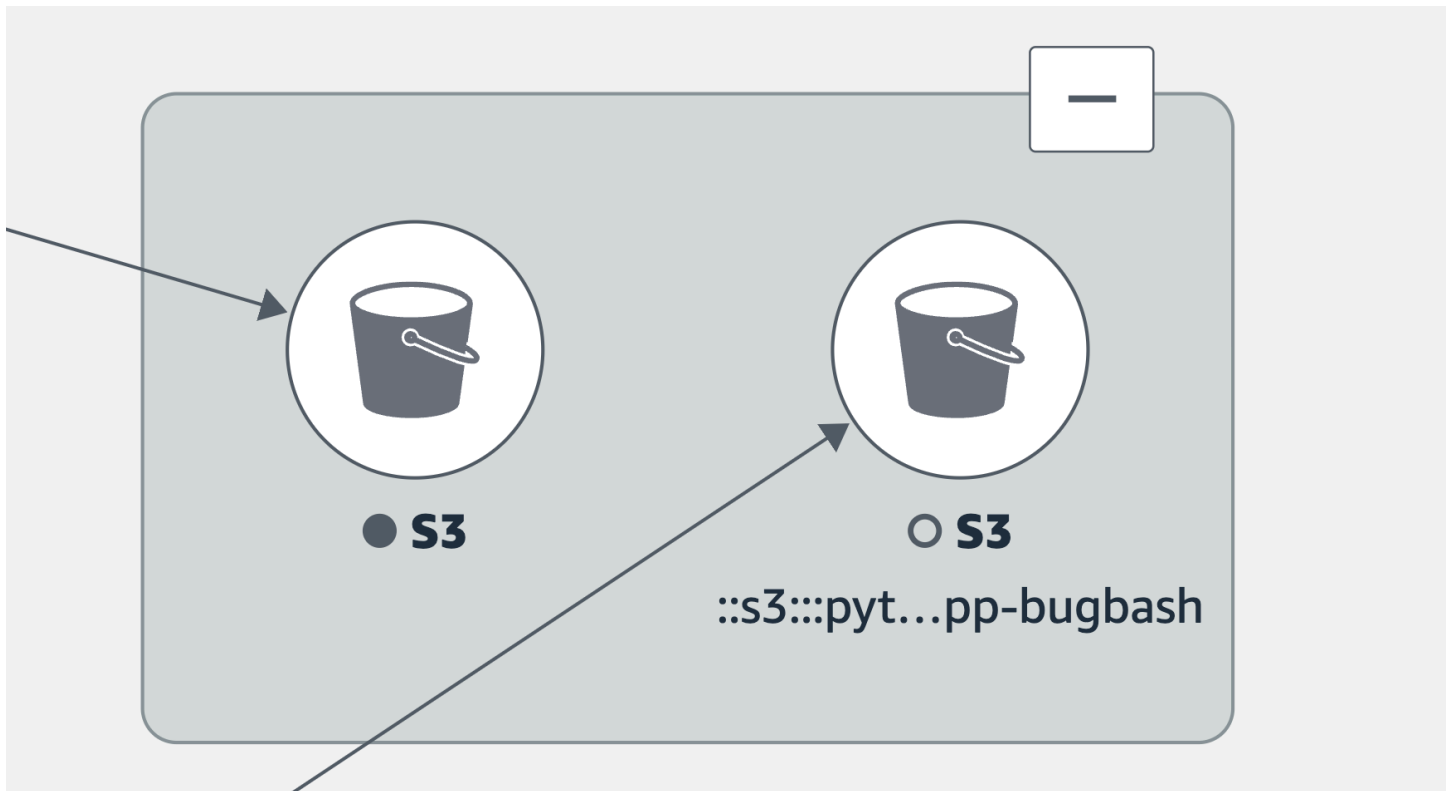
After you have enabled your application for Application Signals, the service map displays nodes representing your services and their dependencies.

Turn on active tracing for your CloudWatch RUM clients and synthetics canaries to see client and canary nodes on the map.

By default, canaries, RUM clients, and AWS service dependencies of the same kind are grouped together into a single expandable icon in the service map. Service dependencies outside of AWS are not grouped together by default. For example, in the following image, all Amazon S3 buckets are grouped together under one expandable icon:



In the previous image, the label between the Amazon S3 grouping and originating service displays the number of edges to the group in parenthesis under the dependency's icon. Select the (+) icon to expand the group and see its individual elements, as shown in the following image:

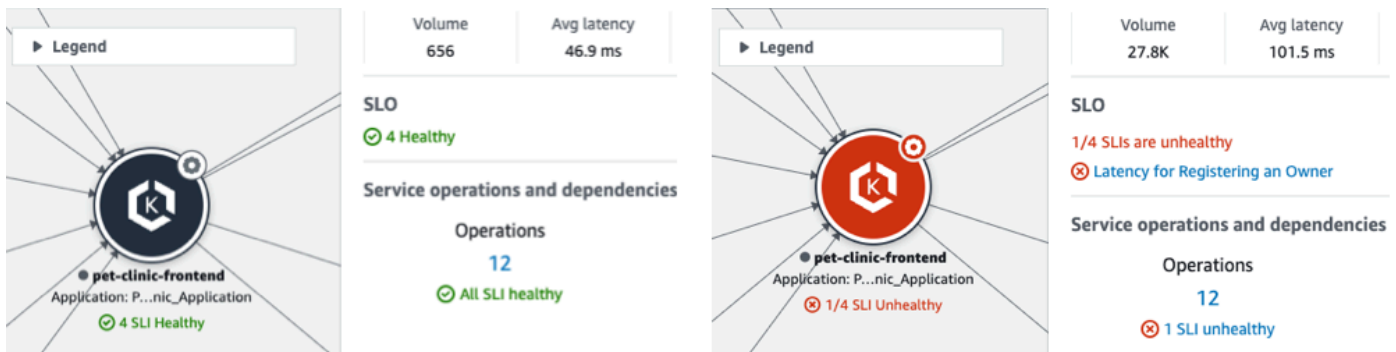


Choose a tab for information about exploring each kind of node and the edges (connections) between them.

View your application services

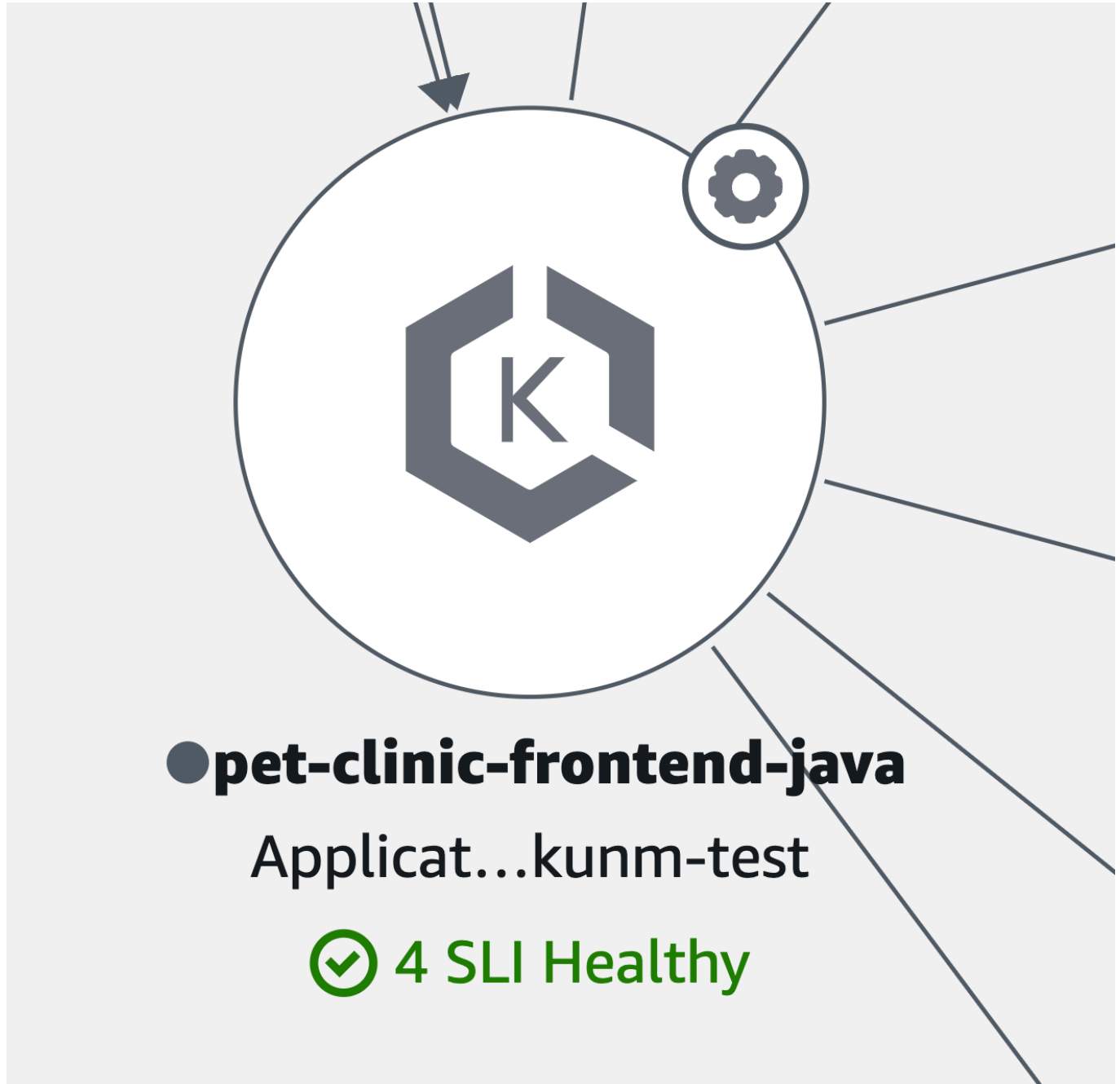
You can view your application services and the status of their SLOs and service level indicators (SLIs) in the **Service Map**. If you didn't create SLOs for a service, choose the **Create SLO** button below the service node.

The **Service Map** displays all of your services. It also shows the customers and canaries that consume the service and the dependencies that your services calls, as shown in the following image:



The following icons represent examples of application services in the service map:

- [Amazon Elastic Kubernetes Service](#):



- A [Kubernetes](#) container:



- Amazon Elastic Compute Cloud (Amazon EC2):



- Other application service types not previously listed:

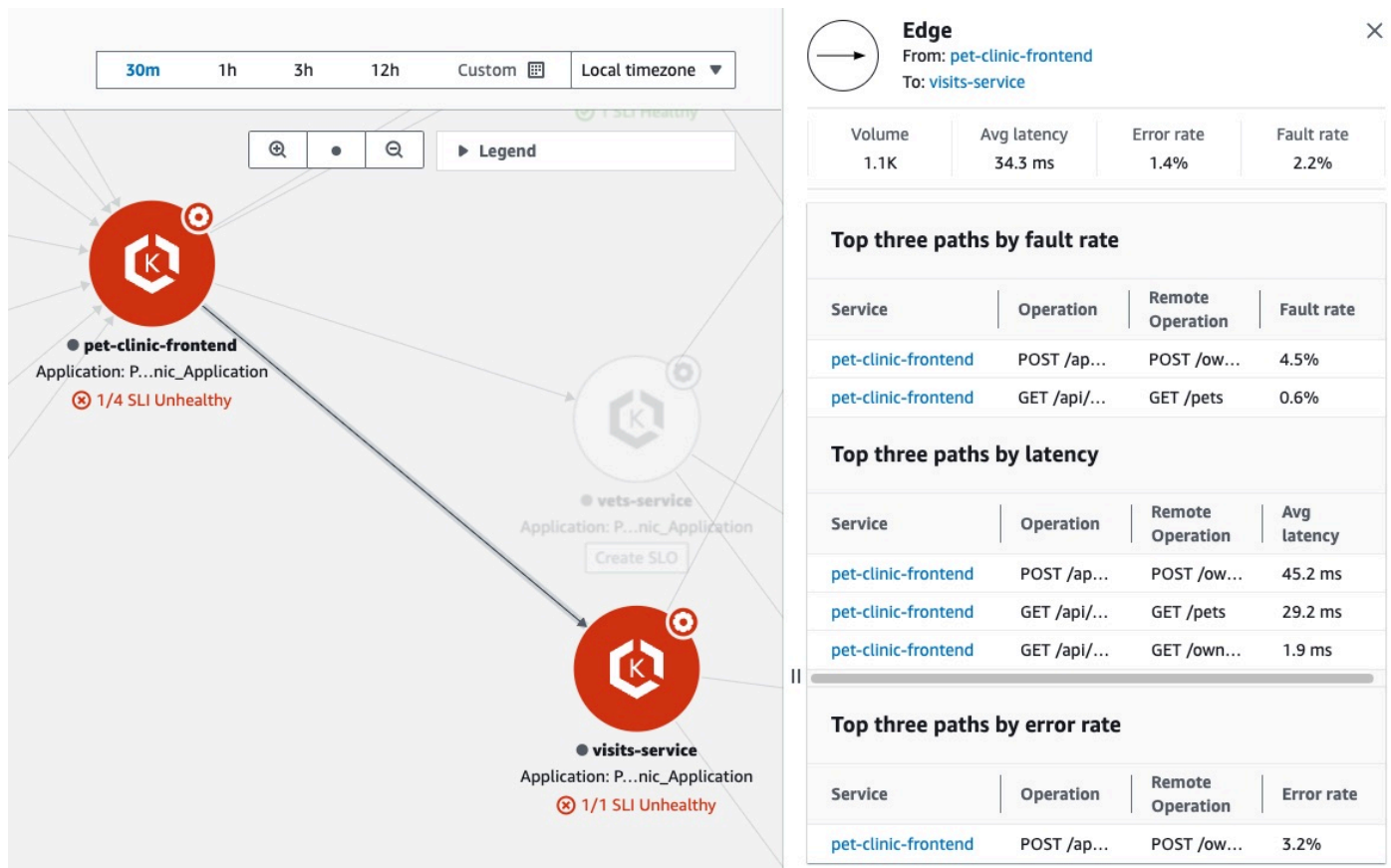


When you select a service node, a pane opens displaying detailed service information:

- Metrics for call volume, latency, error, and fault rate.
- The number of SLIs and SLOs that are healthy or unhealthy.
- The option to view more information about an SLO.
- The number of service operations, dependencies, synthetics canaries, and client pages.
- The option to select each number to open the [Service details](#) page for it.
- The application name, if you have associated the underlying compute resource with an application using AppRegistry or the Applications card on the AWS Management Console home page.

- Choose the application name to display the application details in the [myApplications](#) console page.
- The `Cluster`, `Namespace`, and `Workload` for services hosted in Amazon EKS, or `Environment` for services hosted in Amazon ECS or Amazon EC2. For Amazon EKS-hosted services, choose any link to open CloudWatch Container Insights.

Select an edge or connection between a service node and a downstream service or dependency node. This opens a pane containing top paths by fault rate, latency, and error rate, as shown in the following example image. Choose any link in the pane to open the [Service details](#) page and see detailed information for the chosen service or dependency.

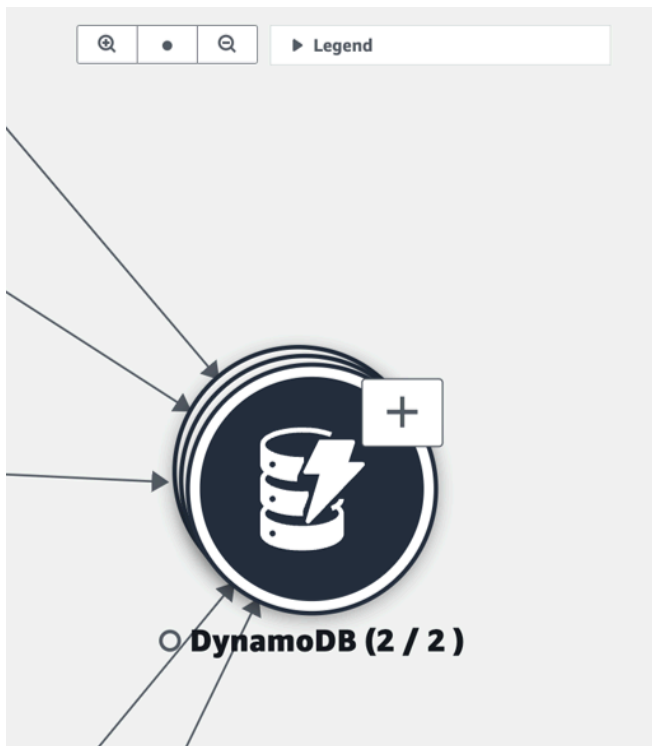


View dependencies

Your application dependencies are displayed on the service map, connected to the services that call them.

Choose a dependency node to open a pane containing top paths by fault rate, latency, and error rate. Choose any service or target link to open the [Service Details](#) page and see detailed

information about the chosen service or dependency target, as shown in the example image below:



Volume	Avg latency	Error rate	Fault rate
-	-	-	-

Top three paths by fault rate		
Service	Remote operation	Fault rate
No paths with faults		

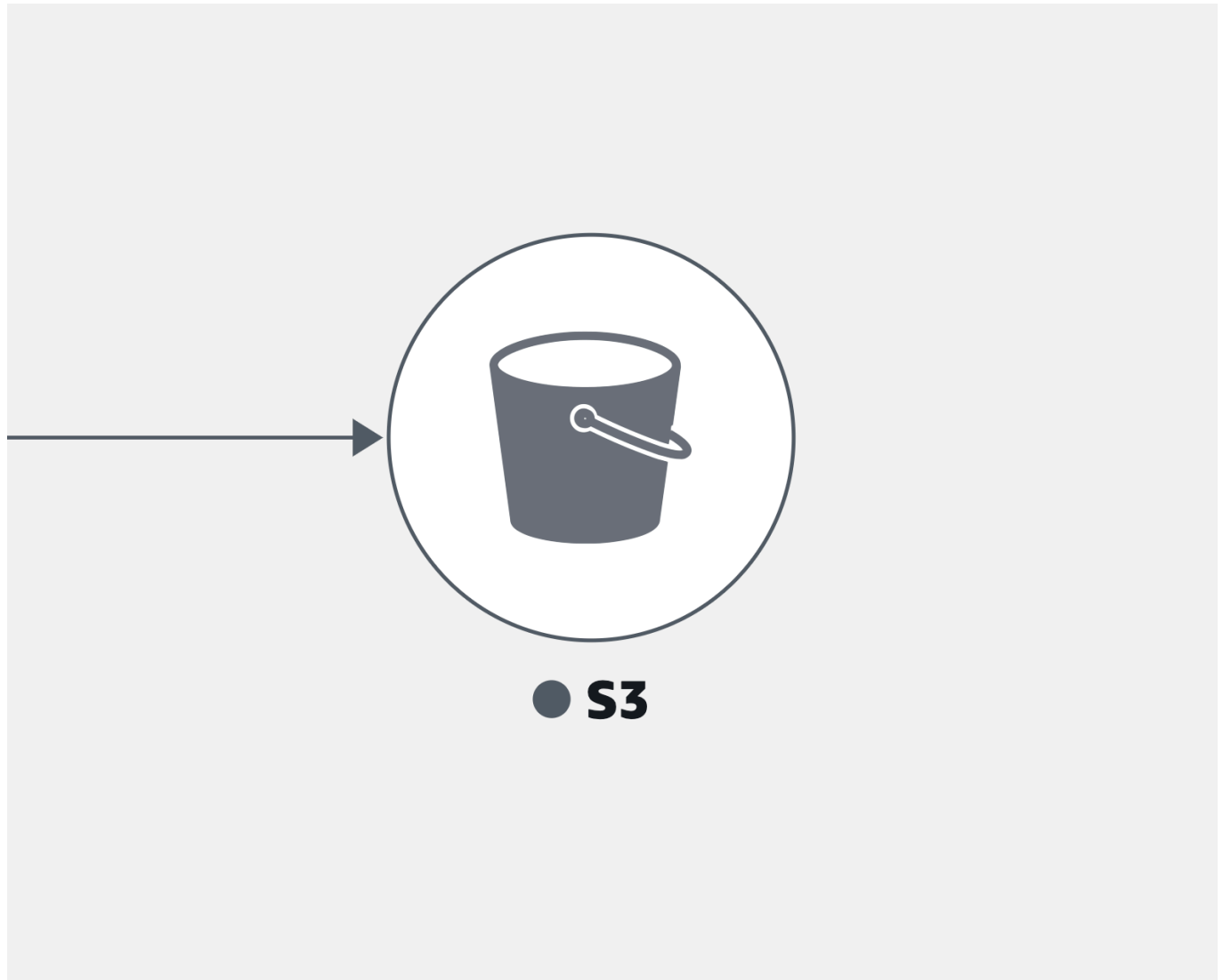
Top three paths by latency		
Service	Remote operation	Avg latency
billing-service-ec2-python	PutItem	282.8 ms
billing-service-python	PutItem	75.6 ms
visits-service-java	PutItem	64.9 ms

Top three paths by error rate		
Service	Remote operation	Error rate
visits-service-java	PutItem	9.6%

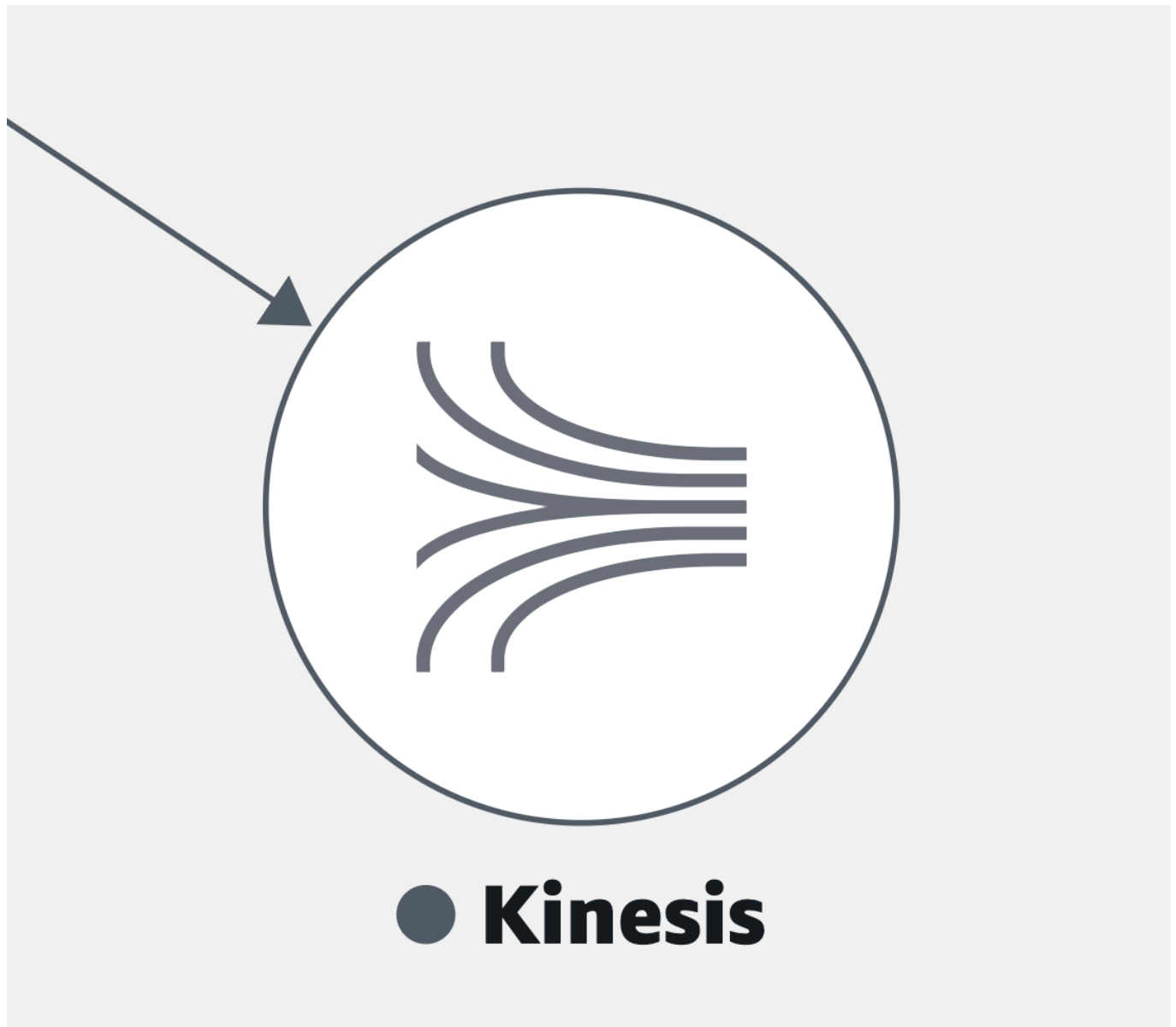
Service dependencies are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the previous image, to expand the group and see its individual elements.

The following icons represent examples of dependency nodes in the service map:

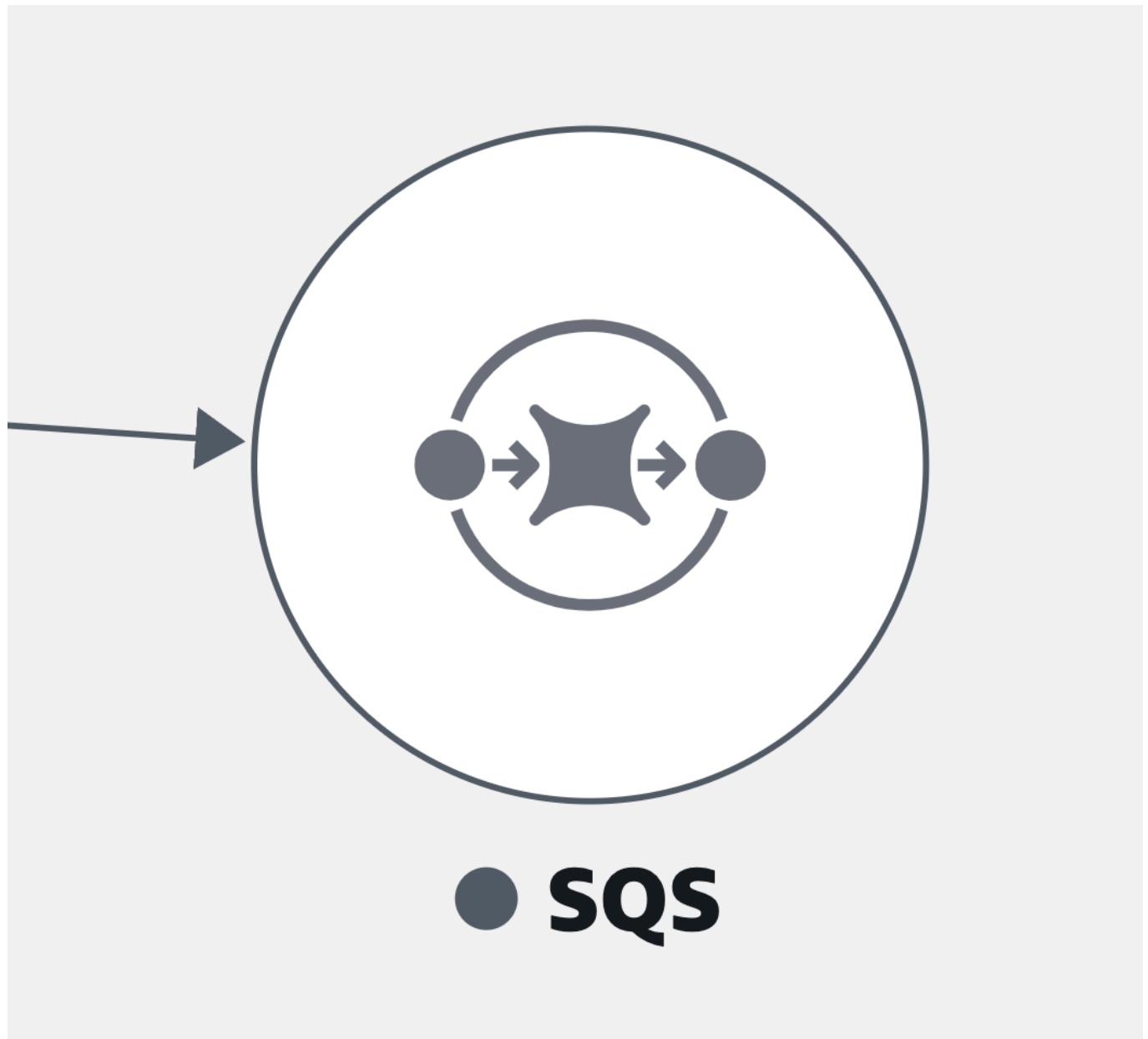
- An [Amazon S3](#) bucket:



- An [Amazon Kinesis](#) stream:



- [Amazon Simple Queue Service](#) (Amazon SQS):



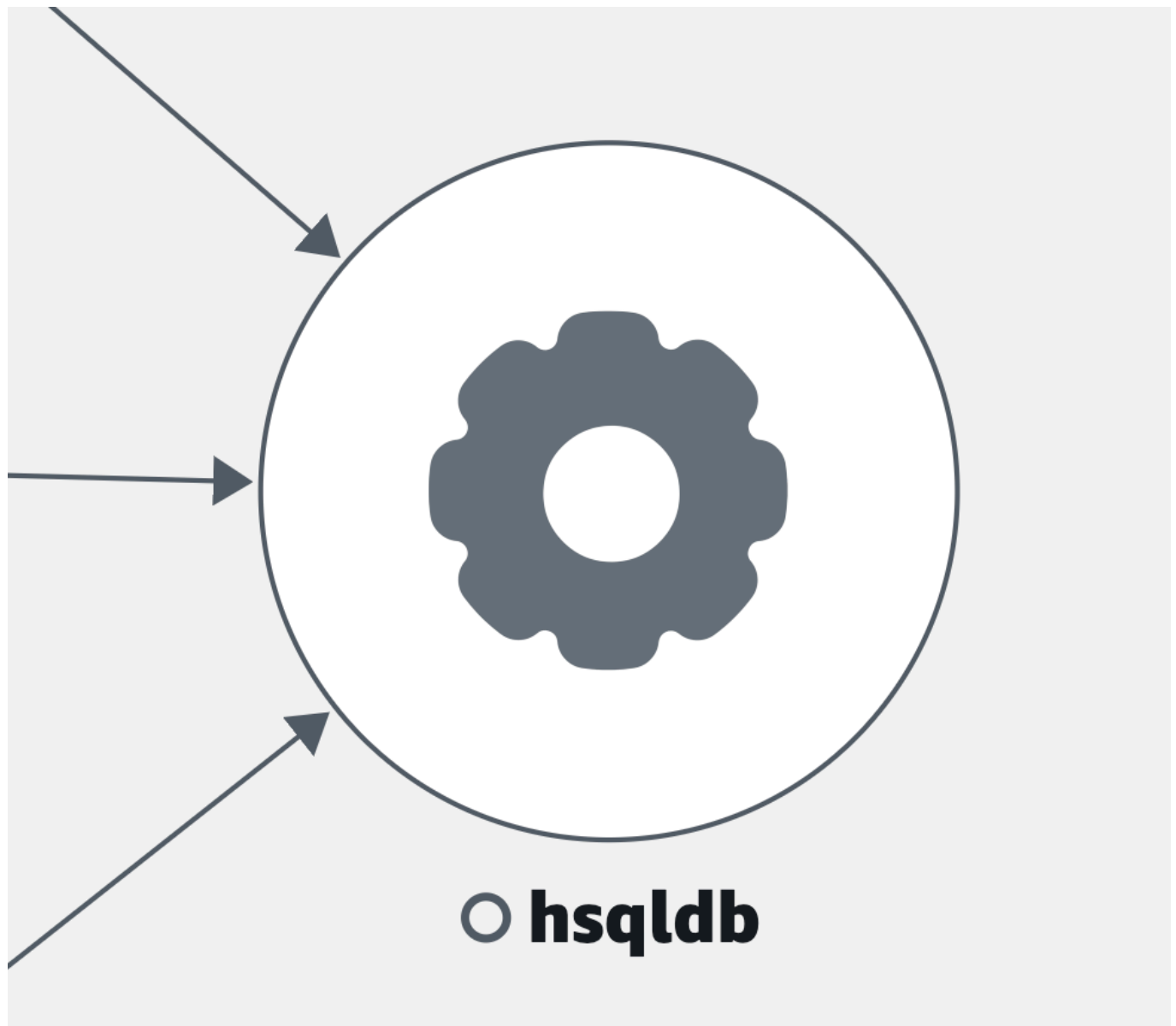
- An [Amazon DynamoDB](#) table:



○ **DynamoDb**

`::dynamodb::table/apm_test`

- Other dependency types not previously listed:




View clients

After you [turn on X-Ray tracing](#) for your CloudWatch RUM web clients, they display on the service map connected to services they call.

Choose a client node to open a pane displaying detailed client information:

- Metrics for page loads, average load time, errors, and average web vitals.
- A graph displaying a breakdown of errors.
- A link to display the client details in CloudWatch RUM.

RUM clients are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the following image, to expand the group and see its individual elements.



Legend

Client: RUM (3 / 3)

Top three clients by largest contentful paint		
Client	Largest co...	Service
PetClinicM...	1.68K	pet-clinic-frontend-java
PetClinicM...	1.62K	pet-clinic-frontend-java
PetClinicM...	1.61K	pet-clinic-frontend-ec2-java

Top three clients by HTTP errors		
Client	Average H...	Service
No clients with HTTP errors		

Top three clients by JS errors		
Client	Average JS...	Service
PetClinicM...	0.25	pet-clinic-frontend-java
PetClinicM...	0	pet-clinic-frontend-ec2-java
PetClinicM...	0	pet-clinic-frontend-java

The following icon represents an example of a RUM client in the service map:

- A RUM client –



○ bugbashappmonitor

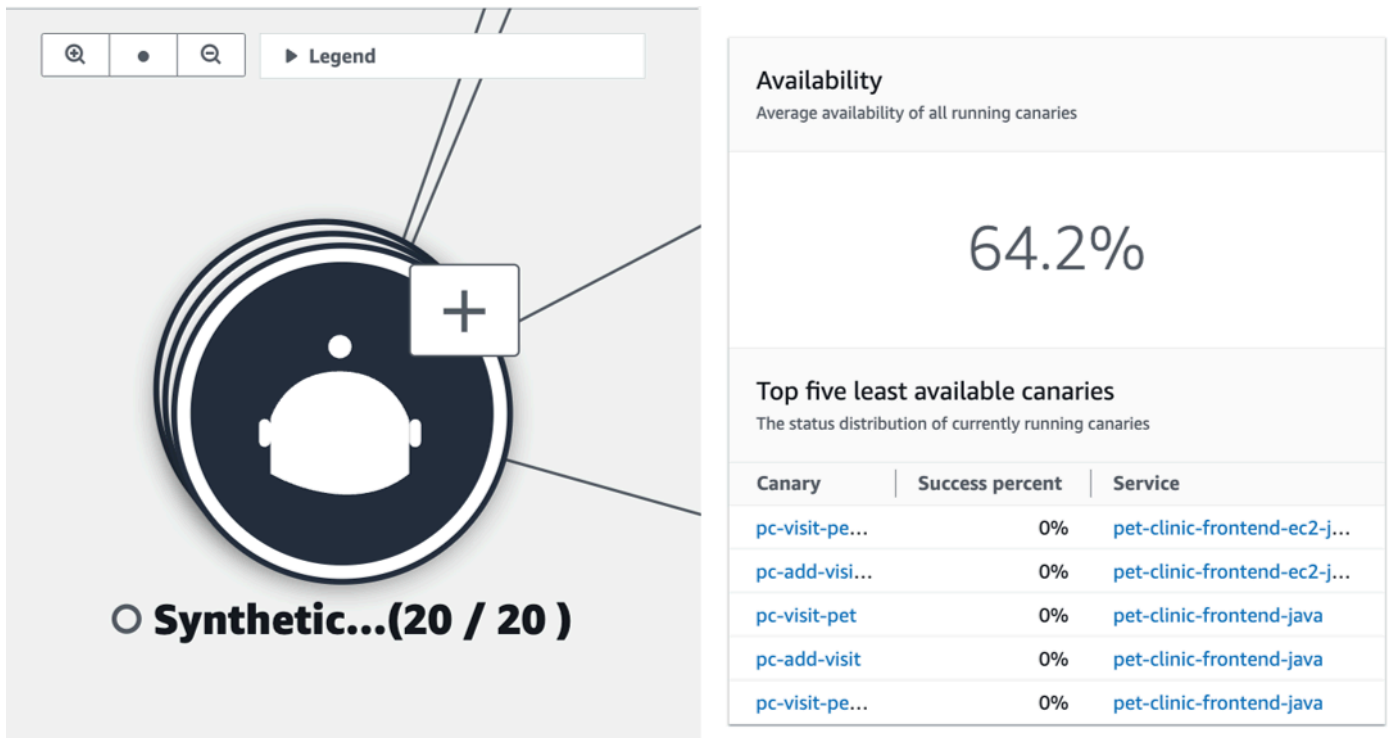
Note

To see AJAX errors within your client pages, use the [CloudWatch RUM web client](#) version 1.15 or newer.

View synthetics canaries

After you [turn on AWS X-Ray tracing](#) for your CloudWatch Synthetics canaries, they display on the service map connected to services they call, as shown in the following example image:

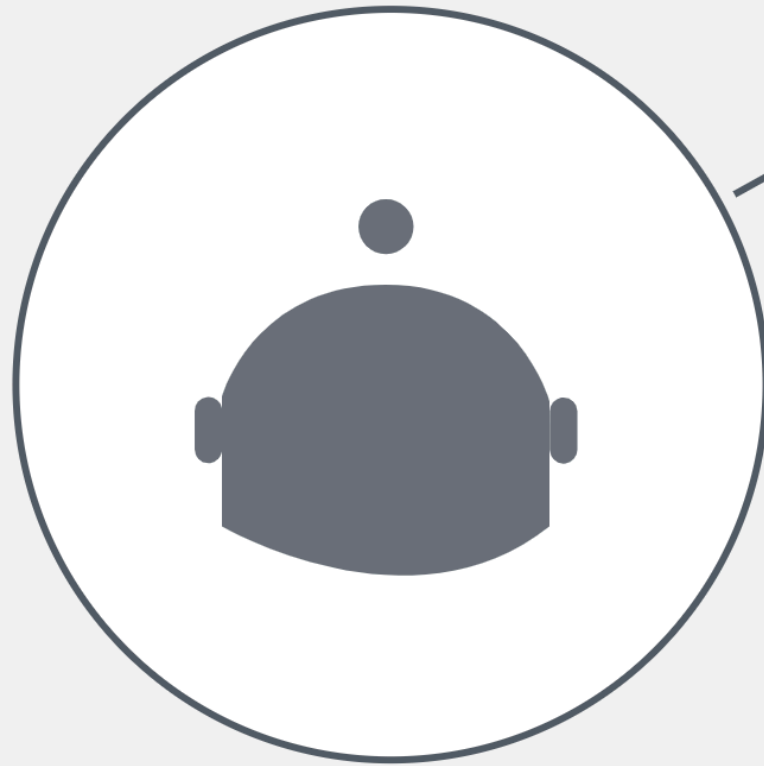
Choose a canary node to open a pane displaying detailed canary information, as shown in the following image:



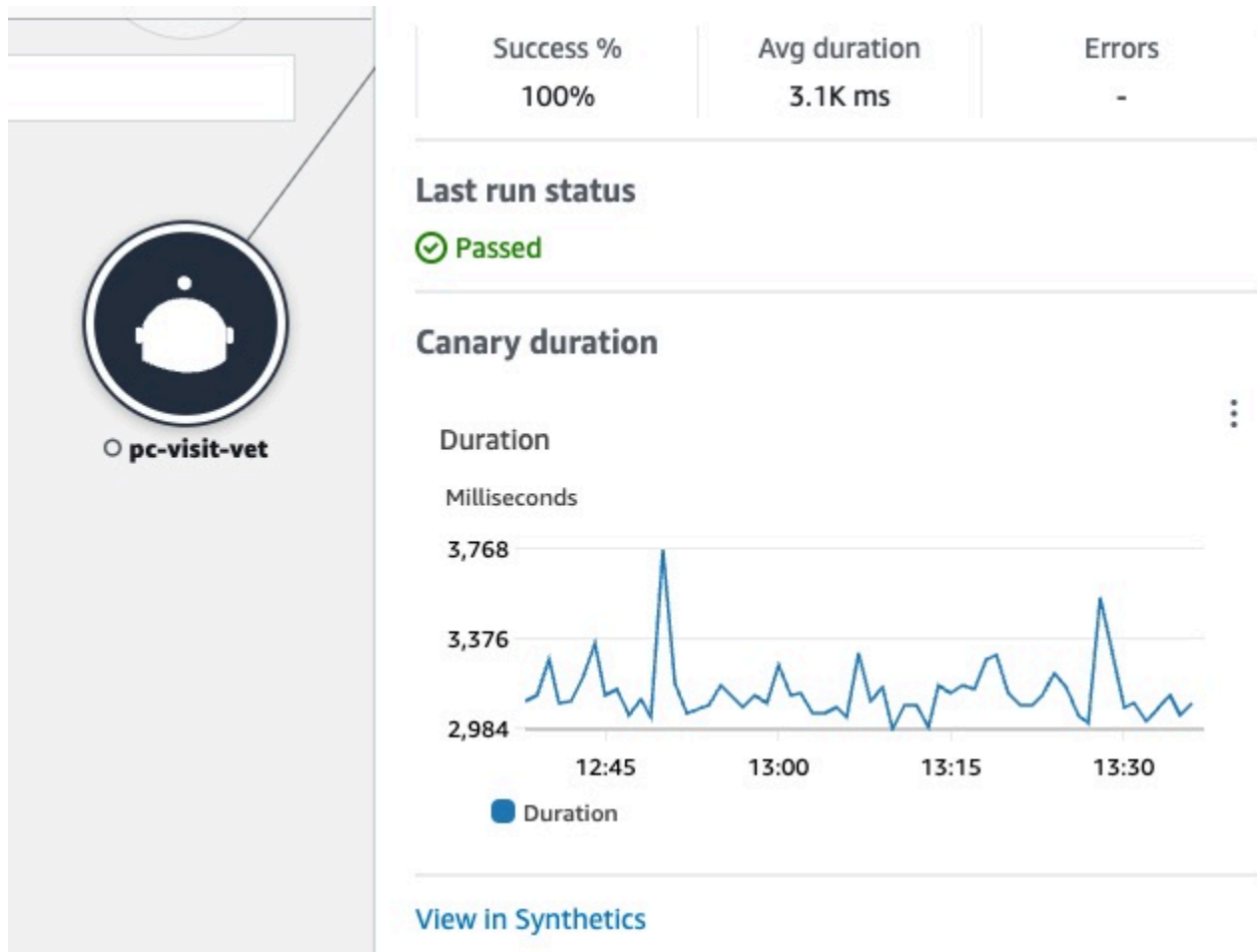
Canaries are grouped together by default into a single expandable icon. Select the (+) icon, as shown in the previous image, to expand the group and see its individual elements.

The following icons represent examples of clients in the service map:

- A synthetics canary –



○ **pc-create-owners**



In the pane for canary nodes, you can see the following:

- Metrics for success percentage, average duration, and errors.
- The status of the last canary run.
- A graph displaying canary run duration. Hover over a graph series to see a pop-up containing more information.
- A link to display canary details in **CloudWatch Synthetics**.

Example: Use Application Signals to resolve an operational health issue

The following scenario provides an example of how Application Signals can be used to monitor your services and identify service quality issues. Drill down to identify potential root causes and take action to resolve the issue. This example is focused on a pet clinic application composed of several microservices that call AWS services such as DynamoDB.

Jane is part of a DevOps team that oversees the operational health of a pet clinic application. Jane's team is committed to ensuring that the application is highly available and responsive. They use [service level objectives \(SLOs\)](#) to measure application performance against these business commitments. She receives an alert about several unhealthy service level indicators (SLIs). She opens the CloudWatch console and navigates to the Services page, where she sees several services in an unhealthy state.

Services [Info](#)

Services by SLI status

■ Healthy (1)
 ■ Unhealthy (2)
 ■ No SLO (1)

Top Services by fault rate

Service	Fault rate
visits-service	1.92%
pet-clinic-frontend	1.04%
customers-service	0.04%

Services (4) [Info](#)

	Name	SLI status	Application
<input type="radio"/>	pet-clinic-frontend	⊗ 2/4 Unhealthy	PetClinic Application
<input type="radio"/>	visits-service	⊗ 1/1 Unhealthy	PetClinic Application
<input type="radio"/>	customers-service	✔ 1 Healthy	PetClinic Application

At the top of the page, Jane sees that the `visits-service` is the top service by fault rate. She selects the link in the graph, which opens the Service detail page for the service. She sees that there is an unhealthy operation in the Service operations table. She selects this operation and sees in the Volume and Availability graph that there are periodic call volume spikes that seem to correlate to dips in availability.

Service operations 1
Dependencies
Synthetics Canaries
Client pages

Selected operation: POST /owners/*/pets/{petId}/visits

Click a point in the graphs to view correlated traces.

Volume and Availability

The graph shows two distinct incidents where service availability drops to 0% and volume drops from 1,162 to approximately 200. The incidents occur around 16:20 and 17:20.

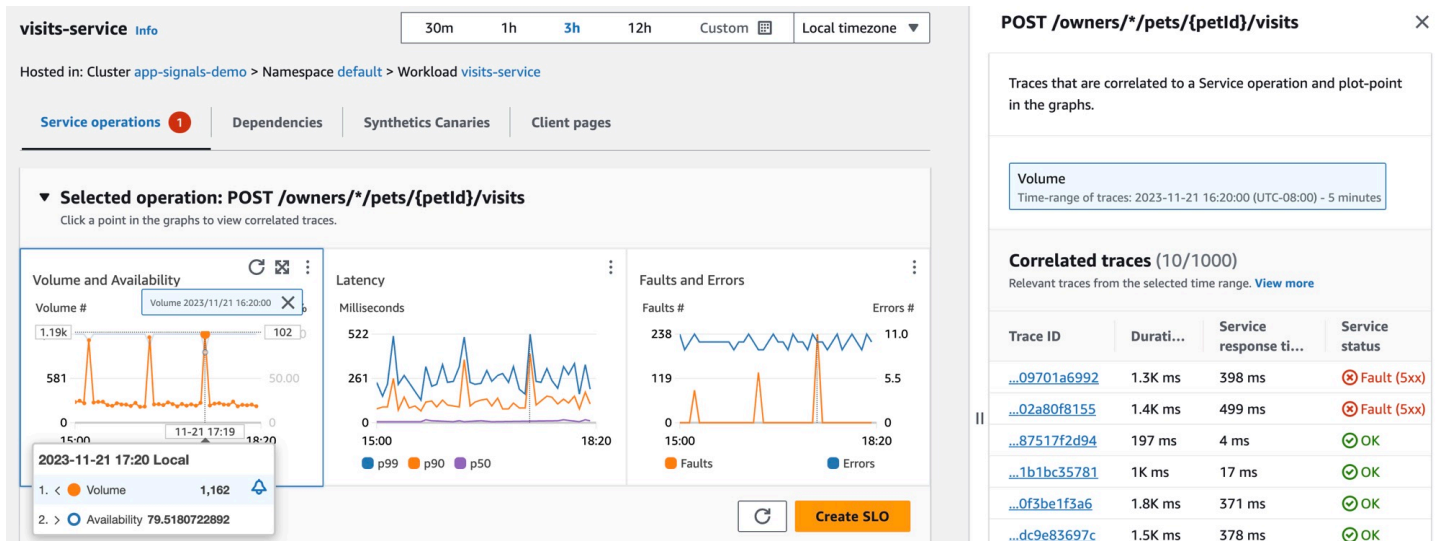
Latency

Latency spikes to 522ms during the two incidents, indicating performance degradation.

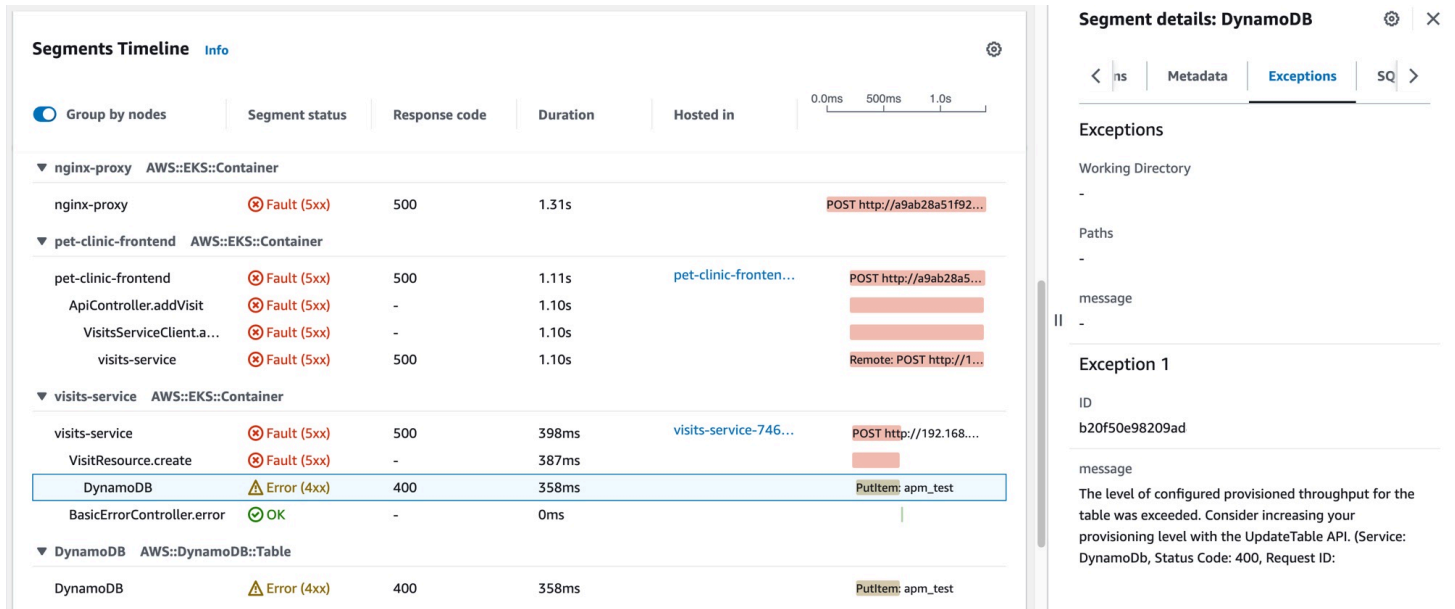
Service operations (4) [Info](#)

	Name	SLI Status	Dependencies
<input checked="" type="radio"/>	POST /owners/*/pets/{petId}/visits	⊗ 1/1 Unhealthy	2
<input type="radio"/>	InternalOperation	Create SLO	2

In order to look closer at the dips in service availability, Jane selects one of the availability data points in the graph. A drawer opens showing X-Ray traces that are correlated to the selected data point. She sees that there are multiple traces containing faults.



Jane selects one of the correlated traces with a fault status, which opens the X-Ray Trace detail page for the selected trace. Jane scrolls down to the Segments Timeline section and follows the call path until she sees that calls to a DynamoDB table are returning errors. She selects the DynamoDB segment and navigates to the Exceptions tab of the right-side drawer.



Jane sees that a DynamoDB resource is misconfigured, resulting in errors during spikes in customer requests. The DynamoDB table's level of provisioned throughput is periodically exceeded, resulting in service availability issues and unhealthy SLIs. Based on this information, her team is able to configure a higher level of provisioned throughput and ensure high availability of the application.

Standard application metrics collected

Application Signals collects *standard application metrics* from the services that it discovers. These metrics relate to the most critical aspects of a service's performance: latency, faults, and errors. They can help you identify issues, monitor performance trends, and optimize resources to improve the overall user experience.

The following table lists the metrics collected by Application Signals. These metrics are sent to CloudWatch in the `ApplicationSignals` namespace.

Metric	Description
Latency	The delay before data transfer begins after the request is made. Units: Milliseconds
Faults	A count of both HTTP 5XX server-side faults and OpenTelemetry span status errors. Units: None
Errors	A count of HTTP 4XX client-side errors. These are considered to be request errors that are not caused by service problems. Therefore, the <code>Availability</code> metric displayed on Application Signals dashboards does not regard these errors as service faults. Units: None

The `Availability` metric displayed on Application Signals dashboards is computed as $(1 - \text{Faults}/\text{Total}) * 100$. Total responses includes all responses and is derived from `SampleCount(Latency)`. Successful responses are all responses without a 5XX error. 4XX responses are treated as successful when Application Signals calculates `Availability`.

Dimensions collected and dimension combinations

The following dimensions are defined for each of the standard application metrics. For more information about dimensions, see [Dimensions](#).

Different dimensions are collected for *service metrics* and *dependency metrics*. Within the services discovered by Application Signals, when microservice A calls microservice B, microservice B is serving the request. In this case, microservice A emits dependency metrics and microservice B emits service metrics. When a client calls microservice A, microservice A is serving the request and emits service metrics.

Dimensions for service metrics

The following dimensions are collected for service metrics.

Dimension	Description
Service	<p>The name of the service.</p> <p>The maximum value is 255 characters.</p>
Operation	<p>The name of the API operation or other activity.</p> <p>The maximum value is 1024 characters. Currently, you can set service level objectives on operations only if the operation name is 194 characters or fewer.</p>
Environment	<p>The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for <code>deployment.environment</code> in the <code>OTEL_ATTRIBUTES_RESOURCES</code> parameter.</p> <p>The maximum value is 259 characters.</p>

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

- [Environment, Service, Operation, [Latency, Error, Fault]]
- [Environment, Service, [Latency, Error, Fault]]

Dimensions for dependency metrics

The following dimensions are collected for dependency metrics:

Dimension	Description
Service	<p>The name of the service.</p> <p>The maximum value is 255 characters.</p>
Operation	<p>The name of the API operation or other operation.</p> <p>The maximum value is 1024 characters.</p>
RemoteService	<p>The name of the remote service being invoked.</p> <p>The maximum value is 255 characters.</p>
RemoteOperation	<p>The name of the API operation being invoked.</p> <p>The maximum value is 1024 characters.</p>
Environment	<p>The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for <code>deployment.environment</code> in the <code>OTEL_ATTRIBUTES_RESOURCES</code> parameter.</p> <p>The maximum value is 259 characters.</p>
RemoteEnvironment	<p>The name of the environment where dependency services are running. The <code>RemoteEnvironment</code> parameter is automatically generated when a service calls a dependency and they are both running in the same cluster. Otherwise, <code>RemoteEnvironment</code> is neither generated nor reported in the service dependency's metrics. Currently only available on Amazon EKS and K8S platforms.</p> <p>The maximum value is 259 characters.</p>
RemoteResourceIdentifier	<p>The name of the resource invoked by a remote call. The <code>RemoteResourceIdentifier</code> parameter is automatically generated if service calls a remote AWS service. Otherwise, <code>RemoteResourceIdentifier</code> is neither generated nor reported in the service dependency's metrics.</p>

Dimension	Description
	The maximum value is 1024 characters.
RemoteResourceType	<p>The type of the resource that is invoked by a remote call. Required only if RemoteResourceIdentifier is defined.</p> <p>The maximum value is 1024 characters.</p>

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

Running on Amazon EKS clusters

- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, Operation, RemoteService, RemoteOperation, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]
- [Environment, Service, RemoteService, RemoteOperation, [Latency, Error, Fault]]
- [RemoteService]

- [RemoteService, RemoteResourceIdentifier, RemoteResourceType]

Service level objectives (SLOs)

You can use Application Signals to create *service level objectives* for the services for your critical business operations. By creating SLOs on these services, you will be able to track them on the SLO dashboard, giving you an at-a-glance view of your most important operations.

In addition to creating a quick view your operators can use to see the current status of critical operations, you can use SLOs to track the longer-term performance of your services, to ensure that they are meeting your expectations. If you have service level agreements with customers, SLOs are a great tool to ensure that they are met.

Assessing your services' health with SLOs starts with setting clear, measurable objectives based on key performance metrics— *service level indicators (SLIs)*. An SLO tracks the SLI performance against the threshold and goal that you set, and reports how far or how close your application performance is to the threshold.

Application Signals helps you set SLOs on your key performance metrics. Application Signals automatically collects Latency and Availability metrics for every service and operation that it discovers, and these metrics are often ideal to use as SLIs. With the SLO creation wizard, you can use these metrics for your SLOs. You can then track the status of all of your SLOs with the Application Signals dashboards.

You can set SLOs on specific operations that your service calls or uses. You can use any CloudWatch metric or metric expression as an SLI, in addition to using the Latency and Availability metrics.

Creating SLOs is very important for getting the most benefit from CloudWatch Application Signals. After you create SLOs, you can view their status in the Application Signals console to quickly see which of your these critical services and operations are performing well and which are unhealthy. Having SLOs to track provides the following major benefits:

- It is easier for your service operators to see the current operational health of critical services as measured against the SLI. Then they can quickly triage and identify unhealthy services and operations.
- You can track your service performance against measurable business goals over longer periods of time.

By choosing what to set SLOs on, you are prioritizing what is important to you. The Application Signals dashboards automatically present information about what you have prioritized.

When you create an SLO, you can also choose to create CloudWatch alarms at the same time to monitor the SLOs. You can set alarms that monitor for breaches of the threshold, and also for warning levels. These alarms can automatically notify you if the SLO metrics are breaching the threshold that you set, or if they are nearing a warning threshold. For example, an SLO nearing its warning threshold can let you know that your team might need to slow down churn in the application to make sure that long-term performance goals are met.

Topics

- [SLO concepts](#)
- [Create an SLO](#)
- [View and triage SLO status](#)
- [Edit an existing SLO](#)
- [Delete an SLO](#)

SLO concepts

An SLO includes the following components:

- A *service level indicator (SLI)*, which is a key performance metric that you specify. It represents the desired level of performance for your application. Application Signals automatically collects the key metrics Latency and Availability for the services and operations that it discovers, and these can often be ideal metrics to set SLOs for.

You choose the threshold to use for your SLI. For example, 200ms for latency.

- A *goal or attainment goal*, which is the percentage of time that the SLI is expected to meet the threshold over each time *interval*. The time intervals can be as short as hours or as long as a year.

Intervals can be either calendar intervals or rolling intervals.

- *Calendar intervals* are aligned with the calendar, such as an SLO that is tracked per month. CloudWatch automatically adjusts health, budget, and attainment numbers based on the number of days in a month. Calendar intervals are better suited for business goals that are measured on a calendar-aligned basis.

- *Rolling intervals* are calculated on a rolling basis. Rolling intervals are better suited for tracking recent user experience of your application.
- The *period* is a shorter length of time, and many periods make up an interval. The application's performance is compared to the SLI during each period within the interval. For each period, the application is determined to have either achieved or not achieved the necessary performance.

For example, a goal of 99% with a calendar interval of one day and a period of 1 minute means that the application must meet or achieve the success threshold during 99% of the 1-minute periods during the day. If it does, then the SLO is met for that day. The next day is a new evaluation interval, and the application must meet or achieve the success threshold during 99% of the 1-minute periods during the second day to meet the SLO for that second day.

An SLI can be based on one of the new standard application metrics collected by Application Signals. Alternatively, it can be any CloudWatch metric or metric expression. The standard application metrics that you can use for an SLI are Latency and Availability. Availability represents the successful responses divided by the total requests. It is calculated as **(1 - Fault Rate)*100**, where Fault responses are 5xx errors. Success responses are responses without a 5XX error. 4XX responses are treated as successful.

Note

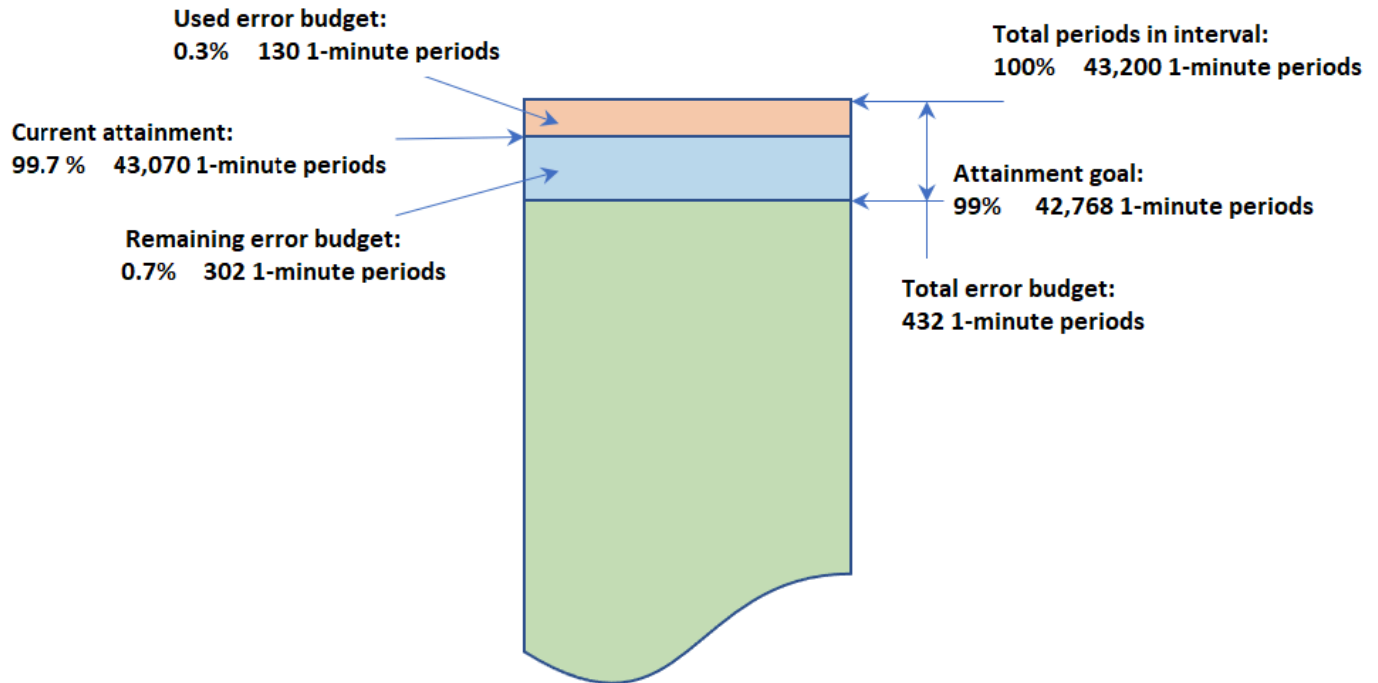
Currently only period-based calculations are supported. Support for volume- or requests-based calculations are planned for future releases.

Calculate error budget and attainment

When you view information about an SLO, you see its current health status and its *error budget*. The error budget is the amount of time within the interval that can breach the threshold but still let the SLO be met. The *total error budget* is the total amount of breaching time that can be tolerated through the entire interval. The *remaining error budget* is the remaining amount of breaching time that can be tolerated during the current interval. This is after the amount of breaching time that has already happened has been subtracted from the total error budget.

The following figure illustrates the attainment and error budget concepts for a goal with a 30-day interval, 1-minute periods, and a 99% attainment goal. 30 days includes 43,200 1-minute periods. 99% of 43,200 is 42,768, so 42,768 minutes during the month must be healthy for the SLO to be met. So far in the current interval, 130 of the 1-minute periods were unhealthy.

SLO with an interval of 30 days and 1-minute periods



Determine success within each period

Within each period, the SLI data is aggregated into a single data point based on the statistic used for the SLI. This data point represents the entire length of the period. That single data point is compared to the SLI threshold to determine if the period is healthy. Seeing unhealthy periods during the current time range on the dashboard can alert your service operators that the service needs to be triaged.

If the period is determined to be unhealthy, the entire length of the period is counted as failed against the error budget. Tracking the error budget lets you know whether the service is achieving the performance that you want over a longer period of time.

Create an SLO

We recommend that you set both latency and availability SLOs on your critical applications. These metrics collected by Application Signals align with common business goals.

You can also set SLOs on any CloudWatch metric or any metric math expression that results in a single time series.

The first time that you create an SLO in your account, CloudWatch automatically creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role in your account, if it doesn't already exist. This service-linked role allows CloudWatch to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications in your account. For more information about CloudWatch service-linked roles, see [Using service-linked roles for CloudWatch](#).

To create an SLO

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. Enter a name for the SLO. Including the name of a service or operation, along with appropriate keywords such as latency or availability, will help you quickly identify what the SLO status indicates during triage.
5. For **Set Service Level Indicator (SLI)**, do one of the following:
 - To set the SLO on either of the standard application metrics Latency or Availability:
 - a. Choose **Service Operation**.
 - b. Select the service that this SLO will monitor.
 - c. Select the operation that this SLO will monitor.

The **Select service** and **Select operation** drop-downs are populated by services and operations that have been active within the past 24 hours.

- d. Choose either **Availability** or **Latency** and then set the threshold.
- To set the SLO on any CloudWatch metric or a CloudWatch metric math expression:
 - a. Choose **CloudWatch Metric**.
 - b. Choose **Select CloudWatch metric**.

The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

After you select the metric that you want, choose the **Graphed metrics** tab and select the **Statistic** and **Period** to use for the SLO. Then choose **Select metric**.

For more information about these screens, see [Graph a metric](#) and [Add a math expression to a CloudWatch graph](#).


- c. For **Set condition**, select a comparison operator and threshold for the SLO to use as the indicator of success.
6. If you selected **Service Operation** in step 5, you can optionally choose **Additional settings** and then adjust the period length for this SLO.
7. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see [SLO concepts](#).
8. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.
 - a. CloudWatch alarms can use Amazon SNS to proactively notify you if an application is unhealthy based on its SLI performance.

To create an alarm, select one of the alarm check boxes and enter or create the Amazon SNS topic to use for notifications when the alarm goes into ALARM state. For more information about CloudWatch alarms, see [Using Amazon CloudWatch alarms](#). Creating alarms incurs charges. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

- b. If you set a warning threshold, it appears in Application Signals screens to help you identify SLOs that are in danger of being unmet, even if they're currently healthy.

To set a warning threshold, enter the threshold value in **Warning threshold**. When the SLO's error budget is lower than the warning threshold, the SLO is marked with **Warning** in several Application Signals screens. Warning thresholds also appear on error budget graphs. You can also create an **SLO warning alarm** that's based on the warning threshold.

9. To add tags to this SLO, choose the **Tags** tab and then choose **Add new tag**. Tags can help you manage, identify, organize, search for, and filter resources. For more information about tagging, see [Tagging your AWS resources](#).

 **Note**

If the application this SLO is related to is registered in AWS Service Catalog AppRegistry, you can use the `awsApplication` tag to associate this SLO with that application in AppRegistry. For more information, see [What is AppRegistry?](#)

10. Choose **Create SLO**. If you also chose to create one or more alarms, the button name changes to reflect this.

View and triage SLO status

You can quickly see the health of your SLOs using either the **Service Level Objectives** or the **Services** options in the CloudWatch console. The **Services** view provides an at-a-glance view of the ratio of unhealthy services, calculated based on SLOs that you have set. For more information about using the **Services** option, see [Monitor the operational health of your applications with Application Signals](#).

The **Service Level Objectives** view provides a macro view of your organization. You can see the met and unmet SLOs as a whole. This gives you a view of how many of your services and operations are performing to your expectations over longer periods of time, according to the SLIs that you chose.

To view all of your SLOs using the Service Level Objectives view

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.

The **Service Level Objectives (SLO)** list appears.

You can quickly see the current status of your SLOs in the **SLI status** column. To sort the SLOs so that all the unhealthy ones are at the top of the list, choose the **SLI status** column until the unhealthy SLOs are all at the top.

The SLO table has the following default columns. You can adjust which columns are displayed by choosing the gear icon above the list. For more information about goals, SLIs, attainment, and intervals, see [SLO concepts](#).

- The name of the SLO.
- The **Goal** column displays the percentage of periods during each interval that must successfully meet the SLI threshold for the SLO goal to be met. It also displays the interval length for the SLO.
- The **SLI status** displays whether the current operational state of the application is healthy or not. If any period during the currently selected time range was unhealthy for the SLO, then the **SLI status** displays **Unhealthy**.

- The **Ending attainment** is the attainment level achieved as of the end of the selected time range. Sort by this column to see the SLOs that are most in danger of not being met.
 - The **Attainment delta** is the difference in attainment level between the start and end of the selected time range. A negative delta means that the metric is trending in a downward direction. Sort by this column to see the latest trends of the SLOs.
 - The **Ending error budget (%)** is the percentage of total time in the period that can have unhealthy periods and still have the SLO be achieved successfully. If you set this to 5%, and the SLI is unhealthy in 5% or fewer of the remaining periods in the interval, the SLO is still achieved successfully.
 - The **Error budget delta** is the difference in error budget between the start and end of the selected time range. A negative delta means that the metric is trending in a failing direction.
 - The **Ending error budget (time)** is the amount of actual time in the interval that can be unhealthy and still have the SLO be achieved successfully. For example, if this is 14 minutes, then if the SLI is unhealthy for fewer than 14 minutes during the remaining interval, the SLO will still be achieved successfully.
 - The **Service, Operation, and Type** columns display information about what service and operation this SLO is set for.
3. To see the attainment and error budget graphs for an SLO, choose the radio button next to the SLO name.

The graphs at the top of the page display the **SLO attainment** and **Error budget** status. A graph about the SLI metric associated with this SLO is also displayed.

4. To further triage an SLO that is not meeting its goal, choose the service name or operation name associated with that SLO. You are taken to the details page where you can triage further. For more information, see [View detailed service activity and operational health with the service detail page](#).
5. To change the time range of the charts and tables on the page, choose a new time range near the top of the screen.

Edit an existing SLO

Follow these steps to edit an existing SLO. When you edit an SLO, you can change only the threshold, interval, attainment goal, and tags. To change other aspects such as service, operation, or metric, create a new SLO instead of editing an existing one.

Changing part of an SLO core configuration, such as period or threshold, invalidates all the previous data points and assessments about attainment and health. It effectively deletes and re-creates the SLO.

Note

If you edit an SLO, alarms associated with that SLO are not automatically updated. You might need to update the alarms to keep them in sync with the SLO.

To edit an existing SLO

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions, Edit SLO**.
4. Make your changes, then choose **Save changes**.

Delete an SLO

Follow these steps to delete an existing SLO.

Note

When you delete an SLO, alarms associated with that SLO are not automatically deleted. You'll need to delete them yourself. For more information, see [Managing alarms](#).

To delete an SLO

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions, Delete SLO**.
4. Choose **Confirm**.

Synthetic monitoring (canaries)

You can use Amazon CloudWatch Synthetics to create *canaries*, configurable scripts that run on a schedule, to monitor your endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your applications. By using canaries, you can discover issues before your customers do.

Canaries are scripts written in Node.js or Python. They create Lambda functions in your account that use Node.js or Python as a framework. Canaries work over both HTTP and HTTPS protocols. Canaries use Lambda layers that contain the CloudWatch Synthetics library. The library contains the NodeJS version of CloudWatch Synthetics for NodeJS canaries and the Python version of CloudWatch Synthetics for Python canaries. The layers belong to the CloudWatch Synthetics service account. Libraries never transmit or store customer information. All customer data is stored in the customer account only.

Canaries offer programmatic access to a headless Google Chrome Browser via Puppeteer or Selenium Webdriver. For more information about Puppeteer, see [Puppeteer](#). For more information about Selenium, see www.selenium.dev/.

Canaries check the availability and latency of your endpoints and can store load time data and screenshots of the UI. They monitor your REST APIs, URLs, and website content, and they can check for unauthorized changes from phishing, code injection and cross-site scripting.

CloudWatch Synthetics is integrated with [Application Signals](#), which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see your canaries in Application Signals, [turn on X-Ray active tracing](#). Your canaries are displayed on the [Service Map](#) connected to your services, and in the [Service detail](#) page of the services they call.

For a video demonstration of canaries, see the following:

- [Introduction to Amazon CloudWatch Synthetics](#)
- [Amazon CloudWatch Synthetics Demo](#)
- [Create Canaries Using Amazon CloudWatch Synthetics](#)
- [Visual Monitoring with Amazon CloudWatch Synthetics](#)

You can run a canary once or on a regular schedule. Canaries can run as often as once per minute. You can use both cron and rate expressions to schedule canaries.

For information about security issues to consider before you create and run canaries, see [Security considerations for Synthetics canaries](#).

By default, canaries create several CloudWatch metrics in the `CloudWatchSynthetics` namespace. These metrics have `CanaryName` as a dimension. Canaries that use the `executeStep()` or `executeHttpStep()` function from the function library also have `StepName` as a dimension. For more information about the canary function library, see [Library functions available for canary scripts](#).

CloudWatch Synthetics integrates well with the X-Ray Trace Map, which uses CloudWatch with AWS X-Ray to provide an end-to-end view of your services to help you more efficiently pinpoint performance bottlenecks and identify impacted users. Canaries that you create with CloudWatch Synthetics appear on the trace map. For more information, see [X-Ray Trace Map](#).

CloudWatch Synthetics is currently available in all commercial AWS Regions and the GovCloud Regions.

Note

In Asia Pacific (Osaka), AWS PrivateLink is not supported. In Asia Pacific (Jakarta), AWS PrivateLink and X-Ray are not supported.

Topics

- [Required roles and permissions for CloudWatch canaries](#)
- [Creating a canary](#)
- [Groups](#)
- [Test a canary locally](#)
- [Troubleshooting a failed canary](#)
- [Sample code for canary scripts](#)
- [Canaries and X-Ray tracing](#)
- [Running a canary on a VPC](#)

- [Encrypting canary artifacts](#)
- [Viewing canary statistics and details](#)
- [CloudWatch metrics published by canaries](#)
- [Edit or delete a canary](#)
- [Start, stop, delete, or update runtime for multiple canaries](#)
- [Monitoring canary events with Amazon EventBridge](#)

Required roles and permissions for CloudWatch canaries

Both the users who create and manage canaries, and the canaries themselves, must have certain permissions.

Required roles and permissions for users who manage CloudWatch canaries

To view canary details and the results of canary runs, you must be signed in as a user with either the `CloudWatchSyntheticsFullAccess` or the `CloudWatchSyntheticsReadOnlyAccess` policies attached. To read all Synthetics data in the console, you also need the `AmazonS3ReadOnlyAccess` and `CloudWatchReadOnlyAccess` policies. To view the source code used by canaries, you also need the `AWSLambda_ReadOnlyAccess` policy.

To create canaries, you must be signed in as a user who has the `CloudWatchSyntheticsFullAccess` policy or a similar set of permissions. To create IAM roles for the canaries, you also need the following inline policy statement:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
        "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
      ]
    }
  ]
}
```

```
]
}
```

⚠ Important

Granting a user the `iam:CreateRole`, `iam:CreatePolicy`, and `iam:AttachRolePolicy` permissions gives that user full administrative access to your AWS account. For example, a user with these permissions can create a policy that has full permissions for all resources and can attach that policy to any role. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see [Changing Permissions for an IAM User](#) and [To embed an inline policy for a user or role](#).

Required roles and permissions for canaries

Each canary must be associated with an IAM role that has certain permissions attached. When you create a canary using the CloudWatch console, you can choose for CloudWatch Synthetics to create an IAM role for the canary. If you do, the role will have the permissions needed.

If you want to create the IAM role yourself, or create an IAM role that you can use when using the AWS CLI or APIs to create a canary, the role must contain the permissions listed in this section.

All IAM roles for canaries must include the following trust policy statement.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Additionally, the canary's IAM role needs one of the following statements.

Basic canary that doesn't use AWS KMS or need Amazon VPC access

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "xray:PutTraceSegments"
      ],
      "Resource": [
```



```

        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
}
]
}

```

Canary that uses AWS KMS to encrypt canary artifacts but does not need Amazon VPC access

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation"
        ],
        "Resource": [
            "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",

```

```

        "logs:PutLogEvents",
        "logs:CreateLogGroup"
    ],
    "Resource": [
        "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:ListAllMyBuckets",
        "xray:PutTraceSegments"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource":
"arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": [
                "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
            ]
        }
    }
}
}

```

```
]
}
```

Canary that does not use AWS KMS but does need Amazon VPC access

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetBucketLocation"
    ],
    "Resource": [
      "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogStream",
      "logs:PutLogEvents",
      "logs:CreateLogGroup"
    ],
    "Resource": [
      "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:ListAllMyBuckets",
      "xray:PutTraceSegments"
    ]
  }
]
```

```

    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2>DeleteNetworkInterface"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

Canary that uses AWS KMS to encrypt canary artifacts and also needs Amazon VPC access

If you update a non-VPC canary to start using a VPC, you'll need to update the canary's role to include the network interface permissions listed in the following policy.

```

{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject"
        ],
        "Resource": [

```

```

        "arn:aws:s3::path/to/your/S3/bucket/canary/results/folder"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation"
    ],
    "Resource": [
        "arn:aws:s3::name/of/the/S3/bucket/that/contains/canary/results"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup"
    ],
    "Resource": [
        "arn:aws:logs:canary_region_name:canary_account_id:log-group:/aws/
lambda/cwsyn-canary_name-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:ListAllMyBuckets",
        "xray:PutTraceSegments"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": "CloudWatchSynthetics"
        }
    }
},
},

```

```

    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2>DeleteNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource":
"arn:aws:kms:KMS_key_region_name:KMS_key_account_id:key/KMS_key_id",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": [
            "s3.region_name_of_the_canary_results_S3_bucket.amazonaws.com"
          ]
        }
      }
    }
  ]
}

```

AWS managed policies for CloudWatch Synthetics

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) AWS managed policies in the IAM User Guide.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally change the permissions in an AWS managed policy. This type of update affects all identities (users, groups, and roles) where the policy is attached.

CloudWatch Synthetics updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch Synthetics since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
Redundant actions removed from CloudWatchSyntheticsFullAccess	CloudWatch Synthetics removed the <code>s3:PutBucketEncryption</code> and <code>lambda:GetLayerVersionByArn</code> actions from CloudWatchSyntheticsFullAccess policy because those actions were redundant with other permissions in the policy. The removed actions did not provide any permissions, and there's no net change to the permissions granted by the policy.	March 12, 2021
CloudWatch Synthetics started tracking changes	CloudWatch Synthetics started tracking changes for its AWS managed policies.	March 10, 2021

CloudWatchSyntheticsFullAccess

Here are the contents of the `CloudWatchSyntheticsFullAccess` policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "synthetics:*"
      ]
    }
  ]
}
```

```
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:CreateBucket",
      "s3:PutEncryptionConfiguration"
    ],
    "Resource": [
      "arn:aws:s3:::cw-syn-results-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:ListRoles",
      "s3:ListAllMyBuckets",
      "s3:GetBucketLocation",
      "xray:GetTraceSummaries",
      "xray:BatchGetTraces",
      "apigateway:GET"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": "arn:aws:s3:::cw-syn-*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObjectVersion"
    ],
    "Resource": "arn:aws:s3:::aws-synthetics-library-*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
  },
```



```

    "Resource":[
      "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ],
    "Condition":{
      "StringEquals":{
        "iam:PassedToService":[
          "lambda.amazonaws.com",
          "synthetics.amazonaws.com"
        ]
      }
    }
  },
  {
    "Effect":"Allow",
    "Action":[
      "iam:GetRole"
    ],
    "Resource":[
      "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "cloudwatch:GetMetricData",
      "cloudwatch:GetMetricStatistics"
    ],
    "Resource":""
  },
  {
    "Effect":"Allow",
    "Action":[
      "cloudwatch:PutMetricAlarm",
      "cloudwatch:DeleteAlarms"
    ],
    "Resource":[
      "arn:aws:cloudwatch::*:alarm:Synthetics-*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "cloudwatch:DescribeAlarms"
    ],

```

```

    "Resource":[
      "arn:aws:cloudwatch:*:*:alarm:*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "lambda:CreateFunction",
      "lambda:AddPermission",
      "lambda:PublishVersion",
      "lambda:UpdateFunctionConfiguration",
      "lambda:GetFunctionConfiguration"
    ],
    "Resource":[
      "arn:aws:lambda:*:*:function:cwsyn-*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "lambda:GetLayerVersion",
      "lambda:PublishLayerVersion"
    ],
    "Resource":[
      "arn:aws:lambda:*:*:layer:cwsyn-*",
      "arn:aws:lambda:*:*:layer:Synthetics:*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "ec2:DescribeVpcs",
      "ec2:DescribeSubnets",
      "ec2:DescribeSecurityGroups"
    ],
    "Resource":[
      "*"
    ]
  },
  {
    "Effect":"Allow",
    "Action":[
      "sns:ListTopics"
    ],
  },

```

```

    "Resource": [
      "*"
    ],
  },
  {
    "Effect": "Allow",
    "Action": [
      "sns:CreateTopic",
      "sns:Subscribe",
      "sns:ListSubscriptionsByTopic"
    ],
    "Resource": [
      "arn:*:sns:*:*:Synthetics-*"
    ]
  }
]
}

```

CloudWatchSyntheticsReadOnlyAccess

Here are the contents of the CloudWatchSyntheticsReadOnlyAccess policy:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "synthetics:Describe*",
        "synthetics:Get*",
        "synthetics:List*"
      ],
      "Resource": "*"
    }
  ]
}

```

Limiting a user to viewing specific canaries

You can limit a user's ability to view information about canaries, so that they can only see information about the canaries you specify. To do this, use an IAM policy with a Condition statement similar to the following, and attach this policy to a user or an IAM role.

The following example limits the user to only viewing information about `name-of-allowed-canary-1` and `name-of-allowed-canary-2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "synthetics:DescribeCanaries",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "synthetics:Names": [
            "name-of-allowed-canary-1",
            "name-of-allowed-canary-2"
          ]
        }
      }
    }
  ]
}
```

CloudWatch Synthetics supports listing as many as five items in the `synthetics:Names` array.

You can also create a policy that uses a `*` as a wildcard in canary names that are to be allowed, as in the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "synthetics:DescribeCanaries",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "synthetics:Names": [
            "my-team-canary-*"
          ]
        }
      }
    }
  ]
}
```

```
    }  
  ]  
}
```

Any user signed in with one of these policies attached can't use the CloudWatch console to view any canary information. They can view canary information only for the canaries authorized by the policy and only by using the [DescribeCanaries](#) API or the [describe-canaries](#) AWS CLI command.

Creating a canary

Important

Ensure that you use Synthetics canaries to monitor only endpoints and APIs where you have ownership or permissions. Depending on the canary frequency settings, these endpoints might experience increased traffic.

When you use the CloudWatch console to create a canary, you can use a blueprint provided by CloudWatch to create your canary or you can write your own script. For more information, see [Using canary blueprints](#).

You can also create a canary using AWS CloudFormation if you are using your own script for the canary. For more information, see [AWS::Synthetics::Canary](#) in the *AWS CloudFormation User Guide*.

If you are writing your own script, you can use several functions that CloudWatch Synthetics has built into a library. For more information, see [Synthetics runtime versions](#).

Note

When you create a canary, one of the layers created is a Synthetics layer prepended with Synthetics. This layer is owned by the Synthetics service account and contains the runtime code.

To create a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.

3. Choose **Create Canary**.
4. Choose one of the following:
 - To base your canary on a blueprint script, choose **Use a blueprint**, and then choose the type of canary you want to create. For more information about what each type of blueprint does, see [Using canary blueprints](#).
 - To upload your own Node.js script to create a custom canary, choose **Upload a script**.

You can then drag your script into the **Script** area or choose **Browse files** to navigate to the script in your file system.

- To import your script from an S3 bucket, choose **Import from S3**. Under **Source location**, enter the complete path to your canary or choose **Browse S3**.

You must have `s3:GetObject` and `s3:GetObjectVersion` permissions for the S3 bucket that you use. The bucket must be in the same AWS Region where you are creating the canary.

5. Under **Name**, enter a name for your canary. The name is used on many pages, so we recommend that you give it a descriptive name that distinguishes it from other canaries.
6. Under **Application or endpoint URL**, enter the URL that you want the canary to test. This URL must include the protocol (such as `https://`).

If you want the canary to test an endpoint on a VPC, you must also enter information about your VPC later in this procedure.

7. If you are using your own script for the canary, under **Lambda handler**, enter the entry point where you want the canary to start. If you use a runtime earlier than `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1`, the string that you enter must end with `.handler`. If you use `syn-nodejs-puppeteer-3.4` or `syn-python-selenium-1.1` or a later runtime, this restriction does not apply.
8. If you are using environment variables in your script, choose **Environment variables** and then specify a value for each environment variable defined in your script. For more information, see [Environment variables](#).
9. Under **Schedule**, choose whether to run this canary just once, run it continuously using a rate expression, or schedule it using a cron expression.
 - When you use the CloudWatch console to create a canary that runs continuously, you can choose a rate anywhere between once a minute and once an hour.

- For more information about writing a cron expression for canary scheduling, see [Scheduling canary runs using cron](#).
10. (Optional) To set a timeout value for the canary, choose **Additional configuration** and then specify the timeout value. Make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.
 11. Under **Data retention**, specify how long to retain information about both failed and successful canary runs. The range is 1-455 days.

This setting affects only the data that CloudWatch Synthetics stores and displays in the console. It does not affect the data stored in your Amazon S3 buckets, or logs or metrics that are published by the canary.

12. Under **Data Storage**, select the S3 bucket to use to store the data from the canary runs. The bucket name can't contain a period (.). If you leave this blank, a default S3 bucket is used or created.

If you are using the `syn-nodejs-puppeteer-3.0` or later runtime, when you enter the URL for the bucket in the text box, you can specify a bucket in the current Region or in a different Region. If you are using an earlier runtime version, the bucket must be in the current Region.

13. (Optional) By default, canaries store their artifacts on Amazon S3, and the artifacts are encrypted at rest using an AWS-managed AWS KMS key. You can use a different encryption option by choosing **Additional configuration** in the **Data Storage** section. You can then choose the type of key to use for encryption. For more information, see [Encrypting canary artifacts](#).
14. Under **Access permissions**, choose whether to create an IAM role to run the canary or use an existing one.

If you have CloudWatch Synthetics create the role, it automatically includes all the necessary permissions. If you want to create the role yourself, see [Required roles and permissions for canaries](#) for information about the necessary permissions.

If you use the CloudWatch console to create a role for a canary when you create the canary, you can't re-use the role for other canaries, because these roles are specific to just one canary. If you have manually created a role that works for multiple canaries, you can use that existing role.

To use an existing role, you must have the `iam:PassRole` permission to pass that role to Synthetics and Lambda. You must also have the `iam:GetRole` permission.

15. (Optional) Under **Alarms**, choose whether you want default CloudWatch alarms to be created for this canary. If you choose to create alarms, they are created with the following name convention: `Synthetics-Alarm-canaryName-index`

`index` is a number representing each different alarm created for this canary. The first alarm has an index of 1, the second alarm has an index of 2, and so on.

16. (Optional) To have this canary test an endpoint that is on a VPC, choose **VPC settings**, and then do the following:
 - a. Select the VPC that hosts the endpoint.
 - b. Select one or more subnets on your VPC. You must select a private subnet because a Lambda instance can't be configured to run in a public subnet when an IP address can't be assigned to the Lambda instance during execution. For more information, see [Configuring a Lambda Function to Access Resources in a VPC](#).
 - c. Select one or more security groups on your VPC.

If the endpoint is on a VPC, you must enable your canary to send information to CloudWatch and Amazon S3. For more information, see [Running a canary on a VPC](#).

17. (Optional) Under **Tags**, add one or more key-value pairs as tags for this canary. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
18. (Optional) Under **Active tracing**, choose whether to enable active X-Ray tracing for this canary. This option is available only if the canary uses runtime version **syn-nodejs-2.0** or later. For more information, see [Canaries and X-Ray tracing](#).

Resources that are created for canaries

When you create a canary, the following resources are created:

- An IAM role with the name `CloudWatchSyntheticsRole-canary-name-uuid` (if you use CloudWatch console to create the canary and specify for a new role to be created for the canary)
- An IAM policy with the name `CloudWatchSyntheticsPolicy-canary-name-uuid`.
- An S3 bucket with the name `cw-syn-results-accountID-region`.

- Alarms with the name `Synthetics-Alarm-MyCanaryName`, if you want alarms to be created for the canary.
- Lambda functions and layers, if you use a blueprint to create the canary. These resources have the prefix `cwsyn-MyCanaryName`.
- CloudWatch Logs log groups with the name `/aws/lambda/cwsyn-MyCanaryName-randomId`.

Using canary blueprints

This section provides details about each of the canary blueprints and the tasks each blueprint is best suited for. Blueprints are provided for the following canary types:

- Heartbeat Monitor
- API Canary
- Broken Link Checker
- Visual Monitoring
- Canary Recorder
- GUI Workflow

When you use a blueprint to create a canary, as you fill out the fields in the CloudWatch console, the **Script editor** area of the page displays the canary you are creating as a Node.js script. You can also edit your canary in this area to customize it further.

Heartbeat monitoring

Heartbeat scripts load the specified URL and store a screenshot of the page and an HTTP archive file (HAR file). They also store logs of accessed URLs.

You can use the HAR files to view detailed performance data about the web pages. You can analyze the list of web requests and catch performance issues such as time to load for an item.

If your canary uses the `syn-nodejs-puppeteer-3.1` or later runtime version, you can use the heartbeat monitoring blueprint to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

API canary

API canaries can test the basic Read and Write functions of a REST API. REST stands for *representational state transfer* and is a set of rules that developers follow when creating an API. One of these rules states that a link to a specific URL should return a piece of data.

Canaries can work with any APIs and test all types of functionality. Each canary can make multiple API calls.

In canaries that use runtime version `syn-nodejs-2.2` or later, the API canary blueprint supports multi-step canaries that monitor your APIs as HTTP steps. You can test multiple APIs in a single canary. Each step is a separate request that can access a different URL, use different headers, and use different rules for whether headers and response bodies are captured. By not capturing headers and response body, you can prevent sensitive data from being recorded.

Each request in an API canary consists of the following information:

- The *endpoint*, which is the URL that you request.
- The *method*, which is the type of request that is sent to the server. REST APIs support GET (read), POST (write), PUT (update), PATCH (update), and DELETE (delete) operations.
- The *headers*, which provide information to both the client and the server. They are used for authentication and providing information about the body content. For a list of valid headers, see [HTTP Headers](#).
- The *data* (or *body*), which contains information to be sent to the server. This is used only for POST, PUT, PATCH, or DELETE requests.

The API canary blueprint supports GET and POST methods. When you use this blueprint, you must specify headers. For example, you can specify **Authorization** as a **Key** and specify the necessary authorization data as the **Value** for that key.

If you are testing a POST request, you also specify the content to post in the **Data** field.

Integration with API Gateway

The API blueprint is integrated with Amazon API Gateway. This enables you to select an API Gateway API and stage from the same AWS account and Region as the canary, or to upload a Swagger template from API Gateway for cross-account and cross-Region API monitoring. You can then choose the rest of the details in the console to create the canary, instead of entering them from scratch. For more information about API Gateway, see [What is Amazon API Gateway?](#)

Using a private API

You can create a canary that uses a private API in Amazon API Gateway. For more information, see [Creating a private API in Amazon API Gateway?](#)

Broken link checker

The broken link checker collects all the links inside the URL that you are testing by using `document.getElementsByTagName(' a ')`. It tests only up to the number of links that you specify, and the URL itself is counted as the first link. For example, if you want to check all the links on a page that contains five links, you must specify for the canary to follow six links.

Broken link checker canaries created using the `syn-nodejs-2.0-beta` runtime or later support the following additional features:

- Provides a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- When viewing canary results, you can filter to see only the broken links and then fix the link based on the reason for failure.
- This version captures annotated source page screenshots for each link and highlights the anchor where the link was found. Hidden components are not annotated.
- You can configure this version to capture screenshots of both source and destination pages, just source pages, or just destination pages.
- This version fixes an issue in the earlier version where the canary script stops after the first broken link even when more links are scraped from the first page.

If you want to update an existing canary using `syn-1.0` to use the new runtime, you must delete and re-create the canary. Updating an existing canary to the new runtime does not make these features available.

A broken link checker canary detects the following types of link errors:

- 404 Page Not Found
- Invalid Host Name
- Bad URL. For example, the URL is missing a bracket, has extra slashes, or uses the wrong protocol.
- Invalid HTTP response code.

- The host server returns empty responses with no content and no response code.
- The HTTP requests constantly time out during the canary's run.
- The host consistently drops connections because it is misconfigured or is too busy.

Visual monitoring blueprint

The visual monitoring blueprint includes code to compare screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium.

The visual monitoring blueprint includes the following line of code in the default blueprint canary script, which enables visual monitoring.

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see [Edit or delete a canary](#).

You can also change the canary run that is used as the baseline by using the `nextRun` or `lastRun` parameters or specifying a canary run ID in the [UpdateCanary](#) API.

When you use the visual monitoring blueprint, you enter the URL where you want the screenshot to be taken, and specify a difference threshold as a percentage. After the baseline run, future runs of the canary that detect a visual difference greater than that threshold trigger a canary failure. After the baseline run, you can also edit the canary to "draw" boundaries on the baseline screenshot that you want to ignore during the visual monitoring.

The visual monitoring feature is powered by the the ImageMagick open source software toolkit. For more information, see [ImageMagick](#).

Canary recorder

With the canary recorder blueprint, you can use the CloudWatch Synthetics Recorder to record your click and type actions on a website and automatically generate a Node.js script that can be used to create a canary that follows the same steps. The CloudWatch Synthetics Recorder is a Google Chrome extension provided by Amazon.

Credits: The CloudWatch Synthetics Recorder is based on the [Headless recorder](#).

For more information, see [Using the CloudWatch Synthetics Recorder for Google Chrome](#).

GUI workflow builder

The GUI Workflow Builder blueprint verifies that actions can be taken on your webpage. For example, if you have a webpage with a login form, the canary can populate the user and password fields and submit the form to verify that the webpage is working correctly.

When you use a blueprint to create this type of canary, you specify the actions that you want the canary to take on the webpage. The actions that you can use are the following:

- **Click**— Selects the element that you specify and simulates a user clicking or choosing the element.

To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Verify selector**— Verifies that the specified element exists on the webpage. This test is useful for verifying that a previous action has causes the correct elements to populate the page.

To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Verify text**— Verifies that the specified string is contained within the target element. This test is useful for verifying that a previous action has caused the correct text to be displayed.

To specify the element in a Node.js script, use a format such as `div[@id=]//h1` because this action uses the `waitForXPath` function in Puppeteer.

To specify the element in a Python script, use xpath format such as `//*[@id=]` or `//*[@class=]` because this action uses the `implicitly_wait` function in Selenium.

- **Input text**— Writes the specified text in the target element.

To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element to verify in a Python script, use xpath `//*[@id=]` or `//*[@class=]`.

- **Click with navigation**— Waits for the whole page to load after choosing the specified element. This is most useful when you need to reload the page.

To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

To specify the element in a Python script, use xpath `//*[@id=]` or `//*[@class=]`.

For example, the following blueprint uses Node.js. It clicks the **firstButton** on the specified URL, verifies that the expected selector with the expected text appears, inputs the name `Test_Customer` into the **Name** field, clicks the **Login** button, and then verifies that the login is successful by checking for the **Welcome** text on the next page.

Application or endpoint URL [Info](#)

https://

Enter the endpoint, API or url that you are testing.

Workflow builder
Select the actions you would like the canary to take.

Action	Selector	Text	
Click	[id='firstButton']		Remove action
Verify selector	div[id='screen2Text']		Remove action
Verify text	[@id='screen2Text']//h3	Type	Remove action
Input text	input[id='Name']	Test_Customer	Remove action
Click with navigation	[id='Login']		Remove action
Verify text	div[@id='welcome']//h1	Welcome	Remove action

GUI workflow canaries that use the following runtimes also provide a summary of the steps executed for each canary run. You can use the screenshots and error message associated with each step to find the root cause of failure.

- syn-nodejs-2.0 or later
- syn-python-selenium-1.0 or later

Using the CloudWatch Synthetics Recorder for Google Chrome

Amazon provides a CloudWatch Synthetics Recorder to help you create canaries more easily. The recorder is a Google Chrome extension.

The recorder records your click and type actions on a website and automatically generates a Node.js script that can be used to create a canary that follows the same steps.

After you start recording, the CloudWatch Synthetics Recorder detects your actions in the browser and converts them to a script. You can pause and resume the recording as needed. When you stop recording, the recorder produces a Node.js script of your actions, which you can easily copy with the **Copy to Clipboard** button. You can then use this script to create a canary in CloudWatch Synthetics.

Credits: The CloudWatch Synthetics Recorder is based on the [Headless recorder](#).

Installing the CloudWatch Synthetics Recorder extension for Google Chrome

To use the CloudWatch Synthetics Recorder, you can start creating a canary and choose the **Canary Recorder** blueprint. If you do this when you haven't already downloaded the recorder, the CloudWatch Synthetics console provides a link to download it.

Alternatively, you can follow these steps to download and install the recorder directly.

To install the CloudWatch Synthetics Recorder

1. Using Google Chrome, go to this website: <https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkkdfplenecpegfno>
2. Choose **Add to Chrome**, then choose **Add extension**.

Using the CloudWatch Synthetics Recorder for Google Chrome

To use the CloudWatch Synthetics Recorder to help you create a canary, you can choose **Create canary** in the CloudWatch console, and then choose **Use a blueprint, Canary Recorder**. For more information, see [Creating a canary](#).

Alternatively, you can use the recorder to record steps without immediately using them to create a canary.

To use the CloudWatch Synthetics Recorder to record your actions on a website

1. Navigate to the page that you want to monitor.
2. Choose the Chrome extensions icon, and then choose **CloudWatch Synthetics Recorder**.
3. Choose **Start Recording**.
4. Perform the steps that you want to record. To pause recording, choose **Pause**.
5. When you are finished recording the workflow, choose **Stop recording**.

6. Choose **Copy to clipboard** to copy the generated script to your clipboard. Or, if you want to start over, choose **New recording**.
7. To create a canary with the copied script, you can paste your copied script into the recorder blueprint inline editor, or save it to an Amazon S3 bucket and import it from there.
8. If you're not immediately creating a canary, you can save your recorded script to a file.

Known limitations of the CloudWatch Synthetics Recorder

The CloudWatch Synthetics Recorder for Google Chrome currently has the following limitations.

- HTML elements that don't have IDs will use CSS selectors. This can break canaries if the webpage structure changes later. We plan to provide some configuration options (such as using data-id) around this in a future version of the recorder.
- The recorder doesn't support actions such as double-click or copy/paste, and doesn't support key combinations such as CMD+O.
- To verify the presence of an element or text on the page, users must add assertions after the script is generated. The recorder doesn't support verifying an element without performing any action on that element. This is similar to the "Verify text" or "Verify element" options in the canary workflow builder. We plan to add some assertions support in a future version of the recorder.
- The recorder records all actions in the tab where the recording is initiated. It doesn't record pop-ups (for instance, to allow location tracking) or navigation to different pages from pop-ups.

Synthetics runtime versions

When you create or update a canary, you choose a Synthetics runtime version for the canary. A Synthetics runtime is a combination of the Synthetics code that calls your script handler, and the Lambda layers of bundled dependencies.

CloudWatch Synthetics currently supports runtimes that use Node.js for scripts and the Puppeteer framework, and runtimes that use Python for scripting and Selenium Webdriver for the framework.

We recommend that you always use the most recent runtime version for your canaries, to be able to use the latest features and updates made to the Synthetics library.

When you create a canary, one of the layers created is a Synthetics layer prepended with `Synthetics`. This layer is owned by the Synthetics service account and contains the runtime code.

Note

Whenever you upgrade a canary to use a new version of the the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically upgraded to the same version of NodeJS that the Synthetics runtime supports.

Topics

- [CloudWatch Synthetics runtime support policy](#)
- [Runtime versions using Node.js and Puppeteer](#)
- [Runtime versions using Python and Selenium Webdriver](#)

CloudWatch Synthetics runtime support policy

Synthetics runtime versions are subject to maintenance and security updates. When any component of a runtime version is no longer supported, that Synthetics runtime version is deprecated.

You can't create canaries using deprecated runtime versions. Canaries that use deprecated runtimes continue to run. You can stop, start, and delete these canaries. You can update an existing canary that uses a deprecated runtime version by updating the canary to use a supported runtime version.

CloudWatch Synthetics notifies you by email if you have canaries that use runtimes that are scheduled to be deprecated in the next 60 days. We recommend that you migrate your canaries to a supported runtime version to benefit from the new functionality, security, and performance enhancements that are included in more recent releases.

How do I update a canary to a new runtime version?

You can update a canary's runtime version by using the CloudWatch console, AWS CloudFormation, the AWS CLI or the AWS SDK. When you use the CloudWatch console, you can update up to five canaries at once by selecting them in the canary list page and then choosing **Actions, Update Runtime**.

You can verify the upgrade by first cloning the canary using the CloudWatch console and updating its runtime version. This creates another canary which is a clone of your original canary. Once you have verified your canary with the new runtime version, you can update the runtime version of your original canary and delete the clone canary.

You can also update multiple canaries using an upgrade script. For more information, see [Canary runtime upgrade script](#).

If you upgrade a canary and it fails, see [Troubleshooting a failed canary](#).

Runtime deprecation dates

Runtime Version	Deprecation date
syn-nodejs-puppeteer-6.1	March 8, 2024
syn-nodejs-puppeteer-6.0	March 8, 2024
syn-nodejs-puppeteer-5.1	March 8, 2024
syn-nodejs-puppeteer-5.0	March 8, 2024
syn-nodejs-puppeteer-4.0	March 8, 2024
syn-nodejs-puppeteer-3.9	January 8, 2024
syn-nodejs-puppeteer-3.8	January 8, 2024
syn-python-selenium-2.0	March 8, 2024
syn-python-selenium-1.3	March 8, 2024
syn-python-selenium-1.2	March 8, 2024

Runtime Version	Deprecation date
syn-python-selenium-1.1	March 8, 2024
syn-python-selenium-1.0	March 8, 2024
syn-nodejs-puppeteer-3.7	January 8, 2024
syn-nodejs-puppeteer-3.6	January 8, 2024
syn-nodejs-puppeteer-3.5	January 8, 2024
syn-nodejs-puppeteer-3.4	November 13, 2022
syn-nodejs-puppeteer-3.3	November 13, 2022
syn-nodejs-puppeteer-3.2	November 13, 2022
syn-nodejs-puppeteer-3.1	November 13, 2022
syn-nodejs-puppeteer-3.0	November 13, 2022
syn-nodejs-2.2	May 28, 2021
syn-nodejs-2.1	May 28, 2021
syn-nodejs-2.0	May 28, 2021
syn-nodejs-2.0-beta	February 8, 2021

Runtime Version	Deprecation date
syn-1.0	May 28, 2021

Canary runtime upgrade script

To upgrade a canary script to a supported runtime version, use the following script.

```
const AWS = require('aws-sdk');

// You need to configure your AWS credentials and Region.
// https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-credentials-node.html
// https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
  /**
   * The number of canaries to upgrade during a single run of this script.
   */
  count: 10,
  /**
   * No canaries are upgraded unless force is specified.
   */
  force: false
};

/**
 * The number of milliseconds to sleep between GetCanary calls when
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
  try {
    const options = getOptions();

    const versions = await getRuntimeVersions();
    const canaries = await getAllCanaries();
    const upgrades = canaries
```

```
.filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
.map(canary => {
  return {
    Name: canary.Name,
    FromVersion: canary.RuntimeVersion,
    ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
  };
});

if (options.force) {
  const promises = [];

  for (const upgrade of upgrades.slice(0, options.count)) {
    const promise = upgradeCanary(upgrade);
    promises.push(promise);
    // Sleep for 100 milliseconds to avoid throttling.
    await usleep(100);
  }

  const succeeded = [];
  const failed = [];
  for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
    const upgrade = upgrades[i];
    const promise = promises[i];
    try {
      await promise;
      console.log(`The update of ${upgrade.Name} succeeded.`);
      succeeded.push(upgrade.Name);
    } catch (e) {
      console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
      failed.push({
        Name: upgrade.Name,
        Reason: e
      });
    }
  }

  if (succeeded.length) {
    console.group('The following canaries were upgraded successfully.');
```

```
    for (const name of succeeded) {
      console.log(name);
    }
    console.groupEnd()
  } else {
```

```
    console.log('No canaries were upgraded successfully.');
```

```
  }
```

```
  if (failed.length) {
```

```
    console.group('The following canaries were not upgraded successfully.');
```

```
    for (const failure of failed) {
```

```
      console.log('\x1b[31m', `${failure.Name}: ${failure.Reason}`, '\x1b[0m');
```

```
    }
```

```
    console.groupEnd();
```

```
  }
```

```
  } else {
```

```
    console.log('Run with --force [--count <count>] to perform the first <count>
```

```
upgrades shown. The default value of <count> is 10.')
```

```
    console.table(upgrades);
```

```
  }
```

```
  } catch (e) {
```

```
    console.error(e);
```

```
  }
```

```
  })());
```

```
function getOptions() {
```

```
  const force = getFlag('--force', DEFAULT_OPTIONS.force);
```

```
  const count = getOption('--count', DEFAULT_OPTIONS.count);
```

```
  return { force, count };
```

```
function getFlag(key, defaultValue) {
```

```
  return process.argv.includes(key) || defaultValue;
```

```
}
```

```
function getOption(key, defaultValue) {
```

```
  const index = process.argv.indexOf(key);
```

```
  if (index < 0) {
```

```
    return defaultValue;
```

```
  }
```

```
  const value = process.argv[index + 1];
```

```
  if (typeof value === 'undefined' || value.startsWith('-')) {
```

```
    throw `The ${key} option requires a value.`;
```

```
  }
```

```
  return value;
```

```
}
```

```
}
```

```
function getAllCanaries() {
```

```
  return new Promise((resolve, reject) => {
```

```
    const canaries = [];
```

```
synthetics.describeCanaries().eachPage((err, data) => {
  if (err) {
    reject(err);
  } else {
    if (data === null) {
      resolve(canaries);
    } else {
      canaries.push(...data.Canaries);
    }
  }
});
});
}

function getRuntimeVersions() {
  return new Promise((resolve, reject) => {
    const jsVersions = [];
    const pythonVersions = [];
    synthetics.describeRuntimeVersions().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          resolve({
            isLatestVersion(version) {
              const latest = this.getLatestVersion(version);
              return latest === version;
            },
            getLatestVersion(version) {
              if (jsVersions.some(v => v.VersionName === version)) {
                return jsVersions[jsVersions.length - 1].VersionName;
              } else if (pythonVersions.some(v => v.VersionName === version)) {
                return pythonVersions[pythonVersions.length - 1].VersionName;
              } else {
                throw Error(`Unknown version ${version}`);
              }
            }
          });
        } else {
          for (const version of data.RuntimeVersions) {
            if (version.VersionName === 'syn-1.0') {
```



```

        jsVersions.push(version);
    } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
        jsVersions.push(version);
    } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
        jsVersions.push(version);
    } else if (version.VersionName.startsWith('syn-python-selenium-')) {
        pythonVersions.push(version);
    } else {
        throw Error(`Unknown version ${version.VersionName}`);
    }
}
}
});
});
}

async function upgradeCanary(upgrade) {
    console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
    ${upgrade.ToVersion}`);
    await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
    upgrade.ToVersion }).promise();
    while (true) {
        await usleep(SLEEP_TIME);
        console.log(`Getting the state of canary ${upgrade.Name}`);
        const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
        const state = response.Canary.Status.State;
        console.log(`The state of canary ${upgrade.Name} is ${state}`);
        if (state === 'ERROR' || response.Canary.Status.StateReason) {
            throw response.Canary.Status.StateReason;
        }
        if (state !== 'UPDATING') {
            return;
        }
    }
}

function usleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

```

Runtime versions using Node.js and Puppeteer

The first runtime version for Node.js and Puppeteer was named `syn-1.0`. Later runtime versions have the naming convention `syn-language-majorversion.minorversion`. Starting with `syn-nodejs-puppeteer-3.0`, the naming convention is `syn-language-framework-majorversion.minorversion`

An additional `-beta` suffix shows that the runtime version is currently in a beta preview release.

Runtime versions with the same major version number are backward compatible.

Important

The following CloudWatch Synthetics runtime versions are scheduled to be deprecated on March 8, 2024.

- `syn-nodejs-puppeteer-6.1`
- `syn-nodejs-puppeteer-6.0`
- `syn-nodejs-puppeteer-5.1`
- `syn-nodejs-puppeteer-5.0`
- `syn-nodejs-puppeteer-4.0`

For more information, see [CloudWatch Synthetics runtime support policy](#).

Important

IMPORTANT: The included AWS SDK for JavaScript v2 dependency will be removed and updated to use AWS SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue referencing and using the included AWS SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

Notes for all runtime versions

When using `syn-nodejs-puppeteer-3.0` runtime version, make sure that your canary script is compatible with Node.js 12.x. If you use an earlier version of a `syn-nodejs` runtime version, make sure that that your script is compatible with Node.js 10.x.

The Lambda code in a canary is configured to have a maximum memory of 1 GB. Each run of a canary times out after a configured timeout value. If no timeout value is specified for a canary, CloudWatch chooses a timeout value based on the canary's frequency. If you configure a timeout value, make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Note

The following CloudWatch Synthetics runtime versions were deprecated on January 8, 2024. This is because AWS Lambda deprecated the Lambda Node.js 14 runtime on December 4, 2023.

- `syn-nodejs-puppeteer-3.9`
- `syn-nodejs-puppeteer-3.8`
- `syn-nodejs-puppeteer-3.7`
- `syn-nodejs-puppeteer-3.6`
- `syn-nodejs-puppeteer-3.5`

The following CloudWatch Synthetics runtime versions were deprecated on November 13, 2022. This is because AWS Lambda deprecated the Lambda Node.js 12 runtime on November 14, 2022.

- `syn-nodejs-puppeteer-3.4`
- `syn-nodejs-puppeteer-3.3`
- `syn-nodejs-puppeteer-3.2`
- `syn-nodejs-puppeteer-3.1`
- `syn-nodejs-puppeteer-3.0`

For more information, see [CloudWatch Synthetics runtime support policy](#).

syn-nodejs-puppeteer-8.0

The `syn-nodejs-puppeteer-8.0` runtime is the newest runtime version for Node.js and Puppeteer. It is not available in AWS GovCloud (US-East) or AWS GovCloud (US-West) because the Lambda runtime Node.js 20.x is not available in those Regions.

Important

Lambda Node.js 18 and later runtimes use AWS SDK for JavaScript V3. If you need to migrate a function from an earlier runtime, follow the [aws-sdk-js-v3 Migration Workshop](#) on GitHub. For more information about AWS SDK for JavaScript version 3, see [this blog post](#).

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.10.0
- Chromium version 125.0.6422.112

New features in syn-nodejs-puppeteer-8.0:

- **Support for two-factor authentication**
- **Bug fixes** for situations where some service clients were losing data in Node.js SDK V3 responses.

syn-nodejs-puppeteer-7.0

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 21.9.0
- Chromium version 121.0.6167.139

Code size:

The size of code and dependencies that you can package into this runtime is 80 MB.

New features in syn-nodejs-puppeteer-7.0:

- **Updated versions of the bundled libraries in Puppeteer and Chromium**— The Puppeteer and Chromium dependencies are updated to new versions.

Important

Moving from Puppeteer 19.7.0 to Puppeteer 21.9.0 introduces breaking changes regarding testing and filters. For more information, see the **BREAKING CHANGES** sections in [puppeteer: v20.0.0](#) and [puppeteer-core: v21.0.0](#).

Recommended upgrade to AWS SDK v3

The Lambda nodejs18.x runtime doesn't support AWS SDK v2. We strongly recommend that you migrate to AWS SDK v3.

syn-nodejs-puppeteer-6.2

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

New features in syn-nodejs-puppeteer-6.2:

- **Updated versions of the bundled libraries in Chromium**
- **Ephemeral storage monitoring**— This runtime adds ephemeral storage monitoring in customer accounts.
- **Bug fixes**

syn-nodejs-puppeteer-5.2

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0

- Chromium version 111.0.5563.146

New features in syn-nodejs-puppeteer-5.2:

- Updated versions of the bundled libraries in Chromium
- Bug fixes

syn-nodejs-puppeteer-6.1

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

New features in syn-nodejs-puppeteer-6.1:

- **Stability improvements**— Added auto-retry logic for handling intermittent Puppeteer launch errors.
- **Dependency upgrades**— Upgrades for some third-party dependency packages.
- **Canaries without Amazon S3 permissions**— Bug fixes, such that canaries that don't have any Amazon S3 permissions can still run. These canaries with no Amazon S3 permissions won't be able to upload screenshots or other artifacts to Amazon S3. For more information about permissions for canaries, see [Required roles and permissions for canaries](#).

Important

IMPORTANT: The included AWS SDK for JavaScript v2 dependency will be removed and updated to use AWS SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue

referencing and using the included AWS SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-6.0

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

New features in syn-nodejs-puppeteer-6.0:

- **Dependency upgrade**— The Node.js dependency is upgraded to 18.x.
- **Intercept mode support**— Puppeteer cooperative intercept mode support was added to the Synthetics canary runtime library.
- **Tracing behavior change**— Changed default tracing behavior to trace only fetch and xhr requests, and not trace resource requests. You can enable the tracing of resource requests by configuring the `traceResourceRequests` option.
- **Duration metric refined**— The `Duration` metric now excludes the operation time the canary uses to upload artifacts, take screenshots, and generate CloudWatch metrics. `Duration` metric values are reported to CloudWatch, and you can also see them in the Synthetics console.
- **Bug fix**— Clean up core dump generated when Chromium crashes during a canary run.

Important

IMPORTANT: The included AWS SDK for JavaScript v2 dependency will be removed and updated to use AWS SDK for JavaScript v3 in a future runtime release. When that

happens, you can update your canary code references. Alternatively, you can continue referencing and using the included AWS SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-5.1

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Bug fixes in syn-nodejs-puppeteer-5.1:

- **Bug fix**— This runtime fixes a bug in `syn-nodejs-puppeteer-5.0` where the HAR files created by the canaries were missing request headers.

syn-nodejs-puppeteer-5.0

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

New features in syn-nodejs-puppeteer-5.0:

- **Dependency upgrade**— The Puppeteer-core version is updated to 19.7.0. The Chromium version is upgraded to 111.0.5563.146.

Important

The new Puppeteer-core version is not completely backward-compatible with previous versions of Puppeteer. Some of the changes in this version can cause existing canaries that use deprecated Puppeteer functions to fail. For more information, see the breaking changes in the change logs for Puppeteer-core versions 19.7.0 through 6.0, in [Puppeteer change logs](#).

syn-nodejs-puppeteer-4.0

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-4.0:

- **Dependency upgrade**— The Node.js dependency is updated to 16.x.

Deprecated runtimes for Node.js and Puppeteer

The following runtimes for Node.js and Puppeteer have been deprecated.

syn-nodejs-puppeteer-3.9

Important

This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.9:

- **Dependency upgrades**— Upgrades some third-party dependency packages.

syn-nodejs-puppeteer-3.8

Important

This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.8:

- **Profile cleanup**— Chromium profiles are now cleaned up after each canary run.

Bug fixes in syn-nodejs-puppeteer-3.8:

- **Bug fixes**— Previously, visual monitoring canaries would sometimes stop working properly after a run with no screenshots. This is now fixed.

syn-nodejs-puppeteer-3.7

Important

This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.7:

- **Logging enhancement**— The canary will upload logs to Amazon S3 even if it times out or crashes.
- **Lambda layer size reduced**— The size of the Lambda layer used for canaries is reduced by 34%.

Bug fixes in syn-nodejs-puppeteer-3.7:

- **Bug fixes**— Japanese, Simplified Chinese, and Traditional Chinese fonts will render properly.

syn-nodejs-puppeteer-3.6

Important

This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.6:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-nodejs-puppeteer-3.5

Important

This runtime version was deprecated on January 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

New features in syn-nodejs-puppeteer-3.5:

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-nodejs-puppeteer-3.4

Important

This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.4:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Expanded HAR file information**— You can now see bad, pending, and incomplete requests in the HAR files produced by canaries.

syn-nodejs-puppeteer-3.3

Important

This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.3:

- **More options for artifact encryption**— For canaries using this runtime or later, instead of using an AWS managed key to encrypt artifacts that the canary stores in Amazon S3, you can choose to use an AWS KMS customer managed key or an Amazon S3-managed key. For more information, see [Encrypting canary artifacts](#).

syn-nodejs-puppeteer-3.2

Important

This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.2:

- **visual monitoring with screenshots**— Canaries using this runtime or later can compare a screenshot taken during a run with a baseline version of the same screenshot. If the screenshots are more different than a specified percentage threshold, the canary fails. For more information, see [Visual monitoring](#) or [Visual monitoring blueprint](#).
- **New functions regarding sensitive data** You can prevent sensitive data from appearing in canary logs and reports. For more information, see [SyntheticsLogHelper class](#).
- **Deprecated function** The `RequestResponseLogHelper` class is deprecated in favor of other new configuration options. For more information, see [RequestResponseLogHelper class](#).

syn-nodejs-puppeteer-3.1

Important

This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0

- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.1:

- **Ability to configure CloudWatch metrics**— With this runtime, you can disable the metrics that you do not require. Otherwise, canaries publish various CloudWatch metrics for each canary run.
- **Screenshot linking**— You can link a screenshot to a canary step after the step has completed. To do this, you take the screenshot by using the **takeScreenshot** method, using the name of the step that you want to associate the screenshot with. For example, you might want to perform a step, add a wait time, and then take the screenshot.
- **Heartbeat monitor blueprint can monitor multiple URLs**— You can use the heartbeat monitoring blueprint in the CloudWatch console to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

syn-nodejs-puppeteer-3.0

Important

This runtime version was deprecated on November 13, 2022. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

New features in syn-nodejs-puppeteer-3.0:

- **Upgraded dependencies**— This version uses Puppeteer version 5.5.0, Node.js 12.x, and Chromium 88.0.4298.0.
- **Cross-Region bucket access**— You can now specify an S3 bucket in another Region as the bucket where your canary stores its log files, screenshots, and HAR files.

- **New functions available**— This version adds library functions to retrieve the canary name and the Synthetics runtime version.

For more information, see [Synthetics class](#).

syn-nodejs-2.2

This section contains information about the syn-nodejs-2.2 runtime version.

Important

This runtime version was deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.2:

- **Monitor your canaries as HTTP steps**— You can now test multiple APIs in a single canary. Each API is tested as a separate HTTP step, and CloudWatch Synthetics monitors the status of each step using step metrics and the CloudWatch Synthetics step report. CloudWatch Synthetics creates SuccessPercent and Duration metrics for each HTTP step.

This functionality is implemented by the `executeHttpStep(stepName, requestOptions, callback, stepConfig)` function. For more information, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#).

The API canary blueprint is updated to use this new feature.

- **HTTP request reporting**— You can now view detailed HTTP requests reports which capture details such as request/response headers, response body, status code, error and performance timings, TCP connection time, TLS handshake time, first byte time, and content transfer time. All HTTP requests which use the HTTP/HTTPS module under the hood are captured here. Headers

and response body are not captured by default but can be enabled by setting configuration options.

- **Global and step-level configuration**— You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs to enable or disable certain options.

For more information, see [SyntheticsConfiguration class](#).

- **Continue on step failure configuration**— You can choose to continue canary execution when a step fails. For the `executeHttpStep` function, this is turned on by default. You can set this option once at global level or set it differently per-step.

syn-nodejs-2.1

Important

This runtime version was deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in syn-nodejs-2.1:

- **Configurable screenshot behavior**— Provides the ability to turn off the capturing of screenshots by UI canaries. In canaries that use previous versions of the runtimes, UI canaries always capture screenshots before and after each step. With `syn-nodejs-2.1`, this is configurable. Turning off screenshots can reduce your Amazon S3 storage costs, and can help you comply with HIPAA regulations. For more information, see [SyntheticsConfiguration class](#).
- **Customize the Google Chrome launch parameters** You can now configure the arguments used when a canary launches a Google Chrome browser window. For more information, see [launch\(options\)](#).

There can be a small increase in canary duration when using `syn-nodejs-2.0` or later, compared to earlier versions of the canary runtimes.

`syn-nodejs-2.0`

Important

This runtime version was deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in `syn-nodejs-2.0`:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Support for X-Ray active tracing.** When a canary has tracing enabled, X-Ray traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the X-Ray Trace Map, even when they don't send requests to other services or applications that have tracing enabled. For more information, see [Canaries and X-Ray tracing](#).
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. With this runtime version, these

metrics are reported only for UI canaries, and are not reported for API canaries. They are also reported for API canaries starting with runtime version `syn-nodejs-puppeteer-2.2`.

- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in `syn-nodejs-2.0`:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

There can be a small increase in canary duration when using `syn-nodejs-2.0` or later, compared to earlier versions of the canary runtimes.

`syn-nodejs-2.0-beta`

Important

This runtime version was deprecated on February 8, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

New features in `syn-nodejs-2.0-beta`:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0

- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. These metrics are reported only for UI canaries, and are not reported for API canaries.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in `syn-nodejs-2.0-beta`:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (`3xx`) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

`syn-1.0`

Important

This runtime version is scheduled to be deprecated on May 28, 2021. For more information, see [CloudWatch Synthetics runtime support policy](#).

The first Synthetics runtime version is `syn-1.0`.

Major dependencies:

- Lambda runtime Node.js 10.x

- Puppeteer-core version 1.14.0
- The Chromium version that matches Puppeteer-core 1.14.0

Runtime versions using Python and Selenium Webdriver

The following sections contain information about the CloudWatch Synthetics runtime versions for Python and Selenium Webdriver. Selenium is an open-source browser automation tool. For more information about Selenium, see www.selenium.dev/

The naming convention for these runtime versions is `syn-language-framework-majorversion.minorversion`.

Important

The following CloudWatch Synthetics runtime versions are scheduled to be deprecated on March 8, 2024.

- `syn-python-selenium-2.0`
- `syn-python-selenium-1.3`
- `syn-python-selenium-1.2`
- `syn-python-selenium-1.1`
- `syn-python-selenium-1.0`

For more information, see [CloudWatch Synthetics runtime support policy](#).

`syn-python-selenium-3.0`

Version 3.0 is the newest CloudWatch Synthetics runtime for Python and Selenium.

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 121.0.6167.139

New features in `syn-python-selenium-3.0`:

- **Updated versions of the bundled libraries in Chromium**— The Chromium dependency is updated to a new version.

syn-python-selenium-2.1

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 111.0.5563.146

New features in syn-python-selenium-2.1:

- **Updated versions of the bundled libraries in Chromium**— The Chromium and Selenium dependencies are updated to new versions.

syn-python-selenium-2.0

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Python 3.8
- Selenium 4.10.0
- Chromium version 111.0.5563.146

New features in syn-python-selenium-2.0:

- **Updated dependencies**— The Chromium and Selenium dependencies are updated to new versions.

Bug fixes in syn-python-selenium-2.0:

- **Timestamp added**— A timestamp has been added to canary logs.
- **Session re-use**— A bug was fixed so that canaries are now prevented from reusing the session from their previous canary run.

syn-python-selenium-1.3

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

New features in syn-python-selenium-1.3:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-python-selenium-1.2

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-python-selenium-1.1

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Configuration options for adding metrics and step failure configurations**— These options were already available in runtimes for Node.js canaries. For more information, see [SyntheticsConfiguration class](#).
- **Custom arguments in Chrome** — You can now open a browser in incognito mode or pass in proxy server configuration. For more information, see [Chrome\(\)](#).
- **Cross-Region artifact buckets**— A canary can store its artifacts in an Amazon S3 bucket in a different Region.
- **Bug fixes, including a fix for the `index.py` issue**— With previous runtimes, a canary file named `index.py` caused exceptions because it conflicted with the name of the library file. This issue is now fixed.

syn-python-selenium-1.0

Important

This runtime version is scheduled to be deprecated on March 8, 2024. For more information, see [CloudWatch Synthetics runtime support policy](#).

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Selenium support**— You can write canary scripts using the Selenium test framework. You can bring your Selenium scripts from elsewhere into CloudWatch Synthetics with minimal changes, and they will work with AWS services.

Writing a canary script

The following sections explain how to write a canary script and how to integrate a canary with other AWS services and with external dependencies and libraries.

Topics

- [Writing a Node.js canary script](#)
- [Writing a Python canary script](#)
- [Changing an existing Selenium script to use a Synthetics canary](#)
- [Changing an existing Puppeteer Synthetics script to authenticate non-standard certificates](#)

Writing a Node.js canary script

Topics

- [Creating a CloudWatch Synthetics canary from scratch](#)
- [Packaging your Node.js canary files](#)

- [Changing an existing Puppeteer script to use as a Synthetics canary](#)
- [Environment variables](#)
- [Integrating your canary with other AWS services](#)
- [Forcing your canary to use a static IP address](#)

Creating a CloudWatch Synthetics canary from scratch

Here is an example minimal Synthetics Canary script. This script passes as a successful run, and returns a string. To see what a failing canary looks like, change `let fail = false;` to `let fail = true;`.

You must define an entry point function for the canary script. To see how files are uploaded to the Amazon S3 location specified as the canary's `ArtifactS3Location`, create these files under the `/tmp` folder. After the script runs, the pass/fail status and the duration metrics are published to CloudWatch and the files under `/tmp` are uploaded to S3.

```
const basicCustomEntryPoint = async function () {

    // Insert your code here

    // Perform multi-step pass/fail check

    // Log decisions made and results to /tmp

    // Be sure to wait for all your code paths to complete
    // before returning control back to Synthetics.
    // In that way, your canary will not finish and report success
    // before your code has finished executing

    // Throw to fail, return to succeed
    let fail = false;
    if (fail) {
        throw "Failed basicCanary check.";
    }

    return "Successfully completed basicCanary checks.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

```
};
```

Next, we'll expand the script to use Synthetics logging and make a call using the AWS SDK. For demonstration purposes, this script will create an Amazon DynamoDB client and make a call to the DynamoDB listTables API. It logs the response to the request and logs either pass or fail depending on whether the request was successful.

```
const log = require('SyntheticsLogger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

  log.info("Starting DynamoDB:listTables canary.");

  let dynamodb = new AWS.DynamoDB();
  var params = {};
  let request = await dynamodb.listTables(params);
  try {
    let response = await request.promise();
    log.info("listTables response: " + JSON.stringify(response));
  } catch (err) {
    log.error("listTables error: " + JSON.stringify(err), err.stack);
    throw err;
  }

  return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
  return await basicCustomEntryPoint();
};
```

Packaging your Node.js canary files

If you are uploading your canary scripts using an Amazon S3 location, your zip file should include your script under this folder structure: `nodejs/node_modules/myCanaryFilename.js file`.

If you have more than a single `.js` file or you have a dependency that your script depends on, you can bundle them all into a single ZIP file that contains the folder structure

`nodejs/node_modules/myCanaryFilename.js` file and other folders and files. If you are using `syn-nodejs-puppeteer-3.4` or later, you can optionally put your canary files in another folder and creating your folder structure like this: `nodejs/node_modules/myFolder/myCanaryFilename.js` file and other folders and files.

Handler name

Be sure to set your canary's script entry point (handler) as `myCanaryFilename.functionName` to match the file name of your script's entry point. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, then `functionName` must be `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, you can choose any function name as the handler. If you are using `syn-nodejs-puppeteer-3.4` or later, you can also optionally store the canary in a separate folder such as `nodejs/node_modules/myFolder/my_canary_filename`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Puppeteer script to use as a Synthetics canary

This section explains how to take Puppeteer scripts and modify them to run as Synthetics canary scripts. For more information about Puppeteer, see [Puppeteer API v1.14.0](#).

We'll start with this example Puppeteer script:

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

The conversion steps are as follows:

- Create and export a `handler` function. The handler is the entry point function for the script. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, the handler function must be named `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using

`syn-nodejs-puppeteer-3.4` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
const basicPuppeteerExample = async function () {};  
  
exports.handler = async () => {  
  return await basicPuppeteerExample();  
};
```

- Use the `Synthetics` dependency.

```
var synthetics = require('Synthetics');
```

- Use the `Synthetics.getPage` function to get a Puppeteer Page object.

```
const page = await synthetics.getPage();
```

The page object returned by the `Synthetics.getPage` function has the `page.on` request, response and `requestfailed` events instrumented for logging. Synthetics also sets up HAR file generation for requests and responses on the page, and adds the canary ARN to the user-agent headers of outgoing requests on the page.

The script is now ready to be run as a Synthetics canary. Here is the updated script:

```
var synthetics = require('Synthetics'); // Synthetics dependency  
  
const basicPuppeteerExample = async function () {  
  const page = await synthetics.getPage(); // Get instrumented page from Synthetics  
  await page.goto('https://example.com');  
  await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp  
  folder  
};  
  
exports.handler = async () => { // Exported handler function  
  return await basicPuppeteerExample();  
};
```

Environment variables

You can use environment variables when creating canaries. This allows you to write a single canary script and then use that script with different values to quickly create multiple canaries that have a similar task.

For example, suppose your organization has endpoints such as `prod`, `dev`, and `pre-release` for the different stages of your software development, and you need to create canaries to test each of these endpoints. You can write a single canary script that tests your software and then specify different values for the endpoint environment variable when you create each of the three canaries. Then, when you create a canary, you specify the script and the values to use for the environment variables.

The names of environment variables can contain letters, numbers, and the underscore character. They must start with a letter and be at least two characters. The total size of your environment variables can't exceed 4 KB. You can't specify any Lambda reserved environment variables as the names of your environment variables. For more information about reserved environment variables, see [Runtime environment variables](#).

Important

The environment variables keys and values are not encrypted. Do not store sensitive information in them.

The following example script uses two environment variables. This script is for a canary that checks whether a webpage is available. It uses environment variables to parameterize both the URL that it checks and the CloudWatch Synthetics log level that it uses.

The following function sets `LogLevel` to the value of the `LOG_LEVEL` environment variable.

```
synthetics.setLogLevel(process.env.LOG_LEVEL);
```

This function sets `URL` to the value of the `URL` environment variable.

```
const URL = process.env.URL;
```

This is the complete script. When you create a canary using this script, you specify values for the `LOG_LEVEL` and `URL` environment variables.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadEnvironmentVariable = async function () {

  // Setting the log level (0-3)
  synthetics.setLogLevel(process.env.LOG_LEVEL);
  // INSERT URL here
  const URL = process.env.URL;

  let page = await synthetics.getPage();
  //You can customize the wait condition here. For instance,
  //using 'networkidle2' may be less restrictive.
  const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
30000});
  if (!response) {
    throw "Failed to load page!";
  }
  //Wait for page to render.
  //Increase or decrease wait time based on endpoint being monitored.
  await page.waitFor(15000);
  await synthetics.takeScreenshot('loaded', 'loaded');
  let pageTitle = await page.title();
  log.info('Page title: ' + pageTitle);
  log.debug('Environment variable:' + process.env.URL);

  //If the response status code is not a 2xx success code
  if (response.status() < 200 || response.status() > 299) {
    throw "Failed to load page!";
  }
};

exports.handler = async () => {
  return await pageLoadEnvironmentVariable();
};
```

Passing environment variables to your script

To pass environment variables to your script when you create a canary in the console, specify the keys and values of the environment variables in the **Environment variables** section on the console. For more information, see [Creating a canary](#).

To pass environment variables through the API or AWS CLI, use the `EnvironmentVariables` parameter in the `RunConfig` section. The following is an example AWS CLI command that creates a canary that uses two environment variables with keys of `Environment` and `Region`.

```
aws synthetics create-canary --cli-input-json '{
  "Name": "nameofCanary",
  "ExecutionRoleArn": "roleArn",
  "ArtifactS3Location": "s3://cw-syn-results-123456789012-us-west-2",
  "Schedule": {
    "Expression": "rate(0 minute)",
    "DurationInSeconds": 604800
  },
  "Code": {
    "S3Bucket": "canarycreation",
    "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
    "Handler": "pageLoadBlueprint.handler"
  },
  "RunConfig": {
    "TimeoutInSeconds": 60,
    "EnvironmentVariables": {
      "Environment": "Production",
      "Region": "us-west-1"
    }
  },
  "SuccessRetentionPeriodInDays": 13,
  "FailureRetentionPeriodInDays": 13,
  "RuntimeVersion": "syn-nodejs-2.0"
}'
```

Integrating your canary with other AWS services

All canaries can use the AWS SDK library. You can use this library when you write your canary to integrate the canary with other AWS services.

To do so, you need to add the following code to your canary. For these examples, AWS Secrets Manager is used as the service that the canary is integrating with.

- Import the AWS SDK.

```
const AWS = require('aws-sdk');
```


- Create a client for the AWS service that you are integrating with.

```
const secretsManager = new AWS.SecretsManager();
```

- Use the client to make API calls to that service.

```
var params = {  
  SecretId: secretName  
};  
return await secretsManager.getSecretValue(params).promise();
```

The following canary script code snippet demonstrates an example of integration with Secrets Manager in more detail.

```
var synthetics = require('Synthetics');  
const log = require('SyntheticsLogger');  
  
const AWS = require('aws-sdk');  
const secretsManager = new AWS.SecretsManager();  
  
const getSecrets = async (secretName) => {  
  var params = {  
    SecretId: secretName  
  };  
  return await secretsManager.getSecretValue(params).promise();  
}  
  
const secretsExample = async function () {  
  let URL = "<URL>";  
  let page = await synthetics.getPage();  
  
  log.info(`Navigating to URL: ${URL}`);  
  const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:  
30000});  
  
  // Fetch secrets  
  let secrets = await getSecrets("secretname")  
  
  /**  
   * Use secrets to login.  
   *  
   * Assuming secrets are stored in a JSON format like:
```

```
* {
*   "username": "<USERNAME>",
*   "password": "<PASSWORD>"
* }
**/
let secretsObj = JSON.parse(secrets.SecretString);
await synthetics.executeStep('login', async function () {
  await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
  await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

  await Promise.all([
    page.waitForNavigation({ timeout: 30000 }),
    await page.click(">SUBMIT-BUTTON-SELECTOR<")
  ]);
});

// Verify login was successful
await synthetics.executeStep('verify', async function () {
  await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
});
};

exports.handler = async () => {
  return await secretsExample();
};
```

Forcing your canary to use a static IP address

You can set up a canary so that it uses a static IP address.

To force a canary to use a static IP address

1. Create a new VPC. For more information, see [Using DNS with Your VPC](#).
2. Create a new internet gateway. For more information, see [Adding an internet gateway to your VPC](#).
3. Create a public subnet inside your new VPC.
4. Add a new route table to the VPC.
5. Add a route in the new route table, that goes from `0.0.0.0/0` to the internet gateway.
6. Associate the new route table with the public subnet.
7. Create an elastic IP address. For more information, see [Elastic IP addresses](#).

8. Create a new NAT gateway and assign it to the public subnet and the elastic IP address.
9. Create a private subnet inside the VPC.
10. Add a route to the VPC default route table, that goes from `0.0.0.0/0` to the NAT gateway
11. Create your canary.

Writing a Python canary script

This script passes as a successful run, and returns a string. To see what a failing canary looks like, change `fail = False` to `fail = True`

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

Packaging your Python canary files

If you have more than one .py file or your script has a dependency, you can bundle them all into a single ZIP file. If you use the `syn-python-selenium-1.1` runtime, the ZIP file must contain your main canary .py file within a python folder, such as `python/my_canary_filename.py`. If you use `syn-python-selenium-1.1` or later, you can optionally use a different folder, such as `python/myFolder/my_canary_filename.py`.

This ZIP file should contain all necessary folders and files, but the other files do not need to be in the python folder.

Be sure to set your canary's script entry point as `my_canary_filename.functionName` to match the file name and function name of your script's entry point. If you are using the `syn-python-selenium-1.0` runtime, then `functionName` must be `handler`. If you are using `syn-python-selenium-1.1` or later, this handler name restriction doesn't apply, and you can also optionally

store the canary in a separate folder such as `python/myFolder/my_canary_filename.py`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Selenium script to use a Synthetics canary

You can quickly modify an existing script for Python and Selenium to be used as a canary. For more information about Selenium, see www.selenium.dev/.

For this example, we'll start with the following Selenium script:

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

The conversion steps are as follows.

To convert a Selenium script to be used as a canary

1. Change the import statement to use Selenium from the `aws_synthetics` module:

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
```

The Selenium module from `aws_synthetics` ensures that the canary can emit metrics and logs, generate a HAR file, and work with other CloudWatch Synthetics features.

2. Create a handler function and call your Selenium method. The handler is the entry point function for the script.

If you are using `syn-python-selenium-1.0`, the handler function must be named `handler`. If you are using `syn-python-selenium-1.1` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-python-selenium-1.1` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

```
def handler(event, context):
```

```
basic_selenium_script()
```

The script is now updated to be a CloudWatch Synthetics canary. Here is the updated script:

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

Changing an existing Puppeteer Synthetics script to authenticate non-standard certificates

One important use case for Synthetics canaries is for you to monitor your own endpoints. If you want to monitor an endpoint that isn't ready for external traffic, this monitoring can sometimes mean that you don't have a proper certificate signed by a trusted third-party certificate authority.

Two possible solutions to this scenario are as follows:

- To authenticate a client certificate, see [How to validate authentication using Amazon CloudWatch Synthetics – Part 2](#).
- To authenticate a self-signed certificate, see [How to validate authentication with self-signed certificates in Amazon CloudWatch Synthetics](#)

You are not limited to these two options when you use CloudWatch Synthetics canaries. You can extend these features and add your business logic by extending the canary code.

Note

Synthetics canaries running on Python runtimes innately have the `--ignore-certificate-errors` flag enabled, so those canaries shouldn't have any issues reaching sites with non-standard certificate configurations.

Library functions available for canary scripts

CloudWatch Synthetics includes several built-in classes and functions that you can call when writing Node.js scripts to use as canaries.

Some apply to both UI and API canaries. Others apply to UI canaries only. A UI canary is a canary that uses the `getPage()` function and uses Puppeteer as a web driver to navigate and interact with webpages.

Note

Whenever you upgrade a canary to use a new version of the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically upgraded to the same version of NodeJS that the Synthetics runtime supports.

Topics

- [Library functions available for Node.js canary scripts](#)
- [Library functions available for Python canary scripts using Selenium](#)

Library functions available for Node.js canary scripts

This section lists the library functions available for Node.js canary scripts.

Topics

- [Node.js library classes and functions that apply to all canaries](#)
- [Node.js library classes and functions that apply to UI canaries only](#)
- [Node.js library classes and functions that apply to API canaries only](#)

Node.js library classes and functions that apply to all canaries

The following CloudWatch Synthetics library functions for Node.js are useful for all canaries.

Topics

- [Synthetics class](#)
- [SyntheticsConfiguration class](#)
- [Synthetics logger](#)

- [SyntheticsLogHelper class](#)

Synthetics class

The following functions for all canaries are in the Synthetics class.

addExecutionError(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `addExecutionError` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `addExecutionError` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try {
    await synthetics.takeScreenshot(stepName, "loaded");
} catch(ex) {
    synthetics.addExecutionError('Unable to take screenshot ', ex);
}
```

getCanaryName();

Returns the name of the canary.

getCanaryArn();

Returns the ARN of the canary.

getCanaryUserAgentString();

Returns the custom user agent of the canary.

getRuntimeVersion();

This function is available in runtime version `syn-nodejs-puppeteer-3.0` and later. It returns the Synthetics runtime version of the canary. For example, the return value could be `syn-nodejs-puppeteer-3.0`.

getLogLevel();

Retrieves the current log level for the Synthetics library. Possible values are the following:

- 0 – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

Sets the log level for the Synthetics library. Possible values are the following:

- 0 – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
synthetics.setLogLevel(0);
```

SyntheticsConfiguration class

This class is available only in the `syn-nodejs-2.1` runtime version or later.

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs.

You can pass in options at the step level. For examples, see [`async executeStep\(stepName, functionToExecute, \[stepConfig\]\)`](#); and [`executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)`](#)

Function definitions:

setConfig(options)

options is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *options*.

setConfig(options) for all canaries

For canaries using `syn-nodejs-puppeteer-3.2` or later, the **(options)** for **setConfig** can include the following parameters:

- `includeRequestHeaders` (boolean)— Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` (boolean)— Whether to include response headers in the report. The default is `false`.
- `restrictedHeaders` (array)— A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing `includeRequestHeaders` as `true` and `restrictedHeaders` as `['Authorization']`.
- `includeRequestBody` (boolean)— Whether to include the request body in the report. The default is `false`.
- `includeResponseBody` (boolean)— Whether to include the response body in the report. The default is `false`.

setConfig(options) regarding CloudWatch metrics

For canaries using `syn-nodejs-puppeteer-3.1` or later, the **(options)** for **setConfig** can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see [CloudWatch metrics published by canaries](#).

- `failedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_2xxMetric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_4xxMetric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_5xxMetric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `stepDurationMetric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `stepSuccessMetric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated2xxMetric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

- `visualMonitoringSuccessPercentMetric` (boolean)— Whether to emit the `visualMonitoringSuccessPercent` metric for this canary. The default is `true`.
- `visualMonitoringTotalComparisonsMetric` (boolean)— Whether to emit the `visualMonitoringTotalComparisons` metric for this canary. The default is `false`.
- `stepsReport` (boolean)— Whether to report a step execution summary. The default is `true`.
- `includeUrlPassword` (boolean)— Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent disclosing sensitive data. The default is `false`.
- `restrictedUrlParameters` (array)— A list of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `logRequest` (boolean)— Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `true`.
- `logResponse` (boolean)— Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `true`.
- `logRequestBody` (boolean)— Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.
- `logResponseBody` (boolean)— Whether to log response bodies along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.
- `logRequestHeaders` (boolean)— Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.

Note that `includeRequestHeaders` enables headers in artifacts.

- `logResponseHeaders` (boolean)— Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

Note that `includeResponseHeaders` enables headers in artifacts.

Note

The `Duration` and `SuccessPercent` metrics are always emitted for each canary, both with and without the `CanaryName` metric.

Methods to enable or disable metrics

disableAggregatedRequestMetrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disableRequestMetrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disableStepMetrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enableAggregatedRequestMetrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enableRequestMetrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enableStepMetrics()

Enables all step metrics, including both step success metrics and step duration metrics.

get2xxMetric()

Returns whether the canary emits a 2xx metric with the `CanaryName` dimension.

get4xxMetric()

Returns whether the canary emits a 4xx metric with the `CanaryName` dimension.

get5xxMetric()

Returns whether the canary emits a 5xx metric with the `CanaryName` dimension.

getAggregated2xxMetric()

Returns whether the canary emits a 2xx metric with no dimension.

getAggregated4xxMetric()

Returns whether the canary emits a 4xx metric with no dimension.

getAggregatedFailedCanaryMetric()

Returns whether the canary emits a Failed metric with no dimension.

getAggregatedFailedRequestsMetric()

Returns whether the canary emits a Failed requests metric with no dimension.

getAggregated5xxMetric()

Returns whether the canary emits a 5xx metric with no dimension.

getFailedCanaryMetric()

Returns whether the canary emits a Failed metric with the CanaryName dimension.

getFailedRequestsMetric()

Returns whether the canary emits a Failed requests metric with the CanaryName dimension.

getStepDurationMetric()

Returns whether the canary emits a Duration metric with the CanaryName dimension for this canary.

getStepSuccessMetric()

Returns whether the canary emits a StepSuccess metric with the CanaryName dimension for this canary.

with2xxMetric(_2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 2xx metric with the CanaryName dimension for this canary.

with4xxMetric(_4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 4xx metric with the CanaryName dimension for this canary.

with5xxMetric(_5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 5xx metric with the CanaryName dimension for this canary.

withAggregated2xxMetric(agggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 2xx metric with no dimension for this canary.

withAggregated4xxMetric(agggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 4xx metric with no dimension for this canary.

withAggregated5xxMetric(agggregated5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 5xx metric with no dimension for this canary.

withAggregatedFailedCanaryMetric(agggregatedFailedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a Failed metric with no dimension for this canary.

withAggregatedFailedRequestsMetric(agggregatedFailedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a Failed requests metric with no dimension for this canary.

withFailedCanaryMetric(failedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a Failed metric with the CanaryName dimension for this canary.

withFailedRequestsMetric(failedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a Failed requests metric with the CanaryName dimension for this canary.

withStepDurationMetric(stepDurationMetric)

Accepts a Boolean argument, which specifies whether to emit a Duration metric with the CanaryName dimension for this canary.

withStepSuccessMetric(stepSuccessMetric)

Accepts a Boolean argument, which specifies whether to emit a StepSuccess metric with the CanaryName dimension for this canary.

Methods to enable or disable other features

withHarFile()

Accepts a Boolean argument, which specifies whether to create a HAR file for this canary.

withStepsReport()

Accepts a Boolean argument, which specifies whether to report a step execution summary for this canary.

withIncludeUrlPassword()

Accepts a Boolean argument, which specifies whether to include passwords that appear in URLs in logs and reports.

withRestrictedUrlParameters()

Accepts an array of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. You can pass an asterisk (*) as a value to redact all URL path and query parameter values

withLogRequest()

Accepts a Boolean argument, which specifies whether to log every request in the canary's logs.

withLogResponse()

Accepts a Boolean argument, which specifies whether to log every response in the canary's logs.

withLogRequestBody()

Accepts a Boolean argument, which specifies whether to log every request body in the canary's logs.

withLogResponseBody()

Accepts a Boolean argument, which specifies whether to log every response body in the canary's logs.

withLogRequestHeaders()

Accepts a Boolean argument, which specifies whether to log every request header in the canary's logs.

withLogResponseHeaders()

Accepts a Boolean argument, which specifies whether to log every response header in the canary's logs.

getHarFile()

Returns whether the canary creates a HAR file.

getStepsReport()

Returns whether the canary reports a step execution summary.

getIncludeUrlPassword()

Returns whether the canary includes passwords that appear in URLs in logs and reports.

getRestrictedUrlParameters()

Returns whether the canary redacts URL path or query parameters.

getLogRequest()

Returns whether the canary logs every request in the canary's logs.

getLogResponse()

Returns whether the canary logs every response in the canary's logs.

getLogRequestBody()

Returns whether the canary logs every request body in the canary's logs.

getLogResponseBody()

Returns whether the canary logs every response body in the canary's logs.

getLogRequestHeaders()

Returns whether the canary logs every request header in the canary's logs.

getLogResponseHeaders()

Returns whether the canary logs every response header in the canary's logs.

Functions for all canaries

- `withIncludeRequestHeaders(includeRequestHeaders)`
- `withIncludeResponseHeaders(includeResponseHeaders)`
- `withRestrictedHeaders(restrictedHeaders)`
- `withIncludeRequestBody(includeRequestBody)`
- `withIncludeResponseBody(includeResponseBody)`
- `enableReportingOptions()`— Enables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .
- `disableReportingOptions()`— Disables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .

setConfig(options) for UI canaries

For UI canaries, **setConfig** can include the following Boolean parameters:

- `continueOnStepFailure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the **executeStep** function). If any steps fail, the canary run will still be marked as failed. The default is `false`.
- `harFile` (boolean)— Whether to create a HAR file. The default is `True`.
- `screenshotOnStepStart` (boolean)— Whether to take a screenshot before starting a step.
- `screenshotOnStepSuccess` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshotOnStepFailure` (boolean)— Whether to take a screenshot after a step fails.

Methods to enable or disable screenshots

disableStepScreenshots()

Disables all screenshot options (`screenshotOnStepStart`, `screenshotOnStepSuccess`, and `screenshotOnStepFailure`).

enableStepScreenshots()

Enables all screenshot options (`screenshotOnStepStart`, `screenshotOnStepSuccess`, and `screenshotOnStepFailure`). By default, all these methods are enabled.

getScreenshotOnStepFailure()

Returns whether the canary takes a screenshot after a step fails.

getScreenshotOnStepStart()

Returns whether the canary takes a screenshot before starting a step.

getScreenshotOnStepSuccess()

Returns whether the canary takes a screenshot after completing a step successfully.

withScreenshotOnStepStart(screenshotOnStepStart)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

withScreenshotOnStepFailure(screenshotOnStepFailure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

Then, set the configuration for each option by calling the `setConfig` method using one of the following options.

```
// Set configuration values
synConfig.setConfig({
  screenshotOnStepStart: true,
  screenshotOnStepSuccess: false,
  screenshotOnStepFailure: false
});
```

Or

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnSt
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synConfig.disableStepScreenshots();
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

setConfig(options) for API canaries

For API canaries, **setConfig** can include the following Boolean parameters:

- `continueOnHttpStepFailure` (boolean)— Whether to continue with running the canary script after an HTTP step fails (this refers to the **executeHttpStep** function). If any steps fail, the canary run will still be marked as failed. The default is `true`.

Visual monitoring

Visual monitoring compares screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails and you can see the areas with differences highlighted in color in the canary run report. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium.

To enable visual monitoring, add the following line of code to the canary script. For more details, see [SyntheticsConfiguration class](#).

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.

- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see [Edit or delete a canary](#).

Other options for visual monitoring

`syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)`

Set the acceptable percentage for screenshot variance in visual comparisons.

`syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafa00")`

Set the highlight color that designates variance areas when you look at canary run reports that use visual monitoring.

`syntheticsConfiguration.withFailCanaryRunOnVisualVariance(failCanary)`

Set whether or not the canary fails when there is a visual difference that is more than the threshold. The default is to fail the canary.

Synthetics logger

`SyntheticsLogger` writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

You can use the `SyntheticsLogger`, assuming you want to run the Synthetics Library at the same log level as your Synthetics canary logging.

Using the `SyntheticsLogger` is not required to create a log file that is uploaded to your S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in S3 as artifacts.

To use the Synthetics Library logger:

```
const log = require('SyntheticsLogger');
```

Useful function definitions:

log.debug(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.debug("Starting step - login.");
```

log.error(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
try {
  await login();
} catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.info("Successfully completed step - login.");
```

log.log(*message*, *ex*);

This is an alias for `log.info`.

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.log("Successfully completed step - login.");
```

log.warn(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper class

The `SyntheticsLogHelper` class is available in the runtime `syn-nodejs-puppeteer-3.2` and later runtimes. It is already initialized in the CloudWatch Synthetics library and is configured with Synthetics configuration. You can add this as a dependency in your script. This class enables you to sanitize URLs, headers, and error messages to redact sensitive information.

Note

Synthetics sanitizes all URLs and error messages it logs before including them in logs, reports, HAR files, and canary run errors based on the Synthetics configuration setting `restrictedUrlParameters`. You have to use `getSanitizedUrl` or `getSanitizedErrorMessage` only if you are logging URLs or errors in your script. Synthetics does not store any canary artifacts except for canary errors thrown by the script. Canary run artifacts are stored in your customer account. For more information, see [Security considerations for Synthetics canaries](#).

`getSanitizedUrl(url, stepConfig = null)`

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized url strings based on the configuration. You can choose to redact sensitive URL parameters such as `password` and `access_token` by setting the property `restrictedUrlParameters`. By default, passwords in URLs are redacted. You can enable URL passwords if needed by setting `includeUrlPassword` to `true`.

This function throws an error if the URL passed in is not a valid URL.

Parameters

- *url* is a string and is the URL to sanitize.
- *stepConfig* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following sample URL: `https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`. In this example, `access_token` contains your sensitive information which shouldn't be logged. Note that the Synthetics services doesn't store any canary run artifacts. Artifacts such as logs, screenshots, and reports are all stored in an Amazon S3 bucket in your customer account.

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
  restrictedUrlParameters: ['access_token'];
});
```

Next, sanitize and log the URL

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200');
```

This logs the following in your canary log.

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

You can override the Synthetics configuration for a URL by passing in an optional parameter containing Synthetics configuration options, as in the following example.

```
const urlConfig = {
```

```
restrictedUrlParameters = ['*']
};
const sanitizedUrl = synthetics.getSanitizedUrl('https://example.com/learn/home?
access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

The preceding example redacts all query parameters, and is logged as follows:

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized error strings by sanitizing any URLs present based on the Synthetics configuration. You can choose to override the global Synthetics configuration when you call this function by passing an optional `stepConfig` parameter.

Parameters

- *error* is the error to sanitize. It can be an Error object or a string.
- *stepConfig* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following error: Failed to load url: `https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('Synthetics');

// Import Synthetics logger for logging url
const log = require('SyntheticsLogger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
```



```
synConfig.setConfig({
  restrictedUrlParameters: ['access_token'];
});
```

Next, sanitize and log the error message

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('SyntheticsLogHelper');

try {
  // Your code which can throw an error containing url which your script logs
} catch (error) {
  const sanitizedErrorMessage = synthetics.getSanitizedErrorMessage(errorMessage);
  logger.info(sanitizedErrorMessage);
}
```

This logs the following in your canary log.

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

getSanitizedHeaders(headers, stepConfig=null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized headers based on the `restrictedHeaders` property of `syntheticsConfiguration`. The headers specified in the `restrictedHeaders` property are redacted from logs, HAR files, and reports.

Parameters

- *headers* is an object containing the headers to sanitize.
- *stepConfig* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the headers.

Node.js library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for UI canaries.

Topics

- [Synthetics class](#)
- [BrokenLinkCheckerReport class](#)

- [SyntheticsLink class](#)

Synthetics class

The following functions are in the Synthetics class.

async addUserAgent(page, userAgentString);

This function appends *userAgentString* to the specified page's user-agent header.

Example:

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

Results in the page's user-agent header being set to *browsers-user-agent-header-valueMyApp-1.0*

async executeStep(stepName, functionToExecute, [stepConfig]);

Executes the provided step, wrapping it with start/pass/fail logging, start/pass/fail screenshots, and pass/fail and duration metrics.

Note

If you are using the `syn-nodejs-2.1` or later runtime, you can configure whether and when screenshots are taken. For more information, see [SyntheticsConfiguration class](#).

The `executeStep` function also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.

- Emits the `stepName SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

If the canary uses the `syn-nodejs-2.0` runtime or later, this function also adds a step execution summary to the canary's report. The summary includes details about each step, such as start time, end time, status (PASSED/FAILED), failure reason (if failed), and screenshots captured during the execution of each step.

Example:

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000)
{
    await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
    timeoutInMillis});});
```

Response:

Returns what `functionToExecute` returns.

Updates with `syn-nodejs-2.2`

Starting with `syn-nodejs-2.2`, you can optionally pass step configurations to override CloudWatch Synthetics configurations at the step level. For a list of options that you can pass to `executeStep`, see [SyntheticsConfiguration class](#).

The following example overrides the default `false` configuration for `continueOnStepFailure` to `true` and specifies when to take screenshots.

```
var stepConfig = {
    'continueOnStepFailure': true,
    'screenshotOnStepStart': false,
    'screenshotOnStepSuccess': true,
    'screenshotOnStepFailure': false
}

await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {
    await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
    timeoutInMillis});
```

```
}, stepConfig);
```

getDefaultLaunchOptions();

The `getDefaultLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics. For more information, see [Launch options type](#)

```
// This function returns default launch options used by Synthetics.  
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage();

Returns the current open page as a Puppeteer object. For more information, see [Puppeteer API v1.14.0](#).

Example:

```
let page = synthetics.getPage();
```

Response:

The page (Puppeteer object) that is currently open in the current browser session.

getRequestResponseLogHelper();

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class](#) instead.

Use this function as a builder pattern for tweaking the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(false));
```

Response:

```
{RequestResponseLogHelper}
```

launch(options)

The options for this function are available only in the `syn-nodejs-2.1` runtime version or later.

This function is used only for UI canaries. It closes the existing browser and launches a new one.

Note

CloudWatch Synthetics always launches a browser before starting to run your script. You don't need to call `launch()` unless you want to launch a new browser with custom options.

(options) is a configurable set of options to set on the browser. For more information, see [Launch options type](#).

If you call this function with no options, Synthetics launches a browser with default arguments, `executablePath`, and `defaultViewport`. The default viewport in CloudWatch Synthetics is 1920 by 1080.

You can override launch parameters used by CloudWatch Synthetics and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600.

```
await synthetics.launch({
  defaultViewport: {
    "deviceScaleFactor": 1,
    "width": 800,
    "height": 600
  }
});
```

The following sample code adds a new `ignoreHTTPSErrors` parameter to the CloudWatch Synthetics launch parameters:

```
await synthetics.launch({
  ignoreHTTPSErrors: true
});
```

You can disable web security by adding a `--disable-web-security` flag to args in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
await synthetics.launch({
  args: launchArgs
});
```

RequestResponseLogHelper class

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this class is deprecated. Any use of this class causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class](#) instead.

Handles the fine-grained configuration and creation of string representations of request and response payloads.

```
class RequestResponseLogHelper {

  constructor () {
    this.request = {url: true, resourceType: false, method: false, headers: false,
postData: false};
    this.response = {status: true, statusText: true, url: true, remoteAddress:
false, headers: false};
  }

  withLogRequestUrl(logRequestUrl);

  withLogRequestResourceType(logRequestResourceType);

  withLogRequestMethod(logRequestMethod);

  withLogRequestHeaders(logRequestHeaders);

  withLogRequestPostData(logRequestPostData);
```

```
withLogResponseStatus(logResponseStatus);

withLogResponseStatusText(logResponseStatusText);

withLogResponseUrl(logResponseUrl);

withLogResponseRemoteAddress(logResponseRemoteAddress);

withLogResponseHeaders(logResponseHeaders);
```

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()
  .withLogRequestPostData(true)
  .withLogRequestHeaders(true)
  .withLogResponseHeaders(true));
```

Response:

```
{RequestResponseLogHelper}
```

setRequestResponseLogHelper();

Important

In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use [RequestResponseLogHelper class](#) instead.

Use this function as a builder pattern for setting the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(true);
```

Response:

```
{RequestResponseLogHelper}
```

async takeScreenshot(name, suffix);

Takes a screenshot (.PNG) of the current page with name and suffix (optional).

Example:

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

This example captures and uploads a screenshot named `01-navigateToUrl-loaded.png` to the canary's S3 bucket.

You can take a screenshot for a particular canary step by passing the `stepName` as the first parameter. Screenshots are linked to the canary step in your reports, to help you track each step while debugging.

CloudWatch Synthetics canaries automatically take screenshots before starting a step (the `executeStep` function) and after the step completion (unless you configure the canary to disable screenshots). You can take more screenshots by passing in the step name in the `takeScreenshot` function.

The following example takes screenshot with the `signupForm` as the value of the `stepName`. The screenshot will be named `02-signupForm-address` and will be linked to the step named `signupForm` in the canary report.

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport class

This class provides methods to add a synthetics link. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `BrokenLinkCheckerReport`, include the following lines in the script:

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');  
  
const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

Useful function definitions:

addLink(*syntheticsLink*, isBroken)

syntheticsLink is a `SyntheticsLink` object representing a link. This function adds the link according to the status code. By default, it considers a link to be broken if the status code is not available or the status code is 400 or higher. You can override this default behavior by passing in the optional parameter `isBrokenLink` with a value of `true` or `false`.

This function does not have a return value.

getLinks()

This function returns an array of `SyntheticsLink` objects that are included in the broken link checker report.

getTotalBrokenLinks()

This function returns a number representing the total number of broken links.

getTotalLinksChecked()

This function returns a number representing the total number of links included in the report.

How to use BrokenLinkCheckerReport

The following canary script code snippet demonstrates an example of navigating to a link and adding it to the broken link checker report.

1. Import `SyntheticsLink`, `BrokenLinkCheckerReport`, and `Synthetics`.

```
const BrokenLinkCheckerReport = require('BrokenLinkCheckerReport');
const SyntheticsLink = require('SyntheticsLink');

// Synthetics dependency
const synthetics = require('Synthetics');
```

2. To add a link to the report, create an instance of `BrokenLinkCheckerReport`.

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. Navigate to the URL and add it to the broken link checker report.

```
let url = "https://amazon.com";
```

```
let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();

// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
  const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout:
    30000});
} catch (ex) {
  // Add failure reason if navigation fails.
  link.withFailureReason(ex);
}

if (response) {
  // Capture screenshot of destination page
  let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

  // Add screenshot result to synthetics link
  link.addScreenshotResult(screenshotResult);

  // Add status code and status description to the link
  link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. Add the report to Synthetics. This creates a JSON file named `BrokenLinkCheckerReport.json` in your S3 bucket for each canary run. You can see a links report in the console for each canary run along with screenshots, logs, and HAR files.

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink class

This class provides methods to wrap information. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `SyntheticsLink`, include the following lines in the script:

```
const SyntheticsLink = require('SyntheticsLink');  
  
const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

This function returns `syntheticsLinkObject`

Useful function definitions:

withUrl(*url*)

url is a URL string. This function returns `syntheticsLinkObject`

withText(*text*)

text is a string representing anchor text. This function returns `syntheticsLinkObject`. It adds anchor text corresponding to the link.

withParentUrl(*parentUrl*)

parentUrl is a string representing the parent (source page) URL. This function returns `syntheticsLinkObject`

withStatusCode(*statusCode*)

statusCode is a string representing the status code. This function returns `syntheticsLinkObject`

withFailureReason(*failureReason*)

failureReason is a string representing the failure reason. This function returns `syntheticsLinkObject`

addScreenshotResult(*screenshotResult*)

screenshotResult is an object. It is an instance of `ScreenshotResult` that was returned by the Synthetics function `takeScreenshot`. The object includes the following:

- `fileName`— A string representing the `screenshotFileName`
- `pageUrl` (optional)
- `error` (optional)

Node.js library classes and functions that apply to API canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for API canaries.

Topics

- [executeHttpRequest\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#)

executeHttpRequest(stepName, requestOptions, [callback], [stepConfig])

Executes the provided HTTP request as a step, and publishes `SuccessPercent` (pass/fail) and `Duration` metrics.

executeHttpRequest uses either HTTP or HTTPS native functions under the hood, depending upon the protocol specified in the request.

This function also adds a step execution summary to the canary's report. The summary includes details about each HTTP request, such as the following:

- Start time
- End time
- Status (PASSED/FAILED)
- Failure reason, if it failed
- HTTP call details such as request/response headers, body, status code, status message, and performance timings.

Parameters

stepName(*String*)

Specifies the name of the step. This name is also used for publishing CloudWatch metrics for this step.

requestOptions(*Object or String*)

The value of this parameter can be a URL, a URL string, or an object. If it is an object, then it must be a set of configurable options to make an HTTP request. It supports all options in [http.request\(options\[, callback\]\)](#) in the Node.js documentation.

In addition to these Node.js options, **requestOptions** supports the additional parameter **body**. You can use the **body** parameter to pass data as a request body.

callback(*response*)

(Optional) This is a user function which is invoked with the HTTP response. The response is of the type [Class: `http.IncomingMessage`](#).

stepConfig(*object*)

(Optional) Use this parameter to override global synthetics configurations with a different configuration for this step.

Examples of using `executeHttpRequest`

The following series of examples build on each other to illustrate the various uses of this option.

This first example configures request parameters. You can pass a URL as **requestOptions**:

```
let requestOptions = 'https://www.amazon.com';
```

Or you can pass a set of options:

```
let requestOptions = {
  'hostname': 'myproductsEndpoint.com',
  'method': 'GET',
  'path': '/test/product/validProductName',
  'port': 443,
  'protocol': 'https:'
};
```

The next example creates a callback function which accepts a response. By default, if you do not specify **callback**, CloudWatch Synthetics validates that the status is between 200 and 299 inclusive.

```
// Handle validation for positive scenario
const callback = async function(res) {
  return new Promise((resolve, reject) => {
    if (res.statusCode < 200 || res.statusCode > 299) {
      throw res.statusCode + ' ' + res.statusMessage;
    }
  });
};
```

```
    }

    let responseBody = '';
    res.on('data', (d) => {
      responseBody += d;
    });

    res.on('end', () => {
      // Add validation on 'responseBody' here if required. For ex, your
      status code is 200 but data might be empty
      resolve();
    });
  });
};
```

The next example creates a configuration for this step that overrides the global CloudWatch Synthetics configuration. The step configuration in this example allows request headers, response headers, request body (post data), and response body in your report and restrict 'X-Amz-Security-Token' and 'Authorization' header values. By default, these values are not included in the report for security reasons. If you choose to include them, the data is only stored in your S3 bucket.

```
// By default headers, post data, and response body are not included in the report for
// security reasons.
// Change the configuration at global level or add as step configuration for individual
// steps
let stepConfig = {
  includeRequestHeaders: true,
  includeResponseHeaders: true,
  restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
  values do not appear in report generated.
  includeRequestBody: true,
  includeResponseBody: true
};
```

This final example passes your request to `executeHttpRequest` and names the step.

```
await synthetics.executeHttpRequest('Verify GET products API', requestOptions, callback,
  stepConfig);
```

With this set of examples, CloudWatch Synthetics adds the details from each step in your report and produces metrics for each step using `stepName`.

You will see `successPercent` and `duration` metrics for the `Verify GET products` API step. You can monitor your API performance by monitoring the metrics for your API call steps.

For a sample complete script that uses these functions, see [Multi-step API canary](#).

Library functions available for Python canary scripts using Selenium

This section lists the Selenium library functions available for Python canary scripts.

Topics

- [Python and Selenium library classes and functions that apply to all canaries](#)
- [Python and Selenium library classes and functions that apply to UI canaries only](#)

Python and Selenium library classes and functions that apply to all canaries

The following CloudWatch Synthetics Selenium library functions for Python are useful for all canaries.

Topics

- [SyntheticsConfiguration class](#)
- [SyntheticsLogger class](#)

SyntheticsConfiguration class

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level.

Function definitions:

`set_config(options)`

```
from aws_synthetics.common import synthetics_configuration
```

options is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *options*.

- `screenshot_on_step_start` (boolean)— Whether to take a screenshot before starting a step.
- `screenshot_on_step_success` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshot_on_step_failure` (boolean)— Whether to take a screenshot after a step fails.

`with_screenshot_on_step_start(screenshot_on_step_start)`

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

`with_screenshot_on_step_success(screenshot_on_step_success)`

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

`with_screenshot_on_step_failure(screenshot_on_step_failure)`

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

`get_screenshot_on_step_start()`

Returns whether to take a screenshot before starting a step.

`get_screenshot_on_step_success()`

Returns whether to take a screenshot after completing a step successfully.

`get_screenshot_on_step_failure()`

Returns whether to take a screenshot after a step fails.

`disable_step_screenshots()`

Disables all screenshot options (`get_screenshot_on_step_start`, `get_screenshot_on_step_success`, and `get_screenshot_on_step_failure`).

`enable_step_screenshots()`

Enables all screenshot options (`get_screenshot_on_step_start`, `get_screenshot_on_step_success`, and `get_screenshot_on_step_failure`). By default, all these methods are enabled.

`setConfig(options)` regarding CloudWatch metrics

For canaries using `syn-python-selenium-1.1` or later, the **(options)** for `setConfig` can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see [CloudWatch metrics published by canaries](#).

- `failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failed_requests_metric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `2xx_metric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `4xx_metric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `5xx_metric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `step_duration_metric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `step_success_metric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `aggregated_failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_failed_requests_metric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_2xx_metric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_4xx_metric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_5xx_metric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

with_2xx_metric(2xx_metric)

Accepts a Boolean argument, which specifies whether to emit a 2xx metric with the `CanaryName` dimension for this canary.

with_4xx_metric(4xx_metric)

Accepts a Boolean argument, which specifies whether to emit a 4xx metric with the `CanaryName` dimension for this canary.

with_5xx_metric(5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a 5xx metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(agggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 2xx metric with no dimension for this canary.

withAggregated4xxMetric(agggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a 4xx metric with no dimension for this canary.

with_aggregated_5xx_metric(agggregated_5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a 5xx metric with no dimension for this canary.

with_aggregated_failed_canary_metric(agggregated_failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

with_aggregated_failed_requests_metric(agggregated_failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

with_failed_canary_metric(failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

`with_failed_requests_metric(failed_requests_metric)`

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

`with_step_duration_metric(step_duration_metric)`

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

`with_step_success_metric(step_success_metric)`

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable metrics

`disable_aggregated_request_metrics()`

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

`disable_request_metrics()`

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

`disable_step_metrics()`

Disables all step metrics, including both step success metrics and step duration metrics.

`enable_aggregated_request_metrics()`

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

`enable_request_metrics()`

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

`enable_step_metrics()`

Enables all step metrics, including both step success metrics and step duration metrics.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration. Then, set the configuration for each option by calling the `setConfig` method using one of the following options.

```
from aws_synthetics.common import synthetics_configuration

synthetics_configuration.set_config(
    {
        "screenshot_on_step_start": False,
        "screenshot_on_step_success": False,
        "screenshot_on_step_failure": True
    }
)

or
```

Or

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(F
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synthetics_configuration.disable_step_screenshots()
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

`set_config(options)` for UI canaries

Starting with `syn-python-selenium-1.1`, for UI canaries, `set_config` can include the following Boolean parameters:

- `continue_on_step_failure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the `executeStep` function). If any steps fail, the canary run will still be marked as failed. The default is `false`.

SyntheticsLogger class

`synthetics_logger` writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

Using `synthetics_logger` is not required to create a log file that is uploaded to your Amazon S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in the S3 bucket as artifacts.

To use `synthetics_logger`:

```
from aws_synthetics.common import synthetics_logger
```

Useful function definitions:

Get log level:

```
log_level = synthetics_logger.get_level()
```

Set log level:

```
synthetics_logger.set_level()
```

Log a message with a specified level. The level can be DEBUG, INFO, WARN, or ERROR, as in the following syntax examples:

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warn(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

For information about debug parameters, see the standard Python documentation at [logging.debug](#)

In these logging functions, the message is the message format string. The args are the arguments that are merged into msg using the string formatting operator.

There are three keyword arguments in kwargs:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.
- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the LogRecord created for the logging event with user-defined attributes.

Examples:

Log a message with the level DEBUG:

```
synthetics_logger.debug('Starting step - login.')
```

Log a message with the level INFO. `logger.log` is a synonym for `logger.info`:

```
synthetics_logger.info('Successfully completed step - login.')
```

or

```
synthetics_logger.log('Successfully completed step - login.')
```

Log a message with the level WARN:

```
synthetics_logger.warn('Warning encountered trying to publish %s', 'CloudWatch Metric')
```

Log a message with the level ERROR:

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Log an exception:

```
synthetics_logger.exception(message, *args, **kwargs)
```

Logs a message with level ERROR. Exception information is added to the logging message. You should call this function only from an exception handler.

For information about exception parameters, see the standard Python documentation at [logging.exception](#)

The message is the message format string. The args are the arguments, which are merged into msg using the string formatting operator.

There are three keyword arguments in kwargs:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.
- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the LogRecord created for the logging event with user-defined attributes.

Example:

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Python and Selenium library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics Selenium library functions for Python are useful only for UI canaries.

Topics

- [SyntheticsBrowser class](#)
- [SyntheticsWebDriver class](#)

SyntheticsBrowser class

When you create a browser instance by calling `synthetics_webdriver.Chrome()`, the returned browser instance is of the type `SyntheticsBrowser`. The `SyntheticsBrowser` class controls the `ChromeDriver`, and enables the canary script to drive the browser, allowing the Selenium `WebDriver` to work with Synthetics.

In addition to the standard Selenium methods, it also provides the following methods.

set_viewport_size(width, height)

Sets the viewport of the browser. Example:

```
browser.set_viewport_size(1920, 1080)
```

save_screenshot(filename, suffix)

Saves screenshots to the `/tmp` directory. The screenshots are uploaded from there to the canary artifacts folder in the S3 bucket.

filename is the file name for the screenshot, and *suffix* is an optional string to be used for naming the screenshot.

Example:

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver class

To use this class, use the following in your script:

```
from aws_synthetics.selenium import synthetics_webdriver
```

add_execution_error(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `add_execution_error` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `add_execution_error` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try:
    browser.save_screenshot("loaded.png")
except Exception as ex:
    self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

Appends the value of `user_agent_str` to the browser's user agent header. You must assign `user_agent_str` before creating the browser instance.

Example:

```
synthetics_webdriver.add_user_agent('MyApp-1.0')
```

execute_step(step_name, function_to_execute)

Processes one function. It also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName SuccessPercent` metric, 100 for pass or 0 for failure.

- Emits the `stepName Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

Example:

```
from selenium.webdriver.common.by import By

def custom_actions():
    #verify contains
    browser.find_element(By.XPATH, "//*[@id=\"id_1\"][contains(text(),'login')]")
    #click a button
    browser.find_element(By.XPATH, '//*[@id="submit"]/a').click()

await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

Chrome()

Launches an instance of the Chromium browser and returns the created instance of the browser.

Example:

```
browser = synthetics_webdriver.Chrome()
browser.get("https://example.com/)
```

To launch a browser in incognito mode, use the following:

```
add_argument('--incognito')
```

To add proxy settings, use the following:

```
add_argument('--proxy-server=%s' % PROXY)
```

Example:

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("--incognito")
```

```
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

Scheduling canary runs using cron

Using a cron expression gives you flexibility when you schedule a canary. Cron expressions contain five or six fields in the order listed in the following table. The fields are separated by spaces. The syntax differs depending on whether you are using the CloudWatch console to create the canary, or the AWS CLI or AWS SDKs. When you use the console, you specify only the first five fields. When you use the AWS CLI or AWS SDKs, you specify all six fields, and you must specify `*` for the Year field.

Field	Allowed values	Allowed special characters
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? L #
Year	*	

Special characters

- The `,` (comma) includes multiple values in the expression for a field. For example, in the Month field, `JAN,FEB,MAR` would include January, February, and March.
- The `-` (dash) special character specifies ranges. In the Day field, `1-15` would include days 1 through 15 of the specified month.
- The `*` (asterisk) special character includes all values in the field. In the Hours field, `*` includes every hour. You cannot use `*` in both the Day-of-month and Day-of-week fields in the same expression. If you use it in one, you must use `?` in the other.
- The `/` (forward slash) specifies increments. In the Minutes field, you can enter `1/10` to specify every tenth minute, starting from the first minute of the hour (for example, the eleventh, twenty-first, and thirty-first minute, and so on).

- The **?** (question mark) specifies one or another. If you enter **7** in the Day-of-month field and you don't care what day of the week the seventh is, you can enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, **3W** specifies the weekday closest to the third day of the month.
- The **#** wildcard in the Day-of-week field specifies a certain instance of the specified day of the week within a month. For example, **3#2** is the second Tuesday of the month. The **3** refers to Tuesday because it is the third day of each week, and the **2** refers to the second day of that type within the month.

Limitations

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or ***** (asterisk) in one of the fields, you must use a **?** (question mark) in the other.
- Cron expressions that lead to rates faster than one minute are not supported.
- You can't set a canary to wait for more than a year before running, so you can specify only ***** in the Year field.

Examples

You can refer to the following sample cron strings when you create a canary. The following examples are the correct syntax for using the AWS CLI or AWS SDKs to create or update a canary. If you are using the CloudWatch console, omit the final ***** in each example.

Expression	Meaning
<code>0 10 * * ? *</code>	Run at 10:00 am (UTC) every day
<code>15 12 * * ? *</code>	Run at 12:15 pm (UTC) every day
<code>0 18 ? * MON-FRI *</code>	Run at 6:00 pm (UTC) every Monday through Friday
<code>0 8 1 * ? *</code>	Run at 8:00 am (UTC) on the first day of each month
<code>0/10 * ? * MON-SAT *</code>	Run every 10 minutes Monday through Saturday of each week

Expression	Meaning
0/5 8-17 ? * MON-FRI *	Run every five minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

Groups

You can create *groups* to associate canaries with each other, including cross-Region canaries. Using groups can help you with managing and automating your canaries, and you can also view aggregated run results and statistics for all canaries in a group.

Groups are global resources. When you create a group, it is replicated across all AWS Regions that support groups, and you can add canaries from any of these Regions to it, and view it in any of these Regions. Although the group ARN format reflects the Region name where it was created, a group is not constrained to any Region. This means that you can put canaries from multiple Regions into the same group, and then use that group to view and manage all of those canaries in a single view.

Groups are supported in all Regions except the Regions that are disabled by default. For more information about these Regions, see [Enabling a Region](#).

Each group can contain as many as 10 canaries. You can have as many as 20 groups in your account. Any single canary can be a member of up to 10 groups.

To create a group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.
3. Choose **Create Group**.
4. Under **Group Name**, enter a name for the group.
5. Select canaries to associate with this group. To select a canary, type its complete name in **Exact canary name** and choose **Search**. Then select the check box next to the canary name. If there are multiple canaries with the same name in different Regions, be sure to select the canaries that you want.

You can repeat this step to associate as many as 10 canaries with the group.

6. (Optional) Under **Tags**, add one or more key-value pairs as tags for this group. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
7. Choose **Create Group**.

Test a canary locally

This section explains how to modify, test, and debug CloudWatch Synthetics canaries directly within the Microsoft Visual Studio code editor or the JetBrains IDE code editor. The local debugging environment uses a Serverless Application Model (SAM) container to simulate a Lambda function to emulate the behavior of a Synthetics canary.

Note

It is impractical to perform locally debug canaries that rely on visual monitoring. Visual monitoring rely on capturing base screenshots during an initial run, and then comparing these screenshots to the screenshots from subsequent runs. In a local development environment, runs are not stored or tracked, and each iteration is an independent, standalone run. The absence of a canary run history makes it impractical to debug canaries that rely on visual monitoring.

Prerequisites

1. Choose or create an Amazon S3 bucket to use to store artifacts from local canary test runs, such as HAR files and screenshots. This requires you to be provisioned with IAM. If you skip setting up Amazon S3 buckets you can still test your canary locally, but you will see an error message about the missing bucket and you won't have access to canary artifacts.

If you use an Amazon S3 bucket, we recommend that you set the bucket lifecycle to delete objects after a few days, to save costs. For more information, see [Managing your storage lifecycle](#).

2. Set up a default AWS profile for your AWS account. For more information, see [Configuration and credential file settings](#).
3. Set the debug environment's default AWS Region to your preferred Region, such as us-west-2.
4. Install the AWS SAM CLI. For more information, see [Installing the AWS SAM CLI](#).

5. Install Visual Studio Code Editor or JetBrains IDE. For more information, see [Visual Studio Code](#) or [JetBrains IDE](#)
6. Install Docker to work with the AWS SAM CLI. Make sure to start the docker daemon. For more information, see [Installing Docker to use with the AWS SAM CLI](#).

Alternatively, You can install other container management software such as Rancher, as long as it uses the Docker runtime.

7. Install an AWS toolkit extension for your preferred editor. For more information, see [Installing the AWS Toolkit for Visual Studio Code](#) or [Installing the AWS Toolkit for JetBrains](#).

Topics

- [Set up the testing and debugging environment](#)
- [Use Visual Studio Code IDE](#)
- [Use JetBrains IDE](#)
- [Run a canary locally with the SAM CLI](#)
- [Integrate your local testing environment into an existing canary package](#)
- [Change the CloudWatch Synthetics runtime](#)
- [Common errors](#)

Set up the testing and debugging environment

First, clone the Github repository that AWS provides by entering the following command. The repository contains code samples for both Node.js canaries and Python canaries.

```
git clone https://github.com/aws-samples/synthetics-canary-local-debugging-sample.git
```

Then do one of the following, depending on the language of your canaries.

For Node.js canaries

1. Go to the Node.js canary source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/nodejs-canary/src
```

2. Enter the following command to install canary dependencies.

```
npm install
```

For Python canaries

1. Go to the Python canary source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/python-canary/src
```

2. Enter the following command to install canary dependencies.

```
pip3 install -r requirements.txt -t .
```

Use Visual Studio Code IDE

The Visual Studio launch configuration file is located at `.vscode/launch.json`. It contains configurations to allow the template file to be discovered by Visual Studio code. It defines a Lambda payload with required parameters to invoke the canary successfully. Here's the launch configuration for a Node.js canary:

```
{
    ...
    ...
    "lambda": {
        "payload": {
            "json": {
                // Canary name. Provide any name you like.
                "canaryName": "LocalSyntheticsCanary",
                // Canary artifact location
                "artifactS3Location": {
                    "s3Bucket": "cw-syn-results-123456789012-us-west-2",
                    "s3Key": "local-run-artifacts",
                },
                // Your canary handler name
                "customerCanaryHandlerName": "heartbeat-canary.handler"
            }
        },
        // Environment variables to pass to the canary code
        "environmentVariables": {}
    }
}
```



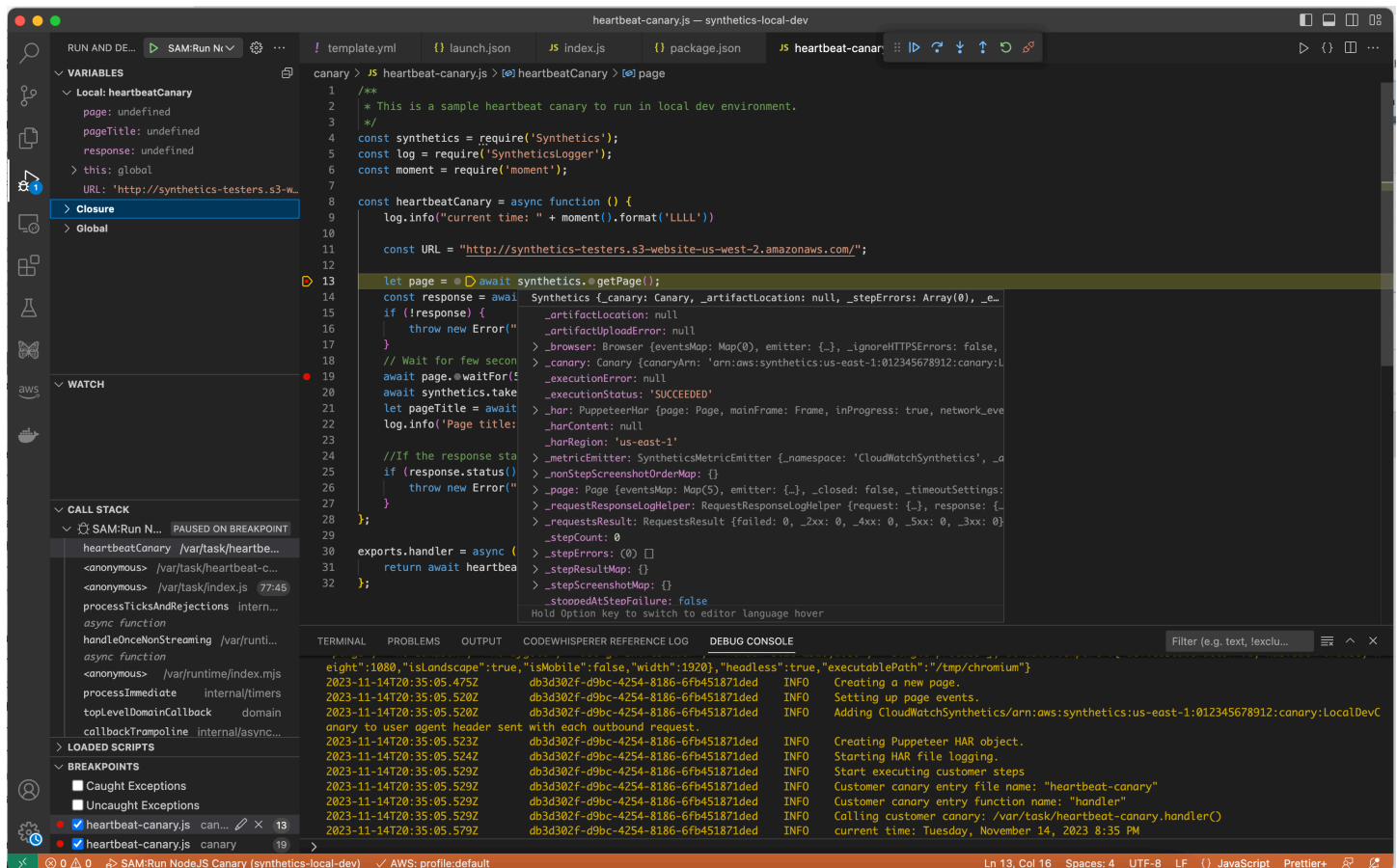
```
    }  
  ]  
}
```

You can also optionally provide the following fields in the payload JSON:

- `s3EncryptionMode` Valid values: `SSE_S3` | `SSE_KMS`
- `s3KmsKeyArn` Valid value: *KMS Key ARN*
- `activeTracing` Valid values: `true` | `false`
- `canaryRunId` Valid value: *UUID* This parameter is required if active tracing is enabled.

To debug the canary in Visual Studio, add breakpoints in the canary code where you want to pause execution. To add a breakpoint, choose the editor margin and go to **Run and Debug** mode in the editor. Run the canary by clicking on the play button. When the canary runs, the logs will be tailed in the debug console, providing you with real-time insights into the canary's behavior. If you added breakpoints, the canary execution will pause at each breakpoint, allowing you to step through code and inspect variable values, instance methods, object attributes, and the function call stack.

There is no cost incurred for running and debugging canaries locally, except for the artifacts stored in the Amazon S3 bucket and the CloudWatch metrics generated by each local run.



Use JetBrains IDE

After you have the AWS Toolkit for JetBrains extension installed, be sure that the Node.js plugin and JavaScript debugger are enabled to run, if you are debugging a Node.js canary. Then follow these steps.

Debug a canary using JetBrains IDE

1. In the left navigation pane of JetBrains IDE, choose **Lambda**, then choose the local configuration template.
2. Enter a name for the run configuration, such as **LocalSyntheticsCanary**
3. Choose **From template**, choose the file browser in the template field, then select the **template.yml** file from the project, either from the **nodejs** directory or the **python** directory.
4. In the **Input** section, enter the payload for the canary as shown in the following screen.

```

{
  "canaryName": "LocalSyntheticsCanary",
  "artifactS3Location": {

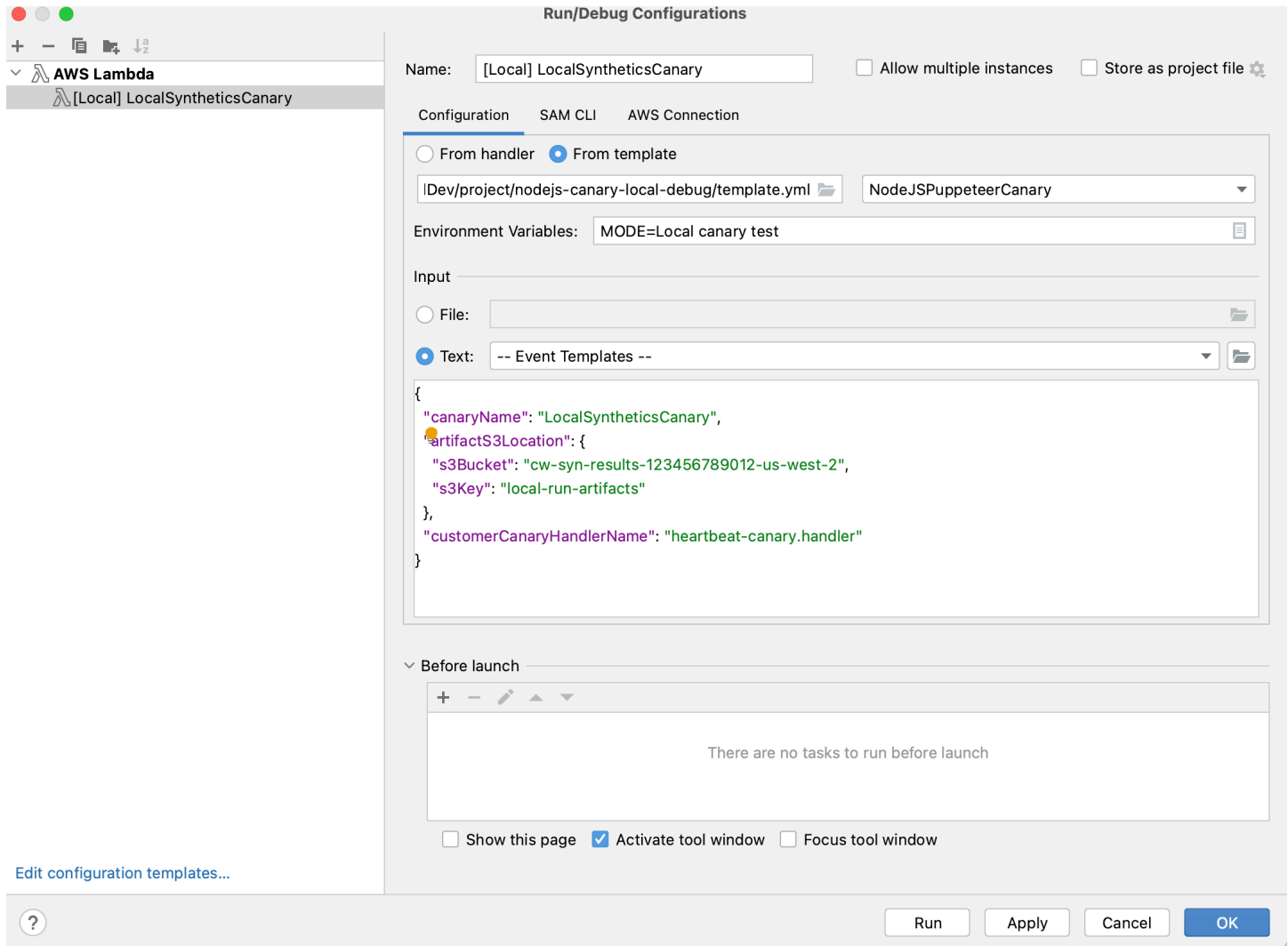
```

```

    "s3Bucket": "cw-syn-results-123456789012-us-west-2",
    "s3Key": "local-run-artifacts"
  },
  "customerCanaryHandlerName": "heartbeat-canary.handler"
}

```

You can also set other environment variables in the payload JSON, as listed in [Use Visual Studio Code IDE](#).



Run a canary locally with the SAM CLI

Use the one of the following procedures to run your canary locally using the Serverless Application Model (SAM) CLI. Be sure to specify your own Amazon S3 bucket name for `s3Bucket` in `event.json`

To use the SAM CLI to run a Node.js canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/nodejs-canary
```

2. Enter the following commands.

```
sam build
sam local invoke -e ../event.json
```

To use the SAM CLI to run a Python canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/python-canary
```

2. Enter the following commands.

```
sam build
sam local invoke -e ../event.json
```

Integrate your local testing environment into an existing canary package

You can integrate local canary debugging into your existing canary package by copying three files:

- Copy the `template.yml` file into your canary package root. Be sure to modify the path for `CodeUri` to point to the directory where your canary code exists.
- If you're working with a Node.js canary, copy the `cw-synthetics.js` file to your canary source directory. If you're working with a Python canary, copy the `cw-synthetics.py` to your canary source directory.
- Copy the launch configuration file `.vscode/launch.json` into the package root. Make sure to put it inside the `.vscode` directory; create it if it doesn't exist already.

Change the CloudWatch Synthetics runtime

As part of your debugging, you might want to try running a canary with a different CloudWatch Synthetics runtime, instead of the latest runtime. To do so, find the runtime that you want to use from one of the following tables. Be sure to select the runtime for the correct Region. Then paste the ARN for that runtime into the appropriate place in your `template.yml` file, and then run the canary.

Node.js runtimes

ARNs for syn-nodejs-puppeteer-7.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-7.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:44</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:46</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:44</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:47</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:44</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:45</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:20</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:26</code>

Region	ARN
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:44
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:46
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:49
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:44
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:44
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:76
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:45
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:46
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:44

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:46
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:44
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:45
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:44
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:44
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:19
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:17
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:44
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:19
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:45
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:41

Region	ARN
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:42

ARNs for syn-nodejs-puppeteer-6.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-6.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:41
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:43
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:41
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:44
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:41
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:42
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15

Region	ARN
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:41
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:46
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:41
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:41
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:41
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:42
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:41
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:43

Region	ARN
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:41
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:42
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:41
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:41
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:41
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:42
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:39
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:39

ARNs for syn-nodejs-puppeteer-5.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-5.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:42</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:44</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:42</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:45</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:42</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:43</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:18</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:24</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:16</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:42</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:28</code>

Region	ARN
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:47
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:42
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:42
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:42
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:74
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:44
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:42
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:44
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:42
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:43

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:42
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:42
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:17
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:15
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:42
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:17
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:43
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:40
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:40

Python runtimes

ARNs for syn-python-selenium-3.0

The following table lists the ARNs to use for version `syn-python-selenium-3.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>aarn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:32</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:34</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:32</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:34</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:32</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:32</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:20</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:26</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:18</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:32</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:30</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:34</code>

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:37
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:32
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:32
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:32
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:76
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:32
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:32
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:32
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:34
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:32
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:33
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:32

Region	ARN
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:20</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:32</code>
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:19</code>
Israel (Tel Aviv)	<code>arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:17</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:32</code>
Middle East (UAE)	<code>arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:19</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:33</code>
AWS GovCloud (US-East)	<code>arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:30</code>
AWS GovCloud (US-West)	<code>arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:31</code>

ARNs for syn-python-selenium-2.1

The following table lists the ARNs to use for version `syn-python-selenium-2.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:29</code>

Region	ARN
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:31
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:29
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:31
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:29
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:29
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:29
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:30
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:34

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:29
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:29
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:29
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:29
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:29
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:29
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:31
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:29
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:30
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:29
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17

Region	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:29
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:29
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:30
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:29
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:29

Common errors

Error: Running AWS SAM projects locally requires Docker. Have you got it installed and running?

Make sure to start Docker on your computer.

SAM local invoke failed: An error occurred (ExpiredTokenException) when calling the GetLayerVersion operation: The security token included in the request is expired

Make sure that the AWS default profile is set up.

More common errors

For more information about common errors with the SAM, see [AWS SAM CLI troubleshooting](#) .

Troubleshooting a failed canary

If your canary fails, check the following for troubleshooting.

General troubleshooting

- Use the canary details page to find more information. In the CloudWatch console, choose **Canaries** in the navigation pane and then choose the name of the canary to open the canary details page. In the **Availability** tab, check the **SuccessPercent** metric to see whether the problem is constant or intermittent.

While still in the **Availability** tab, choose a failed data point to see screenshots, logs, and step reports (if available) for that failed run.

If a step report is available because steps are part of your script, check to see which step has failed and see the associated screenshots to see the issue that your customers are seeing.

You can also check the HAR files to see if one or more requests are failing. You can dig deeper by using logs to drill down on failed requests and errors. Finally, you can compare these artifacts with the artifacts from a successful canary run to pinpoint the issue.

By default, CloudWatch Synthetics captures screenshots for each step in a UI canary. However, your script might be configured to disable screenshots. During debugging, you may want to enable screenshots again. Similarly, for API canaries you might want to see HTTP request and response headers and body during debugging. For information about how to include this data in the report, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#).

- If you had a recent deployment to your application, roll it back and then debug later.
- Connect to your endpoint manually to see if you can reproduce the same issue.

Topics

- [Canary fails after Lambda environment update](#)
- [My canary is blocked by AWS WAF](#)
- [Waiting for an element to appear](#)
- [Node is either not visible or not an HTML element for page.click\(\)](#)
- [Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied](#)

- [Error: Protocol error \(Runtime.callFunctionOn\): Target closed.](#)
- [Canary Failed. Error: No datapoint - Canary Shows timeout error](#)
- [Trying to access an internal endpoint](#)
- [Canary runtime version upgrade and downgrade issues](#)
- [Cross-origin request sharing \(CORS\) issue](#)
- [Canary race condition issues](#)
- [Troubleshooting a canary on a VPC](#)

Canary fails after Lambda environment update

CloudWatch Synthetics canaries are implemented as Lambda functions in your account. These Lambda functions are subject to regular Lambda runtime updates containing security updates, bug fixes, and other improvements. Lambda strives to provide runtime updates that are backward-compatible with existing functions. However, as with software patching, there are rare cases in which a runtime update can negatively impact an existing function. If you believe your canary has been impacted by a Lambda runtime update, you can use the Lambda runtime management manual mode (in supported Regions) to temporarily roll back the Lambda runtime version. This keeps your canary function working and minimizes disruption, providing time to remedy the incompatibility before returning to the latest runtime version.

If your canary is failing after a Lambda runtime update, the best solution is to upgrade to one of the newest Synthetics runtimes. For more information about the latest runtimes, see [Synthetics runtime versions](#).

As an alternative solution, in Regions where Lambda runtime management controls are available, you can revert a canary back to an older Lambda managed runtime, using manual mode for runtime management controls. You can set manual mode using either the AWS CLI or by using the Lambda console, using the steps below in the following sections.

Warning

When you change the runtime settings to manual mode, your Lambda function won't receive automatic security updates until it is reverted back to Auto mode. During this period, your Lambda function might be susceptible to security vulnerabilities.

Prerequisites

- Install [jq](#)
- Install the latest version of the AWS CLI. For more information, see [AWS CLI install and update instructions](#).

Step 1: Obtain the Lambda function ARN

Run the following command to retrieve the `EngineArn` field from the response. This `EngineArn` is the ARN of the Lambda function that is associated with the canary. You will use this ARN in the following steps.

```
aws synthetics get-canary --name my-canary | jq '.Canary.EngineArn'
```

Example output of `EngineArn`:

```
"arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-afb1-EXAMPLE991:8"
```

Step 2: Obtain the last good Lambda runtime version ARN

To help understand whether your canary was impacted by a Lambda runtime update, check whether the date and time when the Lambda runtime version ARN changes in your logs appeared to the date and time when you saw impact to your canary. If they do not match, it is probably not a Lambda runtime update that is causing your issues.

If your canary is impacted by a Lambda runtime update, you must identify the ARN of the working Lambda runtime version that you were previously using. Follow the instructions in [Identifying runtime version changes](#) to find the ARN of the previous runtime. Record the runtime version ARN, and continue to Step 3. for setting the runtime management configuration.

If your canary has not yet been impacted by a Lambda environment update, then you can find the ARN of the Lambda runtime version that you are currently using. Run the following command to retrieve the `RuntimeVersionArn` of the Lambda function from the response.

```
aws lambda get-function-configuration \
--function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991:8" | jq '.RuntimeVersionConfig.RuntimeVersionArn'
```

Example output of `RuntimeVersionArn`:

```
"arn:aws:lambda:us-west-2::runtime:EXAMPLE647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

Step 3: Updating the Lambda runtime management configuration

You can use either the AWS CLI or the Lambda console to update the runtime management configuration.

To set Lambda runtime management configuration manual mode using the AWS CLI

Enter the following command to change the runtime management of the Lambda function to manual mode. Be sure to replace the *function-name* and *qualifier* with the Lambda function ARN and Lambda function version number respectively, using the values you found in Step 1. Also replace the *runtime-version-arn* with the version ARN that you found in Step 2.

```
aws lambda put-runtime-management-config \
  --function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991" \
  --qualifier 8 \
  --update-runtime-on "Manual" \
  --runtime-version-arn "arn:aws:lambda:us-
west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

To change a canary to manual mode using the Lambda console

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose the **Versions** tab, choose the version number link that corresponds to your ARN, and choose the **Code** tab.
3. Scroll down to **Runtime settings**, expand **Runtime management configuration**, and copy the **Runtime version ARN**.

The screenshot shows the 'Runtime settings' section of the AWS Lambda console. It includes buttons for 'Edit' and 'Edit runtime management configuration'. The 'Runtime' is set to 'Node.js 18.x', the 'Handler' is 'index.handler', and the 'Architecture' is 'x86_64'. Under the 'Runtime management configuration' section, the 'Runtime version ARN' is displayed as 'arn:aws:lambda:us-west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f' with a copy icon. The 'Update runtime version' is set to 'Auto'.

4. Choose **Edit runtime management configuration**, choose **Manual**, paste the runtime version ARN that you copied earlier into the **Runtime version ARN** field. Then choose **Save**.

Edit runtime management configuration

Runtime management configuration [Info](#)

Update runtime version
Choose when your function receives security updates from Lambda.

Auto
Automatically update to the most recent and secure runtime version.

Function update
Your function's runtime version is only updated when you make changes to your function.

Manual
Your function's runtime version is not updated and won't receive security updates.

⚠ When you choose **Manual**, your function's runtime version won't receive security updates.

Runtime version ARN [Info](#)
To roll back to an earlier runtime version, get the earlier runtime version ARN from your function logs. If you are using CloudWatch, see [CloudWatch Logs](#).

```
arn:aws:lambda:us-west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f
```

Required format: `arn:aws:lambda:{region}::runtime:{id}`

[Cancel](#)
[Save](#)

My canary is blocked by AWS WAF

To prevent AWS WAF from blocking your canary, set up a AWS WAF string match condition that allows the string `CloudWatchSynthetics`. For more information, see [Working with string match conditions](#) in the AWS WAF documentation.

Waiting for an element to appear

After analyzing your logs and screenshots, if you see that your script is waiting for an element to appear on screen and times out, check the relevant screenshot to see if the element appears on the page. Verify your xpath to make sure that it is correct.

For Puppeteer-related issues, check [Puppeteer's GitHub page](#) or internet forums.

Node is either not visible or not an HTML Element for page.click()

If a node is not visible or is not an HTML Element for `page.click()`, first verify the xpath that you are using to click the element. Also, if your element is at the bottom of the screen, adjust your viewport. CloudWatch Synthetics by default uses a viewport of 1920 * 1080. You can set a different viewport when you launch the browser or by using the Puppeteer function `page.setViewport`.

Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied

If your canary fails because of an Amazon S3 error, CloudWatch Synthetics was unable to upload screenshots, logs, or reports created for the canary because of permission issues. Check the following:

- Check that the canary's IAM role has the `s3:ListAllMyBuckets` permission, the `s3:GetBucketLocation` permission for the correct Amazon S3 bucket, and the `s3:PutObject` permission for the bucket where the canary stores its artifacts. If the canary performs visual monitoring, the role also needs the `s3:GetObject` permission for the bucket. These same permissions are also required in the Amazon VPC S3 Gateway Endpoint Policy, if the canary is deployed in a VPC with a VPC endpoint.
- If the canary uses an AWS KMS customer managed key for encryption instead of the standard AWS managed key (default), the canary's IAM role might not have the permission to encrypt or decrypt using that key. For more information, see [Encrypting canary artifacts](#).
- Your bucket policy might not allow the encryption mechanism that the canary uses. For example, if your bucket policy mandates to use a specific encryption mechanism or KMS key, then you must select the same encryption mode for your canary.

If the canary performs visual monitoring, see [Updating artifact location and encryption when using visual monitoring](#) for more information.

Error: Protocol error (Runtime.callFunctionOn): Target closed.

This error appears if there are some network requests after the page or browser is closed. You might have forgotten to wait for an asynchronous operation. After executing your script, CloudWatch Synthetics closes the browser. The execution of any asynchronous operation after the browser is closed might cause `target closed` error.

Canary Failed. Error: No datapoint - Canary Shows timeout error

This means that your canary run exceeded the timeout. The canary execution stopped before CloudWatch Synthetics could publish success percent CloudWatch metrics or update artifacts such as HAR files, logs and screenshots. If your timeout is too low, you can increase it.

By default, a canary timeout value is equal to its frequency. You can manually adjust the timeout value to be less than or equal to the canary frequency. If your canary frequency is low, you must

increase the frequency to increase the timeout. You can adjust both the frequency and the timeout value under **Schedule** when you create or update a canary by using the CloudWatch Synthetics console.

Be sure that your canary timeout value is no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Canary artifacts are not available to view in the CloudWatch Synthetics console when this error happens. You can use CloudWatch Logs to see the canary's logs.

To use CloudWatch Logs to see the logs for a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group by typing the canary name in the filter box. Log groups for canaries have the name `/aws/lambda/cwsyn-canaryName-randomId`.

Trying to access an internal endpoint

If you want your canary to access an endpoint on your internal network, we recommend that you set up CloudWatch Synthetics to use VPC. For more information, see [Running a canary on a VPC](#).

Canary runtime version upgrade and downgrade issues

If you recently upgraded the canary from runtime version `syn-1.0` to a later version, it may be a cross-origin request sharing (CORS) issue. For more information, see [Cross-origin request sharing \(CORS\) issue](#).

If you recently downgraded the canary to an older runtime version, check to make sure that the CloudWatch Synthetics functions that you are using are available in the older runtime version that you downgraded to. For example, the `executeHttpStep` function is available for runtime version `syn-nodejs-2.2` and later. To check on the availability of functions, see [Writing a canary script](#).

Note

When you plan to upgrade or downgrade the runtime version for a canary, we recommend that you first clone the canary and update the runtime version in the cloned canary. Once you have verified that the clone with the new runtime version works, you can update the runtime version of your original canary and delete the clone.

Cross-origin request sharing (CORS) issue

In a UI canary, if some network requests are failing with 403 or `net::ERR_FAILED`, check whether the canary has active tracing enabled and also uses the Puppeteer function `page.setExtraHTTPHeaders` to add headers. If so, the failed network requests might be caused by cross-origin request sharing (CORS) restrictions. You can confirm whether this is the case by disabling active tracing or removing the extra HTTP headers.

Why does this happen?

When active tracing is used, an extra header is added to all outgoing requests to trace the call. Modifying the request headers by adding a trace header or adding extra headers using Puppeteer's `page.setExtraHTTPHeaders` causes a CORS check for XMLHttpRequest (XHR) requests.

If you don't want to disable active tracing or remove the extra headers, you can update your web application to allow cross-origin access or you can disable web security by using the `disable-web-security` flag when you launch the Chrome browser in your script.

You can override launch parameters used by CloudWatch Synthetics and pass additional `disable-web-security` flag parameters by using the CloudWatch Synthetics launch function. For more information, see [Library functions available for Node.js canary scripts](#).

Note

You can override launch parameters used by CloudWatch Synthetics when you use runtime version `syn-nodejs-2.1` or later.

Canary race condition issues

For the best experience when using CloudWatch Synthetics, ensure that the code written for the canaries is idempotent. Otherwise, in rare cases, canary runs may encounter race conditions when the canary interacts with the same resource across different runs.

Troubleshooting a canary on a VPC

If you have issues after creating or updating a canary on a VPC, one of the following sections might help you troubleshoot the problem.

New canary in error state or canary can't be updated

If you create a canary to run on a VPC and it immediately goes into an error state, or you can't update a canary to run on a VPC, the canary's role might not have the right permissions. To run on a VPC, a canary must have the permissions `ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces`, and `ec2>DeleteNetworkInterface`. These permissions are all contained in the `AWSLambdaVPCLambdaAccessExecutionRole` managed policy. For more information, see [Execution Role and User Permissions](#).

If this issue happened when you created a canary, you must delete the canary, and create a new one. If you use the CloudWatch console to create the new canary, under **Access Permissions**, select **Create a new role**. A new role that includes all permissions required to run the canary is created.

If this issue happens when you update a canary, you can update the canary again and provide a new role that has the required permissions.

"No test result returned" error

If a canary displays a "no test result returned" error, one of the following issues might be the cause:

- If your VPC does not have internet access, you must use VPC endpoints to give the canary access to CloudWatch and Amazon S3. You must enable the **DNS resolution** and **DNS hostname** options in the VPC for these endpoint addresses to resolve correctly. For more information, see [Using DNS with Your VPC](#) and [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints](#).
- Canaries must run in private subnets within a VPC. To check this, open the **Subnets** page in the VPC console. Check the subnets that you selected when configuring the canary. If they have a path to an internet gateway (**igw-**), they are not private subnets.

To help you troubleshoot these issues, see the logs for the canary.

To see the log events from a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the canary's log group. The log group name starts with `/aws/lambda/cwsyn-canary-name`.

Sample code for canary scripts

This section contains code samples that illustrate some possible functions for CloudWatch Synthetics canary scripts.

Samples for Node.js and Puppeteer

Setting cookies

Web sites rely on cookies to provide custom functionality or track users. By setting cookies in CloudWatch Synthetics scripts, you can mimic this custom behavior and validate it.

For example, a web site might display a **Login** link for a revisiting user instead of a **Register** link.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();

    // Set cookies. I found that name, value, and either url or domain are required
    fields.
    const cookies = [{
        'name': 'cookie1',
        'value': 'val1',
        'url': url
    },{
        'name': 'cookie2',
        'value': 'val2',
        'url': url
    },{
        'name': 'cookie3',
        'value': 'val3',
        'url': url
    }
    ];

    await page.setCookie(...cookies);

    // Navigate to the url
```

```
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'],
timeout: timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " +
JSON.stringify(cookiesSet));
    });

};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Device emulation

You can write scripts that emulate various devices so that you can approximate how a page looks and behaves on those devices.

The following sample emulates an iPhone 6 device. For more information about emulation, see [page.emulate\(options\)](#) in the Puppeteer documentation.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];

    // INSERT URL here
    const URL = "https://amazon.com";

    let page = await synthetics.getPage();
    await page.emulate(iPhone);

    //You can customize the wait condition here. For instance,
    //using 'networkidle2' may be less restrictive.
```

```
const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
30000});
if (!response) {
  throw "Failed to load page!";
}

await page.waitFor(15000);

await synthetics.takeScreenshot('loaded', 'loaded');

//If the response status code is not a 2xx success code
if (response.status() < 200 || response.status() > 299) {
  throw "Failed to load page!";
}
};

exports.handler = async () => {
  return await pageLoadBlueprint();
};
```

Multi-step API canary

This sample code demonstrates an API canary with two HTTP steps: testing the same API for positive and negative test cases. The step configuration is passed to enable reporting of request/response headers. Additionally, it hides the Authorization header and X-Amz-Security-Token, because they contain user credentials.

When this script is used as a canary, you can view details about each step and the associated HTTP requests such as step pass/fail, duration, and performance metrics like DNS look up time and first byte time. You can view the number of 2xx, 4xx and 5xx for your canary run.

```
var synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const apiCanaryBlueprint = async function () {

  // Handle validation for positive scenario
  const validatePositiveCase = async function(res) {
    return new Promise((resolve, reject) => {
      if (res.statusCode < 200 || res.statusCode > 299) {
        throw res.statusCode + ' ' + res.statusMessage;
      }
    })
  }
}
```

```
    let responseBody = '';
    res.on('data', (d) => {
      responseBody += d;
    });

    res.on('end', () => {
      // Add validation on 'responseBody' here if required. For ex, your
status code is 200 but data might be empty
      resolve();
    });
  });
};

// Handle validation for negative scenario
const validateNegativeCase = async function(res) {
  return new Promise((resolve, reject) => {
    if (res.statusCode < 400) {
      throw res.statusCode + ' ' + res.statusMessage;
    }

    resolve();
  });
};

let requestOptionsStep1 = {
  'hostname': 'myproductsEndpoint.com',
  'method': 'GET',
  'path': '/test/product/validProductName',
  'port': 443,
  'protocol': 'https:'
};

let headers = {};
headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

requestOptionsStep1['headers'] = headers;

// By default headers, post data and response body are not included in the report
for security reasons.
// Change the configuration at global level or add as step configuration for
individual steps
let stepConfig = {
```



```
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpRequestStep('Verify GET products API with valid name',
requestOptionsStep1, validatePositiveCase, stepConfig);

    let requestOptionsStep2 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/canary/InvalidName(',
        'port': 443,
        'protocol': 'https:'
    };

    headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep2['headers'] = headers;

    // By default headers, post data and response body are not included in the report
for security reasons.
    // Change the configuration at global level or add as step configuration for
individual steps
    stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpRequestStep('Verify GET products API with invalid name',
requestOptionsStep2, validateNegativeCase, stepConfig);

};
```

```
exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Samples for Python and Selenium

The following sample Selenium code is a canary that fails with a custom error message when a target element is not loaded.

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

Canaries and X-Ray tracing

You can choose to enable active AWS X-Ray tracing on canaries that use the `syn-nodejs-2.0` or later runtime. With tracing enabled, traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the [X-Ray Trace Map](#), and within [Application Signals](#) after you have enabled it for your application.

Note

Activating X-Ray tracing on canaries is not yet supported in Asia Pacific (Jakarta).

When a canary appears on the X-Ray trace map, it appears as a new client node type. You can hover on a canary node to see data about latency, requests, and faults. You can also choose the canary node to see more data at the bottom of the page. From this area of the page, you can choose **View in Synthetics** to jump to the CloudWatch Synthetics console for more details about the canary, or choose **View Traces** to see more details about the traces from this canary's runs.

A canary with tracing enabled also has a **Tracing** tab in its details page, with details about traces and segments from the canary's runs.

Enabling tracing increases canary run time by 2.5% to 7%.

A canary with tracing enabled must use a role with the following permissions. If you use the console to create the role when you create the canary, it is given these permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Sid230934",
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments"
      ],
      "Resource": "*"
    }
  ]
}
```

Traces generated by canaries incur charges. For more information about X-Ray pricing, see [AWS X-Ray Pricing](#).

Running a canary on a VPC

You can run canaries on endpoints on a VPC and public internal endpoints. To run a canary on a VPC, you must have both the **DNS Resolution** and **DNS hostnames** options enabled on the VPC. For more information, see [Using DNS with Your VPC](#).

When you run a canary on a VPC endpoint, you must provide a way for it to send its metrics to CloudWatch and its artifacts to Amazon S3. If the VPC is already enabled for internet access, there's nothing more for you to do. The canary executes in your VPC, but can access the internet to upload its metrics and artifacts.

If the VPC is not already enabled for internet access, you have two options:

- Enable it for internet access. For more information, see the following section [Giving internet access to your canary on a VPC](#).
- If you want to keep your VPC private, you can configure the canary to send its data to CloudWatch and Amazon S3 through private VPC endpoints. If you have not already done so, you must create a VPC endpoint for CloudWatch (com.amazonaws.*region*.monitoring) and a gateway endpoint for Amazon S3. For more information, see [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints](#) and [Amazon VPC Endpoints for Amazon S3](#).

Giving internet access to your canary on a VPC

Follow these steps to give internet access to your VPC canary, or to assign your canary a static IP address

To give internet access to a canary on a VPC

1. Create a NAT gateway in a public subnet on the VPC. For instructions, see [Create a NAT gateway](#).
2. Add a new route to the route table in the private subnet where the canary is launched. Specify the following:
 - For **Destination**, enter **0.0.0.0/0**
 - For **Target**, choose **NAT Gateway**, and then choose the ID of the NAT gateway that you created.

- Choose **Save routes**.

For more information about adding the route to the route table, see [Add and remove routes from a route table](#).

Note

Be sure that the routes to your NAT gateway are in an **active** status. If the NAT gateway is deleted and you haven't updated the routes, they're in a blackhole status. For more information, see [Work with NAT gateways](#).

Encrypting canary artifacts

CloudWatch Synthetics stores canary artifacts such as screenshots, HAR files, and reports in your Amazon S3 bucket. By default, these artifacts are encrypted at rest using an AWS managed key. For more information, see [Customer keys and AWS keys](#).

You can choose to use a different encryption option. CloudWatch Synthetics supports the following:

- **SSE-S3**– Server-side encryption (SSE) with an Amazon S3-managed key.
- **SSE-KMS**– Server-side encryption (SSE) with an AWS KMS customer managed key.

If you want to use the default encryption option with an AWS managed key, you don't need any additional permissions.

To use SSE-S3 encryption, you specify **SSE_S3** as the encryption mode when you create or update your canary. You do not need any additional permissions to use this encryption mode. For more information, see [Protecting data using server-side encryption with Amazon S3-managed encryption keys \(SSE-S3\)](#).

To use an AWS KMS customer managed key, you specify **SSE-KMS** as the encryption mode when you create or update your canary, and you also provide the Amazon Resource Name (ARN) of your key. You can also use a cross-account KMS key.

To use a customer managed key, you need the following settings:

- The IAM role for your canary must have permission to encrypt your artifacts using your key. If you are using visual monitoring, you must also give it permission to decrypt artifacts.

```
{
  "Version": "2012-10-17",
  "Statement": [{"Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "Your KMS key ARN"
  }
}
```

- Instead of adding permissions to your IAM role, you can add your IAM role to your key policy. If you use the same role for multiple canaries, you should consider this approach.

```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": "Your synthetics IAM role ARN"
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

- If you are using a cross-account KMS key, see [Allowing users in other accounts to use a KMS key](#).

Viewing encrypted canary artifacts when using a customer managed key

To view canary artifacts, update your customer managed key to give AWS KMS the decrypt permission to the user viewing the artifacts. Alternatively, add decrypt permissions to the user or IAM role that is viewing the artifacts.

The default AWS KMS policy enables IAM policies in the account to allow access to the KMS keys. If you are using a cross-account KMS key, see [Why are cross-account users getting Access Denied errors when they try to access Amazon S3 objects encrypted by a custom AWS KMS key?](#)

For more information about troubleshooting access denied issues because of a KMS key, see [Troubleshooting key access](#).

Updating artifact location and encryption when using visual monitoring

To perform visual monitoring, CloudWatch Synthetics compares your screenshots with baseline screenshots acquired in the run selected as the baseline. If you update your artifact location or encryption option, you must do one of the following:

- Ensure that your IAM role has sufficient permission for both the previous Amazon S3 location and the new Amazon S3 location for artifacts. Also ensure that it has permission for both the previous and new encryption methods and KMS keys.
- Create a new baseline by selecting the next canary run as a new baseline. If you use this option, you only need to ensure that your IAM role has sufficient permissions for the new artifact location and encryption option.

We recommend the second option of selecting the next run as the new baseline. This avoids having a dependency on an artifact location or encryption option that you're not using anymore for the canary.

For example, suppose that your canary uses artifact location A and KMS key K for uploading artifacts. If you update your canary to artifact location B and KMS key L, you can ensure that your IAM role has permissions to both of the artifact locations (A and B) and both of the KMS keys (K and L). Alternatively, you can select the next run as the new baseline and ensure that your canary IAM role has permissions to artifact location B and KMS key L.

Viewing canary statistics and details

You can view details about your canaries and see statistics about their runs.

To be able to see all the details about your canary run results, you must be logged on to an account that has sufficient permissions. For more information, see [Required roles and permissions for CloudWatch canaries](#).

To view canary statistics and details

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.

In the details about the canaries that you have created:

- **Status** visually shows how many of your canaries have passed their most recent runs.
 - **Groups** displays the groups you have created, and displays how many of them have failing or alarming canaries.
 - **Slowest performers** displays the group and the Region with the slowest-performing canaries. These are calculated by adding up the average duration of all canaries (across the time span selected) within a group or Region and dividing it by the number of canaries in the group or Region. If you choose the metric for Slowest group, the table is filtered to display only the slowest groups and their canaries. The table is sorted by Average Duration.
 - Near the bottom of the page is a table displaying all canaries. One column displays the alarms created for each canary. Only alarms that conform to the naming standard for canary alarms are displayed. This standard is `Synthetics-Alarm-canaryName-index`. Canary alarms that you create in the **Synthetics** section of the CloudWatch console automatically use this naming convention. If you create canary alarms in the **Alarms** section of the CloudWatch console or by using AWS CloudFormation, and you don't use this naming convention, the alarms work but they do not appear in this list.
3. To see more details about a single canary, choose the name of the canary in the **Canaries** table.

In the details about that canary:

- The **Availability** tab displays information about the recent runs of this canary.

Under **Canary runs**, you can choose one of the lines to see details about that run.

Under the graph, you can choose **Steps**, **Screenshot**, **Logs**, or **HAR file** to see these types of details. If the canary has active tracing enabled, you can also choose **Traces** to see tracing information from the canary's runs.

The logs for canary runs are stored in S3 buckets and in CloudWatch Logs.

Screenshots show how your customers view your webpages. You can use the HAR files (HTTP Archive files) to view detailed performance data about the webpages. You can analyze the list of web requests and catch performance issues such as time to load for an item. Log files show the record of interactions between the canary run and the webpage and can be used to identify details of errors.

If the canary uses the `syn-nodejs-2.0-beta` runtime or later, you can sort the HAR files by status code, request size, or duration.

The **Steps** tab displays a list of the canary's steps, each step's status, failure reason, URL after step execution, screenshots, and duration of step execution. For API canaries with HTTP steps, you can view steps and corresponding HTTP requests if you are using runtime `syn-nodejs-2.2` or later.

Choose the **HTTP Requests** tab to view the log of each HTTP request made by the canary. You can view request/response headers, response body, status code, error and performance timings (total duration, TCP connection time, TLS handshake time, first byte time, and content transfer time). All HTTP requests which use the HTTP/HTTPS module under the hood are captured here.

By default in API canaries, the request header, response header, request body, and response body are not included in the report for security reasons. If you choose to include them, the data is stored only in your S3 bucket. For information about how to include this data in the report, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#).

Response body content types of text, HTML and JSON are supported. Content types like text/HTML, text/plain, application/JSON and application/x-amz-json-1.0 are supported. Compressed responses are not supported.

- The **Monitoring** tab displays graphs of the CloudWatch metrics published by this canary. For more information about these metrics, see [CloudWatch metrics published by canaries](#).

Below the CloudWatch graphics published by the canary are graphs of Lambda metrics related to the canary's Lambda code.

- The **Configuration** tab displays configuration and schedule information about the canary.
- The **Groups** tab displays the groups that this canary is associated with, if any.
- The **Tags** tab displays the tags associated with the canary.

CloudWatch metrics published by canaries

Canaries publish the following metrics to CloudWatch in the `CloudWatchSynthetics` namespace. For more information about viewing CloudWatch metrics, see [View available metrics](#).

Metric	Description
SuccessPercent	<p>The percentage of the runs of this canary that succeed and find no failures.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Average</p> <p>Units: Percent</p>
Duration	<p>The duration in milliseconds of the canary run.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Average</p> <p>Units: Milliseconds</p>
Errors	<p>The number of times the canary failed to run its full script.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p>
2xx	<p>The number of network requests performed by the canary that returned OK responses, with response codes between 200 and 299.</p> <p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p>
4xx	<p>The number of network requests performed by the canary that returned Error responses, with response codes between 400 and 499.</p>

Metric	Description
	<p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p>
5xx	<p>The number of network requests performed by the canary that returned Fault responses, with response codes between 500 and 599.</p> <p>This metric is reported for UI canaries that use runtime version <code>syn-nodejs-2.0</code> or later, and is reported for API canaries that use runtime version <code>syn-nodejs-2.2</code> or later.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p>
Failed	<p>The number of canary runs that failed to execute. These failures are related to the canary itself.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p>
Failed requests	<p>The number of HTTP requests executed by the canary on the target website that failed with no response.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Sum</p> <p>Units: Count</p>

Metric	Description
VisualMonitoringSuccessPercent	<p>The percentage of visual comparisons that successfully matched the baseline screenshots during a canary run.</p> <p>Valid Dimensions: CanaryName</p> <p>Valid Statistic: Average</p> <p>Units: Percent</p>
VisualMonitoringTotalComparisons	<p>The total number of visual comparisons that happened during a canary run.</p> <p>Valid Dimensions: CanaryName</p> <p>Units: Count</p>

Note

Canaries that use either the `executeStep()` or `executeHttpStep()` methods from the Synthetics library also publish `SuccessPercent` and `Duration` metrics with the dimensions `CanaryName` and `StepName` for each step.

Edit or delete a canary

You can edit or delete an existing canary.

Edit canary

When you edit a canary, even if you don't change its schedule, the schedule is reset corresponding to when you edit the canary. For example, if you have a canary that runs every hour, and you edit that canary, the canary will run immediately after the edit is completed and then every hour after that.

To edit or update a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.
3. Select the button next to the canary name, and choose **Actions, Edit**.
4. (Optional) If this canary performs visual monitoring of screenshots and you want to set the next run of the canary as the baseline, select **Set next run as new baseline**.
5. (Optional) If this canary performs visual monitoring of screenshots and you want to remove a screenshot from visual monitoring or you want to designate parts of the screenshot to be ignored during visual comparisons, under **Visual Monitoring** choose **Edit Baseline**.

The screenshot appears, and you can do one of the following:

- To remove the screenshot from being used for visual monitoring, select **Remove screenshot from visual test baseline**.
 - To designate parts of the screenshot to be ignored during visual comparisons, click and drag to draw areas of the screen to ignore. Once you have done this for all the areas that you want to ignore during comparisons, choose **Save**.
6. Make any other changes to the canary that you'd like, and choose **Save**.

Delete canary

When you delete a canary, you can choose whether to also delete other resources used and created by the canary. When you delete a canary, you should also delete the following:

- Lambda functions and layers used by this canary. Their prefix is `cwsyn-MyCanaryName`.
- CloudWatch alarms created for this canary. These alarms have a name that starts with `Synthetics-Alarm-MyCanaryName`. For more information about deleting alarms, see [Edit or delete a CloudWatch alarm](#).
- Amazon S3 objects and buckets, such as the canary's results location and artifact location.
- IAM roles created for the canary. These have the name `role/service-role/CloudWatchSyntheticsRole-MyCanaryName`.
- Log groups in CloudWatch Logs created for the canary. These logs groups have the following names: `/aws/lambda/cwsyn-MyCanaryName-randomId`.

Before you delete a canary, you might want to view the canary details and make note of this information. That way, you can delete the correct resources after you delete the canary.

To delete a canary

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.
3. If the canary is currently in the RUNNING state, you must stop it. Only canaries in the STOPPED, READY(NOT_STARTED), or ERROR states can be deleted.

To stop the canary, select the button next to the canary name, and choose **Actions, Stop**.

4. Select the button next to the canary name, and choose **Actions, Delete**.
5. Choose whether to also delete the other resources created for and used by the canary. This includes the Lambda function and layers, and the canary's IAM role and IAM policy.

To delete the canary's IAM role and IAM policy, you must have sufficient permissions. For more information, see [AWS managed \(predefined\) policies for CloudWatch Synthetics](#).

6. Enter **Delete** into the box and choose **Delete**.
7. Delete the other resources used by and created for the canary, as listed earlier in this section.

Start, stop, delete, or update runtime for multiple canaries

You can stop, start, delete, or update the runtime of as many as five canaries with one action. If you update the runtime of a canary, it is updated to the latest runtime available for the language and framework that the canary uses.

If you select multiple canaries and only some of them are in a state that is valid for the action that you select, the action is performed only on the canaries where that action is valid. For example, if you select some canaries that are currently running and some that are not, and you select to start the canaries, the canaries that weren't already running will start, and the canaries that were already running are not affected.

If none of the canaries that you select are valid for an action, that action will not be available in the menu.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Synthetics Canaries**.
3. Select the check boxes next to the canaries that you want to stop, start, or delete.
4. Choose **Actions** and then choose either **Start, Stop, Delete, or Update Runtime**.

Monitoring canary events with Amazon EventBridge

Amazon EventBridge event rules can notify you when canaries change status or complete runs. EventBridge delivers a near-real-time stream of system events that describe changes in AWS resources. CloudWatch Synthetics sends these events to EventBridge on a *best effort* basis. Best effort delivery means that CloudWatch Synthetics attempts to send all events to EventBridge, but in some rare cases an event might not be delivered. EventBridge processes all received events at least once. Additionally, your event listeners might not receive the events in the order that the events occurred.

Note

Amazon EventBridge is an event bus service that you can use to connect your applications with data from a variety of sources. For more information, see [What is Amazon EventBridge?](#) in the *Amazon EventBridge User Guide*.

CloudWatch Synthetics emits an event when a canary changes state or completes a run. You can create an EventBridge rule that includes an event pattern to match all event types sent from CloudWatch Synthetics, or that matches only specific event types. When a canary triggers a rule, EventBridge invokes the target actions defined in the rule. This allows you to send notifications, capture event information, and take corrective action, in response to a canary state change or the completion of a canary run. For example, you can create rules for the following use cases:

- Investigating when a canary run fails
- Investigating when a canary has gone into the ERROR state
- Tracking a canary's life cycle
- Monitoring canary run success or failure as part of a workflow

Example events from CloudWatch Synthetics

This section lists example events from CloudWatch Synthetics. For more information about event format, see [Events and Event Patterns in EventBridge](#).

Canary status change

In this event type, the values of `current-state` and `previous-state` can be the following:

CREATING | READY | STARTING | RUNNING | UPDATING | STOPPING | STOPPED | ERROR

```

{
    "version": "0",
    "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",
    "detail-type": "Synthetics Canary Status Change",
    "source": "aws.synthetics",
    "account": "123456789012",
    "time": "2021-02-09T22:19:43Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "account-id": "123456789012",
        "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
        "canary-name": "events-bb-1",
        "current-state": "STOPPED",
        "previous-state": "UPDATING",
        "source-location": "NULL",
        "updated-on": 1612909161.767,
        "changed-config": {
            "executionArn": {
                "previous-value":
"arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                "current-value":
"arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
            },
            "vpcId": {
                "current-value": "NULL"
            },
            "testCodeLayerVersionArn": {
                "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                "current-value":
"arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
            }
        },
        "message": "Canary status has changed"
    }
}

```


Successful canary run completed

```
{
  "version": "0",
  "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
  "detail-type": "Synthetics Canary TestRun Successful",
  "source": "aws.synthetics",
  "account": "123456789012",
  "time": "2021-02-09T22:24:01Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "account-id": "123456789012",
    "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
    "canary-name": "events-bb-1",
    "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
    "artifact-location": "cw-syn-results-123456789012-us-east-1/canary/us-east-1/events-bb-1-ec3-28ddb266797/2021/02/09/22/23-41-200",
    "test-run-status": "PASSED",
    "state-reason": "null",
    "canary-run-timeline": {
      "started": 1612909421,
      "completed": 1612909441
    },
    "message": "Test run result is generated successfully"
  }
}
```

Failed canary run completed

```
{
  "version": "0",
  "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
  "detail-type": "Synthetics Canary TestRun Failure",
  "source": "aws.synthetics",
  "account": "123456789012",
  "time": "2021-02-09T22:24:27Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "account-id": "123456789012",
    "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
    "canary-name": "events-bb-1",
```

```

        "canary-run-id": "0df3823e-7e33-4da1-8194-
b04e4d4a2bf6",
        "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
        "test-run-status": "FAILED",
        "state-reason": "\"Error: net::ERR_NAME_NOT_RESOLVED
\""
        "canary-run-timeline": {
            "started": 1612909461,
            "completed": 1612909467
        },
        "message": "Test run result is generated successfully"
    }
}

```

It's possible that events might be duplicated or out of order. To determine the order of events, use the `time` property.

Prerequisites for creating EventBridge rules

Before you create an EventBridge rule for CloudWatch Synthetics, you should do the following:

- Familiarize yourself with events, rules, and targets in EventBridge.
- Create and configure the targets invoked by your EventBridge rules. Rules can invoke many types of targets, including:
 - Amazon SNS topics
 - AWS Lambda functions
 - Kinesis streams
 - Amazon SQS queues

For more information, see [What is Amazon EventBridge?](#) and [Getting started with Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

Create an EventBridge rule (CLI)

The steps in the following example create an EventBridge rule that publishes an Amazon SNS topic when the canary named `my-canary-name` in `us-east-1` completes a run or changes state.

1. Create the rule.

```
aws events put-rule \  
  --name TestRule \  
  --region us-east-1 \  
  --event-pattern "{\"source\": [\"aws.synthetic\"], \"detail\": {\"canary-name\":  
  [\"my-canary-name\"]}}"
```

Any properties you omit from the pattern are ignored.

2. Add the topic as a rule target.

- Replace *topic-arn* with the Amazon Resource Name (ARN) of your Amazon SNS topic.

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="topic-arn"
```

Note

To allow Amazon EventBridge to call your target topic, you must add a resource-based policy to your topic. For more information, see [Amazon SNS permissions](#) in the *Amazon EventBridge User Guide*.

For more information, see [Events and event patterns in EventBridge](#) in the *Amazon EventBridge User Guide*.

CloudWatch RUM

With CloudWatch RUM, you can perform real user monitoring to collect and view client-side data about your web application performance from actual user sessions in near real time. The data that you can visualize and analyze includes page load times, client-side errors, and user behavior. When you view this data, you can see it all aggregated together and also see breakdowns by the browsers and devices that your customers use.

You can use the collected data to quickly identify and debug client-side performance issues. CloudWatch RUM helps you visualize anomalies in your application performance and find relevant debugging data such as error messages, stack traces, and user sessions. You can also use RUM

to understand the range of end user impact including the number of users, geolocations, and browsers used.

End user data that you collect for CloudWatch RUM is retained for 30 days and then automatically deleted. If you want to keep the RUM events for a longer time, you can choose to have the app monitor send copies of the events to CloudWatch Logs in your account. Then, you can adjust the retention period for that log group.

To use RUM, you create an *app monitor* and provide some information. RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions, which is displayed in a pre-built dashboard. You can specify what percentage of user sessions to gather data from.

CloudWatch RUM is integrated with [Application Signals](#), which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see RUM client page requests in Application Signals, turn on X-Ray active tracing by [creating an app monitor](#), or [manually configuring the RUM web client](#). Your RUM clients are displayed on the [Service Map](#) connected to your services, and in the [Service detail](#) page of the services they call.

The RUM web client is open source. For more information, see [CloudWatch RUM web client](#).

Performance considerations

This section discusses the performance considerations of using CloudWatch RUM.

- **Load performance impact**— The CloudWatch RUM web client can be installed in your web application as a JavaScript module, or loaded into your web application asynchronously from a content delivery network (CDN). It does not block the application's load process. CloudWatch RUM is designed for there to be no perceptible impact to the application's load time.
- **Runtime impact**— The RUM web client performs processing to record and dispatch RUM data to the CloudWatch RUM service. Because events are infrequent and the amount of processing is small, CloudWatch RUM is designed for there to be no detectable impact to the application's performance.
- **Network impact**— The RUM web client periodically sends data to the CloudWatch RUM service. Data is dispatched at regular intervals while the application is running, and also immediately before the browser unloads the application. Data sent immediately before the browser unloads

the application are sent as beacons, which, are designed to have no detectable impact on the application's unload time.

RUM Pricing

With CloudWatch RUM, you incur charges for every RUM event that CloudWatch RUM receives. Each data item collected using the RUM web client is considered a RUM event. Examples of RUM events include a page view, a JavaScript error, and an HTTP error. You have options for which types of events are collected by each app monitor. You can activate or deactivate options to collect performance telemetry events, JavaScript errors, HTTP errors, and X-Ray traces. For more information about choosing these options, see [Step 2: Create an app monitor](#) and [Information collected by the CloudWatch RUM web client](#). For more information about pricing, see [Amazon CloudWatch Pricing](#).

Region availability

CloudWatch RUM is currently available in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Melbourne)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)

- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)

Topics

- [IAM policies to use CloudWatch RUM](#)
- [Set up an application to use CloudWatch RUM](#)
- [Configuring the CloudWatch RUM web client](#)
- [Regionalization](#)
- [Use page groups](#)
- [Specify custom metadata](#)
- [Send custom events](#)
- [Viewing the CloudWatch RUM dashboard](#)
- [CloudWatch metrics that you can collect with CloudWatch RUM](#)
- [Data protection and data privacy with CloudWatch RUM](#)
- [Information collected by the CloudWatch RUM web client](#)
- [Manage your applications that use CloudWatch RUM](#)
- [CloudWatch RUM quotas](#)
- [Troubleshooting CloudWatch RUM](#)

IAM policies to use CloudWatch RUM

To be able to fully manage CloudWatch RUM, you must be signed in as an IAM user or role that has the **AmazonCloudWatchRUMFullAccess** IAM policy. Additionally, you may need other policies or permissions:

- To create an app monitor that creates a new Amazon Cognito identity pool for authorization, you need to have the **Admin** IAM role or the **AdministratorAccess** IAM policy.
- To create an app monitor that sends data to CloudWatch Logs, you must be logged on to an IAM role or policy that has the following permission:

```
{
  "Effect": "Allow",
  "Action": [
    "logs:PutResourcePolicy"
  ],
  "Resource": [
    "*"
  ]
}
```

Other users who need to view CloudWatch RUM data but don't need to create CloudWatch RUM resources, can be granted the **AmazonCloudWatchRUMReadOnlyAccess** policy.

Set up an application to use CloudWatch RUM

Use the steps in these sections to set up your application to begin using CloudWatch RUM to collect performance data from real user sessions.

Topics

- [Step 1: Authorize your application to send data to AWS](#)
- [Step 2: Create an app monitor](#)
- [\(Optional\) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client](#)
- [Step 4: Insert the code snippet into your application](#)
- [Step 5: Test your app monitor setup by generating user events](#)

Step 1: Authorize your application to send data to AWS

To use CloudWatch RUM, your application must have authorization.

You have three options to set up authorization:

- Let CloudWatch RUM create a new Amazon Cognito identity pool for the application. This method requires the least effort to set up. It's the default option.

The identity pool will contain an unauthenticated identity. This allows the CloudWatch RUM web client to send data to CloudWatch RUM without authenticating the user of the application.

The Amazon Cognito identity pool has an attached IAM role. The Amazon Cognito unauthenticated identity allows the web client to assume the IAM role that is authorized to send data to CloudWatch RUM.

- Use an existing Amazon Cognito identity pool. In this case, you must also modify the IAM role that is attached to the identity pool. Use this option for identity pools that support unauthenticated users. You can use identity pools only from the same Region.
- Use authentication from an existing identity provider that you have already set up. In this case, you must get credentials from the identity provider and your application must forward these credentials to the RUM web client.

Use this option for identity pools that support only authenticated users.

The following sections include more details about these options.

CloudWatch RUM creates a new Amazon Cognito identity pool

This is the simplest option to set up, and if you choose this no further setup steps are required. You must have administrative permissions to use this option. For more information, see [IAM policies to use CloudWatch RUM](#).

With this option, CloudWatch RUM creates the following resources:

- A new Amazon Cognito identity pool
- An unauthenticated Amazon Cognito identity. This allows the RUM web client to assume an IAM role without authenticating the user of the application.
- The IAM role that the RUM web client will assume. The IAM policy attached to this role allows it to use the PutRumEvents API with the app monitor resource. In other words, it allows the RUM web client to send data to RUM.

The RUM web client uses the Amazon Cognito identity to obtain AWS credentials. The AWS credentials are associated with the IAM role. The IAM role is authorized to use PutRumEvents with the AppMonitor resource.

Amazon Cognito sends the necessary security token to enable your application to send data to CloudWatch RUM. The JavaScript code snippet that CloudWatch RUM generates includes the following lines to enable authentication.

```
{
  identityPoolId: [identity pool id], // e.g., 'us-west-2:EXAMPLE4a-66f6-4114-902a-
EXAMPLEbad7'
}
);
```

Use an existing Amazon Cognito identity pool

If you choose to use an existing Amazon Cognito identity pool, you specify the identity pool when you add the application to CloudWatch RUM. The pool must support enabling access to unauthenticated identities. You can use identity pools only from the same Region.

You also must add the following permissions to the IAM policy that is attached to the IAM role that is associated with this identity pool.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "rum:PutRumEvents",
      "Resource": "arn:aws:rum:[region]:[accountid]:appmonitor/[app monitor
name]"
    }
  ]
}
```

Amazon Cognito will then send the necessary security token to enable your application to access CloudWatch RUM.

Third-party provider

If you choose to use private authentication from a third-party provider, you must get credentials from the identity provider and forward them to AWS. The best way to do this is by using a *security token vendor*. You can use any security token vendor, including Amazon Cognito with AWS Security

Token Service. For more information about AWS STS, see [Welcome to the AWS Security Token Service API Reference](#).

If you want to use Amazon Cognito as the token vendor in this scenario, you can configure Amazon Cognito to work with an authentication provider. For more information, see [Getting Started with Amazon Cognito Identity Pools \(Federated Identities\)](#).

After you configure Amazon Cognito to work with your identity provider, you also need to do the following:

- Create an IAM role with the following permissions. Your application will use this role to access AWS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "rum:PutRumEvents",
      "Resource": "arn:aws:rum:[region]:[accountID]:appmonitor/[app monitor
name]"
    }
  ]
}
```

- Add the following to your application to have it pass the credentials from your provider to CloudWatch RUM. Insert the line so that it runs after a user has signed in to your application and the application has received the credentials to use to access AWS.

```
cwr('setAwsCredentials', { /* Credentials or CredentialProvider */});
```

For more information about credential providers in the AWS JavaScript SDK, see [Setting credentials in a web browser](#) in the v3 developer guide for SDK for JavaScript, [Setting credentials in a web browser](#) in the v2 developer guide for SDK for JavaScript, , and [@aws-sdk/credential-providers](#).

You can also use the SDK for the CloudWatch RUM web client to configure the web client authentication methods. For more information about the web client SDK, see [CloudWatch RUM web client SDK](#).

Step 2: Create an app monitor

To start using CloudWatch RUM with your application, you create an *app monitor*. When the app monitor is created, RUM generates a JavaScript snippet for you to paste into your application. The snippet pulls in the RUM web client code. The RUM web client captures data from a percentage of your application's user sessions and sends it to RUM.

To create an app monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **Add app monitor**.
4. Enter the information and settings for your application:
 - For **App monitor name**, enter a name to be used to identify this app monitor within the CloudWatch RUM console.
 - For **Application domain**, enter the top-level domain name where your application has administrative authority. This must be in a URL domain format.

Choose **Include sub domains** to have the app monitor also collect data from all subdomains under the top-level domain.
5. For **Configure RUM data collection**, specify whether you want the app monitor to collect each of the following:
 - **Performance telemetry** – Collects information about page load and resource load times
 - **JavaScript errors** – Collects information about unhandled JavaScript errors raised by your application
 - **HTTP errors** – Collects information about HTTP errors thrown by your application

Selecting these options provides more information about your application, but also generates more CloudWatch RUM events and thus incurs more charges.

If you don't select any of these, the app monitor still collects session start events and page IDs so that you can see how many users are using your application, including breakdowns by operating system type and version, browser type and version, device type, and location.

6. Select **Check this option to allow the CloudWatch RUM Web Client to set cookies** if you want to be able to collect user IDs and session IDs from sampled user sessions. The user IDs are

randomly generated by RUM. For more information, see [CloudWatch RUM web client cookies \(or similar technologies\)](#).

7. For **Session samples**, enter the percentage of user sessions that will be used to gather RUM data. The default is 100%. Reducing this number gives you less data, but reduces your charges. For more information about RUM pricing, see [RUM pricing](#).
8. End user data that you collect for CloudWatch RUM is retained for 30 days and then deleted. If you want to keep copies of RUM events in CloudWatch Logs and configure how long to retain these copies, choose **Check this option to store your application telemetry data in your CloudWatch Logs account** under **Data storage**. By default, the CloudWatch Logs log group retains the data for 30 days. You can adjust the retention period in the CloudWatch Logs console.
9. For **Authorization**, specify whether to use a new or existing Amazon Cognito identity pool or use a different identity provider. Creating a new identity pool is the simplest option that requires no other setup steps. For more information, see [Step 1: Authorize your application to send data to AWS](#).

Creating a new Amazon Cognito identity pool requires administrative permissions. For more information, see [IAM policies to use CloudWatch RUM](#).

10. (Optional) By default, when you add the RUM code snippet to your application, the web client injects the JavaScript tag to monitor usage into the HTML code of all pages of your application. To change this, choose **Configure pages** and then choose either **Include only these pages** or **Exclude these pages**. Then, specify the pages to include or exclude. To specify a page to include or exclude, enter its complete URLs. To specify additional pages, choose **Add URL**.
11. To enable AWS X-Ray tracing of the user sessions that are sampled by the app monitor, choose **Active tracing** and select **Trace my service with AWS X-Ray**.

If you select this, XMLHttpRequest and fetch requests made during user sessions sampled by the app monitor are traced. You can then see traces and segments from these user sessions in the RUM dashboard, and the X-Ray trace map and trace details pages. These user sessions will also show up as client pages in [Application Signals](#) after you have enabled it for your application.

By making additional configuration changes to the CloudWatch RUM web client, you can add an X-Ray trace header to HTTP requests to enable end-to-end tracing of user sessions through

to downstream AWS managed services. For more information, see [Enabling X-Ray end-to-end tracing](#).

12. (Optional) To add tags to the app monitor, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

13. Choose **Add app monitor**.

14. In the **Sample code** section, you can copy the code snippet to use to add to your application. We recommend that you choose **JavaScript** or **TypeScript** and use NPM to install the CloudWatch RUM web client as a JavaScript module.

Alternatively, you can choose **HTML** to use a content delivery network (CDN) to install the CloudWatch RUM web client. The disadvantage of using a CDN is that the web client is often blocked by ad blockers.

15. Choose **Copy** or **Download**, and then choose **Done**.

(Optional) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client

You can modify the code snippet before inserting it into your application, to activate or deactivate several options. For more information, see the [CloudWatch RUM web client documentation](#).

There are three configuration options that you should definitely be aware of, as discussed in these sections.

Preventing the collection of resource URLs that might contain personal information

By default, the CloudWatch RUM web client is configured to record the URLs of resources downloaded by the application. These resources include HTML files, images, CSS files, JavaScript files, and so on. For some applications, URLs may contain personally identifiable information (PII).

If this is the case for your application, we strongly recommend that you disable the collection of resource URLs by setting `recordResourceUrl: false` in the code snippet configuration, before inserting it into your application.

Manually recording page views

By default, the web client records page views when the page first loads and when the browser's history API is called. The default page ID is `window.location.pathname`. However, in some cases you might want to override this behavior and instrument the application to record page views programmatically. Doing so gives you control over the page ID and when it is recorded. For example, consider a web application that has a URI with a variable identifier, such as `/entity/123` or `/entity/456`. By default, CloudWatch RUM generates a page view event for each URI with a distinct page ID matching the pathname, but you might want to group them by the same page ID instead. To accomplish this, disable the web client's page view automation by using the `disableAutoPageView` configuration, and use the `recordPageView` command to set the desired page ID. For more information, see [Application-specific Configurations](#) on GitHub.

Embedded script example:

```
cwr('recordPageView', { pageId: 'entityPageId' });
```

JavaScript module example:

```
awsRum.recordPageView({ pageId: 'entityPageId' });
```

Enabling X-Ray end-to-end tracing

When you create the app monitor, selecting **Trace my service with AWS X-Ray** enables the tracing of XMLHttpRequest and fetch requests made during user sessions that are sampled by the app monitor. You can then see traces from these HTTP requests in the CloudWatch RUM dashboard, and the X-Ray Trace Map and Trace details pages.

By default, these client-side traces are not connected to downstream server-side traces. To connect client-side traces to server-side traces and enable end-to-end tracing, set the `addXRayTraceIdHeader` option to `true` in the web client. This causes the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests.

The following code block shows an example of adding client-side traces. Some configuration options are omitted from this sample for readability.

```
<script>  
  (function(n,i,v,r,s,c,u,x,z){...})(
```

```
'cwr',
'000000000-0000-0000-0000-000000000000',
'1.0.0',
'us-west-2',
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
{
  enableXRay: true,
  telemetries: [
    'errors',
    'performance',
    [ 'http', { addXRayTraceIdHeader: true } ]
  ]
}
);
</script>
```

Warning

Configuring the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests can cause cross-origin resource sharing (CORS) to fail or invalidate the request's signature if the request is signed with SigV4. For more information, see the [CloudWatch RUM web client documentation](#). We strongly recommend that you test your application before adding a client-side X-Ray trace header in a production environment.

For more information, see the [CloudWatch RUM web client documentation](#)

Step 4: Insert the code snippet into your application

Next, you insert the code snippet that you created in the previous section into your application.

Warning

The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. Before you insert the code snippet, see [Filtering by metadata attributes in the console](#).

If you don't have the code snippet that was previously generated, you can find it by following the directions in [How do I find a code snippet that I've already generated?](#)

To insert the CloudWatch RUM code snippet into your application

1. Insert the code snippet that you copied or downloaded in the previous section inside the `<head>` element of your application. Insert it before the `<body>` element or any other `<script>` tags.

The following is an example of a generated code snippet:

```
<script>
(function (n, i, v, r, s, c, x, z) {
  x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
  window[n] = function (c, p) {
    x.q.push({c: c, p: p});
  };
  z = document.createElement('script');
  z.async = true;
  z.src = s;
  document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
  '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
  '1.0.0',
  'us-east-2',
  'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
  {
    sessionSampleRate: 1,
    identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cfd21443",
    endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
    telemetries: ["performance", "errors", "http"],
    allowCookies: true,
    enableXRay: false
  });
</script>
```

2. If your application is a multipage web application, you must repeat step 1 for each HTML page that you want included in the data collection.

Step 5: Test your app monitor setup by generating user events

After you have inserted the code snippet and your updated application is running, you can test it by manually generating user events. To test this, we recommend that you do the following. This testing incurs standard CloudWatch RUM charges.

- Navigate between pages in your web application.
- Create multiple user sessions, using different browsers and devices.
- Make requests.
- Cause JavaScript errors.

After you have generated some events, view them in the CloudWatch RUM dashboard. For more information, see [Viewing the CloudWatch RUM dashboard](#).

Data from user sessions might take up to 15 minutes to appear in the dashboard.

If you don't see data 15 minutes after you generated events in the application, see [Troubleshooting CloudWatch RUM](#).

Configuring the CloudWatch RUM web client

Your applications can use one of the code snippets generated by CloudWatch RUM to install the CloudWatch RUM web client. The generated snippets support two installation methods: as a JavaScript module through NPM, or from a content delivery network (CDN). For best performance, we recommend using the NPM installation method. For more information about using this method, see [Installing as a JavaScript Module](#).

If you use the CDN installation option, ad blockers might block the default CDN provided by CloudWatch RUM. This disables application monitoring for users who have ad blockers installed. Because of this, we recommend that you use the default CDN only for initial onboarding with CloudWatch RUM. For more information about the ways to mitigate this issue, see [Instrument the application](#).

The code snippet sits in the <head> tag of an HTML file and installs the web client by downloading the web client, and then configuring the web client for the application it is monitoring. The snippet is a self-executing function which looks similar to the following. In this example, the body of the snippet's function has been omitted for readability.

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(
  'cwI',
  '00000000-0000-0000-0000-000000000000',
  '1.0.0',
  'us-west-2',
  'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwI.js',
```

```
{ /* Configuration Options Here */ }  
);  
<script>
```

Arguments

The code snippet accepts six arguments:

- A namespace for running commands on the web client, such as 'cwr'
- The ID of the app monitor, such as '00000000-0000-0000-0000-000000000000'
- The application version, such as '1.0.0'
- The AWS Region of the app monitor, such as 'us-west-2'
- The URL of the web client, such as 'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'
- Application-specific configuration options. For more information, see the following section.

Ignoring errors

The CloudWatch RUM web client listens to all types of errors that happen in your applications. If your application emits JavaScript errors that you do not want to view in the CloudWatch RUM dashboard, you can configure the CloudWatch RUM web client to filter out these errors so that you see only the relevant error events on the CloudWatch RUM dashboard. For example, you might choose not to view some JavaScript errors in the dashboard because you have already identified a fix for them and the volume of these errors is masking other errors. You might also want to ignore errors that you can't fix because they are owned by a library owned by a third party.

For more information about how to instrument the web client to filter out specific JavaScript errors, see the example in [Errors](#) in the web client Github documentation.

Configuration options

For information about the configuration options available for the CloudWatch RUM web client, see the [CloudWatch RUM web client documentation](#)

Regionalization

This section illustrates strategies for using CloudWatch RUM with applications in different Regions.

My web application is deployed in multiple AWS Regions

If your web application is deployed in multiple AWS Regions, you have three options:

- Deploy one app monitor in one Region, in one account, serving all Regions.
- Deploy separate app monitors for each Region, in unique accounts.
- Deploy separate app monitors for each Region, all in one account.

The advantage of using one app monitor is that all data will be centralized into one visualization, and all logs are written to the same log group in CloudWatch Logs. With a single app monitor there is a small amount of extra latency for requests, and a single point of failure.

Using multiple app monitors removes the single point of failure, but prevents all data from being combined into one visualization.

CloudWatch RUM hasn't launched in some Regions that my application is deployed in

CloudWatch RUM is launched into many Regions and has wide geographical coverage. By setting up CloudWatch RUM in the Regions where it is available, you can get the benefits. End users can be anywhere and still have their sessions included if you have set up an app monitor in the Region that they are connecting to.

However, CloudWatch RUM is not yet launched in AWS GovCloud (US-East), AWS GovCloud (US-West), or any Regions in China. You are not able to send data to CloudWatch RUM from these Regions.

Use page groups

Use page groups to associate different pages in your application with each other so that you can see aggregated analytics for groups of pages. For example, you might want to see the aggregated page load times of all of your landing pages.

You put pages into page groups by adding one or more tags to page view events in the CloudWatch RUM web client. The following examples put the `/home` page into the page group named `en` and the page group named `landing`.

Embedded script example

```
cwr('recordPageView', { pageId: '/home', pageTags: ['en', 'landing']});
```

JavaScript module example

```
awsRum.recordPageView({ pageId: '/home', pageTags: ['en', 'landing']});
```

Note

Page groups are intended to facilitate aggregating analytics across different pages. For information about how to define and manipulate pageIds for your application, see the **Manually recording page views** section in [\(Optional\) Step 3: Manually modify the code snippet to configure the CloudWatch RUM web client](#).

Specify custom metadata

CloudWatch RUM attaches additional data to each event as metadata. Event metadata consists of attributes in the form of key-value pairs. You can use these attributes to search or filter events in the CloudWatch RUM console. By default, CloudWatch RUM creates some metadata for you. For more information about the default metadata, see [RUM event metadata](#).

You can also use the CloudWatch RUM web client to add custom metadata to CloudWatch RUM events. The custom metadata can include session attributes and page attributes.

To add custom metadata, you must use version 1.10.0 or later of the CloudWatch RUM web client.

Requirements and syntax

Each event can include as many as 10 custom attributes in the metadata. The syntax requirements for custom attributes are as follows:

- **Keys**
 - Maximum of 128 characters
 - Can include alphanumeric characters, colons (:), and underscores (_)
 - Can't begin with aws :.
 - Can't consist entirely of any of the reserved keywords listed in the following section. Can use those keywords as part of a longer key name.
- **Values**
 - Maximum of 256 characters
 - Must be strings, numbers, or Boolean values

Reserved keywords

You can't use the following reserved keywords as complete key names. You can use the following keywords as part of a longer key name, such as `applicationVersion`.

- `browserLanguage`
- `browserName`
- `browserVersion`
- `countryCode`
- `deviceType`
- `domain`
- `interaction`
- `osName`
- `osVersion`
- `pageId`
- `pageTags`
- `pageTitle`
- `pageUrl`
- `parentPageId`
- `platformType`
- `referrerUrl`
- `subdivisionCode`
- `title`
- `url`
- `version`

Note

CloudWatch RUM removes custom attributes from RUM events if an attribute includes a key or value that is not valid, or if the limit of 10 custom attributes per event has already been reached.

Add session attributes

If you configure custom session attributes, they are added to all events in a session. You configure session attributes either during CloudWatch RUM web client initialization or at runtime by using the `addSessionAttributes` command.

For example, you can add your application's version as a session attribute. Then, in the CloudWatch RUM console, you can filter errors by version to find whether an increased error rate is associated with a particular version of your application.

Adding a session attribute at initialization, NPM example

The code section in bold adds the session attribute.

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';

try {
  const config: AwsRumConfig = {
    allowCookies: true,
    endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
    guestRoleArn: "arn:aws:iam::000000000000:role/RUM-Monitor-us-west-2-000000000000-00xx-Unauth",
    identityPoolId: "us-west-2:00000000-0000-0000-0000-000000000000",
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance'],
    sessionAttributes: {
      applicationVersion: "1.3.8"
    }
  };

  const APPLICATION_ID: string = '00000000-0000-0000-0000-000000000000';
  const APPLICATION_VERSION: string = '1.0.0';
  const APPLICATION_REGION: string = 'us-west-2';

  const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    config
  );
} catch (error) {
  // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

Adding a session attribute at runtime, NPM example

```
awsRum.addSessionAttributes({
  applicationVersion: "1.3.8"
})
```

Adding a session attribute at initialization, embedded script example

The code section in bold adds the session attribute.

```
<script>
  (function(n,i,v,r,s,c,u,x,z){...})(
    'cwr',
    '00000000-0000-0000-0000-000000000000',
    '1.0.0',
    'us-west-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
      sessionSampleRate:1,
      guestRoleArn:'arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth',
      identityPoolId:'us-west-2:00000000-0000-0000-0000-000000000000',
      endpoint:'https://dataplane.rum.us-west-2.amazonaws.com',
      telemetries:['errors','http','performance'],
      allowCookies:true,
      sessionAttributes: {
        applicationVersion: "1.3.8"
      }
    }
  );
</script>
```

Adding a session attribute at runtime, embedded script example

```
<script>
  function addSessionAttribute() {
    cwr('addSessionAttributes', {
      applicationVersion: "1.3.8"
    })
  }
</script>
```

Add page attributes

If you configure custom page attributes, they are added to all events on the current page. You configure page attributes either during CloudWatch RUM web client initialization or at runtime by using the `recordPageView` command.

For example, you can add your page template as a page attribute. Then, in the CloudWatch RUM console, you can filter errors by page templates to find whether an increased error rate is associated with a particular page template of your application.

Adding a page attribute at initialization, NPM example

The code section in bold adds the page attribute.

```
const awsRum: AwsRum = new AwsRum(  
  APPLICATION_ID,  
  APPLICATION_VERSION,  
  APPLICATION_REGION,  
  { disableAutoPageView: true // optional }  
);  
awsRum.recordPageView({  
  pageId: '/home',  
  pageAttributes: {  
    template: 'artStudio'  
  }  
});  
const credentialProvider = new CustomCredentialProvider();  
if(awsCreds) awsRum.setAwsCredentials(credentialProvider);
```

Adding a page attribute at runtime, NPM example

```
awsRum.recordPageView({  
  pageId: '/home',  
  pageAttributes: {  
    template: 'artStudio'  
  }  
});
```

Adding a page attribute at initialization, embedded script example

The code section in bold adds the page attribute.


```
<script>
  (function(n,i,v,r,s,c,u,x,z){...})(
    'cwr',
    '00000000-0000-0000-0000-000000000000',
    '1.0.0',
    'us-west-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
      disableAutoPageView: true //optional
    }
  );
  cwr('recordPageView', {
    pageId: '/home',
    pageAttributes: {
      template: 'artStudio'
    }
  });
  const awsCreds = localStorage.getItem('customAwsCreds');
  if(awsCreds) cwr('setAwsCredentials', awsCreds)
</script>
```

Adding a page attribute at runtime, embedded script example

```
<script>
  function recordPageView() {
    cwr('recordPageView', {
      pageId: '/home',
      pageAttributes: {
        template: 'artStudio'
      }
    });
  }
</script>
```

Filtering by metadata attributes in the console

To filter the visualizations in the CloudWatch RUM console with any built-in or custom metadata attribute, use the search bar. In the search bar, you can specify as many as 20 filter terms in the form of **key=value** to apply to the visualizations. For example, to filter data for only the Chrome browser, you could add the filter term **browserName=Chrome**.

By default, the CloudWatch RUM console retrieves the 100 most common attributes keys and values to display in the dropdown in the search bar. To add more metadata attributes as filter terms, enter the complete attribute key and value into the search bar.

A filter can include as many as 20 filter terms, and you can save up to 20 filters per app monitor. When you save a filter, it is saved in the **Saved filters** dropdown. You can also delete a saved filter.

Send custom events

CloudWatch RUM records and ingests the events listed in [Information collected by the CloudWatch RUM web client](#). If you use version 1.12.0 or later of the CloudWatch RUM web client, you can define, record, and send additional custom events. You define the event type name and the data to send for each event type that you define. Each custom event payload can be up to 6 KB.

Custom events are ingested only if the app monitor has custom events enabled. To update the configuration settings of your app monitor, use the CloudWatch RUM console or the [UpdateAppMonitor](#) API.

After you enable custom events, and then define and send custom events, you can search for them. To search for them, use the **Events** tab in the CloudWatch RUM console. Search by using the event type.

Requirements and syntax

Custom events consist of an event type and event details. The requirements for these are as follows:

- **Event type**
 - This can be either the **type** or **name** of your event. For example, the CloudWatch RUM built-in event type called **JsError** has an event type of `com.amazon.rum.js_error_event`.
 - Must be between 1 and 256 characters.
 - Can be a combination of alphanumeric characters, underscores, hyphens, and periods.
- **Event details**
 - Contains the actual data that you want to record in CloudWatch RUM.
 - Must be an object that consists of fields and values.

Examples of recording custom events

There are two ways to record custom events in the CloudWatch RUM web client.

- Use the CloudWatch RUM web client's `recordEvent` API.
- Use a customized plugin.

Send a custom event using the `recordEvent` API, NPM example

```
awsRum.recordEvent('my_custom_event', {
  location: 'IAD',
  current_url: 'amazonaws.com',
  user_interaction: {
    interaction_1 : "click",
    interaction_2 : "scroll"
  },
  visit_count:10
})
```

Send a custom event using the `recordEvent` API, embedded script example

```
cwr('recordEvent', {
  type: 'my_custom_event',
  data: {
    location: 'IAD',
    current_url: 'amazonaws.com',
    user_interaction: {
      interaction_1 : "click",
      interaction_2 : "scroll"
    },
    visit_count:10
  }
})
```

Example of sending a custom event using a customized plugin

```
// Example of a plugin that listens to a scroll event, and
// records a 'custom_scroll_event' that contains the timestamp of the event.
class MyCustomPlugin implements Plugin {
  // Initialize MyCustomPlugin.
```

```
constructor() {
  this.enabled;
  this.context;
  this.id = 'custom_event_plugin';
}
// Load MyCustomPlugin.
load(context) {
  this.context = context;
  this.enable();
}
// Turn on MyCustomPlugin.
enable() {
  this.enabled = true;
  this.addEventHandler();
}
// Turn off MyCustomPlugin.
disable() {
  this.enabled = false;
  this.removeEventHandler();
}
// Return MyCustomPlugin Id.
getPluginId() {
  return this.id;
}
// Record custom event.
record(data) {
  this.context.record('custom_scroll_event', data);
}
// EventHandler.
private eventHandler = (scrollEvent: Event) => {
  this.record({timestamp: Date.now()})
}
// Attach an eventHandler to scroll event.
private addEventHandler(): void {
  window.addEventListener('scroll', this.eventHandler);
}
// Detach eventHandler from scroll event.
private removeEventHandler(): void {
  window.removeEventListener('scroll', this.eventHandler);
}
}
```

Viewing the CloudWatch RUM dashboard

CloudWatch RUM helps you collect data from user sessions about your application's performance, including page load times, Apdex score, browsers and devices used, geolocation of user sessions, and sessions with errors. All of this information is displayed in a dashboard.

To view the RUM dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.

The **Overview** tab displays information collected by one of the app monitors that you have created.

The top row of panes displays the following information for this app monitor:

- Number of page loads
- Average page load speed
- Apdex score
- Status of any alarms associated with the app monitor

The application performance index (Apdex) score indicates end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application. For more information about Apdex scores, see [How CloudWatch RUM sets Apdex scores](#).

Several of these panes include links that you can use to further examine the data. Choosing any of these links displays a detailed view with **Performance, Errors, HTTP requests, Sessions, Events Browsers & Devices**, and **User Journey** tabs at the top of the display.

3. To focus further, choose the **List view** tab and then choose the name of the app monitor that you want to focus on. This displays the following tabs for the chosen app monitor.
 - The **Performance** tab displays page performance information including load times, session information, request information, web vitals, and page loads over time. This view includes controls to toggle the view between focusing on **Page loads, Requests**, and **Location**.
 - The **Errors** tab displays Javascript error information including the error message most frequently seen by users and the devices and browsers with the most errors. This view

includes a histogram of the errors and a list view of errors. You can filter the list of errors by user and event details. Choose an error message to see more details.

- The **HTTP requests** tab displays HTTP request information including the request URL with most errors and the devices and browsers with the most errors. This tab includes a histogram of the requests, a list view of requests, and a list view of network errors. You can filter the lists by user and event details. Choose a response code or an error message to see more details about the request or network error, respectively.
- The **Sessions** tab displays session metrics. This tab includes a histogram of session start events and a list view of sessions. You can filter the list of sessions by event type, user details, and event details. Choose a **sessionId** to see more details about a session.
- The **Events** tab displays a histogram of RUM events and a list view of the events. You can filter the list of events by event type, user details, and event details. Choose a RUM event to see the raw event.
- The **Browsers & Devices** tab displays information such as the performance and usage of different browsers and devices to access your application. This view includes controls to toggle the view between focusing on **Browsers** and **Devices**.

If you narrow the scope to a single browser, you see the data broken down by browser version.

- The **User Journey** tab displays the paths that your customers use to navigate your application. You can see where your customers enter your application and what page they exit your application from. You can also see the paths that they take and the percentage of customers that follow those paths. You can pause on a node to get more details about that page. You can choose a single path to highlight the connections for easier viewing.
4. (Optional) On any of the first six tabs, you can choose the **Pages** button and select a page or page group from the list. This narrows down the displayed data to a single page or group of pages of your application. You can also mark pages and page groups in the list as favorites.

How CloudWatch RUM sets Apdex scores

Apdex (Application Performance Index) is an open standard that defines a method to report, benchmark, and rate application response time. An Apdex score helps you understand and identify the impact on application performance over time.

The Apdex score indicates the end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application.

Each individual Apdex score falls into one of three thresholds. Based on the Apdex threshold and actual application response time, there are three kinds of performance, as follows:

- **Satisfied**– The actual application response time is less than or equal to the Apdex threshold. For CloudWatch RUM, this threshold is 2000 ms or less.
- **Tolerable**– The actual application response time is greater than the Apdex threshold, but less than or equal to four times the Apdex threshold. For CloudWatch RUM, this range is 2000–8000 ms.
- **Frustrating**– The actual application response time is greater than four times the Apdex threshold. For CloudWatch RUM, this range is over 8000 ms.

The total 0-1 Apdex score is calculated using the following formula:

$$(\text{positive scores} + \text{tolerable scores}/2)/\text{total scores} * 100$$

CloudWatch metrics that you can collect with CloudWatch RUM

The table in this section lists the metrics that you automatically collect with CloudWatch RUM. You can see these metrics in the CloudWatch console. For more information, see [View available metrics](#).

You can also optionally send extended metrics to CloudWatch or CloudWatch Evidently. For more information, see [Extended metrics](#).

These metrics are published in the metric namespace named AWS/RUM. All of the following metrics are published with an `application_name` dimension. The value of this dimension is the name of the app monitor. Some metrics are also published with additional dimensions, as listed in the table.

Metric	Unit	Description
HttpStatusCodeCount	Count	The count of HTTP responses in the application, by their response status code.

Metric	Unit	Description
		<p>Additional dimensions:</p> <ul style="list-style-type: none"> • <code>event_details.response.status</code> is the response status code, such as 200, 400, 404, and so on. • <code>event_type</code> The type of event. Currently, the only possible value for this dimension is <code>http</code>.
Http4xxCount	Count	<p>The count of HTTP responses in the application, with 4xx response status code.</p> <p>These are calculated based on <code>http_event</code> RUM events that result in 4xx codes.</p>
Http5xxCount	Count	<p>The count of HTTP responses in the application, with 5xx response status code.</p> <p>These are calculated based on <code>http_event</code> RUM events that result in 5xx codes.</p>

Metric	Unit	Description
JsErrorCount	Count	The count of JavaScript error events ingested.
NavigationFrustratedCount	Count	The count of navigation events with a duration higher than the frustrating threshold, which is 8000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.
NavigationSatisfiedCount	Count	The count of navigation events with a duration that is less than the Apdex objective, which is 2000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.

Metric	Unit	Description
NavigationToleratedCount	Count	The count of navigation events with a duration between 2000ms and 8000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.
PageViewCount	Count	The count of page view events ingested by the app monitor. This is calculated by counting the page_view_event RUM events.

Metric	Unit	Description
PerformanceResourceDuration	Milliseconds	<p>The duration of a resource event.</p> <p>Additional dimension s:</p> <ul style="list-style-type: none">• <code>event_details.file_type</code> is the file type of the resource event, such as a stylesheet, document, image, script, or font.• <code>event_type</code> The type of event. Currently, the only possible value for this dimension is <code>resource</code>.
PerformanceNavigationDuration	Milliseconds	The duration of a navigation event.
RumEventPayloadSize	Bytes	The size of every event ingested by CloudWatch RUM. You can also use the <code>SampleCount</code> statistic for this metric to monitor the number of events that an app monitor is ingesting.

Metric	Unit	Description
SessionCount	Count	The count of session start events ingested by the app monitor. In other words, the number of new sessions started.
WebVitalsCumulativeLayoutShift	None	Tracks the value of the cumulative layout shift events.
WebVitalsFirstInputDelay	Milliseconds	Tracks the value of the first input delay events.
WebVitalsLargestContentfulPaint	Milliseconds	Tracks the value of the largest contentful paint events.

Custom metrics and extended metrics that you can send to CloudWatch and CloudWatch Evidently

By default, RUM app monitors send metrics to CloudWatch. These default metrics and dimensions are listed in [CloudWatch metrics that you can collect with CloudWatch RUM](#).

You can also set up an app monitor to export metric. The app monitor can send extended metrics, custom metrics, or both. It can send them to CloudWatch or to CloudWatch Evidently, or to both.

- **Custom metrics**– Custom metrics are metrics that you define. With custom metrics, you can use any metric name and namespace. To derive the metrics, you can use any custom events, built-in events, custom attributes, or default attributes.

You can send custom metrics to both CloudWatch and CloudWatch Evidently.

- **Extended metrics**– Lets you send the default CloudWatch RUM metrics to CloudWatch Evidently to be used in Evidently experiments. You can also send any of the default CloudWatch RUM

metrics to CloudWatch with additional dimensions. This way, these metrics can give you a more fine-grained view.

Topics

- [Custom metrics](#)
- [Extended metrics](#)

Custom metrics

To send custom metrics, you must use the AWS APIs or AWS CLI instead of the console. For more information about using the AWS APIs, see [PutRumMetricsDestination](#) and [BatchCreateRumMetricDefinitions](#).

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each custom metric or extended metric that you send to each destination, each combination of dimension name and dimension value counts toward this limit. This also counts as a CloudWatch custom metric for pricing.

The following example shows how to create a custom metric derived from a custom event. Here is the example custom event that is used:

```
cwr('recordEvent', {
  type: 'my_custom_event',
  data: {
    location: 'IAD',
    current_url: 'amazonaws.com',
    user_interaction: {
      interaction_1 : "click",
      interaction_2 : "scroll"
    },
    visit_count:10
  }
})
```

Given this custom event, you can create a custom metric that counts the number of visits to the `amazonaws.com` URL from Chrome browsers. The following definition creates a metric named `AmazonVisitsCount` in your account, in the `RUM/CustomMetrics/PageVisits` namespace.

```
{
```

```

"AppMonitorName":"customer-appMonitor-name",
"Destination":"CloudWatch",
"MetricDefinitions":[
  {
    "Name":"AmazonVisitsCount",
    "Namespace":"PageVisit",
    "ValueKey":"event_details.visit_count",
    "UnitLabel":"Count",
    "DimensionKeys":{
      "event_details.current_url": "URL"
    },
    "EventPattern":"{\"metadata\":{\"browserName\":[\"Chrome\"]},\"event_type\":[\"my_custom_event\"],\"event_details\":{\"current_url\":[\"amazonaws.com\"]}}"
  }
]
}

```

Extended metrics

If you set up extended metrics, you can do one or both of the following:

- Send default CloudWatch RUM metrics to CloudWatch Evidently to be used in Evidently experiments. Only the **PerformanceNavigationDuration**, **PerformanceResourceDuration**, **WebVitalsCumulativeLayoutShift**, **WebVitalsFirstInputDelay**, and **WebVitalsLargestContentfulPaint** metrics can be sent to Evidently.
- Send any of the default CloudWatch RUM metrics to CloudWatch with additional dimensions so that the metrics give you a more fine-grained view. For example, you can see metrics specific to a certain browser that's used by your users, or metrics for users in a specific geolocation.

For more information about the default CloudWatch RUM metrics, see [CloudWatch metrics that you can collect with CloudWatch RUM](#).

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each extended or custom metric that you send to each destination, each combination of dimension name and dimension value counts as an extended metric for this limit. This also counts as a CloudWatch custom metric for pricing.

When you send extended metrics to CloudWatch, you can use the CloudWatch RUM console to create CloudWatch alarms on them.

Extended metrics are charged as CloudWatch custom metrics. For more information, see [Amazon CloudWatch Pricing](#).

The following dimensions are supported for extended metrics for all the metric names that app monitors can send. These metric names are listed in [CloudWatch metrics that you can collect with CloudWatch RUM](#).

- **BrowserName**

Example dimension values: Chrome, Firefox, Chrome Headless

- **CountryCode** This uses the ISO-3166 format, with two-letter codes.

Example dimension values: US, JP, DE

- **DeviceType**

Example dimension values: desktop, mobile, tablet, embedded

- **FileType**

Example dimension values: Image, Stylesheet

- **OSName**

Example dimension values: Linux, Windows, iOS, Android

- **PageId**

Set up extended metrics using the console

To use the console to send extended metrics to CloudWatch, use the following steps.

To send extended metrics to CloudWatch Evidently, you must use the AWS APIs or AWS CLI instead of the console. For information about using the AWS APIs to send extended metrics to either CloudWatch or Evidently, see [PutRumMetricsDestination](#) and [BatchCreateRumMetricDefinitions](#).

To use the console to set up an app monitor and send RUM extended metrics to CloudWatch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.

5. Choose **Send metrics**.
6. Select one or more metric names to send with additional dimensions.
7. Select one or more factors to use as dimensions for these metrics. As you make your choices, the number of extended metrics that your choices create is displayed in **Number of extended metrics**.

This number is calculated by multiplying the number of chosen metric names by the number of different dimensions that you create. This number represents how many custom metrics you are charged for. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

- a. To send a metric with page ID as a dimension, choose **Browse for page ID** and then select the page IDs to use.
- b. To send a metric with device type as a dimension, choose either **Desktop devices** or **Mobile and tablets**.
- c. To send a metric with operating system as a dimension, select one or more operating systems under **Operating system**.
- d. To send a metric with browser type as a dimension, select one or more browsers under **Browsers**.
- e. To send a metric with geolocation as a dimension, select one or more locations under **Locations**.

Only the locations where this app monitor has reported metrics from will appear in the list to choose from.

8. When you are finished with your choices, choose **Send metrics**.
9. (Optional) In the **Extended metrics** list, to create an alarm that watches one of the metrics, choose **Create alarm** in that metric's row.

For general information about CloudWatch alarms, see [Using Amazon CloudWatch alarms](#). For a tutorial for setting an alarm on a CloudWatch RUM extended metric, see [Tutorial: create an extended metric and alarm it](#).

Stop sending extended metrics

To use the console to stop sending extended metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Select one or more metric name and dimension combinations to stop sending. Then choose **Actions, Delete**.

Tutorial: create an extended metric and alarm it

This tutorial demonstrates how to set up an extended metric to be sent to CloudWatch, and then how to set an alarm on that metric. In this tutorial, you create a metric that tracks JavaScript errors on the Chrome browser.

To set up this extended metric and set an alarm on it

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view** and then choose the name of the app monitor that is to send the metric.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Choose **Send metrics**.
6. Select **JSErrorCount**.
7. Under **Browsers**, select **Chrome**.

This combination of **JSErrorCount** and **Chrome** will send one extended metric to CloudWatch. The metric counts JavaScript errors only for user sessions that use the Chrome browser. The metric name will be **JsErrorCount** and the dimension name will be **Browser**.

8. Choose **Send metrics**.
9. In the **Extended metrics** list, choose **Create alarm** in the row that displays **JsErrorCount** under **Name** and displays **Chrome** under **BrowserName**.
10. Under **Specify metric and conditions**, confirm that the **Metric name** and **BrowserName** fields are pre-filled with the correct values.
11. For **Statistic**, select the statistic that you want to use for the alarm. **Average** is a good choice for this type of counting metric.
12. For **Period**, select **5 minutes**.
13. Under **Conditions**, do the following:

- Choose **Static**.
 - Choose **Greater** to specify that the alarm should go into ALARM state when the number of errors is higher than the threshold you are about to specify.
 - Under **than...**, enter the number for the alarm threshold. The alarm goes into ALARM state when the number of errors over a 5-minute period exceeds this number.
14. (Optional) By default, the alarm goes into ALARM state as soon as the number of errors exceeds the threshold number you set during a 5-minute period. You can optionally change this so that the alarm goes into ALARM state only if this number is exceeded for more than one 5-minute period.

To do so, choose **Additional configuration** and then for **Datapoints to alarm**, specify how many 5-minute periods need to have the error number over the threshold to trigger the alarm. For example, you can select 2 out of 2 to have the alarm trigger only when two consecutive 5-minute periods are over the threshold, or 2 out of 3 to have the alarm trigger if any two of three consecutive 5-minute periods are over the threshold.

For more information about this type of alarm evaluation, see [Evaluating an alarm](#).

15. Choose **Next**.
16. For **Configure actions**, specify what should happen when the alarm goes into alarm state. To receive a notification with Amazon SNS, do the following:
- Choose **Add notification**.
 - Choose **In alarm**.
 - Either select an existing SNS topic or create a new one. If you create a new one, specify a name for it and add at least one email address to it.
17. Choose **Next**.
18. Enter a name and optional description for the alarm, and choose **Next**.
19. Review the details and choose **Create alarm**.

Data protection and data privacy with CloudWatch RUM

The AWS [shared responsibility model](#) applies to data protection and data privacy in Amazon CloudWatch RUM. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information about data privacy, see

the [Data Privacy FAQ](#). For information about data protection in Europe, see [The AWS Shared Responsibility Model and GDPR](#) blog post on the AWS Security Blog. For more resources about complying with GDPR requirements, see the [General Data Protection Regulation \(GDPR\) Center](#).

Amazon CloudWatch RUM generates a code snippet for you to embed into your website or web application code, based on your input of end user data that you want to collect. The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. The use of cookies (or similar technologies) is subject to data privacy regulations in certain jurisdictions. Before using Amazon CloudWatch RUM, we strongly recommend that you assess your compliance obligations under applicable law, including any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for the use of cookies and the processing (including collection) of end user data. For more information about how the web client uses cookies (or similar technologies) and what end-user data the web client collects, see [Information collected by the CloudWatch RUM web client](#) and [CloudWatch RUM web client cookies \(or similar technologies\)](#).

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Amazon CloudWatch RUM or other services might be included in diagnostic logs.

CloudWatch RUM web client cookies (or similar technologies)

The CloudWatch RUM web client collects certain data about user sessions by default. You can choose to enable cookies to have the web client collect a user ID and session ID that persist across page loads. The user ID is randomly generated by RUM.

If these cookies are enabled, RUM is able to display the following types of data when you view the RUM dashboard for this app monitor.

- Aggregated data based on user IDs, such as number of unique users and the number of different users who experienced an error.
- Aggregated data based on session IDs, such as number of sessions and the number of sessions that experienced an error.
- The *user journey*, which is the sequence of pages that each sampled user session includes.

Important

If you do not enable these cookies (or similar technologies), the web client still records certain information about end user sessions such as browser type/version, operating system type/version, device type, and so on. These are collected to provide aggregated page-specific insights, such as web vitals, page views, and pages that experienced errors. For more information about the data recorded, see [Information collected by the CloudWatch RUM web client](#).

Information collected by the CloudWatch RUM web client

This section documents the **PutRumEvents** schema, which defines the structure of the data that you can collect from user sessions using CloudWatch RUM.

A **PutRumEvents** request sends a data structure with the following fields to CloudWatch RUM.

- The ID of this batch of RUM events
- App monitor details, which includes the following:
 - App monitor ID
 - Monitored application version
- User details, which includes the following. **This is collected only if the app monitor has cookies enabled.**
 - A user ID generated by the web client
 - Session ID
- The array of [RUM events](#) in this batch.

RUM event schema

The structure of each RUM event includes the following fields.

- The ID of the event
- A timestamp
- The event type
- The user agent
- [Metadata](#)

- [RUM event details](#)

RUM event metadata

The metadata includes page metadata, user agent metadata, geolocation metadata, and domain metadata.

Page metadata

The page metadata includes the following:

- Page ID
- Page title
- Parent page ID. – **This is collected only if the app monitor has cookies enabled.**
- Interaction depth – **This is collected only if the app monitor has cookies enabled.**
- Page tags – You can add tags to page events to group pages together. For more information, see [Use page groups](#).

User agent metadata

The user agent metadata includes the following:

- Browser language
- Browser name
- Browser version
- Operating system name
- Operating system version
- Device type
- Platform type

Geolocation metadata

The geolocation metadata includes the following:

- Country code
- Subdivision code

Domain metadata

The domain metadata includes the URL domain.

RUM event details

The details of an event follow one of the following type of schemas, depending on the event type.

Session start event

This event contains no fields. **This is collected only if the app monitor has cookies enabled.**

Page view schema

A **Page view** event contains the following properties. You can deactivate page view collection by configuring the web client. For more information, see the [CloudWatch RUM web client documentation](#).

Name	Type	Description
Page ID	String	An ID that uniquely represents this page within the application. By default, this is the URL path.
Parent page ID	String	The ID of the page that the user was on when they navigated to the current page. This is collected only if the app monitor has cookies enabled.
Interaction depth	String	This is collected only if the app monitor has cookies enabled.

JavaScript error schema

JavaScript error events generated by the agent contain the following properties. The web client collects these events only if you selected to collect the errors telemetry.

Name	Type	Description
Error type	String	The error's name, if one exists. For more information, see Error.prototype.name .

Name	Type	Description
		Some browsers might not support error types.
Error message	String	The error's message. For more information, see Error.prototype.message . If the error field does not exist, this is the message of the error event. For more information, see ErrorEvent . Error messages might not be consistent across different browsers.
Stack trace	String	The error's stack trace, if one exists, truncated to 150 characters. For more information, see Error.prototype.stack . Some browsers might not support stack traces.

DOM event schema

Document object model (DOM) events generated by the agent contain the following properties. These events are not collected by default. They are collected only if you activate the interactions telemetry. For more information, see the [CloudWatch RUM web client documentation](#).

Name	Type	Description
Event	String	The type of DOM event, such as click, scroll, or hover. For more information, see Event reference .
Element	String	The DOM element type
Element ID	String	If the element that generated the event has an ID, this property stores that ID. For more information, see Element.id .
CSSLocator	String	The CSS locator used to identify the DOM element.
InteractionId	String	A unique ID for the interaction between the user and the UI.

Navigation event schema

Navigation events are collected only if the app monitor has performance telemetry activated.

Navigation events use [Navigation timing Level 1](#) and [Navigation timing Level 2](#) APIs. Level 2 APIs are not supported on all browsers, so these newer fields are optional.

Note

Timestamp metrics are based on [DOMHighResTimestamp](#). With Level 2 APIs, all timings are by default relative to the `startTime`. But for Level 1, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Navigation events contain the following properties.

Name	Type	Description	Notes
<code>initiatorType</code>	String	Represents the type of resource that initiated the performance event.	<p>Value: "navigation"</p> <p>Level 1: "navigation"</p> <p>Level 2: <code>entryData</code> <code>.initiatorType</code></p>
<code>navigationType</code>	String	Represents the type of navigation. This attribute is not required.	<p>Value: The value must be one of the following:</p> <ul style="list-style-type: none"> <code>navigate</code> is a navigation started by choosing

Name	Type	Description	Notes
			<p>a link, entering a URL in a browser's address bar, form submission, or initializing through a script operation other than reload or back_forward .</p> <ul style="list-style-type: none">• reload is a navigation through the browser's reload operation or location. reload() .• back_forward is a navigation through the browser's history

Name	Type	Description	Notes
			traversal operation. <ul style="list-style-type: none"> • prerender is a navigation initiated by a prerender hint. For more information, see Prerender.
startTime	Number	Indicates when the event is triggered.	Value: 0 Level 1: entryData .navigationStart - entryData .navigationStart Level 2: entryData .startTime

Name	Type	Description	Notes
unloadEventStart	Number	Indicates the time when the previous document in the window began to unload after the unload event was thrown.	<p>Value: If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0.</p> <p>Level 1:</p> <pre data-bbox="1312 997 1507 1556">entryData .unloadEventStart > 0 ? entryData .unloadEventStart - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre data-bbox="1312 1591 1507 1768">entryData .unloadEventStart</pre>

Name	Type	Description	Notes
promptForUnload	Number	The time taken to unload the document. In other words, the time between <code>unloadEventStart</code> and <code>unloadEventEnd</code> . <code>UnloadEventEnd</code> represents the moment in milliseconds when the unload event handler finishes.	<p>Value: If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0.</p> <p>Level 1: <code>entryData.unloadEventEnd - entryData.unloadEventStart</code></p> <p>Level 2: <code>entryData.unloadEventEnd - entryData.unloadEventStart</code></p>

Name	Type	Description	Notes
redirectCount	Number	<p>A number representing the number of redirects since the last non-redirect navigation under the current browsing context.</p> <p>This attribute is not required.</p>	<p>Value: If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0.</p> <p>Level 1: Not available</p> <p>Level 2: entryData .redirect Count</p>

Name	Type	Description	Notes
redirectStart	Number	The time when the first HTTP redirect starts.	<p>Value: If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0.</p> <p>Level 1:</p> <pre data-bbox="1307 903 1510 1417">entryData .redirect Start > 0 ? entryData .redirect Start - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre data-bbox="1307 1459 1510 1585">entryData .redirectStart</pre>

Name	Type	Description	Notes
redirectTime	Number	The time taken for the HTTP redirect. This is the difference between <code>redirectStart</code> and <code>redirectEnd</code> .	Level 1: : entryData .redirectEnd - entryData .redirectStart Level 2: : entryData .redirectEnd - entryData .redirectStart

Name	Type	Description	Notes
workerStart	Number	<p>This is a property of the <code>PerformanceResourceTiming</code> interface. It marks the beginning of worker thread operation.</p> <p>This attribute is not required.</p>	<p>Value: If a Service Worker thread is already running, or immediately before starting the Service Worker thread, this property returns the time immediately before dispatching <code>FetchEvent</code>. It returns 0 if the resource is not intercepted by a Service Worker.</p> <p>Level 1: Not available</p> <p>Level 2: <code>entryData.workerStart</code></p>

Name	Type	Description	Notes
workerTime	Number	<p>If the resource is intercepted by a Service Worker, this returns the time required for worker thread operation.</p> <p>This attribute is not required.</p>	<p>Level 1: Not available</p> <p>Level 2:</p> <pre>entryData .workerStart > 0 ? entryData .fetchStart - entryData .workerStart : 0</pre>
fetchStart	Number	<p>The time when the browser is ready to fetch the document using an HTTP request. This is before checking any application cache.</p>	<p>Level 1:</p> <pre>: entryData .fetchStart > 0 ? entryData .fetchStart - entryData .navigationStart : 0</pre> <p>Level 2: entryData .fetchStart</p>

Name	Type	Description	Notes
domainLookupStart	Number	The time when the domain lookup starts.	<p>Value: If a persistent connection is used or if the information is stored in a cache or local resource, the value will be the same as <code>fetchStart</code>.</p> <p>Level 1:</p> <pre>entryData .domainLookupStart > 0 ? entryData .domainLookupStart - entryData .navigationStart : 0</pre> <p>Level 2:</p> <pre>entryData .domainLookupStart</pre>

Name	Type	Description	Notes
dns	Number	The time required for domain lookup.	<p>Value: If the resources and DNS records are cached, the expected value is 0.</p> <p>Level 1: entryData .domainLookupEnd - entryData .domainLookupStart</p> <p>Level 2: entryData .domainLookupEnd - entryData .domainLookupStart</p>
nextHopProtocol	String	<p>A string representing the network protocol used to fetch the resource.</p> <p>This attribute is not required.</p>	<p>Level 1: Not available</p> <p>Level 2: entryData .nextHopProtocol</p>

Name	Type	Description	Notes
connectStart	Number	The time immediately before the user agent starts establishing the connection to the server to retrieve the document.	<p>Value: If an RFC2616 persistent connection is used, or if the current document is retrieved from relevant application caches or local resources, this attribute returns the value of <code>domainLookupEnd</code>.</p> <p>Level 1:</p> <pre>entryData .connectStart > 0 ? entryData .connectStart - entryData .navigationStart : 0</pre> <p>Level 2:</p> <pre>entryData .connectStart</pre>

Name	Type	Description	Notes
connect	Number	Measures the time required to establish the transport connections or to perform SSL authentication. It also includes the blocked time that is taken when there are too many concurrent requests issued by the browser.	<p>Level 1: entryData .connectEnd - entryData .connectStart</p> <p>Level 2: entryData .connectEnd - entryData .connectStart</p>
secureConnectionStart	Number	If the URL scheme of the current page is "https", this attribute returns the time immediately before the user agent starts the handshake process to secure the current connection. It returns 0 if HTTPS is not used. For more information about URL schemes, see URL representation .	<p>Formula: entryData .secureConnectionStart</p>

Name	Type	Description	Notes
tlsTime	Number	The time taken to complete an SSL handshake.	<p>Level 1:</p> <pre>entryData .secureCo nnectionS tart > 0 ? entryData .connectE nd - entryData .secureCo nnectionS tart : 0</pre> <p>Level 2:</p> <pre>entryData .secureCo nnectionS tart > 0 ? entryData .connectE nd - entryData .secureCo nnectionS tart : 0</pre>

Name	Type	Description	Notes
requestStart	Number	The time immediately before the user agent starts requesting the resource from the server, or from relevant application caches, or from local resources.	<p>Level 1:</p> <pre> : entryData .requestStart > 0 ? entryData .requestStart - entryData .navigationStart : 0 </pre> <p>Level 2: entryData.requestStart</p>
timeToFirstByte	Number	The time taken to receive the first byte of information after a request is made. This time is relative to the <code>startTime</code> .	<p>Level 1: entryData.responseStart - entryData.requestStart</p> <p>Level 2: entryData.responseStart - entryData.requestStart</p>

Name	Type	Description	Notes
responseStart	Number	The time immediately after the user agent's HTTP parser receives the first byte of the response from the relevant application caches, or from local resources, or from the server.	<p>Level 1:</p> <pre>entryData .response Start > 0 ? entryData .response Start - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre>entryData .response Start</pre>

Name	Type	Description	Notes
<p>responseTime</p>	<p>String</p>	<p>The time taken to receive a complete response in the form of bytes from the relevant application caches, or from local resources, or from the server.</p>	<p>Level 1:</p> <pre>entryData .response Start > 0 ? entryData .response End - entryData .response Start : 0</pre> <p>Level 2:</p> <pre>entryData .response Start > 0 ? entryData .response End - entryData .response Start : 0</pre>

Name	Type	Description	Notes
domInteractive	Number	The time when the parser finished its work on the main document, and the HTML DOM is constructed. At this time, its Document .readyState changes to "interactive" and the corresponding readystatechange event is thrown.	<p>Level 1:</p> <pre>entryData .domInteractive > 0 ? entryData .domInteractive - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre>entryData .domInter active</pre>

Name	Type	Description	Notes
domContentLoadedStart	Number	Represents the time value equal to the time immediately before the user agent fires the DOMContentLoaded event at the current document. The DOMContentLoaded event fires when the initial HTML document has been completely loaded and parsed. At this time, the main HTML document has finished parsing, the browser begins constructing the render tree, and subresources still have to be loaded. This does not wait for style sheets, images, and subframes to finish loading.	<p>Level 1:</p> <pre>entryData .domContentLoadedStart > 0 ?</pre> <pre>entryData .domContentLoadedStart -</pre> <pre>entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre>entryData .domContentLoadedStart</pre>

Name	Type	Description	Notes
domContentLoaded	Number	<p>This start and end time of render tree construction is marked by the <code>domContentLoadedEventStart</code> and <code>domContentLoadedEventEnd</code>. It lets CloudWatch RUM track execution. This property is the difference between <code>domContentLoadedStart</code> and <code>domContentLoadedEnd</code>.</p> <p>During this time, DOM and CSSOM are ready. This property waits on script execution, except for asynchronous and dynamically created scripts. If the scripts depend on style sheets, <code>domContentLoaded</code> waits on the style sheets, too. It does not wait on images.</p> <div data-bbox="592 909 1268 1703" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>The actual values of <code>domContentLoadedStart</code> and <code>domContentLoadedEnd</code> approximate to <code>domContentLoaded</code> in Google Chrome's Network panel. It indicates HTML DOM + CSSOM render tree construction time from the beginning of the page loading process. In the case of navigation metrics, the <code>domContentLoaded</code> value represents the difference between start and end values, which is the time required for downloading subresources and render-tree construction only.</p> </div>	<p>Level 2: <code>entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart</code></p> <p>Level 2: <code>entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart</code></p>

Name	Type	Description	Notes
domComplete	Number	The time immediately before the browser sets the current document readiness of the current document to complete. At this point, the loading of subresources, such as images, is complete. This includes the time taken for downloading blocking content such as CSS and synchronous JavaScript. This approximates to <code>loadTime</code> in Google Chrome's Network panel.	<p>Level 1:</p> <pre>entryData .domComplete > 0 ? entryData .domComplete - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre>entryData .domCompl ete</pre>
domProcessingTime	Number	The total time between the response and the load event start.	<p>Level 1:</p> <pre>entryData .loadEven tStart - entryData .responseEnd</pre> <p>Level 2:</p> <pre>entryData .loadEven tStart - entryData .responseEnd</pre>

Name	Type	Description	Notes
loadEventStart	Number	The time immediately before the load event of the current document is fired.	<p>Level 1:</p> <pre>entryData .loadEventStart > 0 ? entryData .loadEventStart - entryData .navigati onStart : 0</pre> <p>Level 2:</p> <pre>entryData .loadEventStart</pre>
loadEventTime	Number	The difference between loadEventStart and loadEventEnd . Additional functions or logic waiting for this load event will be fired during this time.	<p>Level 1:</p> <pre>entryData .loadEventEnd - entryData .loadEventStart</pre> <p>Level 2:</p> <pre>entryData .loadEventEnd - entryData .loadEventStart</pre>

Name	Type	Description	Notes
duration	String	Duration is the total page load time. It records the timing for downloading the main page and all of its synchronous subresources, and also for rendering the page. Asynchronous resources such as scripts continue to download later. This is the difference between the <code>loadEventEnd</code> and <code>startTime</code> properties.	<p>Level 1: entryData .loadEventEnd - entryData .navigationStart</p> <p>Level 2: entryData .duration</p>
headerSize	Number	<p>Returns the difference between <code>transferSize</code> and <code>encodedBodySize</code> .</p> <p>This attribute is not required.</p>	<p>Level 1: Not available</p> <p>Level 2: entryData .transferSize - entryData .encodedBodySize</p> <p>Level 2: entryData .transferSize - entryData .encodedBodySize</p>

Name	Type	Description	Notes
compressionRatio	Number	<p>The ratio of <code>encodedBodySize</code> and <code>decodedBodySize</code>. The value of <code>encodedBodySize</code> is the compressed size of the resource excluding the HTTP headers. The value of <code>decodedBodySize</code> is the decompressed size of the resource excluding the HTTP headers.</p> <p>This attribute is not required.</p>	<p>Level 1: Not available.</p> <p>Level 2:</p> <pre>entryData .encodedBodySize > 0 ? entryData .decodedBodySize / entryData .encodedBodySize : 0</pre>
navigationTimingLevel	Number	The navigation timing API version.	Value: 1 or 2

Resource event schema

Resource events are collected only if the app monitor has performance telemetry activated.

Timestamp metrics are based on [The DOMHighResTimeStamp typedef](#). With Level 2 APIs, by default all timings are relative to the `startTime`. But for Level 1 APIs, the `navigationStartTime` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Resource events generated by the agent contain the following properties.

Name	Type	Description	Notes
targetUrl	String	Returns the resource's URL.	Formula: entryData.name
initiatorType	String	Represents the type of resource that initiated the performance resource event.	Value: "resource" Formula: entryData.initiatorType
duration	String	Returns the difference between the <code>responseEnd</code> and <code>startTime</code> properties. This attribute is not required.	Formula: entryData.duration
transferSize	Number	Returns the size (in octets) of the fetched resource, including the response header fields and the response payload body. This attribute is not required.	Formula: entryData.transferSize
fileType	String	Extensions derived from the target URL pattern.	

Largest contentful paint event schema

Largest contentful paint events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals .		

First input delay event

First input delay events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals .		

Cumulative layout shift event

Cumulative layout shift events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals .		

HTTP event

HTTP events can contain the following properties. It will contain either a Response field or an Error field, but not both.

These events are collected only if the app monitor has HTTP telemetry activated.

Name	Description
Request	The request field includes the following: <ul style="list-style-type: none">• The Method field, which can have values such as GET, POST, and so on.• The URL

Name	Description
Response	The response field includes the following: <ul style="list-style-type: none"> • Status, such as 2xx, 4xx, or 5xx • Status text
Error	The error field includes the following: <ul style="list-style-type: none"> • Type • Message • File name • Line number • Column number • Stack trace

X-Ray trace event schema

These events are collected only if the app monitor has X-Ray tracing activated.

For information about X-Ray trace event schemas, see [AWS X-Ray segment documents](#).

Route change timing for single-page applications

In a traditional multi-page application, when a user requests for new content to be loaded, the user is actually requesting a new HTML page from the server. As a result, the CloudWatch RUM web client captures the load times using the regular performance API metrics.

However, single-page web applications use JavaScript and Ajax to update the interface without loading a new page from the server. Single-page updates are not recorded by the browser timing API, but instead use route change timing.

CloudWatch RUM supports the monitoring of both full page loads from the server and single-page updates, with the following differences:

- For route change timing, there are no browser-provided metrics such as `tlsTime`, `timeToFirstByte`, and so on.
- For route change timing, the `initiatorType` field will be `route_change`.

The CloudWatch RUM web client listens to user interactions that may lead to a route change, and when such a user interaction is recorded, the web client records a timestamp. Then route change timing will begin if both of the following are true:

- A browser history API (except browser forward and back buttons) was used to perform the route change.
- The difference between the time of route change detection and latest user interaction timestamp is less than 1000 ms. This avoids data skew.

Then, once route change timing begins, that timing completes if there are no ongoing AJAX requests and DOM mutations. Then the timestamp of the latest completed activity will be used as the completion timestamp.

Route change timing will time out if there are ongoing AJAX requests or DOM mutations for more than 10 seconds (by default). In this case, the CloudWatch RUM web client will no longer record timing for this route change.

As a result, the duration of a route change event is calculated as the following:

```
(time of latest completed activity) - (latest user interaction timestamp)
```

Manage your applications that use CloudWatch RUM

Use the steps in these sections to manage your applications' use of CloudWatch RUM.

How do I find a code snippet that I've already generated?

To find a CloudWatch RUM code snippet that you've already generated for an application, follow these steps.

To find a code snippet that you've already generated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view**.
4. Next to the name of the app monitor, choose **View JavaScript**.
5. In the **JavaScript Snippet** pane, choose **Copy to clipboard**.

Edit your application

To change an app monitor's settings, follow these steps. You can change any settings except the app monitor name.

To edit how your application uses CloudWatch RUM

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions, Edit**.
5. Change any settings except the application name. For more information about the settings, see [Step 2: Create an app monitor](#).
6. When finished, choose **Save**.

Changing the settings changes the code snippet. You must now paste the updated code snippet into your application.

7. After the JavaScript code snippet is created, choose **Copy to clipboard** or **Download**, and then choose **Done**.

To start monitoring with the new settings, you insert the code snippet into your application. Insert the code snippet inside the <head> element of your application, before the <body> element or any other <script> tags.

Stop using CloudWatch RUM or delete an app monitor

To stop using CloudWatch RUM with an application, remove the code snippet that RUM generated from your application's code.

To delete a RUM app monitor, follow these steps.

To delete an app monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, RUM**.
3. Choose **List view**.
4. Choose the button next to the name of the application, and then choose **Actions, Delete**.

5. In the confirmation box, enter **Delete** and then choose **Delete**.
6. If you haven't done so already, delete the CloudWatch RUM code snippet from your application's code.

CloudWatch RUM quotas

CloudWatch RUM has the following quotas.

Resource	Default quota
App monitors	20 per account You can request a quota increase.
RUM ingestion rate	50 PutRunEvents requests per second (TPS). You can request a quota increase.

Troubleshooting CloudWatch RUM

This section contains tips to help you troubleshoot CloudWatch RUM.

There is no data for my application

First, make sure that the code snippet has been correctly inserted into your application. For more information, see [Step 4: Insert the code snippet into your application](#).

If that is not the issue, then maybe there has been no traffic to your application yet. Generate some traffic by accessing your application the same way that a user would.

Data has stopped being recorded for my application

Your application might have been updated and now no longer contains a CloudWatch RUM code snippet. Check your application code.

Another possibility is that someone may have updated the code snippet but then didn't insert the updated snippet into the application. Find the current correct code snippet by following the

directions in [How do I find a code snippet that I've already generated?](#) and compare it to the code snippet that is pasted into your application.

Perform launches and A/B experiments with CloudWatch Evidently

You can use Amazon CloudWatch Evidently to safely validate new features by serving them to a specified percentage of your users while you roll out the feature. You can monitor the performance of the new feature to help you decide when to ramp up traffic to your users. This helps you reduce risk and identify unintended consequences before you fully launch the feature.

You can also conduct A/B experiments to make feature design decisions based on evidence and data. An experiment can test as many as five variations at once. Evidently collects experiment data and analyzes it using statistical methods. It also provides clear recommendations about which variations perform better. You can test both user-facing features and backend features.

Evidently pricing

Evidently charges your account based on Evidently events and Evidently analysis units. Evidently events include both data events such as clicks and page views, and assignment events that determine the feature variation to serve to a user.

Evidently analysis units are generated from Evidently events, based on rules that you have created in Evidently. Analysis units are the number of rule matches on events. For example, a user click event might produce a single Evidently analysis unit, a click count. Another example is a user checkout event that might produce two Evidently analysis units, checkout value and the number of items in cart. For more information about pricing, see [Amazon CloudWatch Pricing](#).

CloudWatch Evidently is currently available in the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Frankfurt)

- Europe (Ireland)
- Europe (Stockholm)

Topics

- [IAM policies to use Evidently](#)
- [Create projects, features, launches, and experiments](#)
- [Manage features, launches, and experiments](#)
- [Adding code to your application](#)
- [Project data storage](#)
- [How Evidently calculates results](#)
- [View launch results in the dashboard](#)
- [View experiment results in the dashboard](#)
- [How CloudWatch Evidently collects and stores data](#)
- [Using service-linked roles for Evidently](#)
- [CloudWatch Evidently quotas](#)
- [Tutorial: A/B testing with the Evidently sample application](#)

IAM policies to use Evidently

To fully manage CloudWatch Evidently, you must be signed in as an IAM user or role that has the following permissions:

- The **AmazonCloudWatchEvidentlyFullAccess** policy
- The **ResourceGroupsandTagEditorReadOnlyAccess** policy

Additionally, to be able to create a project that stores evaluation events in Amazon S3 or CloudWatch Logs, you need the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```



```

        "s3:GetBucketPolicy",
        "s3:PutBucketPolicy",
        "s3:GetObject",
        "s3:ListBucket"
    ],
    "Resource": "arn:aws:s3:::*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogDelivery",
        "logs>DeleteLogDelivery",
        "logs:DescribeResourcePolicies",
        "logs:PutResourcePolicy"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

Additional permissions for CloudWatch RUM integration

Additionally, if you intend to manage Evidently launches or experiments that integrate with Amazon CloudWatch RUM and use CloudWatch RUM metrics for monitoring, you need the **AmazonCloudWatchRUMFullAccess** policy. To create an IAM role to give the CloudWatch RUM web client permission to send data to CloudWatch RUM, you need the following permissions:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchRUMEvidentlyRole-*",
                "arn:aws:iam::*:policy/service-role/CloudWatchRUMEvidentlyPolicy-*"
            ]
        }
    ]
}

```

```
    }  
  ]  
}
```

Permissions for read-only access to Evidently

For other users who need to view Evidently data but don't need to create Evidently resources, you can grant the **AmazonCloudWatchEvidentlyReadOnlyAccess** policy.

Create projects, features, launches, and experiments

To get started with CloudWatch Evidently, for either a feature launch or an A/B experiment, you first create a *project*. A project is a logical grouping of resources. Within the project, you create *features* that have variations that you want to test or launch. You can create a feature either before you create a launch or experiment, or at the same time.

Topics

- [Create a new project](#)
- [Use client-side evaluation - powered by AWS AppConfig](#)
- [Add a feature to a project](#)
- [Use segments to focus your audience](#)
- [Create a launch](#)
- [Create an experiment](#)

Create a new project

Use these steps to set up a new CloudWatch Evidently project.

To create a new CloudWatch Evidently project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose **Create project**.
4. For **Project name**, enter a name to be used to identify this project within the CloudWatch Evidently console.

You can optionally add a project description.

5. For **Evaluation event storage**, choose whether you want to store the evaluation events that you collect with Evidently. Even if you don't store these events, Evidently aggregates them to create metrics and other experiment data that you can view in the Evidently dashboard. For more information, see [Project data storage](#).
6. For **Use client-side evaluation**, choose whether you want to enable client-side evaluation for this project. With client-side evaluation, your application can assign variations to user sessions locally instead of by calling the [EvaluateFeature](#) operation. This mitigates the latency and availability risks that come with an API call. For more information, see [Use client-side evaluation - powered by AWS AppConfig](#).

To create a project with client-side evaluation, you must have the `evidently:ExportProjectAsConfiguration` permission.

If you enable client-side evaluation, also do the following:

- a. Choose whether to use an existing AWS AppConfig application or create a new one.
- b. Choose whether to use an existing AWS AppConfig environment or create a new one.

For more information about applications and environments in AWS AppConfig, see [How AWS AppConfig works](#).

7. (Optional) To add tags to this project, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

8. Choose **Create project**.

Use client-side evaluation - powered by AWS AppConfig

You can use *client-side evaluation - powered by AWS AppConfig (client-side evaluation)* in a project, which lets your application assign variations to user sessions locally instead of assigning variations by calling the [EvaluateFeature](#) operation. This mitigates the latency and availability risks that come with an API call.

To use client-side evaluation, attach the AWS AppConfig Lambda extension as a layer to your Lambda functions and configure the environment variables. The client-side evaluation runs as

a side process on the local host. Then, you can call the **EvaluateFeature** and **PutProjectEvent** operations against `localhost`. The client-side evaluation process handles the variation assignment, caching, and data synchronization. For more information about AWS AppConfig, see [How AWS AppConfig works](#).

When you integrate with AWS AppConfig, you specify an AWS AppConfig *application ID* and an AWS AppConfig *environment ID* to Evidently. You can use the same application ID and environment ID across Evidently projects.

When you create a project with client-side evaluation enabled, Evidently creates an AWS AppConfig configuration profile for that project. The configuration profile for each project will be different.

Client-side evaluation access control

Evidently client-side evaluation uses a different access control mechanism than the rest of Evidently does. We strongly recommend that you understand this so that you can implement the proper security measures.

With Evidently, you can create IAM policies that limit the actions a user can perform on individual resources. For example, you can create a user role that disallows a user from having the **EvaluateFeature** action. For more information about the Evidently actions that can be controlled with IAM policies, see [Actions defined by Amazon CloudWatch Evidently](#).

The client-side evaluation model allows local evaluations of Evidently features that use project metadata. A user of a project with client-side evaluation enabled can call the **EvaluateFeature** API against a local host endpoint, and this API call does not reach Evidently and is not authenticated by the Evidently service's IAM policies. This call is successful even if the user doesn't have the IAM permission to use the **EvaluateFeature** action. However, a user still needs the **PutProjectEvents** permission for the agent to buffer the evaluation events or custom events and to offload data to Evidently asynchronously.

Additionally, a user must have the `evidently:ExportProjectAsConfiguration` permission to be able to create a project that uses client-side evaluation. This helps you control access to **EvaluateFeature** actions that are called during client-side evaluation.

If you aren't careful, the client-side evaluation security model can subvert the policies that you have set on the rest of Evidently. A user who has the `evidently:ExportProjectAsConfiguration` permission can create a project with client-side evaluation enabled, and then use the **EvaluateFeature** action for client-side evaluation with that project even if they are expressly denied the **EvaluateFeature** action in an IAM policy.

Get started with Lambda

Evidently currently supports client-side evaluation by using an AWS Lambda environment. To get started, first decide which AWS AppConfig application and environment to use. Choose an existing application and environment, or create new ones.

The following sample AWS AppConfig AWS CLI commands create an application and environment.

```
aws appconfig create-application --name YOUR_APP_NAME
```

```
aws appconfig create-environment --application-id YOUR_APP_ID --  
name YOUR_ENVIRONMENT_NAME
```

Next, create an Evidently project by using these AWS AppConfig resources. For more information, see [Create a new project](#).

Client-side evaluation is supported in Lambda by using a Lambda layer. This is a public layer that is part of AWS-AppConfig-Extension, a public AWS AppConfig extension created by the AWS AppConfig service. For more information about Lambda layers, see [Layer](#).

To use client-side evaluation, you must add this layer to your Lambda function and configure permissions and environment variables.

To add the Evidently client-side evaluation Lambda layer to your Lambda function and configure it

1. Create a Lambda function if you haven't already.
2. Add the client-side evaluation layer to your function. You can either specify its ARN or select it from the list of AWS layers if you haven't already. For more information, see [Configuring functions to use layers](#) and [Available versions of the AWS AppConfig Lambda extension](#).
3. Create an IAM policy named **EvidentlyAppConfigCachingAgentPolicy** with the following contents, and attach it to the function's execution role. For more information, see [Lambda execution role](#).

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "VisualEditor0",
```

```

        "Effect": "Allow",
        "Action": [
            "appconfig:GetLatestConfiguration",
            "appconfig:StartConfigurationSession",
            "evidently:PutProjectEvents"
        ],
        "Resource": "*"
    }
]
}

```

4. Add the required environment variable

`AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` to your Lambda function. This environment variable specifies the mapping between the Evidently project and the AWS AppConfig resources.

If you are using this function for one Evidently project, set the value of the environment variable to: `applications/APP_ID/environments/ENVIRONMENT_ID/configurations/PROJECT_NAME`

If you are using this function for multiple Evidently projects, use a comma to separate the values, as in the following example: `applications/APP_ID_1/environments/ENVIRONMENT_ID_1/configurations/PROJECT_NAME_1, applications/APP_ID_2/environments/ENVIRONMENT_ID_2/configurations/PROJECT_NAME_2`

- (Optional) Set other environment variables. For more information, see [Configuring the AWS AppConfig Lambda extension](#).
- In your application, get Evidently evaluations locally by sending `EvaluateFeature` to `localhost`.

Python example:

```

import boto3
from botocore.config import Config

def lambda_handler(event, context):
    local_client = boto3.client(
        'evidently',
        endpoint_url="http://localhost:2772",
        config=Config(inject_host_prefix=False)
    )

```

```
)
response = local_client.evaluate_feature(
    project=event['project'],
    feature=event['feature'],
    entityId=event['entityId']
)
print(response)
```

Node.js example:

```
const AWS = require('aws-sdk');
const evidently = new AWS.Evidently({
    region: "us-west-2",
    endpoint: "http://localhost:2772",
    hostPrefixEnabled: false
});

exports.handler = async (event) => {

    const evaluation = await evidently.evaluateFeature({
        project: 'John_ETCProject_Aug2022',
        feature: 'Feature_IceCreamFlavors',
        entityId: 'John'
    }).promise()

    console.log(evaluation)
    const response = {
        statusCode: 200,
        body: evaluation,
    };
    return response;
};
```

Kotlin example:

```
String localhostEndpoint = "http://localhost:2772/"
public AmazonCloudWatchEvidentlyClient getEvidentlyLocalClient() {
    return AmazonCloudWatchEvidentlyClientBuilder.standard()

        .withEndpointConfiguration(AwsClientBuilder.EndpointConfiguration(localhostEndpoint,
            region))
}
```

```
.withClientConfiguration(ClientConfiguration().withDisableHostPrefixInjection(true))
    .withCredentials(credentialsProvider)
    .build();
}

AmazonCloudWatchEvidentlyClient evidently = getEvidentlyLocalClient();

// EvaluateFeature via local client.
EvaluateFeatureRequest evaluateFeatureRequest = new
    EvaluateFeatureRequest().builder()
        .withProject(${YOUR_PROJECT}) //Required.
        .withFeature(${YOUR_FEATURE}) //Required.
        .withEntityId(${YOUR_ENTITY_ID}) //Required.
        .withEvaluationContext(${YOUR_EVAL_CONTEXT}) //Optional: a JSON object of
        attributes that you can optionally pass in as part of the evaluation event sent to
        Evidently.
        .build();

EvaluateFeatureResponse evaluateFeatureResponse =
    evidently.evaluateFeature(evaluateFeatureRequest);

// PutProjectEvents via local client.
PutProjectEventsRequest putProjectEventsRequest = new
    PutProjectEventsRequest().builder()
        .withData(${YOUR_DATA})
        .withTimeStamp(${YOUR_TIMESTAMP})
        .withType(${YOUR_TYPE})
        .build();

PutProjectEventsResponse putProjectEventsResponse =
    evidently.putProjectEvents(putProjectEventsRequest);
```

Configure how often the client sends data to Evidently

To specify how often client-side evaluation sends data to Evidently, you can optionally configure two environment variables.

- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_EVENT_BATCH_SIZE` specifies the number of events per project to batch before sending them to Evidently. Valid values are integers between 1 and 50, and the default is 40.

- `AWS_APPCONFIG_EXTENSION_EVIDENTLY_BATCH_COLLECTION_DURATION` specifies the duration in seconds to wait for events before sending them to Evidently. The default is 30.

Troubleshooting

Use the following information to help troubleshoot problems with using CloudWatch Evidently with client-side evaluation - powered by AWS AppConfig.

An error occurred (BadRequestException) when calling the EvaluateFeature operation: HTTP method not supported for provided path

Your environment variables might be configured incorrectly. For example, you might have used `EVIDENTLY_CONFIGURATIONS` as the environment variable name instead of `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS`.

ResourceNotFoundException: Deployment not found

Your update to the project metadata has not been deployed to AWS AppConfig. Check for an active deployment in the AWS AppConfig environment that you used for client-side evaluation.

ValidationException: No Evidently configuration for project

Your `AWS_APPCONFIG_EXTENSION_EVIDENTLY_CONFIGURATIONS` environment variable might be configured with the incorrect project name.

Add a feature to a project

A *feature* in CloudWatch Evidently represents a feature that you want to launch or that you want to test variations of.

Before you can add a feature, you must create a project. For more information, see [Create a new project](#).

To add a feature to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project.
4. Choose **Add feature**.

5. For **Feature name**, enter a name to be used to identify this feature within this project.

You can optionally add a feature description.

6. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see [Variation types](#).
7. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. This is also the variation that is served to users who are not added to a launch or experiment for this feature.

8. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start a launch or an experiment.

For more information, see [Adding code to your application](#).

9. (Optional) To specify that certain users always see a certain variation, choose **Overrides**, **Add override**. Then, specify a user by entering their user ID, account ID, or some other identifier in **Identifier**, and specify which variation they should see.

This can be useful for members of your own testing team or other internal users when you want to make sure they see a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.

You can repeat this for as many as 20 users by choosing **Add override** again.

10. (Optional) To add tags to this feature, choose **Tags**, **Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

11. Choose **Add feature**.

Variation types

When you create a feature and define the variations, you must select a *variation type*. The possible types are:

- Boolean
- Long integer
- Double precision floating-point number
- String

The variation type sets how the different variations are differentiated in your code. You can use the variation type to simplify the implementation of CloudWatch Evidently and also to simplify the process of modifying the features in your launches and experiments.

For example, if you define a feature with the long integer variation type, the integers that you specify to differentiate the variations can be numbers passed directly into your code. One example might be testing the pixel size of a button. The values for the variation types can be the number of pixels used in each variation. The code for each variation can read the variation type value and use that as the button size. To test a new button size, you can change the number used for the value of the variation, without making any other code changes.

When you set the values for your variation types within a feature, you should avoid assigning the same values to multiple variations, unless you want to do A/A testing to initially try out CloudWatch Evidently, or have other reasons to do so.

Evidently doesn't have native support for JSON as a type, but you can pass in JSON in the String variation type, and parse that JSON in your code.

Use segments to focus your audience

You can define audience *segments* and use them in your launches and experiments. A segment is a portion of your audience that shares one or more characteristics. Examples could be Chrome browser users, users in Europe, or Firefox browser users in Europe who also fit other criteria that your application collects, such as age.

Using a segment in an experiment limits that experiment to evaluating only the users who match the segment criteria. When you use one or more segments in a launch, you can define different traffic splits for the different audience segments.

Segment rule pattern syntax

To create a segment, define a segment rule *pattern*. Specify the attributes that you want to use to evaluate whether a user session will be in the segment. The pattern that you create is compared to the value of `evaluationContext` that Evidently finds in a user session. For more information, see [Using EvaluateFeature](#).

To create an segment rule pattern, specify the fields that you want the pattern to match. You can also use logic in your pattern, such as `And`, `Or`, `Not` and `Exists`.

For an `evaluationContext` to match a pattern, the `evaluationContext` must match all parts of the rule pattern. Evidently ignores the fields in the `evaluationContext` that aren't included in the rule pattern.

The values that rule patterns match follow JSON rules. You can include strings enclosed in quotation marks (`"`), numbers, and the keywords `true`, `false`, and `null`.

For strings, Evidently uses exact character-by-character matching without case-folding or any other string normalization. Therefore, rule matches are case-sensitive. For example, if your `evaluationContext` includes a `browser` attribute but your rule pattern checks for `Browser`, it will not match.

For numbers, Evidently uses string representation. For example, `300`, `300.0`, and `3.0e2` are not considered equal.

When you write rule patterns to match `evaluationContext`, you can use the `TestSegmentPattern` API or the `test-segment-pattern` CLI command to test that your pattern matches the correct JSON. For more information, see [TestSegmentPattern](#).

The following summary shows all the comparison operators that are available in Evidently segment patterns.

Comparison	Example	Rule syntax
Null	UserID is null	<pre>{ "UserID": [null] }</pre>
Empty	LastName is empty	<pre>{</pre>

Comparison	Example	Rule syntax
		<pre>"LastName": [""]</pre>
Equals	Browser is "Chrome"	<pre>{ "Browser": ["Chrome"] }</pre>
And	Country is "France" and Device is "Mobile"	<pre>{ "Country": ["France"], "Device": ["Mobile"] }</pre>
Or (multiple values of a single attribute)	Browser is "Chrome" or "Firefox"	<pre>{ "Browser": ["Chrome", "Firefox"] }</pre>
Or (different attributes)	Browser is "Safari" or Device is "Tablet"	<pre>{ "\$or": [{"Browser": ["Safari"]}, {"Device": ["Tablet"] }]</pre>
Not	Browser is anything but "Safari"	<pre>{ "Browser": [{ "anything-but": ["Safari"] }] }</pre>

Comparison	Example	Rule syntax
Numeric (equals)	Price is 100	<pre data-bbox="1073 226 1500 459"> { "Price": [{ "numeric": ["=", 100] }] } </pre>
Numeric (range)	Price is more than 10, and less than or equal to 20	<pre data-bbox="1073 501 1500 735"> { "Price": [{ "numeric": [">", 10, "<=", 20] }] } </pre>
Exists	Age field exists	<pre data-bbox="1073 777 1500 968"> { "Age": [{ "exists": true }] } </pre>
Does not exist	Age field does not exist	<pre data-bbox="1073 1010 1500 1201"> { "Age": [{ "exists": false }] } </pre>
Begins with a prefix	Region is in the United States	<pre data-bbox="1073 1243 1500 1434"> { "Region": [{"prefix": "us-" }] } </pre>
Ends with a suffix	Location has a suffix "West"	<pre data-bbox="1073 1476 1500 1709"> { "Region": [{"suffix": "West" }] } </pre>

Segment rule examples

All of the following examples assume that you are passing values for `evaluationContext` with the same field labels and values that you are using in your rule patterns.

The following example matches if `Browser` is Chrome or Firefox and `Location` is US-West.

```
{
  "Browser": ["Chrome", "Firefox"],
  "Location": ["US-West"]
}
```

The following example matches if `Browser` is any browser except Chrome, the `Location` starts with US, and an `Age` field exists.

```
{
  "Browser": [ {"anything-but": ["Chrome"]} ],
  "Location": [ {"prefix": "US"} ],
  "Age": [ {"exists": true} ]
}
```

The following example matches if the `Location` is Japan and either `Browser` is Safari or `Device` is Tablet.

```
{
  "Location": ["Japan"],
  "$or": [
    {"Browser": ["Safari"]},
    {"Device": ["Tablet"]}
  ]
}
```

Create a segment

After you create a segment, you can use it in any launch or experiment in any project.

To create a segment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.

3. Choose the **Segments** tab.
4. Choose **Create segment**.
5. For **Segment name**, enter a name to use to identify this segment.

Optionally, add a description.

6. For **Segment pattern**, enter a JSON block that defines the rule pattern. For more information about rule pattern syntax, see [Segment rule pattern syntax](#).

Create a launch

To expose a new feature or change to a specified percentage of your users, create a launch. You can then monitor key metrics such as page load times and conversions before you roll out the feature to all of your users.

Before you can add a launch, you must have created a project. For more information, see [Create a new project](#).

When you add a launch, you can use a feature that you have already created, or create a new feature while you create the launch.

To add a launch to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Select the button next to the name of the project and choose **Project actions, Create launch**.
4. For **Launch name**, enter a name to be used to identify this feature within this project.

You can optionally add a description.

5. Choose either **Select from existing features** or **Add new feature**.

If you are using an existing feature, select it under **Feature name**.

If you choose **Add new feature**, do the following:

- a. For **Feature name**, enter a name to be used to identify this feature within this project. You can optionally add a description.
- b. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. For more information, see [Variation types](#).

- c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop an experiment, this default variation will then be served to all users.

- d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start the launch.

For more information, see [Adding code to your application](#).

6. For **Launch configuration**, choose whether to start the launch immediately or schedule it to start later.
7. (Optional) To specify different traffic splits for audience segments that you have defined, instead of the traffic split that you will use for your general audience, choose **Add Segment Overrides**.

In **Segment Overrides**, select a segment and define the traffic split to use for that segment.

You can optionally define more segments to define traffic splits for by choosing **Add Segment Override**. A launch can have up to six segment overrides.

For more information, see [Use segments to focus your audience](#).

8. For **Traffic configuration**, select the traffic percentage to assign to each variation for the general audience that doesn't match the segment overrides. You can also choose to exclude variations from being served to users.

The **Traffic summary** shows how much of your overall traffic is available for this launch.

9. If you choose to schedule the launch to start later, you can add multiple steps to the launch. Each step can use different percentages for serving the variations. To do this, choose **Add another step** and then specify the schedule and traffic percentages for the next step. You can include as many as five steps in a launch.
10. If you want to track your feature performance with metrics during the launch, choose **Metrics, Add metric**. You can use either CloudWatch RUM metrics or custom metrics.

To use a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:

- Choose **Custom metrics** and enter a name for the metric.
- Under **Metric rule**, for **Entity ID**, enter the way to identify the entity. This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
- For **Value key**, enter the value that is to be tracked to produce the metric.
- Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

As you enter those fields, the box shows examples of how to code the EventBridge rule to create the metric. For more information about EventBridge, see [What Is Amazon EventBridge?](#)

To use RUM metrics, you must already have a RUM app monitor set up for your application. For more information, see [Set up an application to use CloudWatch RUM](#).

 **Note**

If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions that participate in the launch will send metrics to Evidently. To ensure that the launch metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

11. (Optional) If you create at least one metric for the launch, you can associate an existing CloudWatch alarm with this launch. To do so, choose **Associate CloudWatch alarms**.

When you associate an alarm with a launch, CloudWatch Evidently must add tags to the alarm with the project name and launch name. This is so that CloudWatch Evidently can display the correct alarms in the launch information in the console.

To acknowledge that CloudWatch Evidently will add these tags, choose **Allow Evidently to tag the alarm resource identified below with this launch resource**. Then, choose **Associate alarm** and enter the alarm name.

For information about creating CloudWatch alarms, see [Using Amazon CloudWatch alarms](#).

12. (Optional) To add tags to this launch, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

13. Choose **Create launch**.

Create an experiment

Use experiments to test different versions of a feature or website and collect data from real user sessions. This way, you can make choices for your application based on evidence and data.

Before you can add an experiment, you must have created a project. For more information, see [Create a new project](#).

When you add an experiment, you can use a feature that you have already created, or create a new feature while you create the experiment.

To add an experiment to a project

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Select the button next to the name of the project and choose **Project actions, Create experiment**.
4. For **Experiment name**, enter a name to be used to identify this feature within this project.

You can optionally add a description.

5. Choose either **Select from existing features** or **Add new feature**.

If you are using an existing feature, select it under **Feature name**.

If you choose **Add new feature**, do the following:

- a. For **Feature name**, enter a name to be used to identify this feature within this project. You can also optionally enter a description.
- b. For **Feature variations**, for **Variation type** choose **Boolean**, **Long**, **Double**, or **String**. The type defines which type of value is used for each variation. For more information, see [Variation types](#).

- c. Add up to five variations for your feature. The **Value** for each variation must be valid for the **Variation type** that you selected.

Specify one of the variations to be the default. This is the baseline that the other variations will be compared to, and should be the variation that is being served to your users now. If you stop an experiment that uses this feature, the default variation is then served to the percentage of users that were in the experiment previously.

- d. Choose **Sample code**. The code example shows what you need to add to your application to set up the variations and assign user sessions to them. You can choose between JavaScript, Java, and Python for the code.

You don't need to add the code to your application right now, but you must do so before you start the experiment. For more information, see [Adding code to your application](#).

6. For **Audience**, optionally select a segment that you have created if you want this experiment to apply only to the users who match that segment. For more information about segments, see [Use segments to focus your audience](#).
7. For **Traffic split for the experiment**, specify the percentage of the selected audience whose sessions will be used in the experiment. Then allocate the traffic for the different variations that the experiment uses.

If a launch and an experiment are both running at the same time for the same feature, the audience is first directed to the launch. Then, the percentage of traffic specified for the launch is taken from the overall audience. After that, the percentage that you specify here is the percentage of the remaining audience that is used for the experiment. Any remaining traffic after that is served the default variation.

8. For **Metrics**, choose the metrics to use to evaluate the variations during the experiment. You must use at least one metric for evaluation.
 - a. For **Metric source**, choose whether to use CloudWatch RUM metrics or custom metrics.
 - b. Enter a name for the metric. For **Goal**, choose **Increase** if you want a higher value for the metric to indicate a better variation. Choose **Decrease** if you want a lower value for the metric to indicate a better variation.
 - c. If you are using a custom metric, you can create the metric here using an Amazon EventBridge rule. To create a custom metric, do the following:

- Under **Metric rule**, for **Entity ID**, enter a way to identify the entity, This can be a user or session that does an action that causes a metric value to be recorded. An example is `userDetails.userID`.
- For **Value key**, enter the value that is to be tracked to produce the metric.
- Optionally, enter a name for the units for the metric. This unit name is for display purposes only, for use on graphs in the Evidently console.

You can use RUM metrics only if you have set up RUM to monitor this application. For more information, see [CloudWatch RUM](#).

 **Note**

If you use RUM metrics, and the app monitor is not configured to sample 100% of user sessions, then not all of the user sessions in the experiment will send metrics to Evidently. To ensure that the experiment metrics are accurate, we recommend that the app monitor uses 100% of user sessions for sampling.

- d. (Optional) To add more metrics to evaluate, choose **Add metric**. You can evaluate as many as three metrics during the experiment.
9. (Optional) To create CloudWatch alarms to use with this experiment, choose **CloudWatch alarms**. The alarms can monitor whether the difference in results between each variation and the default variation is larger than a threshold that you specify. If a variation's performance is worse than the default variation, and the difference is greater than your threshold, it goes into alarm state and notifies you.

Creating an alarm here creates one alarm for each variation that is not the default variation.

If you create an alarm, specify the following:

- For **Metric name**, choose the experiment metric to use for the alarm.
- For **Alarm condition** choose what condition causes the alarm to go into alarm state, when the variation metric values are compared to the default variation metric values. For example, choose **Greater** or **Greater/Equal** if higher numbers indicate for the variation indicates that it is performing poorly. This would be appropriate if the metric is measuring page load time, for example.

- Enter a number for the threshold, which is the percentage difference in performance that will cause the alarm to go into ALARM state.
- For **Average over period**, choose how much metric data for each variation is aggregated together before being compared.

You can choose **Add new alarm** again to add more alarms to the experiment.

Next, choose **Set notifications for the alarm** and select or create an Amazon Simple Notification Service topic to send alarm notifications to. For more information, see [Setting up Amazon SNS notifications](#),

10. (Optional) To add tags to this experiment, choose **Tags, Add new tag**.

Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see [Tagging AWS Resources](#).

11. Choose **Create experiment**.
12. If you haven't already, build the feature variants into your application.
13. Choose **Done**. The experiment does not start until you start it.

After you complete the steps in the following procedure, the experiment starts immediately.

To start an experiment that you have created

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project.
4. Choose the **Experiments** tab.
5. Choose the button next to the name of the experiment, and choose **Actions, Start experiment**.
6. (Optional) To view or modify the experiment settings you made when you created it, choose **Experiment setup**.
7. Choose a time for the experiment to end.
8. Choose **Start experiment**.

The experiment starts immediately.

Manage features, launches, and experiments

Use the procedures in these sections to manage the features, launches, and experiments that you have created.

Topics

- [See the current evaluation rules and audience traffic for a feature](#)
- [Modify launch traffic](#)
- [Modify a launch's future steps](#)
- [Modify experiment traffic](#)
- [Stop a launch](#)
- [Stop an experiment](#)

See the current evaluation rules and audience traffic for a feature

You can use the CloudWatch Evidently console to see how the feature's evaluation rules are allocating the audience traffic among the feature's current launches, experiments, and variations.

To view the audience traffic for a feature

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the feature.
4. Choose the **Features** tab.
5. Choose the name of the feature.

In the **Evaluation rules** tab, you can see the flow of audience traffic for your feature, as follows:

- First, the overrides are evaluated. These specify that certain users are always served a specific variation. The sessions of users who are assigned overrides do not contribute to launch or experiment metrics.

- Next, the remaining traffic is available for the ongoing launch, if there is one. If there is a launch in progress, the table in the **Launches** section displays the launch name and the launch traffic split among the feature variations. On the right side of the **Launches** section, a **Traffic** indicator displays how much of the available audience (after overrides) is allocated to this launch. The rest of the traffic not allocated to the launch flows to the experiment (if any) and then the default variation.
- Next, the remaining traffic is available for the ongoing experiment, if there is one. If there is an experiment in progress, the table in the **Experiments** section displays the experiment name and progress. On the right side of the **Experiments** section, a **Traffic** indicator displays how much of the available audience (after overrides and launches) is allocated to this experiment. The rest of the traffic not allocated to the launch or the experiment is served the default variation of the feature.

Modify launch traffic

You can modify the traffic allocation for a launch at any time, including while the launch is ongoing.

If you have both an ongoing launch and an ongoing experiment for the same feature, any changes to the feature traffic will cause a change in the experiment traffic. This is because the audience available to the experiment is the portion of your total audience that is not already allocated to the launch. Increasing launch traffic will decrease the audience available to the experiment, and decreasing launch traffic or ending the launch will increase the audience available to the experiment.

To modify the traffic allocation for a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launches** tab.
5. Choose the name of the launch.

Choose **Modify launch traffic**.

6. For **Serve**, select the new traffic percentage to assign to each variation. You can also choose to exclude variations from being served to users. As you change these values, you can see the updated effects on your overall feature traffic under **Traffic summary**.

The **Traffic summary** shows how much of your overall traffic is available for this launch, and how much of that available traffic is allocated to this launch.

7. Choose **Modify**.

Modify a launch's future steps

You can modify the configuration of launch steps that haven't happened yet, and also add more steps to a launch.

To modify the steps for a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launches** tab.
5. Choose the name of the launch.

Choose **Modify launch traffic**.

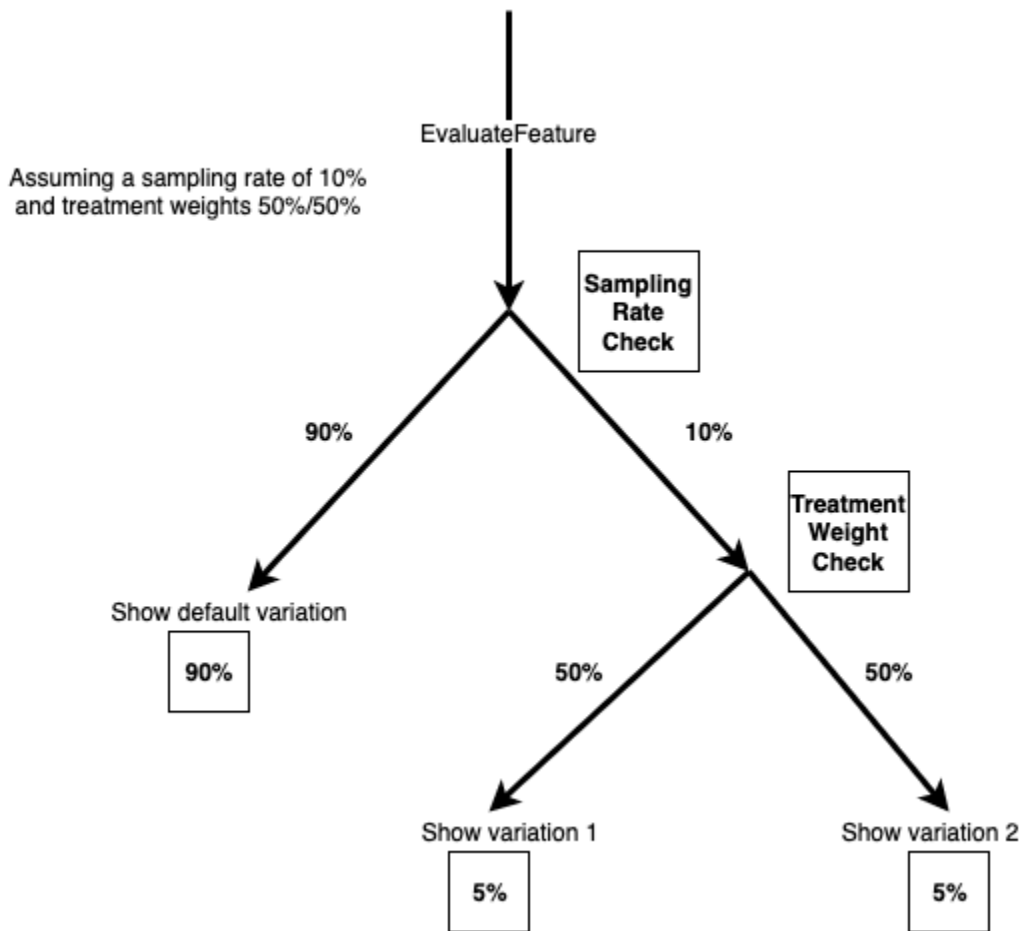
6. Choose **Schedule launch**.
7. For any steps that have not started yet, you can modify the percentage of the available audience to use in the experiment. You can also modify how their traffic is allocated among the variations.

You can add more steps to the launch by choosing **Add another step**. A launch can have a maximum of five steps.

8. Choose **Modify**.

Modify experiment traffic

You can modify the sampling rate for an experiment at any time, including while the experiment is ongoing. However, you can't update the treatment weights after an experiment is running. Therefore, you can change the total traffic exposed to the experiment after an experiment is running, but not the relative allocation to each treatment. If you modify the traffic of an ongoing experiment, we recommend that you only increase the traffic allocation, so that you don't introduce bias.



To modify the traffic allocation for an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Experiments** tab.
5. Choose the name of the launch.
6. Choose **Modify experiment traffic**.
7. Enter a percentage or use the slider to specify how much of the available traffic to allocate to this experiment. The available traffic is the total audience minus the traffic that is allocated to a current launch, if there is one. The traffic that is not allocated to the launch or experiment is served the default variation.
8. Choose **Modify**.

Stop a launch

If you stop an ongoing launch, you will not be able to resume it or restart it. Also, it will not be evaluated as a rule for traffic allocation, and the traffic that was allocated to the launch will instead be available to the feature's experiment, if there is one. Otherwise, all traffic will be served the default variation after the launch is stopped.

To permanently stop a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the button to the left of the name of the launch.
6. Choose **Actions, Cancel launch** or **Actions, Mark as complete**.

Stop an experiment

If you stop an ongoing experiment, you will not be able to resume it or restart it. The portion of traffic that was previously used in the experiment will be served the default variation.

When an experiment is not manually stopped and passes its end date, the traffic does not change. The portion of traffic allocated to the experiment still goes to the experiment. To stop this, and cause the experiment traffic to instead be served the default variation, mark the experiment as complete.

When you stop an experiment, you can choose to cancel it or mark it as complete. If you cancel, it will be shown as **Cancelled** in the list of experiments. If you choose to mark it as complete, it is shown as **Completed**.

To permanently stop an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.

5. Choose the button to the left of the name of the experiment.
6. Choose **Actions, Cancel experiment** or **Actions, Mark as complete**.

Adding code to your application

To work with CloudWatch Evidently, you add code to your application to assign a variation to each user session, and to send metrics to Evidently. You use the CloudWatch Evidently `EvaluateFeature` operation to assign variations to user sessions, and you use the `PutProjectEvents` operation to send events to Evidently to be used to calculate metrics for your launches or experiments.

When you create variations or custom metrics, the CloudWatch Evidently console provides samples of the code you need to add.

For an end-to-end example, see [Tutorial: A/B testing with the Evidently sample application](#).

Using EvaluateFeature

When feature variations are used in a launch or experiment, the application uses the [EvaluateFeature](#) operation to assign each user session a variation. The assignment of a variation to a user is an *evaluation event*. When you call this operation, you pass in the following:

- **Feature name**– Required. Evidently processes the evaluation according to the feature evaluation rules of the launch or experiment, and selects a variation for the entity.
- **entityId**– Required. Represents a unique user.
- **evaluationContext**– Optional. A JSON object representing additional information about a user. Evidently will use this value to match the user to a *segment* of your audience during feature evaluations, if you have created segments. For more information, see [Use segments to focus your audience](#).

The following is an example of an `evaluationContext` value that you can send to Evidently.

```
{
  "Browser": "Chrome",
  "Location": {
    "Country": "United States",
    "Zipcode": 98007
  }
}
```

```
}
```

Sticky evaluations

CloudWatch Evidently uses "sticky" evaluations. A single configuration of `entityId`, `feature`, `feature configuration`, and `evaluationContext` always receives the same variation assignment. The only time this variation assignment changes is when an entity is added to an override or the experiment traffic is dialed up.

A feature configuration includes the following:

- The feature variations
- The variation configuration (percentages assigned to each variation) for a currently running experiment for this feature, if any.
- The variation configuration for a currently running launch for this feature, if any. The variation configuration includes the defined segment overrides, if any.

If an experiment's traffic allocation is *increased*, any `entityId` that was previously assigned to an experiment treatment group will continue to receive the same treatment. Any `entityId` that was previously assigned to the control group, might be assigned to an experiment treatment group, according to the variation configuration specified for the experiment.

If an experiment's traffic allocation is *decreased*, an `entityId` might go from a treatment group to a control group, but it will not go into a different treatment group.

Using PutProjectEvents

To code a custom metric for Evidently, you use the [PutProjectEvents](#) operation. The following is a simple payload example.

```
{
  "events": [
    {
      "timestamp": {{$timestamp}},
      "type": "aws.evidently.custom",
      "data": "{\"details\": {\"pageLoadTime\": 800.0}, \"userDetails\": {\"userId\": \"test-user\"}}"
```

```
}
```

The `entityIdKey` can just be an `entityId` or you can rename it to anything else, such as `userId`. In the actual event, `entityId` can be a username, a session ID, and so on.

```
"metricDefinition":{
  "name": "noFilter",
  "entityIdKey": "userDetails.userId", //should be consistent with jsonValue in
  events "data" fields
  "valueKey": "details.pageLoadTime"
},
```

To ensure that events are associated with the correct launch or experiment, you must pass the same `entityId` when you call both `EvaluateFeature` and `PutProjectEvents`. Be sure to call `PutProjectEvents` after the `EvaluateFeature` call, otherwise data is dropped and won't be used by CloudWatch Evidently.

The `PutProjectEvents` operation does not require the feature name as an input parameter. This way, you can use a single event in multiple experiments. For example, suppose you call `EvaluateFeature` with the `entityId` set to `userDetails.userId`. If you have two or more experiments running, you can have a single event from that user's session emit metrics for each of those experiments. To do this, you call `PutProjectEvents` once for each experiment, using that same `entityId`.

Timing

After your application calls `EvaluateFeature`, there is a one-hour time period where metric events from `PutProjectEvents` are attributed based on that evaluation. If any more events occur after the one-hour period, they are not attributed.

However, if the same `entityId` is used for a new `EvaluateFeature` call during that initial call's one-hour window, the later `EvaluateFeature` result is now used instead, and the one-hour timer is restarted. This can only happen in certain circumstances, such as when experiment traffic is dialed up between the two assignments, as explained in the previous **Sticky evaluations** section.

For an end-to-end example, see [Tutorial: A/B testing with the Evidently sample application](#).

Project data storage

Evidently collects two types of events:

- **Evaluation events** are related to which feature variation is assigned to a user session. Evidently uses these events to produce metrics and other experiment and launch data, which you can view in the Evidently console.

You can also choose to store these evaluation events in Amazon CloudWatch Logs or Amazon S3.

- **Custom events** are used to generate metrics from user actions such as clicks and checkouts. Evidently doesn't provide a method for you to store custom events. If you want to save them, you must modify your application code to send them to a storage option outside of Evidently.

Format of evaluation event logs

If you choose to store evaluation events in CloudWatch Logs or Amazon S3, each evaluation event is stored as a log event with the following format:

```
{
  "event_timestamp": 1642624900215,
  "event_type": "evaluation",
  "version": "1.0.0",
  "project_arn": "arn:aws:evidently:us-east-1:123456789012:project/petfood",
  "feature": "petfood-upsell-text",
  "variation": "Variation1",
  "entity_id": "7",
  "entity_attributes": {},
  "evaluation_type": "EXPERIMENT_RULE_MATCH",
  "treatment": "Variation1",
  "experiment": "petfood-experiment-2"
}
```

Here are more details about the preceding evaluation event format:

- The timestamp is in UNIX time with milliseconds
- The variation is the name of the variation of the feature which was assigned to this user session.
- The entity ID is a string.
- Entity attributes are a hash of arbitrary values sent by the client. For example, if the `entityId` is mapped to blue or green, then you can optionally send userIDs, session data, or whatever else that you want from a correlation and data warehouse perspective.

IAM policy and encryption for evaluation event storage in Amazon S3

If you choose to use Amazon S3 to store evaluation events, you must add an IAM policy like the following to allow Evidently to publish logs to the Amazon S3 bucket. This is because Amazon S3 buckets and the objects they contain are private, and they don't allow access to other services by default.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AWSLogDeliveryWrite",
      "Effect": "Allow",
      "Principal": {"Service": "delivery.logs.amazonaws.com"},
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::bucket_name/optional_folder/AWSLogs/account_id/*",
      "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}}
    },
    {
      "Sid": "AWSLogDeliveryCheck",
      "Effect": "Allow",
      "Principal": {"Service": "delivery.logs.amazonaws.com"},
      "Action": ["s3:GetBucketAcl", "s3:ListBucket"],
      "Resource": "arn:aws:s3:::bucket_name"
    }
  ]
}
```

If you store Evidently data in Amazon S3, you can also choose to encrypt it with Server-Side Encryption with AWS Key Management Service Keys (SSE-KMS). For more information, see [Protecting data using server-side encryption](#).

If you use a customer managed key from AWS KMS, you must add the following to the IAM policy for your key. This allows Evidently to write to the bucket.

```
{
  "Sid": "AllowEvidentlyToUseCustomerManagedKey",
  "Effect": "Allow",
  "Principal": {
    "Service": [
```



```
        "delivery.logs.amazonaws.com"
    ]
},
"Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
],
"Resource": "*"
}
```

How Evidently calculates results

You can use Amazon CloudWatch Evidently A/B testing as a tool for data-driven decision making. In an A/B test, users are randomly assigned to either the control group (also called the default variation), or one of the treatment groups (also called the tested variations). For example, users in the control group might experience the website, service, or application in the same way that they did before the experiment started. Meanwhile, users in the treatment group might experience the change.

CloudWatch Evidently supports up to five different variations in an experiment. Evidently randomly assigns traffic to these variations. This way, you can track business metrics (such as revenue) and performance metrics (such as latency) for each group. Evidently does the following:

- Compares the treatment with the control. (For example, compares whether revenue increases or decreases with a new checkout process.)
- Indicates whether the observed difference between the treatment and the control is *significant*. For this, Evidently offers two approaches: *Frequentist significance levels* and *Bayesian probabilities*.

Why use Frequentist and Bayesian approaches?

Consider a case where the treatment has no effect compared to the control, or a case where the treatment is identical to the control (an *A/A test*). You would still observe a small difference between the treatment and the control in the data. This is because the test participants consist of a finite sample of users, representing a small percentage of all users of the website, service, or

application. Frequentist significance levels and Bayesian probabilities provide insights into whether the observed difference is significant or due to chance.

Evidently considers the following to determine whether the observed difference is significant:

- How big the difference is
- How many samples are part of the test
- How the data is distributed

Frequentist analysis in Evidently

Evidently uses sequential testing, which avoids the usual problems of *peeking*, a common pitfall of frequentist statistics. Peeking is the practice of checking the results of an ongoing A/B test in order to stop it and make a decision based on the observed results. For more information about sequential testing, see [Time-uniform, nonparametric, nonasymptotic confidence sequences](#) by Howard et al. (Ann. Statist. 49 (2) 1055 - 1080, 2021).

Because Evidently's results are valid at any time (*anytime-valid* results), you can peek at results during the experiment and still draw sound conclusions. This can reduce some of the costs of experimentation, because you can stop an experiment before the scheduled time if the results already have significance.

Evidently generates anytime-valid significance levels and anytime-valid 95% confidence intervals of the difference between the tested variation and the default variation in the target metric. The **Result** column in the experiment results indicates the tested variation performance, which can be one of the following:

- **Inconclusive** – The significance level is less than 95%
- **Better** – The significance level is 95% or higher and one of the following is true:
 - The lower bound of the 95%-confidence interval is higher than zero and the metric should increase
 - The upper bound of the 95%-confidence interval is lower than zero and the metric should decrease
- **Worse** – The significance level is 95% or higher and one of the following is true:
 - The upper bound of the 95%-confidence interval is higher than zero and the metric should increase

- The lower bound of the 95%-confidence interval is lower than zero and the metric should decrease
- **Best** – The experiment has two or more tested variations in addition to the default variation, and the following conditions are met:
 - The variation qualifies for the *Better* designation
 - One of the following is true:
 - The lower bound of the 95%-confidence interval is higher than the upper bound of the 95%-confidence intervals of all the other variations and the metric should increase
 - The upper bound of the 95%-confidence interval is lower than the lower bound of the 95%-confidence intervals of all the other variations and the metric should decrease

Bayesian analysis in Evidently

With Bayesian analysis, you can calculate the probability that the mean in the tested variation is larger or smaller than the mean in the default variation. Evidently performs Bayesian inference for the mean of the target metric by using *conjugate priors*. With conjugate priors, Evidently can more efficiently infer the posterior distribution needed for the Bayesian analysis.

Evidently waits until the end date of the experiment to compute the results of the Bayesian analysis. The results page displays the following:

- *probability of increase* – The probability that the mean of the metric in the tested variation is at least 3% larger than the mean in the default variation
- *probability of decrease* – The probability that the mean of the metric in the tested variation is at least 3% smaller than the mean in the default variation
- *probability of no change* – The probability that the mean of the metric in the tested variation lies within $\pm 3\%$ of the mean in the default variation

The **Result** column indicates the performance of the variation, and can be one of the following:

- **Better** – The probability of increase is at least 90% and the metric should increase, or the probability of decrease is at least 90% and the metric should decrease
- **Worse** – The probability of decrease is at least 90% and the metric should increase, or the probability of increase is at least 90% and the metric should decrease

View launch results in the dashboard

You can see the progress and metric results of an experiment while it is ongoing and after it is completed.

To see the progress and results of a launch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the launch.
4. Choose the **Launch** tab.
5. Choose the name of the launch.
6. To see the launch steps and the traffic allocations for each step, choose the **Launch** tab.
7. To see the number of user sessions assigned to each variation over time, and to view the performance metrics for each variation in the launch, choose the **Monitoring** tab.

This view also displays whether any launch alarms have gone into ALARM state during the launch.

8. To see the variations, metrics, alarms, and tags for this launch, choose the **Configuration** tab.

View experiment results in the dashboard

You can see the statistical results of an experiment while it is ongoing and after it is completed. Experiment results are available up to 63 days after the start of the experiment. They are not available after that because of CloudWatch data retention policies.

No statistical results are displayed until each variation has at least 100 events.

Evidently performs an additional offline p-value analysis at the end of the experiment. Offline p-value analysis can detect statistical significance in some cases where the anytime p-values used during the experiment do not find statistical significance.

For more information about how CloudWatch Evidently calculates experiment results, see [How Evidently calculates results](#).

To see the results of an experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the name of the project that contains the experiment.
4. Choose the **Experiments** tab.
5. Choose the name of the experiment, and then choose the **Results** tab.
6. By **Variation performance**, there is a control where you can select which experiment statistics to display. If you select more than one statistic, Evidently displays a graph and table for each statistic.

Each graph and table displays the results of the experiment so far.

Each graph can display the following results. You can use the control at the right of the graph to determine which of the following items is displayed:

- The number of user session events recorded for each variation.
- The average value of the metric that is selected at the top of the graph, for each variation.
- The statistical significance of the experiments. This compares the difference for the metric selected at the top of the graph with the default variation and each of the other variations.
- The 95% upper and lower confidence bounds on the difference of the selected metric, between each of the variations and the default variation.

The table displays a row for each variation. For each variation that is not the default, Evidently displays whether it has received enough data to declare the results statistically significant. It also shows whether the variation's improvement in the statistical value has reached a 95% confidence level.

Finally, in the **Result** column, Evidently provides a recommendation about which variation performs best based on this statistic, or whether the results are inconclusive.

How CloudWatch Evidently collects and stores data

Amazon CloudWatch Evidently collects and stores data related to project configurations so that customers can run experiments and launches. The data includes the following:

- Metadata about projects, features, launches, and experiments
- Metric events
- Evaluation data

Resource metadata is stored in Amazon DynamoDB. The data is encrypted at rest by default, using AWS owned keys. These keys are a collection of AWS KMS keys that an AWS service owns and manages for use in multiple AWS accounts. Customers can't view, manage, or audit the use of these keys. Customers are also not required to take action or change programs to protect the keys that encrypt their data.

For more information, see [AWS owned keys](#) in the AWS Key Management Service Developer Guide.

Evidently metric events and evaluation events are delivered directly to customer-owned locations.

Data in transit is automatically encrypted with HTTPS. This data will be delivered to customer-owned locations.

You can also choose to store evaluation events in Amazon Simple Storage Service or Amazon CloudWatch Logs. For more information about how you can secure your data in these services, see [Enabling Amazon S3 default bucket encryption](#) and [Encrypting log data in CloudWatch Logs using AWS KMS](#).

Retrieving data

You can retrieve your data using CloudWatch Evidently APIs. To retrieve project data, use [GetProject](#) or [ListProjects](#).

To retrieve feature data, use [GetFeature](#) or [ListFeatures](#).

To retrieve launch data, use [GetLaunch](#) or [ListLaunches](#).

To retrieve experiment data, use [GetExperiment](#), [ListExperiments](#), or [GetExperimentResults](#).

Modifying and deleting data

You can modify and delete your data using CloudWatch Evidently APIs. For project data, use [UpdateProject](#) or [DeleteProject](#).

For feature data, use [UpdateFeature](#) or [DeleteFeature](#).

For launch data, use [UpdateLaunch](#) or [DeleteLaunch](#).

For experiment data, use [UpdateExperiment](#) or [DeleteExperiment](#).

Using service-linked roles for Evidently

CloudWatch Evidently uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Evidently. Service-linked

roles are predefined by Evidently and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Evidently easier because you don't have to manually add the necessary permissions. Evidently defines the permissions of its service-linked roles, and unless defined otherwise, only Evidently can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your Evidently resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Evidently

Evidently uses the service-linked role named **AWSServiceRoleForCloudWatchEvidently** – Allows CloudWatch Evidently to manage associated AWS resources on behalf of the customer.

The **AWSServiceRoleForCloudWatchEvidently** service-linked role trusts the following services to assume the role:

- CloudWatch Evidently

The role permissions policy named **AmazonCloudWatchEvidentlyServiceRolePolicy** allows Evidently to complete the following actions on the specified resources:

- Actions: `appconfig:StartDeployment`, `appconfig:StopDeployment`, `appconfig:ListDeployments`, and `appconfig:TagResource` on Evidently thick clients.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.

Creating a service-linked role for Evidently

You don't need to manually create a service-linked role. When you start using an Evidently thick client in the AWS Management Console, the AWS CLI, or the AWS API, Evidently creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you start using an Evidently thick client, Evidently creates the service-linked role for you again.

Editing a service-linked role for Evidently

Evidently does not allow you to edit the `AWSServiceRoleForCloudWatchEvidently` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Evidently

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it. You must delete all Evidently projects that are using thick clients.

Note

If the Evidently service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Evidently resources used by `AWSServiceRoleForCloudWatchEvidently`

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application monitoring, Evidently**.
3. In the list of projects, select the check box next to the projects that used thick clients.
4. Choose **Project actions, Delete project**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForCloudWatchEvidently` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Supported regions for Evidently service-linked roles

Evidently supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS regions and endpoints](#).

CloudWatch Evidently quotas

CloudWatch Evidently has the following quotas.

Resource	Default quota
Projects	50 per Region per account You can request a quota increase.
Segments	500 per Region per account You can request a quota increase.
Quotas per project	<ul style="list-style-type: none"> • 100 total features • 500 total launches • 50 running launches • 500 total experiments • 50 running experiments <p>You can request a quota increase for all of these quotas.</p>
API quotas (all quotas are per Region)	<ul style="list-style-type: none"> • <code>PutProjectEvents</code>: 1000 transactions per second (TPS) in US East (N. Virginia), US West (Oregon), and Europe (Ireland). 200 TPS in all other Regions. • <code>EvaluateFeature</code>: 1000 TPS in US East (N. Virginia), US West (Oregon), and Europe (Ireland). 200 TPS in all other Regions. • <code>BatchEvaluateFeature</code>: 50 TPS

Resource	Default quota
	<ul style="list-style-type: none">Create, Read, Update, Delete (CRUD) APIs: 10 TPS combined across all CRUD APIs <p>You can request a quota increase for all of these quotas.</p>

Tutorial: A/B testing with the Evidently sample application

This section provides a tutorial for using Amazon CloudWatch Evidently for A/B testing. This tutorial the Evidently sample application, which is a simple react application. The sample app will be configured to either display a `showDiscount` feature or not. When the feature is shown to a user, the price displayed on the shopping website it shown at a 20% discount.

In addition to showing the discount to some users and not to others, in this tutorial you set up Evidently to collect page load time metrics from both variations.

Warning

This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed. Access keys can be updated if necessary. For more information, see [Updating access keys](#) in the *IAM User Guide*.

Step 1: Download the sample application

Start by downloading the Evidently sample application.

To download the sample application

1. Download the sample application from the following Amazon S3 bucket:

```
https://evidently-sample-application.s3.us-west-2.amazonaws.com/evidently-sample-shopping-app.zip
```

2. Unzip the package.

Step 2: Add the Evidently endpoint and set up credentials

Next, add the Region and endpoint for Evidently to the `config.js` file in the `src` directory in the sample app package, as in the following example:

```
evidently: {  
  REGION: "us-west-2",  
  ENDPOINT: "https://evidently.us-west-2.amazonaws.com (https://evidently.us-west-2.amazonaws.com/)",  
},
```

You also must make sure that the application has permission to call CloudWatch Evidently.

To grant the sample app permissions to call Evidently

1. Federate to your AWS account.
2. Create an IAM user and attach the **AmazonCloudWatchEvidentlyFullAccess** policy to this user.
3. Make a note of the IAM user's access key id and secret access key, because you will need them in the next step.
4. In the same `config.js` file that you modified earlier in this section, enter the values of the access key ID and the secret access key, as in the following example:

```
credential: {  
  accessKeyId: "Access key ID",  
  secretAccessKey: "Secret key"  
}
```

Important

We use this step to make the sample app as simple as possible for you to try out. We do not recommend that you put your IAM user credential into your actual production application. Instead, we recommend that you use Amazon Cognito for authentication. For more information, see [Integrating Amazon Cognito with web and mobile apps](#).

Step 3: Set up code for the feature evaluation

When you use CloudWatch Evidently to evaluate a feature, you must use the **EvaluateFeature** operation to randomly select a feature variation for each user session. This operation assigns user

sessions to each variation of the feature, according to the percentages that you specified in the experiment.

To set up the feature evaluation code for the bookstore demo app

1. Add the client builder in the `src/App.jsx` file so that the sample app can call Evidently.

```
import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
  region,
) => {
  const credentials = {
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
    region,
    credentials,
  });
};
```

2. Add the following to the `const App` code section to initiate the client.

```
if (client == null) {
  client = defaultClientBuilder(
    config.evidently.ENDPOINT,
    config.evidently.REGION,
  );
}
```

3. Construct `evaluateFeatureRequest` by adding the following code. This code pre-fills the project name and feature name that we recommend later in this tutorial. You can substitute your own project and feature names, as long as you also specify those project and feature names in the Evidently console.

```
const evaluateFeatureRequest = {
  entityId: id,
  // Input Your feature name
  feature: 'showDiscount',
}
```

```
    // Input Your project name'  
    project: 'EvidentlySampleApp',  
  };
```

4. Add the code to call Evidently for feature evaluation. When the request is sent, Evidently randomly assigns the user session to either see the `showDiscount` feature or not.

```
client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {  
  if(res.value?.boolValue !== undefined) {  
    setShowDiscount(res.value.boolValue);  
  }  
  getPageLoadTime()  
})
```

Step 4: Set up code for the experiment metrics

For the custom metric, use Evidently's `PutProjectEvents` API to send metric results to Evidently. The following examples show how to set up the custom metric and send experiment data to Evidently.

Add the following function to calculate the page load time and use `PutProjectEvents` to send the metric values to Evidently. Add the following function to `Home.tsx` and call this function within the `EvaluateFeature` API:

```
const getPageLoadTime = () => {  
  const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;  
  const pageLoadTimeData = `{  
    "details": {  
      "pageLoadTime": ${timeSpent}  
    },  
    "UserDetails": { "userId": "${id}", "sessionId": "${id}"  
  }`;  
  const putProjectEventsRequest = {  
    project: 'EvidentlySampleApp',  
    events: [  
      {  
        timestamp: new Date(),  
        type: 'aws.evidently.custom',  
        data: JSON.parse(pageLoadTimeData)  
      },  
    ],  
  ],  
}
```

```
};
  client.putProjectEvents(putProjectEventsRequest).promise();
}
```

Here is what the `App.js` file should look like after all the editing that you have done since downloading it.

```
import React, { useEffect, useState } from "react";
import { BrowserRouter as Router, Switch } from "react-router-dom";
import AuthProvider from "contexts/auth";
import CommonProvider from "contexts/common";
import ProductsProvider from "contexts/products";
import CartProvider from "contexts/cart";
import CheckoutProvider from "contexts/checkout";
import RouteWrapper from "layouts/RouteWrapper";
import AuthLayout from "layouts/AuthLayout";
import CommonLayout from "layouts/CommonLayout";
import AuthPage from "pages/auth";
import HomePage from "pages/home";
import CheckoutPage from "pages/checkout";
import "assets/scss/style.scss";
import { Spinner } from 'react-bootstrap';

import Evidently from 'aws-sdk/clients/evidently';
import config from './config';

const defaultClientBuilder = (
  endpoint,
  region,
) => {
  const credentials = {
    accessKeyId: config.credential.accessKeyId,
    secretAccessKey: config.credential.secretAccessKey
  }
  return new Evidently({
    endpoint,
    region,
    credentials,
  });
};

const App = () => {
  const [isLoading, setIsLoading] = useState(true);
```

```
const [startTime, setStartTime] = useState(new Date());
const [showDiscount, setShowDiscount] = useState(false);
let client = null;
let id = null;

useEffect(() => {
  id = new Date().getTime().toString();
  setStartTime(new Date());
  if (client == null) {
    client = defaultClientBuilder(
      config.evidently.ENDPOINT,
      config.evidently.REGION,
    );
  }
  const evaluateFeatureRequest = {
    entityId: id,
    // Input Your feature name
    feature: 'showDiscount',
    // Input Your project name'
    project: 'EvidentlySampleApp',
  };

  // Launch
  client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
      setShowDiscount(res.value.boolValue);
    }
  });

  // Experiment
  client.evaluateFeature(evaluateFeatureRequest).promise().then(res => {
    if(res.value?.boolValue !== undefined) {
      setShowDiscount(res.value.boolValue);
    }
    getPageLoadTime()
  })

  setIsLoading(false);
}, []);

const getPageLoadTime = () => {
  const timeSpent = (new Date().getTime() - startTime.getTime()) * 1.000001;
  const pageLoadTimeData = `{
    "details": {
```



```
        </CartProvider>
      </ProductsProvider>
    </CommonProvider>
  </AuthProvider> ) : (
    <Spinner animation="border" />
  )
);
};

export default App;
```

Each time a user visits the sample app, a custom metric is sent to Evidently for analysis. Evidently analyzes each metric and displays results in real time on the Evidently dashboard. The following example shows a metric payload:

```
[ {"timestamp": 1637368646.468, "type": "aws.evidently.custom", "data": "{\"details\n\":{\n\"pageLoadTime\n\":2058.002058},\n\"userDetails\n\":{\n\"userId\n\":\n\"1637368644430\n\", \n\"sessionId\n\":\n\"1637368644430\n\"}}\" } ]
```

Step 5: Create the project, feature, and experiment

Next, you create the project, feature, and experiment in the CloudWatch Evidently console.

To create the project, feature, and experiment for this tutorial

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose **Create project** and fill out the fields. You must use **EvidentlySampleApp** for the project name for the sample to work correctly. For **Evaluation event storage**, choose **Don't store Evaluation events**.

After filling out the fields, choose **Create Project**.

For more details, see [Create a new project](#).

4. After the project is created, create a feature in that project. Name the feature **showDiscount**. In this feature, create two variations of the **Boolean** type. Name the first variation **disable** with a value of **False** and name the second variation **enable** with a value of **True**.

For more information about creating a feature, see [Add a feature to a project](#).

5. After you have finished creating the feature, create an experiment in the project. Name the experiment **pageLoadTime**.

This experiment will use a custom metric called `pageLoadTime` that measures the page load time of the page being tested. Custom metrics for experiments are created using Amazon EventBridge. For more information about EventBridge, see [What Is Amazon EventBridge?](#).

To create that custom metric, do the following when you create the experiment:

- Under **Metrics**, for **Metric source**, choose **Custom metrics**.
- For **Metric name**, enter **pageLoadTime**.
- For **Goal** choose **Decrease**. This indicates that we want a lower value of this metric to indicate the best variation of the feature.
- For **Metric rule**, enter the following:
 - For **Entity ID**, enter **UserDetails.userId**.
 - For **Value key**, enter **details.pageLoadTime**.
 - For **Units**, enter **ms**.
- Choose **Add metric**.

For **Audiences**, select **100%** so that all users are entered in the experiment. Set up the traffic split between the variations to be 50% each.

Then, choose **Create experiment** to create the experiment. After you create it, it does not start until you tell Evidently to start it.

Step 6: Start the experiment and test CloudWatch Evidently

The final steps are starting the experiment and starting the sample app.

To start the tutorial experiment

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Application Signals, Evidently**.
3. Choose the **EvidentlySampleApp** project.
4. Choose the **Experiments** tab.
5. Choose the button next to **pageLoadTime** and choose **Actions, Start experiment**.

6. Choose a time for the experiment to end.
7. Choose **Start experiment**.

The experiment starts immediately.

Next, start the Evidently sample app with the following command:

```
npm install -f && npm start
```

Once the app has started, you will be assigned to one of the two feature variations being tested. One variation displays "20% discount" and the other doesn't. Keep refreshing the page to see the different variations.

Note

Evidently has sticky evaluations. Feature evaluations are deterministic, meaning for the same `entityId` and feature, a user will always receive the same variation assignment. The only time variation assignments change is when an entity is added to an override or experiment traffic is dialed up.

However, to make the use of the sample app tutorial easy for you, Evidently reassigns the sample app feature evaluation every time that you refresh the page, so that you can experience both variations without having to add overrides.

Troubleshooting

We recommend that you use npm version 6.14.14. If you see any errors about building or starting the sample app and you are using a different version of npm, do the following.

To install npm version 6.14.14

1. Use a browser to connect to <https://nodejs.org/download/release/v14.17.5/>.
2. Download [node-v14.17.5.pkg](#) and run this pkg to install npm.

If you see a `webpack not found` error, navigate to the `evidently-sample-shopping-app` folder and try the following:

- a. Delete `package-lock.json`

- b. Delete `yarn-lock.json`
- c. Delete `node_modules`
- d. Delete the webpack dependency from `package.json`
- e. Run the following:

```
npm install -f && npm
```

Network monitoring

The topics in this section describe CloudWatch network and internet monitoring capabilities provided by Amazon CloudWatch Internet Monitor and Amazon CloudWatch Network Monitor. These services help you to gain operational visibility into the network and internet performance and availability of your applications hosted on AWS.

- Internet Monitor uses the connectivity data that AWS captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. You can see a global view of traffic patterns and health events, and easily drill down into information about events. You can also get alerts for internet health events that affect your application clients. In addition, you can use insights that Internet Monitor provides to explore potential improvements to your client experience, by using Amazon CloudFront or routing through different AWS Regions.
- Network Monitor uses a fully-managed agent approach to enable you to track and visualize latency and packet loss for hybrid network connections. To gather measurements and enable Network Monitor to create health event alerts for your application, you create probes that are sent from your resources hosted on AWS to on-premises destination IP addresses. You don't need to install additional agents to monitor your network performance. As with Internet Monitor, you can set alerts and thresholds, get information to help you quickly troubleshoot issues, and then take action to improve your end user experience.

Topics

- [Using Amazon CloudWatch Internet Monitor](#)
- [Using Amazon CloudWatch Network Monitor](#)

Using Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor provides visibility into how internet issues impact the performance and availability between your applications hosted on AWS and your end users. It can reduce the time it takes for you to diagnose internet issues from days to minutes. Internet Monitor uses the connectivity data that AWS captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. This is the same data that AWS uses to monitor internet uptime and availability. With those measurements as a baseline, Internet

Monitor raises awareness for you when there are significant problems for your end users (clients) in the different geographic locations where your application runs.

In the Amazon CloudWatch console, you can see a global view of traffic patterns and health events, and easily drill down into information about events, at different geographic granularities (locations). You can clearly visualize impact, and pinpoint the client locations and networks (ASNs, typically internet service providers or ISPs) that are affected. If Internet Monitor determines that an internet availability or performance issue is caused by a specific ASN or by the AWS network, it provides that information.

Key features of Internet Monitor

- Internet Monitor suggests insights and recommendations that can help you improve your end users' experience. You can explore, in near real-time, how to improve the projected latency of your application by switching to use other services, or by rerouting traffic to your workload through different AWS Regions.
- With Internet Monitor, you can quickly identify what's impacting your application's performance and availability, so that you can track down and address issues.
- Internet Monitor publishes internet measurements to CloudWatch Logs and CloudWatch Metrics, to support using CloudWatch tools with health information for locations and ASNs (internet service providers) specific to your application. Optionally, you can also publish internet measurements to Amazon S3.
- Internet Monitor sends health events to Amazon EventBridge so that you can set up notifications. If an issue is caused by the AWS network, you also automatically receive an AWS Health Dashboard notification with the steps that AWS is taking to mitigate the problem.

How to use Internet Monitor

To use Internet Monitor, you create a *monitor* and associate your application's resources with it—VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories—to enable Internet Monitor to know where your application's internet-facing traffic is. Internet Monitor then publishes internet measurements from AWS that are specific to the *city-networks*, that is, the client locations and ASNs (typically internet service providers or ISPs), where clients access your application. For more information, see [How Amazon CloudWatch Internet Monitor works](#). To begin working with Internet Monitor, see [Getting started with Amazon CloudWatch Internet Monitor using the console](#).

Contents

- [Supported AWS Regions for Amazon CloudWatch Internet Monitor](#)
- [Pricing for Amazon CloudWatch Internet Monitor](#)
- [Components and terms for Amazon CloudWatch Internet Monitor](#)
- [Global internet weather map in Amazon CloudWatch Internet Monitor](#)
- [How Amazon CloudWatch Internet Monitor works](#)
- [Amazon CloudWatch Internet Monitor example use cases](#)
- [Internet Monitor cross-account observability](#)
- [Getting started with Amazon CloudWatch Internet Monitor using the console](#)
- [Examples of using the CLI with Amazon CloudWatch Internet Monitor](#)
- [Monitor and optimize with the Internet Monitor dashboard](#)
- [Exploring your data with CloudWatch tools and the Internet Monitor query interface](#)
- [Creating alarms with Amazon CloudWatch Internet Monitor](#)
- [Using Amazon CloudWatch Internet Monitor with Amazon EventBridge](#)
- [Troubleshoot CloudWatch logs and metrics access errors](#)
- [Data protection and data privacy with Amazon CloudWatch Internet Monitor](#)
- [Identity and Access Management for Amazon CloudWatch Internet Monitor](#)
- [Quotas in Amazon CloudWatch Internet Monitor](#)

Supported AWS Regions for Amazon CloudWatch Internet Monitor

The AWS Regions where Amazon CloudWatch Internet Monitor is supported are listed in this section. For a current list of Regions that Internet Monitor is supported in, including opt-in Regions, see [Amazon CloudWatch Internet Monitor endpoints and quotas](#) in the *Amazon Web Services General Reference*.

Note that Internet Monitor stores data for a monitor in only the AWS Region in which you create the monitor, although a monitor can include resources in multiple Regions.

Region name (Opt-in support)	Region
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1

Region name (Opt-in support)	Region
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Zurich)	eu-central-2
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1

Region name (Default support)	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1

Region name (Default support)	Region
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

Pricing for Amazon CloudWatch Internet Monitor

With Amazon CloudWatch Internet Monitor, there are no upfront costs or long-term commitments. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A *city-network* is the location where clients access your application resources from and the network (ASN, such as an internet service provider or ISP) that clients access the resources through. Note that you are also charged standard CloudWatch prices for logs and any additional metrics, dashboards, alarms, or insights that you create.

You choose a percentage of traffic to monitor when you create a monitor. To help control your bill, you can also set a limit for the maximum number of city-networks to monitor. You can update the percentage of traffic to monitor or the maximum city-networks limit at any time by editing your monitor. The first 100 city-networks (across all monitors per account) are included. After that, you only pay for the actual additional number of city-networks that you monitor, up to the maximum number.

You pay only the actual additional number of city-networks that you monitor, up to the maximum number, with no charge for the first 100 city-networks (across all monitors per account). A flat amount equivalent to the cost of 100 city-networks is deducted from your monthly bill.

For example, a large global company could choose to monitor 100% of its internet-facing traffic, and set a city-networks maximum of 50,000, for one monitor with one resource. Assuming the traffic reached 50,000 city-networks, that portion of its bill would be around 2,700 USD/month.

For another company, in fewer geographic areas, with one monitor with one resource and 200 city-networks, this portion of the bill would be around 13 USD/month. For more information, see [Choosing a city-networks maximum limit](#).

You can try out different options with the pricing calculator. To explore pricing options, on the [Pricing calculator for CloudWatch page](#), scroll down to Internet Monitor.

For more information about Internet Monitor and CloudWatch pricing, see the [Amazon CloudWatch pricing](#) page.

Components and terms for Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor uses or references the following.

Monitor

A monitor includes the resources for a single application that you want to view internet performance and availability measurements for, and that you want to get health event alerts about. When you create a monitor for an application, you add resources for the application to define the cities (locations) for Internet Monitor to monitor. Internet Monitor uses the traffic patterns from the application resources that you add so that it can publish internet performance and availability measurements specific to just the locations and ASNs (typically, internet service providers or ISPs) that communicate with your application. In other words, the resources that you add create a scope of the *city-networks* that you want Internet Monitor to monitor and that you want it to publish measurements for.

Resource added to monitor ("monitored resource")

A resource that you add to a monitor is a "monitored resource" in Internet Monitor. That is:

- Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or an AWS Fargate container.
- Each Network Load Balancer that you add in a Region is a monitored resource.
- Each WorkSpaces directory that you add in a Region is a monitored resource.
- Each CloudFront distribution that you add is a monitored resource.

Autonomous System Number (ASN)

In Internet Monitor, an ASN typically refers to an internet service provider (ISP), such as Verizon or Comcast. An ASN is a network provider that a client uses to access your internet

application. An Autonomous System (AS) is a set of internet routable internet protocol (IP) prefixes that belong to a network or a collection of networks that are all managed, controlled, and supervised by one organization.

City-network (location and ASN)

A city-network is the location (such as a city) that clients access your application resources from and the ASN, typically an internet service provider (ISP), that clients access the resources through. To help control your bill, you can set a limit for the maximum number of city-networks for Internet Monitor to monitor for each monitor. You pay only for the actual number of city-networks that you monitor, up to the maximum number. For more information, see [Choosing a city-network maximum limit](#).

Internet measurements

Internet Monitor publishes internet measurements into log files in CloudWatch Logs every five minutes for the top 500 city-networks (client locations and ASNs, typically internet service providers or ISPs) in your account. These measurements quantify the performance score, availability score, bytes transferred (bytes in and bytes out), and round-trip time for your application's city-networks. These are measurements for the city-networks specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories. Optionally, you can choose to publish internet measurements and events for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket.

Metrics

Internet Monitor generates aggregated metrics for CloudWatch metrics, for global traffic to your application and global traffic to each AWS Region. For more information, see [Using CloudWatch Metrics with Amazon CloudWatch Internet Monitor](#).

Health event

Internet Monitor creates a health event to alert you to a specific problem that affects your application. Internet Monitor detects internet issues, such as increased network latency, across the world. It then uses its historical internet measurements from across the AWS global infrastructure footprint to calculate the impact of current issues on your application, and creates health events. Internet Monitor, by default, creates health events based on both overall impact and local impact thresholds. For more information about configuring thresholds, see [Change health event thresholds](#).

Each health event includes information about the impacted city-networks. You can view health events in the CloudWatch console, or by using the AWS SDK or AWS CLI with Internet Monitor

API actions. Internet Monitor also sends Amazon EventBridge notifications for health events. For more information, see [When Internet Monitor creates and resolves health events](#).

Internet event

Internet Monitor displays information about recent global health events, called internet events, on an internet weather map that is available to all AWS customers. You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events, internet events are not specific to individual customers or their application traffic. For more information, see [Global internet weather map in Amazon CloudWatch Internet Monitor](#).

Thresholds

Internet Monitor creates health events based on both overall thresholds and local thresholds. You can change the default thresholds and configure other options, such as turning off local thresholds. For more information about configuring thresholds, see [Change health event thresholds](#).

Performance and availability scores

By analyzing the data that AWS collects, Internet Monitor can detect when the performance and availability for your application has dropped, compared to estimated baselines that Internet Monitor calculates. To make it easier to see those drops, Internet Monitor reports the information to you as scores. A performance score represents the estimated percentage of traffic that is **not** seeing a performance drop. Similarly, an availability score represents the estimated percentage of traffic that is **not** seeing a availability drop. For more information, see [How AWS calculates performance and availability scores](#).

Bytes transferred and monitored bytes transferred

Bytes transferred is the total number of bytes of ingress and egress traffic between an application in AWS and the city-network (that is, the location and the ASN, typically the internet service provider) where clients access an application. Monitored bytes transferred is a similar metric, but includes only bytes for monitored traffic.

Round-trip time

Round-trip time (RTT) is how long it takes for a request from a client user to return a response to the user. When RTT is aggregated across client locations (cities or other geographies), the value is weighted by how much of your application traffic is driven by each client location.

Global internet weather map in Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor displays a global internet weather map that is available to all AWS customers. To view the map, in the Amazon CloudWatch console, navigate to Internet Monitor.

The map highlights internet events ("outages") all over the world that affect AWS customers, with the specific cities and networks (ASNs, typically internet service providers) where there are issues with performance or availability. The internet weather map includes internet events from the past 24 hours.

You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events in Internet Monitor, internet events are not specific to individual customers or their application traffic.

On the internet weather map, you can choose an internet event to learn details about it. For an internet event, you can see the start time, end time (if the event is over), the current status (Active or Resolved), and the outage issue type (Availability or Performance). To learn more about how the internet weather map is created and what is included, see the [global internet weather map FAQ](#).

To view and work with detailed information that is specific to your application traffic and client locations, you can easily set up a monitor in Internet Monitor for your application. Then, you'll see performance and availability patterns and events, current and historical, as well as get health event alerts, tailored to just your application footprint and customers. The internet weather map gives you an overall view, while a specific monitor filters the information to just the measurements and details that are relevant to your application. With a monitor, you can also explore historical metrics and get recommendations for improving client experience for your application. To learn more, see [Getting started with Amazon CloudWatch Internet Monitor using the console](#).

How Amazon CloudWatch Internet Monitor works

This section provides information about how Amazon CloudWatch Internet Monitor works. This includes descriptions of how AWS collects the data that it uses to help detect connectivity issues across the internet, and how performance and availability scores are calculated.

Contents

- [How Internet Monitor focuses on just your application traffic footprint](#)
- [How AWS measures connectivity issues and calculates measurements](#)

- [Geolocation accuracy in Internet Monitor](#)
- [When Internet Monitor creates and resolves health events](#)
- [Health event report timing](#)
- [How Internet Monitor works with IPv4 and IPv6 traffic](#)
- [How Internet Monitor selects the subset of city-networks to include](#)
- [How the global internet weather map is created \(Frequently Asked Questions\)](#)

How Internet Monitor focuses on just your application traffic footprint

Internet Monitor focuses monitoring on just the subset of the internet that's accessed by the users of your AWS resources, instead of broadly monitoring your website from every Region in the world as other tools do. It's also a cost effective solution, affordable for large and small companies.

Internet Monitor uses the same powerful probes and issue-detection algorithms that AWS takes advantage of internally and alerts you to connectivity issues that affect your application by creating health events in Internet Monitor. Internet Monitor then gives you access to the resulting performance and availability map, by overlaying the traffic profile that it creates from your active viewers, based on your application resources.

Using this information, Internet Monitor shows you just relevant events (that is, the events from places where you have active viewers), and just the impact those events have on your overall viewer volume. So, how much impact an event has, percentage-wise, is based on your total traffic worldwide.

Internet Monitor publishes to CloudWatch Logs internet measurements every five minutes for the top 500 city-networks (client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

The benefits of Internet Monitor include the following:

- Using Internet Monitor doesn't place additional load or cost on your application that's hosted on AWS.
- You don't need to include performance measurement code in your client-side resources, or in your application.

- You can get visibility into performance and availability across the internet that your application is connected to, including "last mile" information.

Note that because Internet Monitor creates measurements based on your AWS resources, Internet Monitor only creates events that are specific to your application traffic. Global internet issues in general are not reported. In addition, when the service location is an AWS Region, the measurements and events emitted are designed to represent connectivity at a Regional level and don't accurately represent connectivity between an end user location and an Availability Zone.

How AWS measures connectivity issues and calculates measurements

Amazon CloudWatch Internet Monitor uses internet connectivity data between different AWS Regions and Amazon CloudFront points of presence (POPs) to different client locations through Autonomous System Numbers (ASNs), typically internet service providers (ISPs). This is the connectivity data that is used internally by AWS operators, on a daily basis, to proactively detect connectivity issues across the global internet.

For every AWS Region, we know which portions of the internet communicate with the Region and do the following:

- We actively monitor those portions of the internet, with a rolling 30-day window.
- We use both network and higher-level protocol probes, including both inbound and outbound probing.

AWS has active and passive probes that measure the latency (performance) at the 90th percentile and reachability (availability) from every AWS Region and from the CloudFront service to the entire internet. Abnormal patterns in connectivity between a service and a customer location are monitored, and then reported as alerts to the customer.

Calculating availability and RTT

Round-trip time (RTT) is how long it takes for a request from the user to return a response to the user. When round-trip time is aggregated across end user locations, the value is weighted by the amount of your traffic that is driven by each end user location.

As an example, with two end user locations, one serving 90% of traffic with a 5 ms RTT, and the other serving 10% of traffic with a 10 ms RTT, the result is an aggregated RTT of 5.5 ms (which comes from $5 \text{ ms} * 0.9 + 10 \text{ ms} * 0.1$).

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Calculating performance and availability scores

AWS has substantial historical data about internet performance and availability between AWS services and different city-networks (locations and ASNs). By applying statistical analysis to the data, Internet Monitor can detect when the performance and availability for your application has dropped, compared to an estimated baseline that it has calculated. To make it easier to see those drops, that information is reported to you in the form of health scores: a performance score and an availability score.

Health scores are calculated at different granularities. At the finest granularity, we compute the health score for a geographic region, such as a city or a metro area, and an ASN (a *city-network*). We also roll up the individual health scores to overall health score numbers for an application in a monitor. If you view performance or availability scores without filtering for any specific geography or service provider, Internet Monitor provides overall health scores.

Overall health scores span your whole application for the specified time period. When the performance or availability score for your application's city-network pairs across your application reaches or drops below the corresponding health event threshold for performance or availability Internet Monitor triggers a health event. By default, the threshold is 95% for both overall performance and availability. Internet Monitor also creates health events based on local thresholds—if the option is enabled, as it is by default—based on values that you configure. To learn more about configuring health event thresholds, see [Change health event thresholds](#).

When you explore information in the monitor and log files to investigate issues and learn more, you can filter by specific cities (locations), networks (ASNs or internet service providers), or both. So, you can use filters to see health scores for different cities, ASNs, or city-network pairs, depending on the filters that you choose.

- An *availability score* represents the estimated percentage of traffic that is **not** seeing an availability drop. Internet Monitor estimates the percentage of traffic experiencing a drop from the total traffic seen and availability metrics measurements. For example, an availability score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing an availability drop for that pair.

- A *performance score* represents the percentage of traffic that is **not** seeing a performance drop. For example, a performance score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing a performance drop for that pair.

Calculating TTFB and RTT (latency)

Time to first byte (TTFB) refers to the time between when a client makes a request and when it receives the first byte of information from the server. AWS calculations for TTFB measure the time elapsed from Amazon EC2 or Amazon CloudFront to the Internet Monitor measurement node (including the last mile of the node). That is, Internet Monitor measures time from the user to the Amazon EC2 Region for TTFB for EC2, and from the user to CloudFront for TTFB for CloudFront.

For round-trip time (RTT), Internet Monitor includes the time from the city-network (that is, the client location and ASN, typically an internet service provider), as mapped by the public IP address, to the AWS Region. This means that Internet Monitor does not have last mile visibility for users who access the internet from behind a gateway or VPN.

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Internet Monitor includes average TTFB information in the **Traffic optimization suggestions** section of the **Traffic insights** tab on the CloudWatch dashboard, to help you evaluate options for different setups for your application that can improve performance.

Regional and Availability Zone measurements and aggregation

Although Internet Monitor aggregates measurements and shares impact at a Regional level, it calculates impact at an Availability Zone (AZ) level. This means that, if, for an event, only one AZ is impacted and most of your traffic flows through that AZ, you do see impact for your traffic. However, for the same event, if your application traffic does not flow through an impacted AZ, you do not see impact.

Note that this applies only to resources that aren't WorkSpaces directories. WorkSpaces directories are measured only on a Regional level.

Geolocation accuracy in Internet Monitor

For location information, Internet Monitor uses IP-geolocation data supplied by [MaxMind](#). The accuracy of the location information in Internet Monitor measurements depends on the accuracy of MaxMind's data.

Be aware that Metro level measurements might not be accurate for locations outside of the United States.

When Internet Monitor creates and resolves health events

Internet Monitor creates and closes health events for the application traffic that you monitor based on the current thresholds that are set. Internet Monitor has a default threshold configuration, and you can also set your own configuration for thresholds. Internet Monitor determines the overall impact that connectivity issues are having on your application, and the impact on local areas where your application has clients, and creates health events when the thresholds are crossed.

Internet Monitor calculates the impact of connectivity issues on a client location based on the historical data about internet performance and availability for network traffic that's available to the service through AWS. It applies the information relevant to your application, based on the geographic locations for ASNs and services where clients use your application: the city-network pairs that are affected. The locations are determined from the resources that you add to your monitor. Then Internet Monitor uses statistical analysis to detect when performance and availability has dropped, affecting the client experience for your application.

The performance and availability scores that Internet Monitor calculates are represented as the percentage of traffic that is **not** seeing a drop. Impact is the opposite of this: it's a representation of how much an issue is problematic for a customer's end users. So if there is a global availability drop of 93%, for example, the corresponding impact would be 7%.

When the performance or availability score for your application's city-network pairs globally reaches or drops below the corresponding health event threshold for performance or availability, this triggers Internet Monitor to generate a health event. By default, the threshold is 95% for both performance and availability. The values to meet, or drop below, the threshold are cumulative, so it could mean several smaller events combine to meet the threshold percentage, or that a single event meets or falls below the threshold level.

As long as performance or availability scores that triggered the event are at or below the corresponding health event threshold percentage for overall impact, the health event stays active. When the score or combined scores that triggered the event rise above the threshold, Internet Monitor resolves the health event.

Internet Monitor also creates health events based on local thresholds and the percentage of overall traffic that an issue has an impact on. You can configure options for local thresholds, or turn off local thresholds altogether.

To learn more about configuring health event thresholds, see [Change health event thresholds](#).

Health event report timing

Internet Monitor uses an aggregator to gather all signals about internet issues, to create health events in monitors within minutes.

When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by AWS or an ASN. Health event analysis continues after an event is resolved. Internet Monitor can update events with new information for up to an hour.

How Internet Monitor works with IPv4 and IPv6 traffic

Internet Monitor measures health toward a network over only IPv4, and shows you health events, and availability and performance metrics, if you serve traffic to that network over any IP family (IPv4 or IPv6). If you serve traffic from a dual-stack resource, such as a dual-stack CloudFront distribution, Internet Monitor raises a health event and shows a drop in a performance score or availability score only if IPv4 traffic has the same issues for the resource as IPv6 traffic does.

Note that the Internet Monitor metrics for overall bytes in and bytes out accurately reflect all internet traffic (IPv4 and IPv6).

How Internet Monitor selects the subset of city-networks to include

When you set a maximum limit for the number of city-networks monitored by your monitor or choose a percentage of traffic to monitor, Internet Monitor chooses the city-networks to include (monitor) based on highest recent traffic volume.

For example, if you set a maximum city-networks limit of 100, Internet Monitor monitors (up to) 100 city-networks based on your application traffic during a recent one hour period. Specifically, Internet Monitor monitors the top 100 city-networks that have had the most traffic in the most recent one hour window *before* the latest one hour window.

To illustrate this, say that the current time is 2:30 PM. In this scenario, the traffic that you see in your monitor was captured between 1:00 PM and 2:00 PM, and the traffic volume measurement that Internet Monitor uses to determine the top 100 city-networks was captured between 12:00 PM and 1:00 PM.

How the global internet weather map is created (Frequently Asked Questions)

The Amazon CloudWatch Internet Monitor internet weather map is available on the Internet Monitor console to all authenticated AWS customers. This section includes details about how the internet weather map is created and how to use it.

What is the Internet Monitor internet weather map?

The internet weather map provides a visual representation of internet issues across the world. It highlights impacted client locations, that is, cities plus ASN (typically internet service providers). The map shows a combination of availability and performance issues that have recently impacted clients' internet experience for top client locations and AWS services globally.

Where does data for the map come from?

The data is based on a combination of active and passive probing of the internet. To learn more about how Internet Monitor measures data you can read the section [How AWS measures connectivity issues](#).

How often is the map updated?

The internet weather map is updated every 15 minutes.

Which networks are tracked for outages?

AWS tracks networks all around the world that represent important IP prefixes used by customers for making internet connections to AWS. We scope outages to client locations that are top talkers for volume of traffic sent to and received from the AWS network.

What determines whether an internet event is included on the map?

Here are some high level criteria that we use to determine whether an internet event is included on the internet weather map:

- AWS detects that there is an availability or performance event.
- If the event is short lived, for example, it lasts less than 5 minutes, we ignore it.
- Then, if the event is in a client location that is classified as a top talker, it's considered an outage.

What thresholds are used for the internet weather map?

Thresholds for determining outages are not static for the internet weather map. Internet Monitor determines what constitutes an event based on detecting a deviation from expected values. You can learn more about how this works by reviewing [how Internet Monitor](#)

[determines when to create health events](#) for monitors that you create with the service.

When you create a monitor, Internet Monitor generates internet traffic health measurements that are specific to your own application traffic. Internet Monitor also alerts you to health events for issues that affect your application's internet traffic.

What can I do with this data?

The internet weather map provides a quick summary of key internet events that happened around the world in the last 24 hours. It helps you to get a flavor of the internet monitoring experience, without needing to onboard your own internet traffic to Internet Monitor. To leverage the full potential of the internet monitoring capabilities of AWS and to personalize it for your applications and services hosted on AWS, you can create a monitor in Internet Monitor.

When you create a monitor, you enable Internet Monitor to identify the specific internet paths that affect your application clients, and you get access to features and capabilities that can help you improve your client experience. You'll also be proactively notified of new internet issues that specifically impact your application traffic and clients.

How can I get more details about events?

Click an outage on the map to see details that include when an event started and ended, the impacted city and ASN, and what type of issue it was (that is, a performance issue or an availability issue).

To get more detailed information about events, and to get custom measurements for your application traffic, [create a monitor in Internet Monitor](#).

Amazon CloudWatch Internet Monitor example use cases

In this section, we describe several specific examples, with links to blog posts with more details. These examples show how you can use the capabilities of Amazon CloudWatch Internet Monitor to help you monitor your application and improve your users' experience.

Set up alerts and decide on actions to take

You can use Internet Monitor to get insights about average internet performance metrics over time, and about health events by city-network (client location and ASN, typically an internet service provider). Using Internet Monitor, you can identify the events that are impacting end user experience for applications hosted on Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon WorkSpaces, or Amazon CloudFront.

After you create a monitor, you have several options for how to be alerted about Internet Monitor health events. These include notifications based on CloudWatch Alarms using event metrics or Amazon EventBridge rules to filter for health events. You can choose different options for notifications or actions based on alarms, including, for example, AWS SMS notifications or updates to a CloudWatch log group.

To see an example with detailed guidance, see the following blog post: [Introducing Amazon CloudWatch Internet Monitor](#).

Identify latency issues and improve TTFB to improve multiplayer gameplay experience

Use Internet Monitor to help you to quickly identify where game players in global cloud gaming apps are experiencing latency issues globally, and provide insights into improving performance. By identifying where the most players currently have the slowest time to first byte (TTFB), you know how to improve latency to make your biggest player base happier.

Now, when you're ready to deploy the next EC2 server for your game, choose the AWS Region that Internet Monitor suggests will lower TTFB in the area with the high latency and large group of players.

For details about setting up and using Internet Monitor for this use case, see the following blog post: [Using Amazon CloudWatch Internet Monitor for a Better Gaming Experience](#).

Internet Monitor cross-account observability

With Internet Monitor cross-account observability, you can monitor your applications that span multiple AWS accounts within a single AWS Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your AWS accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a *sink* in your monitoring account. A sink is a resource that represents an attachment point in a monitoring account. For Internet Monitor, the resource attachment point is a monitor. You use the sink to create a link from your source account to your monitoring account. For more information, see [CloudWatch cross-account observability](#).

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, ensure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Monitors in Internet Monitor
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs

Getting started with Amazon CloudWatch Internet Monitor using the console

To get started with Amazon CloudWatch Internet Monitor, you must create a *monitor* in Internet Monitor for your application by adding AWS resources that it uses and setting several configuration options. This chapter provides the procedure for adding a monitor in the console. It also includes a section with more details about resources in Internet Monitor, and then additional sections with descriptions about and limitations for the different options that you can or must configure for your monitor.

Contents

- [Creating a monitor in Amazon CloudWatch Internet Monitor using the console](#)
- [Adding resources to your monitor](#)
- [Choosing an application traffic percentage to monitor](#)
- [Choosing a city-networks maximum limit](#)
- [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#)
- [Using an Internet Monitor monitor](#)
- [Editing or deleting an Internet Monitor monitor](#)
- [Add or create an Amazon CloudWatch Internet Monitor monitor with Amazon VPC](#)
- [Add or create an Amazon CloudWatch Internet Monitor monitor with CloudFront](#)

Creating a monitor in Amazon CloudWatch Internet Monitor using the console

You create a monitor in Amazon CloudWatch Internet Monitor for your application by adding AWS resources that it uses, and then setting several configuration options. The resources that you add, Amazon Virtual Private Clouds (VPCs), Network Load Balancers (NLBs), CloudFront distributions, or WorkSpaces directories, provide the information for Internet Monitor to map internet traffic information for your application. After you create your monitor, wait 15-30 minutes to generate the traffic profile specific to where your application is used. Then, you can use the Internet Monitor

monitor, or other tools, to visualize and explore performance and availability about your client usage. These tools provide insights for you using your application traffic's measurements, gathered and published by the monitor, for example, to CloudWatch Logs.

Typically, it's simplest to create one monitor in Internet Monitor for one application. Within the same monitor, you can search and sort through measurements and metrics in Internet Monitor log files by different locations and ASNs (typically internet service providers), or other information. It's not necessary to create separate monitors for applications in different areas, for example.

The steps here walk you through setting up your monitor by using the console. To see examples of using the AWS Command Line Interface with the Internet Monitor API actions, to create a monitor, view events, and so on, see [Examples of using the CLI with Amazon CloudWatch Internet Monitor](#).

To create a monitor using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name you want to use for this monitor in Internet Monitor.
5. Choose **Add resources**, and then select the resources to set the monitoring boundaries for Internet Monitor to use for this monitor.

Note

Be aware of the following:

- To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
- You can add a combination of VPCs and CloudFront distributions, or you can add WorkSpaces directories, or you can add Network Load Balancers. You can't add Network Load Balancers or WorkSpaces directories together with other types of resources.

6. Choose a percentage of your internet traffic to monitor.
7. Optionally, specify additional options under **Advanced settings**.
 - For **City-networks maximum**, you can select a limit for the number of city-networks (locations and ASNs, or internet service providers) that Internet Monitor will monitor traffic

for. You can change this at any time by editing your monitor. See [Choosing a city-networks maximum limit](#).

To reset to the default, enter 500000.

If you set a city-networks maximum limit, it sets a cap for the number of city-networks that Internet Monitor monitors for your application, regardless of the percentage of traffic that you choose to monitor.

- Optionally, you can specify an Amazon S3 bucket name and custom prefix to publish internet measurements to Amazon S3 for all monitored city-networks.

Internet Monitor publishes the top 500 (by traffic volume) internet measurements for your application to CloudWatch Logs every five minutes. If you choose to publish measurements to S3, measurements are still published to CloudWatch Logs. For more information, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

- Optionally, you can add a tag for your monitor.

8. Choose **Create monitor**.

After you create a monitor, you can edit the monitor at any time, for example, to change the application traffic percentage, update the maximum city-networks limit or add or remove resources. You can also delete the monitor. To do these tasks in the Internet Monitor console, choose a monitor, and then choose an option in the **Action** menu. Note that you can't change the name of a monitor.

To view the Internet Monitor dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Network monitoring**, then **Internet Monitor**.

The **Monitors** tab displays a list of the monitors that you have created.

To see more information about a specific monitor, choose a monitor.

Adding resources to your monitor

When you create a monitor, you must associate your application's resources with it: Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon CloudFront distributions, Network Load Balancers (NLBs), or Amazon WorkSpaces directories. Then, Internet Monitor knows where your

application's internet-facing traffic and clients are located, and it can create and maintain a traffic profile that determines the relevant measurements to publish for your monitor.

You can add the following types of resources to a monitor in Internet Monitor as "monitored resources". Note that Internet Monitor doesn't support adding different types of resources together in one monitor.

- **VPCs:** Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or in an AWS Fargate container.
- **Network Load Balancers:** Each NLB that you add is a monitored resource.
- **CloudFront distributions:** Each CloudFront distribution that you add is a monitored resource.
- **WorkSpaces directories:** Each WorkSpaces directory that you add in a Region is a monitored resource.

When you monitor traffic for VPCs, traffic for applications that are hosted on load balancers behind the VPC is monitored. You can choose to monitor traffic for individual Network Load Balancer load balancers instead of monitoring a VPC with multiple load balancers. This can be helpful, for example, if you need to understand and configure features for better performance or efficiencies at the load balancer level. Or, you might need compliance information at the Network Load Balancer level.

When you add resources to a monitor in Internet Monitor, be aware of the following:

- To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
- Internet Monitor doesn't support adding different types of resources together in one monitor.

There are Regional differences for opt-in Regions to keep in mind when you add VPCs or NLBs as resources. For more information, see [Supported AWS Regions for Amazon CloudWatch Internet Monitor](#).

In addition, there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, NLBs, and WorkSpaces directories do not include last-mile latency.

Choosing an application traffic percentage to monitor

The coverage that you choose for the percentage of application traffic to monitor determines how many city-networks (client locations and ASNs, typically internet service providers) for your application are monitored, up to an optional city-networks maximum limit that you can also set.

If you choose to monitor less than 100% of your application traffic, an observability gap might exist with your monitor. That's because if there are health events that Amazon CloudWatch Internet Monitor creates where you aren't monitoring traffic, you won't be aware of those issues. You might also have less coverage for the performance and availability score information about client access to your application.

The following sections describe options to explore traffic percentage settings and coverage, and to get an idea about the impact of increasing or decreasing coverage.

- [Explore changing your application traffic percentage](#)
- [View the number of city-networks monitored at different traffic percentage settings](#)

Explore changing your application traffic percentage

You can explore values that you might want to change your application traffic percentage to, by viewing the number of city-networks monitored when you change the percentage. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can try increasing or decreasing the application traffic percentage for your monitor, and view the estimated number of your city-networks that would be covered as a result. With this option, you can quickly see how changing your traffic percentage affects the number of city-monitors are monitored. This can help you to get a feel for what a good application traffic percentage to choose might be, for your application.

To explore monitoring coverage by increasing and decreasing application traffic percentage

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. In your list of monitors, choose a monitor.
4. On the **Overview** tab, in the Monitored traffic section, choose the percentage graph, and then choose **Update monitoring coverage**.

5. In the **Explore and set traffic monitoring coverage** dialog, click the arrows to increase or decrease the percentage of traffic to monitor. By choosing 100% traffic, you can see how many city-networks are monitored with full coverage for monitoring your application.
6. To learn more about how the number of city-networks monitored (estimated here) might affect your costs, choose the link to the [CloudWatch Pricing calculator](#), and then scroll down to Internet Monitor.
7. To set a new percentage of traffic to monitor, choose **Update monitor coverage**. Or, to keep the current coverage level, choose **Cancel**.

View the number of city-networks monitored at different traffic percentage settings

You can view the number of city-networks that would be monitored for your application at different application traffic percentages. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can view graphs that show how coverage for your city-networks would change at different of application traffic percentages, over a time interval that you specify. This is a quick way to visualize and compare the monitoring coverage for your application at specific traffic percentages, all on one graph.

To view graphs of application traffic percentage and corresponding city-networks coverage

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. In your list of monitors, choose a monitor.
4. Choose the **Traffic insights** tab, and scroll down to **Internet traffic graphs**.
5. Under **Compare options for traffic coverage**, in the drop-down list, select one or more percentages. You can choose one or more application traffic percentages, and the graph of **Total monitored city-networks** is updated to display the monitoring coverage Internet Monitor provides for that traffic percentage. By choosing **City-networks at 100% traffic**, you can see how many city-networks are monitored with full coverage for monitoring your application.

Keep in mind the following:

- Traffic coverage is computed based on the number of city-networks in the previous hour of your application traffic. This means that, after you choose a specific percentage of traffic to monitor,

fewer city-networks might be monitored for your application than is shown here in a traffic coverage comparison graph.

- To make sure that all your application traffic is monitored, set `TrafficPercentageToMonitor` to 100 and don't set `MaxCityNetworksToMonitor`. Alternatively, you can set `MaxCityNetworksToMonitor` to 500,000, the upper limit in Internet Monitor.
- If you set a city-networks maximum limit, the total number of monitored city-networks never exceeds that limit, regardless of the application traffic percentage option that you select.
- You can learn more about how the number of city-networks monitored might affect your costs. On the [Pricing calculator for CloudWatch page](#), scroll down to Internet Monitor.

To set a new percentage of traffic to monitor, under **Explore other traffic coverage options**, choose **Update monitoring coverage**. In the dialog, choose a percentage of traffic, and then choose **Update monitor coverage**.

Choosing a city-networks maximum limit

Amazon CloudWatch Internet Monitor can monitor your application traffic for some or all the locations where clients access your application resources, and all the ASNs (typically internet service providers) that they access your application through—that is, the city-networks for your application internet traffic. You choose a [percentage of application traffic](#) to monitor when you create your monitor, which you can update at any time by editing the monitor.

In addition to setting a traffic percentage, you can also set a maximum limit for the number of city-networks monitored. This section describes how the city-networks limit can help you manage billing costs, and provides information and an example to help you determine a limit to set.

The maximum limit that you set for the number of city-networks helps to make sure that your bill is predictable. For more information, see [Amazon CloudWatch Pricing](#). You can also learn how different values for the number of city-networks actually monitored can affect your bill by using the CloudWatch price calculator. To explore options, on the [Pricing calculator for CloudWatch page](#), scroll down to Internet Monitor.

To update your monitor and change the maximum city-networks limit, see [Editing or deleting an Internet Monitor monitor](#).

How billing works with city-networks maximum limits

Setting a maximum limit for the number of city-networks monitored can help prevent unexpected costs in your bill. This is useful, for example, if your traffic patterns vary widely. Billing costs

increase for each city-network that is monitored after the first 100 city-networks, which are included (across all monitors per account). If you set a city-networks maximum limit, it caps the number of city-networks that Internet Monitor monitors for your application, regardless of the percentage of traffic that you choose to monitor.

You only pay for the number of city-networks that are actually monitored. The city-network maximum limit that you choose lets you set a cap on the total that can be included when Internet Monitor monitors traffic with your monitor. You can change the maximum limit at any time by editing your monitor.

To explore options, on the [Pricing calculator for CloudWatch](#) page, scroll down to Internet Monitor. For more information on Internet Monitor pricing, see the Internet Monitor section on the [Amazon CloudWatch Pricing](#) page.

How to choose a city-networks maximum limit

To help you decide on a city-networks maximum limit to select, consider how much traffic you want to monitor for your application. The following Internet Monitor metrics can help you analyze your traffic usage and coverage after you create your monitor: `CityNetworksMonitored`, `TrafficMonitoredPercent`, and one or more of the `CityNetworksForNNPercentTraffic` metrics, where `NN` is a percentage value that is one of the following: 25, 50, 90, 95, 99, or 100. To review definitions for these metrics, and all other Internet Monitor metrics, see [Using CloudWatch Metrics with Amazon CloudWatch Internet Monitor](#).

To see an overview graph of your internet traffic coverage, go to the **Traffic insights** tab on the CloudWatch dashboard and, in the **Internet traffic graphs** section, choose an option for **Compare options for traffic coverage**. The graph shown in the section displays the actual number of city-networks that are monitored for your application, as well as graph lines for different application traffic percentages that you select in the drop-down list. To learn more, see [Setting your application traffic percentage](#).

To explore your options in more detail, you can use the Internet Monitor metrics, as described in the following examples. These examples show how to select a maximum city-networks limit that is best for you, depending on the breadth of application internet traffic coverage you want. Using the [queries for Internet Monitor metrics in CloudWatch Metrics](#) can help you understand more about your application internet traffic coverage.

Example of determining a city-networks maximum limit

As an example, say that you've set a monitoring maximum limit of 100 city-networks and that your application is accessed by clients across 2637 city-networks. In CloudWatch Metrics, you'd see the following Internet Monitor metrics returned:

```
CityNetworksMonitored 100
TrafficMonitoredPercent 12.5
CityNetworksFor90PercentTraffic 2143
CityNetworksFor100PercentTraffic 2637
```

From this example, you can see that you're currently monitoring 12.5% of your internet traffic, with the maximum limit set to 100 city-networks. If you want to monitor 90% of your traffic, the next metric provides information about that: `CityNetworksFor90PercentTraffic` indicates that you would need to monitor 2,143 city-networks for 90% coverage. To do that, you would update your monitor and set the maximum city-networks limit to 2,143.

Similarly, say you'd like to have 100% internet traffic monitoring for your application. The next metric, `CityNetworksFor100PercentTraffic`, indicates that to do this, you should update your monitor to set the maximum city-networks limit to 2,637.

If you now set the maximum to 5,000 city-networks, since that's greater than 2,637, you see the following metrics returned:

```
CityNetworksMonitored 2637
TrafficMonitoredPercent 100
CityNetworksFor90PercentTraffic 2143
CityNetworksFor100PercentTraffic 2637
```

From these metrics, you can see that with the higher limit, you monitor all 2,637 city-networks, which is 100% of your internet traffic.

Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor

You can choose to have Amazon CloudWatch Internet Monitor publish internet measurements to Amazon S3 for your internet-facing traffic to the monitored city-networks (client locations and ASNs, typically internet service providers) in your monitor, up to the 500,000 city-networks service limit. Internet Monitor automatically publishes internet measurements to CloudWatch Logs every

five minutes for the top 500 (by traffic volume) city-networks for each monitor. Measurements that it publishes to S3 include the top 500 that are published to CloudWatch Logs.

You can choose the option to publish to S3, and specify the bucket to publish the measurements, to when you create or update your monitor. The bucket must already be created in S3 before you can specify it in Internet Monitor. There's a service limit of 500,000 city-networks for internet measurements published to S3. Internet Monitor publishes internet measurements to S3 as events, a series of compressed log file objects that are stored in the bucket.

When you create the S3 bucket for Internet Monitor to publish measurements to, make sure that you follow the permissions guidance provided by CloudWatch Logs. Doing so ensures that Internet Monitor can publish logs directly to S3, and that AWS can, if needed, create and change the resource policies associated with the log group receiving the logs. For more information, see [Logs sent to CloudWatch Logs](#) in the Amazon CloudWatch Logs User Guide.

The published log files are compressed. If you open the log files using the Amazon S3 console, they are decompressed and the internet measurement events are displayed. If you download the files, you must decompress them to view the events.

You can also query the internet measurements in the log files using Amazon Athena. Amazon Athena is an interactive query service that makes it easier to analyze data in Amazon S3, by using standard SQL. For more information, see [Using Amazon Athena to query internet measurements in Amazon S3 log files](#).

Using an Internet Monitor monitor

There are several ways to use an Amazon CloudWatch Internet Monitor monitor after you create it: for example, you can view information in the CloudWatch dashboard, get information by using the AWS Command Line Interface, and set health alerts.

Your monitor provides information about your application and configuration preferences so that Internet Monitor can customize measurements and metrics to publish in events for you. Internet Monitor collects measurements from the global infrastructure footprint for AWS. These measurements are a tremendous amount of network performance and availability information, from all over the world. By using information from the resources that you add for your application, Internet Monitor publishes performance and availability measurements for you that is scoped to the city-networks (that is, client locations and ASNs, typically internet service providers or ISPs) where your application is active. So, the measurements and metrics in the Internet Monitor dashboard and in CloudWatch Logs —about availability, performance, monitored bytes transferred, and round-trip time—are specific to your client locations and ASNs.

Internet Monitor also determines when there are anomalies in performance and availability. By default, Internet Monitor overlays your traffic with the availability and performance measurements that AWS has collected for each source-destination pair in your client locations, to determine when there are notable drops in performance or availability. When there's significant degradation for your application's locations and scope, Internet Monitor generates a *health event*, and publishes information about the issue to your monitor.

After you create a monitor, you can use it to access or be alerted to the information that Internet Monitor provides, in the following ways:

- **Use the CloudWatch dashboard** to view and explore performance, availability, and health events; explore your application's historical data; and get insights into new ways to configure your application for better performance. To learn more, see the following:
 - [Tracking real-time performance and availability in Amazon CloudWatch Internet Monitor \(Overview tab\)](#)
 - [Filtering and viewing historical data in Amazon CloudWatch Internet Monitor \(Historical explorer tab\)](#)
 - [Getting insights to improve application performance in Amazon CloudWatch Internet Monitor \(Traffic insights tab\)](#)
- **Configure health event thresholds** to change what triggers Internet Monitor to create a health event for your application. You can configure overall thresholds and local (city-network) thresholds. To learn more, see [Change health event thresholds](#).
- **Use AWS CLI commands** with Internet Monitor API actions to view traffic profile information, view measurements, list health events, and so on. To learn more, see [Examples of using the CLI with Amazon CloudWatch Internet Monitor](#).
- **Use standard CloudWatch tools**, such as CloudWatch Contributor Insights, CloudWatch Metrics explorer, and CloudWatch Logs Insights to visualize the data in CloudWatch. To learn more, see [Exploring your data with CloudWatch tools and the Internet Monitor query interface](#).
- **Use Athena with S3 logs** to access and analyze Internet Monitor internet measurements for your application, if you turned on publishing measurements to S3.
- **Create Amazon EventBridge notifications** to alert you when Internet Monitor determines there is a health event. To learn more, see [Using Amazon CloudWatch Internet Monitor with Amazon EventBridge](#).

- **Receive an AWS Health Dashboard notification** automatically, when Internet Monitor determines that an issue is caused by the AWS network. The notification includes the steps that AWS is taking to mitigate the problem.

Editing or deleting an Internet Monitor monitor

Using the **Action** menu, you can edit or delete a monitor in Amazon CloudWatch Internet Monitor after you create it. For example, you can edit a monitor to do the following:

- Change the percentage of application traffic to monitor
- Set or update the city-networks maximum limit
- Change health event thresholds for availability or performance scores
- Add or remove resources
- Enable or update publishing events to Amazon S3

You can also delete a monitor. Note that you can't change the name of a monitor after you create it.

To make changes to a monitor or delete a monitor, use one of the following procedures.

To edit a monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Make the desired updates. For example, to change the percentage of traffic to monitor, under **Application traffic to monitor**, select or enter a percentage.
6. Choose **Update**.

To delete a monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.

4. Choose **Disable**.
5. Choose the **Action** menu again, and then choose **Delete**.

For more information about the options that you can update, see the following:

- To learn more about resources that you add in Internet Monitor, see [Adding resources to your monitor](#).
- To learn more about the application traffic percentage, see [Choosing an application traffic percentage to monitor](#).
- To learn more about changing the threshold for health events, see [Change health event thresholds](#).
- To learn more about the city-networks maximum limit, see [Choosing a city-networks maximum limit](#).
- To learn more about opting to publish events to S3, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

Add or create an Amazon CloudWatch Internet Monitor monitor with Amazon VPC

When you create a Amazon Virtual Private Cloud VPC in the AWS Management Console, you can optionally choose to also set up monitoring for it in Amazon CloudWatch Internet Monitor. You can add the VPC to an existing monitor, or you can opt to create a new monitor for the VPC in the Amazon VPC console.

By using Internet Monitor with your VPC, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see [Using an Internet Monitor monitor](#).

Important

To create a monitor, or add a VPC to an existing monitor, you must have the correct permissions in place. For more information, see [Identity and Access Management for Amazon CloudWatch Internet Monitor](#).

Add a VPC to an existing monitor

You can choose to have Amazon CloudWatch Internet Monitor add a new VPC to an existing monitor for you when you create the VPC in the AWS Management Console. After you add the VPC, wait a few minutes, and then metrics for the VPC will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the VPC or add another VPC or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the VPC from the monitor, traffic from clients to that VPC is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see [Editing or deleting an Internet Monitor monitor](#).

Create a monitor for a VPC

If you opt to create a monitor for a VPC, the **Create monitor** wizard walks you through the steps. You add the VPC as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in [Creating a monitor in Amazon CloudWatch Internet Monitor using the console](#).

Pricing

With Amazon CloudWatch Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see [Pricing for Amazon CloudWatch Internet Monitor](#)

Stop monitoring a VPC

If you'd like to stop monitoring your VPC resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the VPC to remove, and then choose **Remove**.
7. Choose **Update**.

Add or create an Amazon CloudWatch Internet Monitor monitor with CloudFront

On the metrics dashboard for a distribution in Amazon CloudFront console, you can set up additional monitoring for a distribution in Amazon CloudWatch Internet Monitor. You can add the distribution to an existing monitor, or you can create a new monitor for the distribution.

By using Internet Monitor with your CloudFront distribution, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see [Using an Internet Monitor monitor](#).

Important

To create a monitor, or add a distribution to an existing monitor, you must have the correct permissions in place. For more information, see [Identity and Access Management for Amazon CloudWatch Internet Monitor](#).

Add a distribution to an existing monitor

You can choose to have Internet Monitor add a distribution to an existing monitor directly from the CloudFront metrics dashboard in the AWS Management Console. After you add the distribution, wait a few minutes, and then metrics for the distribution will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the distribution or add another distribution or other resources. You can also change the percentage of traffic that you're monitoring, or make

other changes. If you choose to remove the distribution from the monitor, traffic from clients to that distribution is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see [Editing or deleting an Internet Monitor monitor](#).

Create a monitor for a distribution

If you opt to create a monitor for a distribution, the **Create monitor** wizard walks you through the steps. You add the distribution as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in [Creating a monitor in Amazon CloudWatch Internet Monitor using the console](#).

Pricing

With Amazon CloudWatch Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see [Pricing for Amazon CloudWatch Internet Monitor](#)

Stop monitoring a distribution

If you'd like to stop monitoring your distribution resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the distribution to remove, and then choose **Remove**.
7. Choose **Update**.

Examples of using the CLI with Amazon CloudWatch Internet Monitor

This section includes examples for using the AWS Command Line Interface with Amazon CloudWatch Internet Monitor operations.

Before you begin, make sure that you log in to use the AWS CLI with the same AWS account that has the Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon CloudFront distributions, or Amazon WorkSpaces directories that you want to monitor. Internet Monitor doesn't support accessing resources across accounts. For more information about using the AWS CLI, see the [AWS CLI Command Reference](#). For more information about using API actions with Amazon CloudWatch Internet Monitor, see the [Amazon CloudWatch Internet Monitor API Reference Guide](#).

Topics

- [Create a monitor](#)
- [View monitor details](#)
- [List health events](#)
- [View specific health event](#)
- [View monitor list](#)
- [Edit monitor](#)
- [Delete monitor](#)

Create a monitor

When you create a monitor in Internet Monitor, you provide a name and associate resources with the monitor to show where your application's internet traffic is. You specify a traffic percentage that defines how much of your application traffic is monitored. That also determines the number of city-networks, that is, client locations and ASNs, typically internet service providers or ISPs, that are monitored. You can also opt to set a limit for the maximum number of city-networks to monitor for your application resources, to help control your bill. For more information, see [Choosing a city-networks maximum limit](#).

Finally, you can choose if you want to publish all internet measurements for your application to Amazon S3. Internet measurements for the top 500 city-networks (by traffic volume) are automatically published to CloudWatch Logs by Internet Monitor, but you can choose to publish all measurements to S3 as well.

To create a monitor with the AWS CLI, you use the `create-monitor` command. The following command creates a monitor that monitors 100% of traffic but sets a maximum city-networks limit of 10,000, adds a VPC resource, and opts to publish internet measurements to Amazon S3.

Note

Internet Monitor publishes to CloudWatch Logs internet measurements every five minutes for the top 500 city-networks (client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

```
aws internetmonitor --create-monitor monitor-name "TestMonitor" \  
  --traffic-percentage-to-monitor 100 \  
  --max-city-networks-to-monitor 10000 \  
  --resources "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \  
  --internet-measurements-log-delivery  
S3Config="{BucketName=MyS3Bucket,LogDeliveryStatus=ENABLED}"
```

```
{  
  "Arn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",  
  "Status": "ACTIVE"  
}
```

Note

You can't change the name of a monitor.

View monitor details

To view information about a monitor with the AWS CLI, you use the `get-monitor` command.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor"
```

```
{  
  "ClientLocationType": "city",
```



```

"CreatedAt": "2022-09-22T19:27:47Z",
"ModifiedAt": "2022-09-22T19:28:30Z",
"MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
"MonitorName": "TestMonitor",
"ProcessingStatus": "OK",
"ProcessingStatusInfo": "The monitor is actively processing data",
"Resources": [
  "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
],
"MaxCityNetworksToMonitor": 10000,
"Status": "ACTIVE"
}

```

List health events

When performance degrades for your application's internet traffic, Internet Monitor creates health events in your monitor. To see a list of current health events with the AWS CLI, use the `list-health-events` command

```
aws internetmonitor list-health-events --monitor-name "TestMonitor"
```

```

{
  "HealthEvents": [
    {
      "EventId": "2022-06-20T01-05-05Z/latency",
      "Status": "RESOLVED",
      "EndedAt": "2022-06-20T01:15:14Z",
      "ServiceLocations": [
        {
          "Name": "us-east-1"
        }
      ],
      "PercentOfTotalTrafficImpacted": 1.21,
      "ClientLocations": [
        {
          "City": "Lockport",
          "PercentOfClientLocationImpacted": 60.370000000000005,
          "PercentOfTotalTraffic": 2.01,
          "Country": "United States",
          "Longitude": -78.6913,
          "AutonomousSystemNumber": 26101,
          "Latitude": 43.1721,

```

```

        "Subdivision": "New York",
        "NetworkName": "YAH00-BF1"
    }
],
"StartedAt": "2022-06-20T01:05:05Z",
"ImpactType": "PERFORMANCE",
"EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-05-05Z/latency"
},
{
    "EventId": "2022-06-20T01-17-56Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:30:23Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "PercentOfTotalTrafficImpacted": 1.29,
    "ClientLocations": [
        {
            "City": "Toronto",
            "PercentOfClientLocationImpacted": 75.32,
            "PercentOfTotalTraffic": 1.05,
            "Country": "Canada",
            "Longitude": -79.3623,
            "AutonomousSystemNumber": 14061,
            "Latitude": 43.6547,
            "Subdivision": "Ontario",
            "CausedBy": {
                "Status": "ACTIVE",
                "Networks": [
                    {
                        "AutonomousSystemNumber": 16509,
                        "NetworkName": "Amazon.com"
                    }
                ],
                "NetworkEventType": "AWS"
            },
            "NetworkName": "DIGITALOCEAN-ASN"
        }
    ],
    {
        "City": "Lockport",
        "PercentOfClientLocationImpacted": 22.91,

```

```

        "PercentOfTotalTraffic": 2.01,
        "Country": "United States",
        "Longitude": -78.6913,
        "AutonomousSystemNumber": 26101,
        "Latitude": 43.1721,
        "Subdivision": "New York",
        "NetworkName": "YAH00-BF1"
    },
    {
        "City": "Hangzhou",
        "PercentOfClientLocationImpacted": 2.88,
        "PercentOfTotalTraffic": 0.7799999999999999,
        "Country": "China",
        "Longitude": 120.1612,
        "AutonomousSystemNumber": 37963,
        "Latitude": 30.2994,
        "Subdivision": "Zhejiang",
        "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
    }
],
"StartedAt": "2022-06-20T01:17:56Z",
"ImpactType": "PERFORMANCE",
"EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-17-56Z/latency"
},
{
    "EventId": "2022-06-20T01-34-20Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:35:04Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "PercentOfTotalTrafficImpacted": 1.15,
    "ClientLocations": [
        {
            "City": "Lockport",
            "PercentOfClientLocationImpacted": 39.45,
            "PercentOfTotalTraffic": 2.01,
            "Country": "United States",
            "Longitude": -78.6913,
            "AutonomousSystemNumber": 26101,
            "Latitude": 43.1721,

```

```

        "Subdivision": "New York",
        "NetworkName": "YAH00-BF1"
    },
    {
        "City": "Toronto",
        "PercentOfClientLocationImpacted": 29.770000000000003,
        "PercentOfTotalTraffic": 1.05,
        "Country": "Canada",
        "Longitude": -79.3623,
        "AutonomousSystemNumber": 14061,
        "Latitude": 43.6547,
        "Subdivision": "Ontario",
        "CausedBy": {
            "Status": "ACTIVE",
            "Networks": [
                {
                    "AutonomousSystemNumber": 16509,
                    "NetworkName": "Amazon.com"
                }
            ],
            "NetworkEventType": "AWS"
        },
        "NetworkName": "DIGITALOCEAN-ASN"
    },
    {
        "City": "Hangzhou",
        "PercentOfClientLocationImpacted": 2.88,
        "PercentOfTotalTraffic": 0.7799999999999999,
        "Country": "China",
        "Longitude": 120.1612,
        "AutonomousSystemNumber": 37963,
        "Latitude": 30.2994,
        "Subdivision": "Zhejiang",
        "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
    }
],
"StartedAt": "2022-06-20T01:34:20Z",
"ImpactType": "PERFORMANCE",
"EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-34-20Z/latency"
}
]
}

```

View specific health event

To see a more detailed information about a specific health event with the CLI, run the `get-health-event` command with your monitor name and a health event ID.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor" --event-id "health-event/
TestMonitor/2021-06-03T01:02:03Z/latency"
```

```
{
  "EventId": "2022-06-20T01-34-20Z/latency",
  "Status": "RESOLVED",
  "EndedAt": "2022-06-20T01:35:04Z",
  "ServiceLocations": [
    {
      "Name": "us-east-1"
    }
  ],
  "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor/
health-event/2022-06-20T01-34-20Z/latency",
  "LastUpdatedAt": "2022-06-20T01:35:04Z",
  "ClientLocations": [
    {
      "City": "Lockport",
      "PercentOfClientLocationImpacted": 39.45,
      "PercentOfTotalTraffic": 2.01,
      "Country": "United States",
      "Longitude": -78.6913,
      "AutonomousSystemNumber": 26101,
      "Latitude": 43.1721,
      "Subdivision": "New York",
      "NetworkName": "YAH00-BF1"
    },
    {
      "City": "Toronto",
      "PercentOfClientLocationImpacted": 29.7700000000000003,
      "PercentOfTotalTraffic": 1.05,
      "Country": "Canada",
      "Longitude": -79.3623,
      "AutonomousSystemNumber": 14061,
      "Latitude": 43.6547,
      "Subdivision": "Ontario",
      "CausedBy": {
        "Status": "ACTIVE",

```

```
    "Networks": [
      {
        "AutonomousSystemNumber": 16509,
        "NetworkName": "Amazon.com"
      }
    ],
    "NetworkEventType": "AWS"
  },
  "NetworkName": "DIGITALOCEAN-ASN"
},
{
  "City": "Shenzhen",
  "PercentOfClientLocationImpacted": 4.07,
  "PercentOfTotalTraffic": 0.61,
  "Country": "China",
  "Longitude": 114.0683,
  "AutonomousSystemNumber": 37963,
  "Latitude": 22.5455,
  "Subdivision": "Guangdong",
  "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
},
{
  "City": "Hangzhou",
  "PercentOfClientLocationImpacted": 2.88,
  "PercentOfTotalTraffic": 0.7799999999999999,
  "Country": "China",
  "Longitude": 120.1612,
  "AutonomousSystemNumber": 37963,
  "Latitude": 30.2994,
  "Subdivision": "Zhejiang",
  "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
}
],
"StartedAt": "2022-06-20T01:34:20Z",
"ImpactType": "PERFORMANCE",
"PercentOfTotalTrafficImpacted": 1.15
}
```

View monitor list

To see a list of all monitors in your account with the CLI, run the `list-monitors` command.

```
aws internetmonitor list-monitors
```

```
{
  "Monitors": [
    {
      "MonitorName": "TestMonitor",
      "ProcessingStatus": "OK",
      "Status": "ACTIVE"
    }
  ],
  "NextToken": " zase12"
}
```

Edit monitor

To update information about your monitor by using the CLI, use the `update-monitor` command and specify the name of the monitor to update. You can update the percentage of traffic to monitor, the limit of the maximum number of city-networks to monitor, add or remove the resources that Internet Monitor uses to monitor traffic, and change the monitor status from `ACTIVE` to `INACTIVE`, or vice versa. Note that you can't change the name of the monitor.

The response for an `update-monitor` call returns just the `MonitorArn` and the `Status`.

The following example shows how to use the `update-monitor` command to change the maximum number of city-networks to monitor to `50000`:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --max-city-networks-to-monitor 50000
```

```
{
  "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
  "Status": " ACTIVE "
}
```

The following example shows how to add and remove resources:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" \
  --resources-to-add "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
  --resources-to-remove "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-2222444455556666"
```

```
{
  "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
```

```
"Status": "ACTIVE"
}
```

The following example shows how to use the `update-monitor` command to change the monitor status to `INACTIVE`:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --status "INACTIVE"
```

```
{
  "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
  "Status": "INACTIVE"
}
```

Delete monitor

You can delete a monitor with the CLI by using the `delete-monitor` command. First, you must set the monitor to be inactive. To do that, use the `update-monitor` command to change the status to `INACTIVE`. Confirm that the monitor is inactive by using the `get-monitor` command and checking the status.

When the monitor status is `INACTIVE`, then you can use the CLI to run the `delete-monitor` command to delete the monitor. The response for a successful `delete-monitor` call is empty.

```
aws internetmonitor delete-monitor --monitor-name "TestMonitor"
```

```
{}
```

Monitor and optimize with the Internet Monitor dashboard

The information in this section describes how to filter and view information on the Amazon CloudWatch Internet Monitor dashboard to visualize and get insights about your AWS application's internet traffic and setup.

After you create a monitor to monitor your application's internet performance and availability, Amazon CloudWatch Internet Monitor publishes CloudWatch logs containing internet measurements for client location-network (city-network) pairs, and publishes aggregated CloudWatch metrics for traffic to your application, and to each AWS Region and edge location. You can filter, explore, and get action-oriented suggestions from this information from Internet Monitor in several different ways.

To get started, on the CloudWatch console, under **Network monitoring**, choose **Internet Monitor**.

This section primarily describes how to filter and view Internet Monitor metrics using the AWS Management Console. Alternatively, you can use Internet Monitor API operations with the AWS CLI or an SDK to work directly with Internet Monitor events stored in CloudWatch Logs files. For more information, see [Using your monitor and measurements information](#). For more information about using API operations, see [Examples of using the CLI with Amazon CloudWatch Internet Monitor](#) and the [Amazon CloudWatch Internet Monitor API Reference](#).

There are three tabs in the Internet Monitor dashboard:

- On the **Overview** tab, you can see current and historical performance and availability information about your application, and health events impacting your client locations.
- On the next tab, **Historical explorer**, you can filter by location, ASN, date, and so on, and visualize metrics for your internet traffic over time, using the graphs.
- On the **Traffic insights** tab, in addition to viewing information about top monitored traffic summarized in several customizable ways, you can get suggestions for optimized setups to improve performance for different location and ASN pairs. Internet Monitor predicts your application's performance improvement, based on your traffic patterns and past performance, when you change how you route your traffic or the AWS resources you use. You can also see a graph to compare how many city-networks are included in your monitoring coverage, based on the application traffic percentage that you choose for your monitor.

In addition, because Internet Monitor generates and publishes log files with the measurements about your traffic, you can use other CloudWatch tools in the console to further visualize the data published by Internet Monitor, including CloudWatch Contributor Insights, CloudWatch Metrics, and CloudWatch Logs Insights. For more information, see [Exploring your data with CloudWatch tools and the Internet Monitor query interface](#).

Learn about using Internet Monitor to explore your performance and availability measurements in the following sections.

Topics

- [Tracking real-time performance and availability in Amazon CloudWatch Internet Monitor \(Overview tab\)](#)
- [Filtering and viewing historical data in Amazon CloudWatch Internet Monitor \(Historical explorer tab\)](#)

- [Getting insights to improve application performance in Amazon CloudWatch Internet Monitor \(Traffic insights tab\)](#)

Tracking real-time performance and availability in Amazon CloudWatch Internet Monitor (Overview tab)

Use the **Overview** tab in the CloudWatch console, under Internet Monitor, to get a high-level view of performance and availability for the traffic that your monitor tracks. The tab also displays an internet traffic overview map, with traffic clusters that can help you visualize your application's global traffic, and the location and impact of health events.

Health scores

The **Health scores** graph shows you performance and availability information for your global traffic. AWS has substantial historical data about internet performance and availability for network traffic between geographic locations for different ASNs and AWS services. Internet Monitor uses the connectivity data that AWS has captured from its global networking footprint to calculate a baseline of performance and availability for internet traffic. This is the same data that we use at AWS to monitor our own internet uptime and availability.

With those measurements as a baseline, Internet Monitor can detect when the performance and availability for your application has dropped, compared to the baseline. To make it easier to see those drops, we report that information to you as a performance score and an availability score. For more information, see [Exploring your data with CloudWatch tools and the Internet Monitor query interface](#).

The **Health scores** graph includes health events that occurred during a time period that you choose. When there's a health event, you see a drop in the performance or availability line on the graph. If you select the event, you see more details and bands appear on the graph, with date and time information showing how long the event lasted.

You can also look at these metrics by accessing the log files directly for each data point. In the **Actions** menu, choose **View CloudWatch Logs**.

Internet traffic overview

The **Internet traffic overview** map shows you the internet traffic and health events that are specific to the locations and ASNs where your users access your application from. The countries that are gray on the map are those that include traffic for your application.

Each circle on the map indicates a health event in an area, for a time period that you select. Internet Monitor creates health events when it detects a problem, at a specific threshold, with connectivity between one of your resources hosted in AWS and a city-network where a user is accessing your application. Choosing a circle on the map displays more details about the health event for that location. In addition, for clusters that have health events, you can see detailed information in the **Health events** table below the map.

Note that Internet Monitor creates health events in a monitor when it determines that an event has significant global impact on your application. If aren't any health events that exceed the threshold for impact on traffic for client locations in the time period that you've selected, the map is blank. For more information, see [When Internet Monitor creates and resolves health events](#).

Change health event thresholds

You can configure several options around how and when Internet Monitor creates health events for your application. Choose **Update thresholds** to make changes.

You can change the overall threshold that triggers Internet Monitor to create a health event. The default health event threshold, for both performance scores and availability scores, is 95%. That is, when the overall performance or availability score for your application falls to 95% or below, Internet Monitor creates a health event. For the overall threshold, the health event can be triggered by a single large issue, or by the combination of multiple smaller issues.

You can also change the local—that is, city-network—threshold, combined with a percentage of the overall level of impact, that combined will trigger a health event. By setting a threshold that creates a health event when a score drops below the threshold for one or more city-networks (locations and ASNs, typically ISPs), you can get insights into when there are issues in locations with lower traffic, for example.

An additional local threshold option works together with the local threshold for availability or performance scores. The second factor is the percentage of your overall traffic that must be impacted before Internet Monitor creates a health event based on the local threshold.

By configuring the threshold options for overall traffic and local traffic, you can fine-tune how frequently health events are created, to align with your application usage and your needs. Be aware that when you set the local threshold to be lower, typically more health events are created, depending on your application and the other threshold configuration values that you set.

In summary, you can configure health event thresholds—for performance scores, availability scores, or both—in the following ways:

- Choose different global thresholds for triggering a health event.
- Choose different local thresholds for triggering a health event. With this option, you can also change the percentage of impact on your overall application that must be exceeded before Internet Monitor creates an event.
- Choose to turn off triggering a health event based on local thresholds, or enable local threshold options.

You can also configure options for performance scores, availability scores, or both. You can configure a combination of the options, or just one of them.

To update thresholds and other configuration options for performance scores, availability scores, or both, do the following:

To change threshold configuration options

1. In the AWS Management Console, navigate to CloudWatch, and then, in the left navigation pane, choose Internet Monitor.
2. On the **Overview** tab, in the **Health events timeline** section, choose **Update thresholds**.
3. On the dialog page that opens, choose the new values and options that you want for thresholds and other options that trigger Internet Monitor to create a health event. You can do any of the following:
 - Choose a new value for **Availability score threshold**, **Performance score threshold**, or both.

The graphs in the sections for each setting display the current threshold setting and the actual recent health event scores, for availability or performance, for your application. By viewing the typical values, you can get an idea of values that you might want to change a threshold to.

Tip: To view a larger graph and change the timeframe, choose the expander in the upper right corner of the graph.

- Choose to turn on or off a local threshold for availability or performance, or both. When an option is enabled, you can set the threshold and impact level for when you want Internet Monitor to create a health event.

4. After you configure threshold options, save your updates by choosing **Update health event thresholds**.

To learn more about how health events work, see [When Internet Monitor creates and resolves health events](#).

Health events table

The **Health events** table lists client locations that have been affected by health events, along with information about the events. The following columns are included in the table.

	Description
Client location	<p>The location of the end users who were impacted by the event, who experienced increased latency or reduced availability.</p> <p>To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.</p>
Traffic impact	<p>How much impact was caused by the event, in increased latency or reduced availability. For latency, this is the percentage of how much latency increased during the event compared to typical performance for traffic, from this client location to this AWS location using this client network.</p>
Client network	<p>The network that the traffic traveled over. Typically, this is the internet service provider (ISP) or Autonomous System Number (ASN) for the network traffic.</p>
AWS location	<p>The AWS location for the network traffic, which can be an AWS Region or an internet edge location.</p>

	Description
Impact type	<p>The type of impact for the health event. Health events are typically caused by latency increases (performance issues) or reachability (availability issues).</p> <p>You might also be able to click on the impact type to see the cause of the impairment. When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by AWS or an ASN (internet service provider).</p> <p>Note that this analysis continues after the event is resolved. Internet Monitor can update events with new information for up to an hour.</p>

If you choose one of the client locations in the **Health events** table, you can see more details about the health event at that location. For example, you can see when the event started, when it ended, and the local traffic impact.

Network path visualization

Impairment analysis that is complete has a full network path under **Network path visualization**. The full path shows you each node along the network path for your application for the health event, between the AWS location and the client, for a client-location pair.

If Internet Monitor determines the cause of an impairment, it's marked with a dashed red circle. Impairments can be caused by ASNs, typically internet service providers (ISPs), or the cause can be AWS. If there were multiple causes for an impairment, multiple nodes are circled.

Filtering and viewing historical data in Amazon CloudWatch Internet Monitor (Historical explorer tab)

Use the **Historical explorer** tab in the CloudWatch console, under Internet Monitor, to filter and view data for your application that's in CloudWatch Logs. Internet Monitor publishes

measurements to CloudWatch Logs specific to your application for availability, performance, monitored bytes transferred (or client connection count, for WorkSpaces directories only), and round-trip time for your monitored city-networks in AWS Regions.

Note

Internet Monitor publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks (that is, client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

To start exploring your application's data, select a time period. Then, choose a specific geographical location, such a city, and (optionally) other filters. Internet Monitor applies the filters to the data in the internet measurements logs that it has published for the city-networks for your application traffic. Then you can see graphs of the data that show the performance score, availability score, monitored bytes transferred (for VPCs, Network Load Balancers, and CloudFront distributions) or client connection counts (for WorkSpaces directories), and round-trip time (RTT) for your application over time.

The **All events** table below the graphs shows you the health events that your filter returns for your application traffic, with information about each event. It includes the following columns.

	Description
Event start	The time that the health event started.
Status	Whether the event is still active or is resolved.
Client location	The location of the end users who were impacted by the event, who experienced increased latency or reduced performance.

	Description
	To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor .
Traffic impact	The weighted impact of the event on the location of the health event. That is, for example, the impact on latency, compared to typical performance for traffic from a client location to the AWS location over the client ASN, typically an internet service provider (ISP). Similarly, for an event that affects availability, you see the impact on availability compared to typical availability for the client location for the AWS location over the client ASN.
Event duration	How long the event lasted. Internet Monitor terminates health events when they no longer affect more than 5% (in total) of your application's client locations.
Client ISP	The ASN, typically the internet service provider (ISP), that was the carrier for the network traffic.
Service location	The service location that the network traffic originated from, which can be an AWS Region or internet edge location.

Alternatively, you can look at your application's measurements by accessing the logs directly for each data point. In the **Actions** menu, choose **View CloudWatch Logs**. Note that because measurement events are published to your account when they're created, you can create other CloudWatch dashboards or alarms based on them as well. For more information, see [Getting insights to improve application performance in Amazon CloudWatch Internet Monitor \(Traffic insights tab\)](#) and [Creating alarms with Amazon CloudWatch Internet Monitor](#).

In addition to exploring and analyzing Internet Monitor measurements and metrics, and creating dashboards and alarms based on them, you can use Internet Monitor to help you understand ways that you could improve performance for your application. The **Traffic insights** tab has several ways to help you explore options. For more information, see **Traffic optimization suggestions** on the [Traffic insights](#) tab. In addition, you can see the specific examples in the [Internet Monitor use cases](#) chapter.

Getting insights to improve application performance in Amazon CloudWatch Internet Monitor (Traffic insights tab)

Use the **Traffic insights** tab in the CloudWatch console, under Internet Monitor, to look at summary information for top traffic (by volume) for your application. You can filter and sort your application traffic in multiple ways. Then, scroll down, and select different setup combinations for your application to see what Internet Monitor suggests for the best alternatives to obtain the fastest time to first byte (TTFB) performance.

Internet Monitor publishes to CloudWatch Logs internet measurements every five minutes for the top 500 (by traffic volume) city-networks (that is, client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see [Publishing internet measurements to Amazon S3 in Amazon CloudWatch Internet Monitor](#).

Top traffic summaries

You can start by viewing high-level summaries of your application's overall traffic and performance, over a specific time period, filtered by client location. You can also look at performance for your application for the top (or bottom) client locations by traffic volume, filtered and sorted in multiple ways. For example, you can sort by granularity (that is, city, subdivision, country, or metro area), by total traffic, average time to first byte (TTFB), and other factors.

To learn more about client location accuracy in Internet Monitor, see [Geolocation information and accuracy in Internet Monitor](#).

Note

The filters that you use apply to whole page, so they affect which city-networks are included in the summary graphs and information for total traffic, and also which city-networks are included in the **Traffic optimization suggestions** section that follows.

Traffic optimization suggestions

The **Traffic optimization suggestions** section displays a filtered set of monitored city-networks (locations and ASNs, internet service providers) for your traffic, along with the total client traffic for each one. The entries in the table is based on the filters that you chose for your application traffic for **Traffic insights** at the top of the page. The default is the top 10 cities by traffic volume. You typically see more than 10 rows in the table, because there's entry for each unique city-network pair. That is, there's one row for each combination of location (city) and ASN (network provider) that clients access your application through—such as, Dallas, Texas, USA and Comcast, for example.

Note

To see traffic optimization suggestions for all your monitored city-networks, you can run a query directly in CloudWatch Insights. For a sample query that doesn't include the geographical granularity filter that limits the city-networks list on this page, see [Using CloudWatch Logs Insights with Amazon CloudWatch Internet Monitor](#).

In this section, select different options: Amazon EC2, CloudFront, or both. This lets you see what the predicted average time to first byte (TTFB) values are for clients when you use your application with those services in different AWS Regions, compared to the current TTFB. For more information about TTFB calculations, see [AWS calculations for TTFB and latency](#).

By selecting different the options, and then viewing the results in the table, you can start planning setups and deployments that can improve performance for your clients. Note that you might see a dash (-) instead of a value in a column, when data is not available to display. To review a specific example of how to improve performance, see [Using Amazon CloudWatch Internet Monitor for a Better Gaming Experience](#).

For example, to get started, for a specific city-network (client location and ASN pair), experiment with selecting either the EC2 or CloudFront option, or both. For each city-network

listed in the table, Internet Monitor shows you the potential performance improvements to the TTFB, based on a traffic routing choice (through a specific AWS Region) with that option, compared to the current setup. (Note that, for completeness, the table also includes routes that are already optimized.) For example, you might see a predicted average TTFB that is 50ms for using EC2 routing through us-east-1 compared to your current setup with a TTFB of 100ms where you're using EC2 routing through us-west-2. So you might consider routing through us-west-2.

As another example, you might select EC2, and then see that it doesn't make a measurable performance difference for one client location and ASN, but then note that when you select CloudFront with the same Region, it lowers the TTFB somewhat. This suggests that if you add a CloudFront distribution in front of your application, it could result in a performance improvement and might be worth trying, for this client location and ASN.

Exploring your data with CloudWatch tools and the Internet Monitor query interface

In addition to visualizing your performance and availability for your application with the Amazon CloudWatch Internet Monitor dashboard, there are several methods that you can use to dive deeper into the data that Internet Monitor generates for you. These methods include using CloudWatch tools with Internet Monitor data stored in CloudWatch Log files and using the Internet Monitor query interface. The tools that you can use include CloudWatch Logs Insights, CloudWatch Metrics, CloudWatch Contributor Insights, and Amazon Athena. You can use some or all these tools, as well as the dashboard, to explore Internet Monitor data, depending on your needs.

Internet Monitor aggregates CloudWatch metrics about traffic to your application and to each AWS Region, and includes data such as total traffic impact, availability, and round-trip time. This data is published to CloudWatch Logs and is also available to use with the Internet Monitor query interface. Details about geo-granularity and other aspects of the information available to explore for each one varies.

Amazon CloudWatch Internet Monitor publishes data for your monitor at 5 minute intervals, and then makes the data available in several ways. The following table lists scenarios for accessing Internet Monitor data, and describes features of the data that is collected for each one.

Feature	CloudWatch Logs	Export to S3	Query interface	CloudWatch dashboard
Enabled by default	Yes	No	Yes	Yes
Number of city-networks that data is collected for	Top 500 (see note below)	All	All	All
Data retention	User controlled	User controlled	30 days	30 days
Geo-granularities that data is collected for	All (city-network, metro+network, subdivision+network, country+network)	City-network	All (city-network, metro+network, subdivision+network, country+network)	All (city-network, metro+network, subdivision+network, country+network)
How to query and filter data	Using CloudWatch Logs Insights with Amazon CloudWatch Internet Monitor	Using Amazon Athena to query internet measurements in Amazon S3 log files	Using the Amazon CloudWatch Internet Monitor query interface	Monitor and optimize with the Internet Monitor dashboard

Note: Top 500 measurements are captured for city-networks; top 250 for metro+networks, top 100 for subdivision+networks, top 50 for country+networks.

This chapter describes how to query and explore your data by using CloudWatch tools or the Internet Monitor query interface, together with examples for each method.

Contents

- [Using CloudWatch Logs Insights with Amazon CloudWatch Internet Monitor](#)
- [Using Contributor Insights with Amazon CloudWatch Internet Monitor](#)
- [Using CloudWatch Metrics with Amazon CloudWatch Internet Monitor](#)

- [Using Amazon Athena to query internet measurements in Amazon S3 log files](#)
- [Using the Amazon CloudWatch Internet Monitor query interface](#)

Using CloudWatch Logs Insights with Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor publishes granular measurements of availability and round-trip time to CloudWatch Logs, and you can use CloudWatch Logs Insights queries to filter a subset of logs for a specific city or geography (client location), client ASN (ISP), and AWS source location.

To learn more about client location accuracy in Internet Monitor, see [Geolocation information and accuracy in Internet Monitor](#).

The examples in this section can help you create CloudWatch Logs Insights queries to learn more about your own application traffic measurements and metrics. If you use these examples in CloudWatch Logs Insights, replace *monitorName* with your own monitor name.

View traffic optimization suggestions

On the **Traffic insights** tab in Internet Monitor, you can view traffic optimization suggestions, filtered by a location. To see the same information that is displayed in the **Traffic optimization suggestions** section on that tab, but without the location granularity filter, you can use the following CloudWatch Logs Insights query.

1. In the AWS Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/monitorName/byCity` and `/aws/internet-monitor/monitorName/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

```
fields @timestamp,
clientLocation.city as @city, clientLocation.subdivision as @subdivision,
clientLocation.country as @country,
`trafficInsights.timeToFirstByte.currentExperience.serviceName` as @serviceNameField,
concat(@serviceNameField, ` (`, `serviceLocation`, `)` as @currentExperienceField,
concat(`trafficInsights.timeToFirstByte.ec2.serviceName`, ` (`,
`trafficInsights.timeToFirstByte.ec2.serviceLocation`, `)` as @ec2Field,
`trafficInsights.timeToFirstByte.cloudfront.serviceName` as @cloudfrontField,
concat(`clientLocation.networkName`, ` (AS`, `clientLocation.asn`, `)` as @networkName
| filter ispresent(`trafficInsights.timeToFirstByte.currentExperience.value`)
```

```

| stats avg(`trafficInsights.timeToFirstByte.currentExperience.value`) as @averageTTFB,
avg(`trafficInsights.timeToFirstByte.ec2.value`) as @ec2TTFB,
avg(`trafficInsights.timeToFirstByte.cloudfront.value`) as @cloudfrontTTFB,
sum(`bytesIn` + `bytesOut`) as @totalBytes,
latest(@ec2Field) as @ec2,
latest(@currentExperienceField) as @currentExperience,
latest(@cloudfrontField) as @cloudfront,
count(*) by @networkName, @city, @subdivision, @country
| display @city, @subdivision, @country, @networkName, @totalBytes, @currentExperience,
@averageTTFB, @ec2, @ec2TTFB, @cloudfront, @cloudfrontTTFB
| sort @totalBytes desc

```

View internet availability and RTT (p50, p90, and p95)

To view the internet availability and round-trip time (p50, p90, and p95) for traffic, you can use the following CloudWatch Logs Insights query.

End user geography: Chicago, IL, United States

End user network (ASN): AS7018

AWS service location: US East (N. Virginia) Region

To view the logs, do the following:

1. In the AWS Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/monitorName/byCity` and `/aws/internet-monitor/monitorName/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

The query returns all the performance data for users connecting from AS7018 in Chicago, IL towards US East (N. Virginia) Region over the selected time period.

```

fields @timestamp,
internetHealth.availability.experienceScore as availabilityExperienceScore,
internetHealth.availability.percentageOfTotalTrafficImpacted as
percentageOfTotalTrafficImpacted,
internetHealth.performance.experienceScore as performanceExperienceScore,
internetHealth.performance.roundTripTime.p50 as roundTripTimep50,
internetHealth.performance.roundTripTime.p90 as roundTripTimep90,

```

```
internetHealth.performance.roundTripTime.p95 as roundTripTimep95
| filter clientLocation.country == `United States`
and clientLocation.city == `Chicago`
and serviceLocation == `us-east-1`
and clientLocation.asn == 7018
```

For more information, see [Analyzing log data with CloudWatch Logs Insights](#).

Using Contributor Insights with Amazon CloudWatch Internet Monitor

CloudWatch Contributor Insights can help you identify top client locations and networks (ASNs or internet service providers) for your application. Use the following sample Contributor Insights rules to get started with rules that are useful with Amazon CloudWatch Internet Monitor. For more information, see [Create a Contributor Insights rule](#).

To learn more about client location accuracy in Internet Monitor, see [Geolocation information and accuracy in Internet Monitor](#).

Note

Internet Monitor publishes data every five minutes, so after you set up a Contributor Insights rule, you must adjust the period to five minutes to see a graph.

View top locations and ASNs impacted by an availability impact

To view top client locations and ASNs impacted by a drop in availability, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Sum",
  "Contribution": {
    "Filters": [
      {
        "Match": "$.clientLocation.city",
```

```

        "IsPresent": true
    }
],
"Keys": [
    "$.clientLocation.city",
    "$.clientLocation.networkName"
],
"ValueOf": "$.awsInternetHealth.availability.percentageOfTotalTrafficImpacted"
},
"LogFormat": "JSON",
"LogGroupNames": [
    "/aws/internet-monitor/monitor-name/byCity"
]
}

```

View top client locations and ASNs impacted by a latency impact

To view top client locations and ASNs impacted by an increase in round-trip time (latency), you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```

{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Sum",
  "Contribution": {
    "Filters": [
      {
        "Match": "$.clientLocation.city",
        "IsPresent": true
      }
    ],
    "Keys": [
      "$.clientLocation.city",
      "$.clientLocation.networkName"
    ],
    "ValueOf": "$.awsInternetHealth.performance.percentageOfTotalTrafficImpacted"
  },
  "LogFormat": "JSON",
  "LogGroupNames": [
    "/aws/internet-monitor/monitor-name/byCity"
  ]
}

```



```
}
```

View top client locations and ASNs impacted by total percentage of traffic

To view top client locations and ASNs impacted by total percentage of traffic, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Sum",
  "Contribution": {
    "Filters": [
      {
        "Match": "$.clientLocation.city",
        "IsPresent": true
      }
    ],
    "Keys": [
      "$.clientLocation.city",
      "$.clientLocation.networkName"
    ],
    "ValueOf": "$.percentageOfTotalTraffic"
  },
  "LogFormat": "JSON",
  "LogGroupNames": [
    "/aws/internet-monitor/monitor-name/byCity"
  ]
}
```

Using CloudWatch Metrics with Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor publishes metrics to your account, including metrics for performance, availability, round-trip time, and throughput (bytes per second), which you can view in CloudWatch Metrics in the CloudWatch console. To find all metrics for your monitor, in the CloudWatch Metrics dashboard, see the custom namespace `AWS/InternetMonitor`.

Metrics are aggregated across all internet traffic to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories in the monitor, and to all traffic to each AWS Region and

internet edge location that is monitored. Regions are defined by the service location, which can either be all locations or a specific Region, such as us-east-1.

Note: *city-networks* are client locations and ASNs (typically internet service providers or ISPs).

Internet Monitor provides the following metrics.

Metric	Description
PerformanceScore	A performance score represents the estimated percentage of traffic that is not seeing a performance drop.
AvailabilityScore	An availability score represents the estimated percentage of traffic that is not seeing an availability drop.
BytesIn	Bytes transferred in for your application internet traffic at all application city-networks.
BytesOut	Bytes transferred out for your application internet traffic at all application city-networks.
BytesInMonitored	Bytes transferred in for your application internet traffic at monitored city-networks.
BytesOutMonitored	Bytes transferred out for your application internet traffic at monitored city-networks.
Round-trip time (RTT)	Round-trip time between the AWS Regions, ASNs (typically internet service providers or ISPs), and locations (such as cities) specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories.
CityNetworksMonitored	The number of city-networks Internet Monitor monitored for your application internet traffic. This is never more than the upper limit that

Metric	Description
	you set as the maximum city-networks for the monitor.
TrafficMonitoredPercent	The percentage of total application internet traffic for this monitor that is represented (included) by the city-networks that Internet Monitor is monitoring. This is less than 100 (that is, less than 100%) if clients access your application in more city-networks than the maximum city-networks limit that you have set for the monitor.
CityNetworksFor100PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 100% of your application internet traffic in Internet Monitor.
CityNetworksFor99PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 99% of your application internet traffic in Internet Monitor.
CityNetworksFor95PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 95% of your application internet traffic in Internet Monitor.
CityNetworksFor90PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 90% of your application internet traffic in Internet Monitor.
CityNetworksFor75PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 75% of your application internet traffic in Internet Monitor.

Metric	Description
CityNetworksFor50PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 50% of your application internet traffic in Internet Monitor.
CityNetworksFor25PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 25% of your application internet traffic in Internet Monitor.

Note

To see examples for using several of these metrics to help determine values to choose for a city-networks maximum for your monitor, see [Choosing a city-network maximum value](#).

For more information, see [Use Amazon CloudWatch metrics](#).

Using Amazon Athena to query internet measurements in Amazon S3 log files

You can use Amazon Athena to query and view the internet measurements that Amazon CloudWatch Internet Monitor publishes to an Amazon S3 bucket. There's an option in Internet Monitor to publish internet measurements for your application to an S3 bucket for internet-facing traffic for your monitored city-networks (client locations and ASNs, typically internet service providers or ISPs). Regardless of whether you choose to publish measurements to S3, Internet Monitor automatically publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks for each monitor.

This chapter includes steps for how to create a table in Athena for internet measurements located in an S3 log file, and then provides [example queries](#) to see different views of the measurements. For example, you can query for your top 10 impacted city-networks by latency impact.

Using Amazon Athena to create a table for internet measurements in Internet Monitor

To start using Athena with your Internet Monitor S3 log files, you first create a table for the internet measurements.

Follow the steps in this procedure to create a table in Athena based on the S3 log files. Then, you can run Athena queries on the table, such as [these example internet measurements queries](#), to get information about your measurements.

To create an Athena table

1. Open the Athena console at <https://console.aws.amazon.com/athena/>.
2. In the Athena query editor, enter a query statement to generate a table with Internet Monitor internet measurements. Replace the value for the LOCATION parameter with the location of S3 bucket where your Internet Monitor internet measurements are stored.

```
CREATE EXTERNAL TABLE internet_measurements (  
    version INT,  
    timestamp INT,  
    clientlocation STRING,  
    servicelocation STRING,  
    percentageoftotaltraffic DOUBLE,  
    bytesin INT,  
    bytesout INT,  
    clientconnectioncount INT,  
    internethealth STRING,  
    trafficinsights STRING  
)  
PARTITIONED BY (year STRING, month STRING, day STRING)  
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  
LOCATION  
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/'  
TBLPROPERTIES ('skip.header.line.count' = '1');
```

3. Enter a statement to create a partition to read the data. For example, the following query creates a single partition for a specified date and location:

```
ALTER TABLE internet_measurements  
ADD PARTITION (year = 'YYYY', month = 'MM', day = 'dd')  
LOCATION  
's3://bucket_name/bucket_prefix/AWSLogs/account_id/internetmonitor/AWS_Region/YYYY/  
MM/DD';
```

4. Choose **Run**.

Example Athena statements for internet measurements

The following is an example of a statement to generate a table:

```
CREATE EXTERNAL TABLE internet_measurements (  
    version INT,  
    timestamp INT,  
    clientlocation STRING,  
    servicelocation STRING,  
    percentageoftotaltraffic DOUBLE,  
    bytesin INT,  
    bytesout INT,  
    clientconnectioncount INT,  
    internethealth STRING,  
    trafficinsights STRING  
)  
PARTITIONED BY (year STRING, month STRING, day STRING)  
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'  
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/  
us-east-2/'  
TBLPROPERTIES ('skip.header.line.count' = '1');
```

The following is an example of a statement to create a partition to read the data:

```
ALTER TABLE internet_measurements  
ADD PARTITION (year = '2023', month = '04', day = '07')  
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/  
us-east-2/2023/04/07/'
```

Sample Amazon Athena queries to use with internet measurements in Internet Monitor

This section includes example queries that you can use with Amazon Athena to get information about your application's internet measurements published to Amazon S3.

Query your top 10 impacted (by total percentage of traffic) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,  
    json_extract_scalar(clientLocation, '$.networkname') as networkName,  
    sum(percentageoftotaltraffic) as percentageoftotaltraffic  
FROM internet_measurements  
GROUP BY json_extract_scalar(clientLocation, '$.city'),
```

```

    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageoftotaltraffic desc
limit 10

```

Query your top 10 impacted (by availability) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```

SELECT json_extract_scalar(clientLocation, '$.city') as city,
       json_extract_scalar(clientLocation, '$.networkname') as networkName,
       sum(
         cast(
           json_extract_scalar(
             internetHealth,
             '$.availability.percentageoftotaltrafficimpacted'
           )
         as double )
       ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
         json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10

```

Query your top 10 impacted (by latency) client locations and ASNs

Run this Athena query to return your top 10 impacted (by latency impact) city-networks—that is, client locations and ASNs, typically internet service providers.

```

SELECT json_extract_scalar(clientLocation, '$.city') as city,
       json_extract_scalar(clientLocation, '$.networkname') as networkName,
       sum(
         cast(
           json_extract_scalar(
             internetHealth,
             '$.performance.percentageoftotaltrafficimpacted'
           )
         as double )
       ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),

```

```
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

Query traffic highlights for your client locations and ASNs

Run this Athena query to return traffic highlights, including availability score, performance score, and time to first byte for your city-networks—that is, client locations and ASNs, typically internet service providers. .

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
       json_extract_scalar(clientLocation, '$.subdivision') as subdivision,
       json_extract_scalar(clientLocation, '$.country') as country,
       avg(cast(json_extract_scalar(internetHealth, '$.availability.experiencescore') as
double)) as availabilityScore,
       avg(cast(json_extract_scalar(internetHealth, '$.performance.experiencescore') as
double)) performanceScore,
       avg(cast(json_extract_scalar(trafficinsights,
'$.timetofirstbyte.currentexperience.value') as double)) as averageTTFB,
       sum(bytesIn) as bytesIn,
       sum(bytesOut) as bytesOut,
       sum(bytesIn + bytesOut) as totalBytes
FROM internet_measurements
where json_extract_scalar(clientLocation, '$.city') != 'N/A'
GROUP BY
json_extract_scalar(clientLocation, '$.city'),
       json_extract_scalar(clientLocation, '$.subdivision'),
       json_extract_scalar(clientLocation, '$.country')
ORDER BY totalBytes desc
limit 100
```

For more information about using Athena, see the [Amazon Athena User Guide](#).

Using the Amazon CloudWatch Internet Monitor query interface

An option for understanding more about internet traffic for your AWS application is to use the Amazon CloudWatch Internet Monitor *query interface*. To use the query interface, you create a query with data filters that you choose, and then run the query to return a subset of your Internet Monitor data. Exploring the data that the query returns can give you insights into how your application is performing on the internet.

You can query and explore all the metrics that Internet Monitor captures with your monitor, including availability and performance scores, bytes transferred, round-trip times, and time to first byte (TTFB).

Internet Monitor uses the query interface to provide the data that you can explore in the Internet Monitor console dashboard. By using search options in the dashboard—on the **Historical explorer** tab or the **Traffic insights** tab—you can query and filter internet data for your application.

If you'd like more flexibility to explore and filter your data than the dashboard provides, you can use the query interface yourself, by using Internet Monitor API operations with the AWS Command Line Interface or with an AWS SDK. This section introduces the types of queries that you can use with the query interface, and the filters that you can specify to create a subset of data, to get insights about internet traffic for your application.

Topics

- [How to use the query interface](#)
- [Query examples](#)
- [Get query results](#)
- [Troubleshooting](#)

How to use the query interface

You create a query with the query interface by choosing a *query type*, and then specifying filter values, to return a specific desired subset of your log file data. Then, you can work with the data subset, to further filter and sort, create reports, and so on.

The query process works like this:

1. When you run a query, Internet Monitor returns a `query ID` that is unique to the query. This section describes the query types that are available, and options for filtering data in queries. To understand how this works, you can also review the section on [query examples](#).
2. You specify the query ID with your monitor name with the [GetQueryResults](#) API operation to return data results for the query. Each query type returns a different set of data fields. To learn more, see [Get query results](#).

The query interface provides the following three query types. Each query type returns a different set of information about your traffic from the log files, as shown.

- **Measurements:** Provides availability score, performance score, total traffic, and round-trip times, at 5 minute intervals.
- **Top locations:** Provides availability score, performance score, total traffic, and time to first byte (TTFB) information, for the top location and ASN combinations that you're monitoring, by traffic volume.
- **Top locations details:** Provides TTFB for Amazon CloudFront, your current configuration, and the best performing Amazon EC2 configuration, at 1 hour intervals.

With each of these query types, you can filter the data more by specifying one or more of the following criteria:

- **AWS location:** For AWS location, you can specify CloudFront or an AWS Region, such as us-east-2, us-west-2, and so on.
- **ASN:** Specify an ASN, which is typically an internet service provide (ISP).
- **Client location:** For location, specify a city, metro, subdivision, or country.
- **Geo:** Specify geo for some queries. This is required for queries that use the Top locations query type, but not allowed for other query types. To understand when to specify geo for filter parameters, see the [query examples](#) section.

The operators that you can use for filtering your data are EQUALS and NOT_EQUALS. For details about filtering parameters, see the [FilterParameter](#) API operation.

To see details about the query interface operations, see the following API operations in the Amazon CloudWatch Internet Monitor API Reference Guide:

- To create and run a query, see the [StartQuery](#) API operation.
- To stop a query, see the [StopQuery](#) API operation.
- To return data for a query that you've created, see the [GetQueryResults](#) API operation.
- To retrieve the status of a query, see the [GetQueryStatus](#) API operation.

Query examples

To create a query that you can use to retrieve a filtered set of data from your monitor's log file, you use the [StartQuery](#) API operation. You specify a query type and filter parameters for the query.

Then, when you use the Internet Monitor query interface API operation to get query results using the query, it will retrieve the subset of your data that you want to work with.

To illustrate how query types and filter parameters work, let's look at some examples.

Example 1

Let's say that you want to retrieve all of your monitor's log file data for a specific country, except for one city. The following example shows filter parameters for a query that you could create with the `StartQuery` operation for this scenario.

```
{
  MonitorName: "TestMonitor"
  StartTime: "2023-07-12T20:00:00Z"
  EndTime: "2023-07-12T21:00:00Z"
  QueryType: "MEASUREMENTS"
  FilterParameters: [
    {
      Field: "country",
      Operator: "EQUALS",
      Values: ["Germany"]
    },
    {
      Field: "city",
      Operator: "NOT_EQUALS",
      Values: ["Berlin"]
    },
  ]
}
```

Example 2

As another example, let's say that you want to see your top locations by metropolitan area. You could use the following example query for this scenario.

```
{
  MonitorName: "TestMonitor"
  StartTime: "2023-07-12T20:00:00Z"
  EndTime: "2023-07-12T21:00:00Z"
  QueryType: "TOP_LOCATIONS"
  FilterParameters: [
    {
      Field: "geo",
```

```
    Operator: "EQUALS",
    Values: ["metro"]
  },
]
}
```

Example 3

Now, let's say that you want to see the top city-network combinations in the Los Angeles metro area. To do this, specify `geo=city`, and then set `metro` to Los Angeles. Now, the query returns the top city-networks in the Los Angeles metro area instead of the top metro+networks overall.

Here's the example query that you could use:

```
{
  MonitorName: "TestMonitor"
  StartTime: "2023-07-12T20:00:00Z"
  EndTime: "2023-07-12T21:00:00Z"
  QueryType: "TOP_LOCATIONS"
  FilterParameters: [
    {
      Field: "geo",
      Operator: "EQUALS",
      Values: ["city"]
    },
    {
      Field: "metro",
      Operator: "EQUALS",
      Values: ["Los Angeles"]
    }
  ]
}
```

Example 4

Finally, let's say that you want to retrieve TTFB data for a specific subdivision (for example, a U.S. state).

The following is an example query for this scenario:

```
{
  MonitorName: "TestMonitor"
  StartTime: "2023-07-12T20:00:00Z"
```

```
EndTime: "2023-07-12T21:00:00Z"
QueryType: "TOP_LOCATION_DETAILS"
FilterParameters: [
  {
    Field: "subdivision",
    Operator: "EQUALS",
    Values: ["California"]
  },
]
}
```

Get query results

After you define a query, you can return a set of results with the query by running another Internet Monitor API operation, [GetQueryResults](#). When you run `GetQueryResults`, you specify the query ID for the query that you've defined, along with the name of your monitor. `GetQueryResults` retrieves data for the specified query into a result set.

When you run a query, make sure that the query has finished running before you use `GetQueryResults` to look at the results. You can determine if the query has completed by using the [GetQueryStatus](#) API operation. When the Status for the query is `SUCCEEDED`, you can go ahead with reviewing the results.

When your query completes, you can use the following information to help you review the results. Each query type that you use to create a query includes a unique set of data fields from the log files, as described in the following list:

Measurements

The measurements query type returns the following data:

```
timestamp, availability, performance, bytes_in, bytes_out, rtt_p50,
rtt_p90, rtt_p95
```

Top locations

The top locations query type groups data by location, and provides the data averaged over the time period. The data that it returns includes the following:

```
aws_location, city, metro, subdivision, country, asn, availability,
performance, bytes_in, bytes_out, current_fbl, best_ec2,
best_ec2_region, best_cf_fbl
```

Note that `city`, `metro`, and `subdivision` are only returned if you choose that location type for the `geo` field. The following location fields are returned, depending on the location type that you specify for `geo`:

```
city = city, metro, subdivision, country
metro = metro, subdivision, country
subdivision = subdivision, country
country = country
```

Top locations details

The `top locations details` query type returns data grouped hour by hour. The query returns the following data:

```
timestamp, current_service, current_fbl, best_ec2_fbl, best_ec2_region,
best_cf_fbl
```

When you run the `GetQueryResults` API operation, Internet Monitor returns the following in the response:

- A *data string array* that contains the results that the query returns. The information is returned in arrays that are aligned with the `Fields` field, also returned by the API call. Using the `Fields` field, you can parse the information from the Data repository and then further filter or sort it for your purposes.
- An *array of fields* that lists the fields that the query returned data for (in the Data field response). Each item in the array is a name-datatype pair, such as `availability_score-float`.

Troubleshooting

If errors are returned when you use query interface API operations, verify that you have the required permissions to use Amazon CloudWatch Internet Monitor. Specifically, make sure that you have the following permissions:

```
internetmonitor:StartQuery
internetmonitor:GetQueryStatus
internetmonitor:GetQueryResults
internetmonitor:StopQuery
```

These permissions are included in the recommended AWS Identity and Access Management policy to use the Internet Monitor dashboard in the console. For more information, see [IAM permissions for Amazon CloudWatch Internet Monitor](#).

Creating alarms with Amazon CloudWatch Internet Monitor

You can create Amazon CloudWatch alarms based on Amazon CloudWatch Internet Monitor metrics, just as you can for other Amazon CloudWatch metrics.

For example, you can create an alarm based on the Internet Monitor metric `PerformanceScore`, and configure it to send a notification when the metric is lower than a value that you choose. You configure alarms for Internet Monitor metrics following the same guidelines as for other CloudWatch metrics.

Following are the example Internet Monitor metrics that you might choose to create an alarm for:

- **PerformanceScore**
- **AvailabilityScore**
- **RoundtripTime**

To see all the metrics available for Internet Monitor, see [Using CloudWatch Metrics with Amazon CloudWatch Internet Monitor](#).

The following procedure provides an example of setting an alarm on **PerformanceScore** by navigating to the metric in the CloudWatch dashboard. Then, you follow the standard CloudWatch steps to create an alarm based on a threshold that you choose, and set up a notification or choose other options.

To create an alarm for PerformanceScore in CloudWatch Metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Metrics**, and then choose **All metrics**.
3. Filter for Internet Monitor by choosing `AWS/InternetMonitor`.
4. Choose **MeasurementSource**, **MonitorName**.
5. In the list, select **PerformanceScore**.
6. On the **GraphedMetrics** tab, under **Actions**, choose the bell icon to create an alarm based on a static threshold.

Now, follow the standard CloudWatch steps to choose options for the alarm. For example, you can choose to be notified by an Amazon SNS message if **PerformanceScore** is below a specific threshold number. Alternatively, or in addition, you can add the alarm to a dashboard.

Keep in mind the following:

- Internet Monitor metrics are typically calculated and published within 20 minutes.
- When you create an alarm based on Internet Monitor metrics, make sure that you take into account the short delay before publication when you set an alarm's lookback period. We recommend that you configure **Evaluation Periods** with lookback period that is a minimum of 25 minutes.

To learn more about using CloudWatch alarms with Internet Monitor, see the following blog post: [Using Amazon CloudWatch Internet Monitor for enhanced internet observability](#).

For more information about options when you create a CloudWatch alarm, see [Create a CloudWatch alarm based on a static threshold](#).

Using Amazon CloudWatch Internet Monitor with Amazon EventBridge

The health events that Amazon CloudWatch Internet Monitor creates for networking issues are published with Amazon EventBridge, so that you can send notifications about any degradation in end users' experience for your application.

To use EventBridge to work with Internet Monitor health events, follow the guidance here.

To set up a rule for Internet Monitor in EventBridge

1. In the AWS Management Console, in EventBridge, choose **Rules**, then enter a name and a description. Create the rule on the **Default** event bus.
2. In Step 2, select **Other** for the event source, and then, under **Event pattern**, match the following source.

```
{
  "source": ["aws.internetmonitor"]
}
```

3. In Step 3, for the target, select **AWS Service** and **CloudWatch Logs Group**, then select an existing log group or create a new one.

4. Add any desired tags, and then create the rule. This should populate your selected CloudWatch Logs Group with events from EventBridge.

For more information about how EventBridge rules work with event patterns, see [Amazon EventBridge event patterns](#) in the Amazon EventBridge User Guide.

Troubleshoot CloudWatch logs and metrics access errors

To support some features, Amazon CloudWatch Internet Monitor must interact with certain Amazon CloudWatch resources, including logs and metrics. If Internet Monitor can't access the CloudWatch resources that it requires access to, Internet Monitor sets a status code of `FAULT_ACCESS_CLOUDWATCH` for the monitor.

There are several reasons that your monitor might have the state `FAULT_ACCESS_CLOUDWATCH`. The following sections list possible causes for these errors, and suggested troubleshooting steps.

Internet Monitor couldn't access CloudWatch logs in your account

Internet Monitor publishes diagnostic logs about the application traffic that your monitor tracks. It publishes these logs to log groups in CloudWatch Logs in the following location: `/aws/internet-monitor/monitor_name/[byCity|byMetro|bySubdivision|byCountry]`. Internet Monitor was unable to access these log groups.

Error states and potential solutions:

- **PutLogEvents throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's logs to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **Log group not found:** Disable, and then re-enable your monitor. Enabling a monitor restarts log group creation, which might correct the problem.
- **PutLogEvents access denied error:** Contact AWS support for assistance.
- **PutLogEvents unknown or general error:** Contact AWS support for assistance.

Internet Monitor couldn't access CloudWatch metrics in your account

Internet Monitor delivers specific CloudWatch metrics about the application traffic that is tracked by a monitor. An error occurred when Internet Monitor tried to deliver these metrics to CloudWatch.

Error states and potential solutions:

- **PutMetricData throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's metrics to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **PutMetricData access denied error:** Contact AWS support for assistance.
- **PutMetricData unknown or general error:** Contact AWS support for assistance.

Data protection and data privacy with Amazon CloudWatch Internet Monitor

The AWS [shared responsibility model](#) applies to data protection and data privacy in Amazon CloudWatch Internet Monitor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see [The AWS Shared Responsibility Model and GDPR](#) blog post on the AWS Security Blog. For more resources about complying with GDPR requirements, see the [General Data Protection Regulation \(GDPR\) Center](#).

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Amazon CloudWatch Internet Monitor or other services might be included in diagnostic logs.

Identity and Access Management for Amazon CloudWatch Internet Monitor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Internet Monitor resources. IAM is an AWS service that you can use with no additional charge.

Important

Internet Monitor resource changes on February 24, 2023

If you created IAM policies that included Internet Monitor resources before February 24, 2023, be aware of the following changes to Internet Monitor resources and resource types.

- **HealthEvents** resource was renamed to **HealthEvent**.
- The ARN and Regex formats for the **HealthEvent** resource were updated.
- The ARN and Regex formats for the **Monitor** resource were updated.
- Resource-level permissions for the **GetHealthEvent** action are now supported only on the **HealthEvent** resource type. They're not supported on the **Monitor** resource.
- **TagResource**, **UntagResource**, and **ListTagsForResource** for the **Monitor** resource type were updated to be required.

To see more information about the actions, resources, and condition keys that you can specify in policies to manage access to AWS resources in Internet Monitor, see [Actions, resources, and condition keys for Amazon CloudWatch Internet Monitor](#).

Contents

- [How Amazon CloudWatch Internet Monitor works with IAM](#)
- [AWS managed policies for Amazon CloudWatch Internet Monitor](#)
- [IAM permissions for Amazon CloudWatch Internet Monitor](#)
- [Service-linked role for Amazon CloudWatch Internet Monitor](#)

How Amazon CloudWatch Internet Monitor works with IAM

Before you use IAM to manage access to Internet Monitor, learn what IAM features are available to use with Internet Monitor.

To see tables showing a similar high-level view of how AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

IAM features you can use with Amazon CloudWatch Internet Monitor

IAM feature	Internet Monitor support
Identity-based policies	Yes

IAM feature	Internet Monitor support
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Internet Monitor

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Resource-based policies within Internet Monitor

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource.

Policy actions for Internet Monitor

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Internet Monitor actions, see [Actions defined by Amazon CloudWatch Internet Monitor](#) in the *Service Authorization Reference*.

Policy actions in Internet Monitor use the following prefix before the action:

```
internetmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
  "internetmonitor:action1",  
  "internetmonitor:action2"
```

```
] ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "internetmonitor:Describe*"
```

Policy resources for Internet Monitor

Supports policy resources	Yes
---------------------------	-----

In the *Service Authorization Reference*, you can see the following information related to Internet Monitor:

- To see a list of Internet Monitor resource types and their ARNs, see [Resources defined by Amazon CloudWatch Internet Monitor](#).
- To learn the actions that you can specify with the ARN of each resource, see [Actions defined by Amazon CloudWatch Internet Monitor](#).

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*" ]
```

Policy condition keys for Internet Monitor

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or *Condition block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of Internet Monitor condition keys, see [Condition keys for Amazon CloudWatch Internet Monitor](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon CloudWatch Internet Monitor](#).

ACLs in Internet Monitor

Supports ACLs

No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Internet Monitor

Supports ABAC (tags in policies)

Partial

Internet Monitor has *partial* support for tags in policies. It supports tagging for one resource, monitors.

To use tags with Internet Monitor, use the AWS Command Line Interface or an AWS SDK. Tagging for Internet Monitor is not supported with the AWS Management Console.

To learn more about using tags in policies in general, review the following information.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using temporary credentials with Internet Monitor

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for Internet Monitor

Supports forward access sessions (FAS)	Yes
--	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for Internet Monitor

Supports service roles	No
------------------------	----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Service-linked role for Internet Monitor

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For more information about the service-linked role for Internet Monitor, see [Service-linked role for Amazon CloudWatch Internet Monitor](#).

For details about creating or managing service-linked roles in general in AWS, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

AWS managed policies for Amazon CloudWatch Internet Monitor

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

AWS managed policy: CloudWatchInternetMonitorServiceRolePolicy

This policy is attached to the service-linked role named **AWSServiceRoleForInternetMonitor** to allow Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources or Network Load Balancers, so that you can select them when you create a monitor. For more information, see [Service-linked role for Amazon CloudWatch Internet Monitor](#).

IAM permissions for Amazon CloudWatch Internet Monitor

To access actions for working with monitors and data in Amazon CloudWatch Internet Monitor, users must have the correct permissions.

For more information about security in Amazon CloudWatch, see [Identity and access management for Amazon CloudWatch](#).

Permissions for read-only access in Amazon CloudWatch Internet Monitor

To access read-only actions to work with monitors and data in Amazon CloudWatch Internet Monitor, users must be signed in as a user or role that has the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "internetmonitor:Get*",
        "internetmonitor:List*",
        "internetmonitor:StartQuery",
        "internetmonitor:StopQuery",
        "logs:DescribeLogGroups",
        "logs:GetQueryResults",
        "logs:StartQuery",
        "logs:StopQuery"
      ],
      "Resource": "*"
    }
  ]
}
```

Permissions for full access in Amazon CloudWatch Internet Monitor

To create a monitor in Amazon CloudWatch Internet Monitor, and to have full access to actions for working with monitors and data in Internet Monitor, users must be signed in with a user or role that has the following permissions:

- Permissions to create a service-linked role that is associated with Internet Monitor. For more information, see [Service-linked role for Amazon CloudWatch Internet Monitor](#).
- Permissions to actions that enable full access to work with monitors and data in Internet Monitor.

Note

If you create an identity-based permissions policy that is more restrictive, users with that policy might not have full access to create and work with monitors and data in Internet Monitor.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "internetmonitor:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "internetmonitor.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:PutRolePolicy"
      ],
      "Resource": "arn:aws:iam::*:role/aws-service-role/
internetmonitor.amazonaws.com/AWSServiceRoleForInternetMonitor"
    },
    {
      "Action": [
        "ec2:DescribeVpcs",
        "elasticloadbalancing:DescribeLoadBalancers",
        "workspaces:DescribeWorkspaceDirectories",
        "cloudfront:GetDistribution"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Service-linked role for Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor uses an AWS Identity and Access Management (IAM) [service-linked role](#). A service-linked role is a unique type of IAM role that is linked directly to Internet Monitor. The service-linked role is predefined by Internet Monitor and includes all the permissions that the service requires to call other AWS services on your behalf.

Internet Monitor defines the permissions of the service-linked role, and unless defined otherwise, only Internet Monitor can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your Internet Monitor resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Internet Monitor

Internet Monitor uses the service-linked role named **AWSServiceRoleForInternetMonitor**. This role allows Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources, Amazon CloudFront distributions, Amazon WorkSpaces directories, and Network Load Balancers, so that you can select them when you create a monitor.

This service-linked role uses the managed policy `CloudWatchInternetMonitorServiceRolePolicy`.

The **AWSServiceRoleForInternetMonitor** service-linked role trusts the following service to assume the role:

- `internetmonitor.amazonaws.com`

To view the permissions for this policy, see [CloudWatchInternetMonitorServiceRolePolicy](#) in the *AWS Managed Policy Reference*.

Creating a service-linked role for Internet Monitor

You do not need to manually create the service-linked role for Internet Monitor. The first time that you create a monitor, Internet Monitor creates **AWSServiceRoleForInternetMonitor** for you.

For more information, see [Creating a service-linked role](#) in the *IAM User Guide*.

Editing a service-linked role for Internet Monitor

After Internet Monitor creates a service-linked role in your account, you cannot change the name of the role because various entities might reference the role. You can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Internet Monitor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete the role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for the service-linked role before you can manually delete it.

After you've removed your resources from your monitors in Internet Monitor and then deleted the monitors, you can delete the service-linked role **AWSServiceRoleForInternetMonitor**.

Note

If the Internet Monitor service is using the role when you try to delete it, then the deletion might fail. If that happens, wait for a few minutes and then try again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForInternetMonitor** service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Updates to the Internet Monitor service-linked role

For updates to **AWSServiceRoleForInternetMonitor**, the AWS managed policy for the Internet Monitor service-linked role, see [CloudWatch updates to AWS managed policies](#). For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch [Document history](#) page.

Quotas in Amazon CloudWatch Internet Monitor

Amazon CloudWatch Internet Monitor has the following quotas.

Resource	Default quota
Monitors per Region	50
Resources per monitor	50
Days that resolved Internet Monitor health events are retained	400

Using Amazon CloudWatch Network Monitor

Amazon CloudWatch Network Monitor provides visibility into the performance of the network connecting your AWS hosted applications to your on-premises destinations, and allows you to identify the source of any network performance degradation within minutes. Network Monitor is fully managed by AWS. Therefore, you don't need to install additional agents to monitor your network performance. You can quickly visualize packet loss and latency of your hybrid network connections, set alerts and thresholds, and then take action to improve your end users' network experience.

Network Monitor is intended for network operators and application developers who want real-time insights into network performance.

Key features

- Use Network Monitor to benchmark your changing hybrid network environment with continuous real-time packet loss and latency metrics.
- When you connect by using AWS Direct Connect, Network Monitor rapidly diagnoses network degradation by writing the AWS Network Health Indicator (NHI) to your Amazon CloudWatch account. The NHI metric provides a probabilistic score to determine if the network degradation was within AWS.
- Network Monitor provides a fully-managed agent approach to monitoring, so you don't need to install agents either on VPCs or on-premises. To get started, you just need to specify a VPC subnet and an on-premises IP address.
- Network Monitor publishes metrics to CloudWatch Metrics. You can create dashboards to view your metrics, and also create actionable thresholds and alarms on the metrics that are specific to your application.

For more details, see [the section called “How Network Monitor works”](#).

Network Monitor terminology and components

- **Monitor** — A monitor displays the resources that you can view network performance and availability measurements for, and that you want to get health event alerts about. When you create a monitor for an application, you add an AWS hosted resource as the network source. Network Monitor then creates a list of all possible probes between the AWS hosted resource and your destination IP addresses.
- **Probes** — A probe is the traffic that's sent from the AWS hosted resource to your on-premises destination IP address. Network Monitor metrics are written into your CloudWatch account for every probe that's configured in a monitor.
- **AWS network source** — An AWS network source is a monitor probe's originating AWS source, which is a subnet in one of your VPCs.
- **Destination** — A destination is the target in your on-premises network for the AWS network source. A destination is a combination of your on-premises IP addresses, network protocols, ports, and network packet size. IPv4 and IPv6 addresses are both supported.

Network Monitor limitations and requirements

- Network Monitor supports a maximum of four destination IP addresses, and up to 24 probes per monitor.
- You can have up to 100 monitors per Region per account.
- Monitor subnets must be owned by the same account as the monitor.
- Network Monitor doesn't provide automatic network failover in the event of an AWS network issue.
- There's a charge for each probe that you create. For pricing details, see [the section called “Pricing”](#).

How Amazon CloudWatch Network Monitor works

Network Monitor is fully managed by AWS, and doesn't require agents on monitored resources. When you create a monitor for AWS hosted resources, AWS creates and manages all the infrastructure in the background to perform round-trip time and packet loss measurements.

Because AWS manages the required configurations, you can scale your monitoring rapidly, without needing to install or uninstall agents within your AWS infrastructure.

Network Monitor focuses monitoring on the routes taken by flows from your AWS hosted resources instead of broadly monitoring all flows from your AWS Region. If your workloads spread across multiple Availability Zones, Network Monitor can monitor routes from each of your private subnets.

Network Monitor publishes round-trip time and packet loss metrics to your Amazon CloudWatch account, based on the aggregation interval that you set when you create a monitor. You can also use CloudWatch to set individual latency and packet loss thresholds for each monitor. For example, you might create an alarm for a packet loss sensitive workload to notify you if the packet loss average is higher than a static 0.1% threshold. You can also use CloudWatch anomaly detection to alarm on packet loss or latency metrics that are outside your desired ranges.

Availability and performance measurements

Network Monitor sends periodic active probes from your AWS resource to your on-premises destinations. When you create a monitor, you specify the following:

- **Aggregation interval.** The time, in seconds, that CloudWatch receives the measured results. This will be either every 30 or 60 seconds. The aggregation period you choose for the monitor applies to all probes in that monitor.
- **Probe protocol.** One of the supported protocols, ICMP or TCP. For more information, see [the section called “Communication protocols”](#).
- **Packet size.** The size, in bytes, of each packet transmitted between your AWS hosted resource and your destination on a single probe. You can specify a different packet size for each probe in a monitor.

The metrics that the monitor publishes are the following:

- **Round-trip time.** This metric, measured in milliseconds, is a measure of performance. It records the time it takes for the probe to be transmitted to the destination IP address and for the associated response to be received.
- **Packet loss.** This metric measures the percentage of total packets sent and records the number of transmitted probes that didn't receive an associated response. No response implies that the packets were lost along the network path.

Supported communication protocols

Network Monitor supports two protocols for probes: ICMP and TCP.

ICMP-based probes carry ICMP echo requests from your AWS hosted resources to the destination address and expect an ICMP echo reply in response. Network Monitor uses the information on the ICMP echo request and reply messages to calculate round-trip time and packet loss metrics.

TCP-based probes carry TCP SYN packets from your AWS hosted resources to the destination address and port and expect a TCP SYN+ACK or RST packet in response. Network Monitor uses the information on the TCP SYN and TCP SYN+ACK or RST messages to calculate round-trip time and packet loss metrics. Network Monitor periodically switches source TCP ports to increase network coverage, which increases the probability of detecting packet loss.

AWS Network Health Indicator

Network Monitor publishes a Network Health Indicator (NHI) metric that provides information on network performance and availability for destinations connected through AWS Direct Connect. NHI is a statistical measure of the health of the AWS controlled network path from the AWS hosted resource, where the monitor is deployed, to the Direct Connect location.

Network Monitor uses anomaly detection to calculate availability drops or lower performance along the network paths.

Note

Each time that you create a new monitor, add a probe, or re-activate a probe, the NHI for the monitor is delayed by a few hours while AWS collects data to perform anomaly detection.

To provide the NHI metric, Network Monitor applies statistical correlation across AWS sample datasets, as well as to the packet loss and round-trip latency metrics for traffic simulating your network path. NHI can be one of two values: 1 or 0. A value of 1 indicates that Network Monitor observed a network degradation within the AWS controlled network path. A value of 0 indicates that Network Monitor did not observe any network degradation along the path. Observing the NHI value enables you to gain awareness of network issues more quickly. For example, you can set alerts on the NHI metric to be notified about ongoing issues along your network paths.

Support for IPv4 and IPv6 addresses

Network Monitor provides availability and performance metrics over IPv4 or IPv6 networks and can monitor either IPv4 or IPv6 addresses from dual-stack VPCs. Network Monitor doesn't allow both IPv4 and IPv6 destinations to be configured in the same monitor; you can create separate monitors for IPv4-only and IPv6-only destinations.

Region availability

Network Monitor is currently available in the following AWS Regions:

Region	
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1

Region	
Canada West (Calgary)	ca-west-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2

Region	
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Creating a monitor in Amazon CloudWatch Network Monitor

The following sections describe how to create a monitor, and then add the required probes. For each probe, you choose a source subnet and up to four destination IP addresses, for a maximum of 24 probes per monitor. You can create a monitor by using either the Amazon CloudWatch console or the AWS Command Line Interface.

Topics

- [Creating a monitor in Amazon CloudWatch Network Monitor using the console](#)
- [Creating a monitor in Amazon CloudWatch Network Monitor using the AWS Command Line Interface](#)

Creating a monitor in Amazon CloudWatch Network Monitor using the console

The following steps describe how to create a monitor by using the Amazon CloudWatch console. To create a monitor, you choose the source subnets for the monitor, and then add up to four destinations, to create up to 24 probes per monitor.

Important

These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

Define monitor details

The first step to create a monitor is to define the basic details, by giving the monitor a name and defining the aggregation period. Optionally, you can also add tags.

To define monitor details

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. Choose **Create monitor**.
3. For **Monitor name**, enter a name for the monitor.
4. For **Aggregation period**, choose how often you want to send metrics to CloudWatch **30 seconds** or **60 seconds**.

Note

A shorter aggregation period provides faster detection of network issues; however, the aggregation period you choose can affect your billing structure. For more information about pricing, see the [Amazon CloudWatch pricing](#) page.

5. (Optional) For **Tags**, add **Key** and **Value** pairs to help identify this resource, so that you can search or filter on specific information.
 1. Choose **Add new tag**.
 2. Enter a **Key** name and associated **Value**.
 3. Choose **Add new tag** to add the new tag.

You can add multiple tags by choosing **Add new tag**, or you can remove a tag by choosing **Remove**.
 4. If you want to associate your tags with the probes for the monitor, keep **Add tags to probes created by monitor** selected. This adds the tags to the monitor probes, which can be helpful for tag-based authentication or metering.
6. To [the section called "Choosing source and destination"](#), choose **Next**.

Choosing source and destination

A monitor in Network Monitor uses an AWS source for the VPCs and associated subnets in the Regions where your network operates. A monitor destination is the combination of on-premises IP addresses, network protocols, ports, and network packet size.

The combination of a source and destination is a *probe*. You can have up to four probes per subnet, and up to a total of 24 probes per monitor.

Important

These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To choose a source and destination

1. Under **AWS network source**, choose one or more subnets to include in the monitor. To choose all subnets within a VPC, choose the VPC. Or, choose specific subnets within a VPC. The VPCs and subnets that you choose are the monitor source.
2. For **Destination 1**, enter the destination IP address of the on-premises network. IPv4 and IPv6 addresses are both supported.
3. Choose **Advanced settings**.
4. Choose the network **Protocol** for this customer-managed destination. The protocol can be either **ICMP** or **TCP**.
5. If you choose **TCP**, enter the following information:
 1. Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
 2. Enter the **Packet size**. This is the size, in bytes, of each packet that's sent on the probe between the source and destination. Packet size must be a number from **56** to **8500**.
6. Choose **Add destination** to add another on-premises destination to this monitor. Repeat these steps for each destination that you want to add.
7. Choose **Next** when finished to confirm the probes.

Confirming probes

The **Confirm probes** page shows all the possible combinations of the sources and destinations for the probe specifications that you've provided. For example, if you have six source subnets and four destination IP addresses, there is a total of 24 possible probe combinations.

Important

- These steps are intended to be completed in one session. You can't save in-process work to continue later.
- The **Confirm probes** page does not indicate whether a probe is valid. We recommend that you review this page carefully, and then delete any invalid probes. You might be charged for invalid probes that you don't remove.

To confirm monitor probes

1. Prerequisite: [the section called "Choosing source and destination"](#).
2. On the **Confirm probes** page, review the list of source and destination probe combinations.
3. Choose any probes that you want to remove from the monitor, and then choose **Remove**.

Note

You are not prompted to confirm deleting a probe. If you delete a probe and want to restore it, you must set it up again. You can add a probe to an existing monitor by following the steps in [the section called "Add a probe to a monitor"](#).

4. Choose **Next** to review details before you create the monitor.

Reviewing and creating

The final step is to review the details of the monitor and the probes for the monitor. You can change any information about the monitor at this point. When you have finished reviewing, change any information that isn't correct, and then create the monitor.

As soon as you create the monitor, Network Monitor begins tracking metrics and you'll start being charged for probes in the monitor.

Important

- This step is intended to be completed in one session. You can't save in-process work to continue later.
- If you choose to edit a section, you must step through the process to create a monitor from the point that you make the edits. Earlier monitor creation pages maintain the information that you already entered.

To review and create a monitor

1. On the **Review and create probes** page, choose **Edit** for any section where you want to make changes.
2. Make any changes in that section, and then choose **Next**.
3. When you're finished making edits, choose **Create monitor**.

The Network Monitor page displays the current state of monitor creation in the **Network monitors** section. While Network Monitor is creating the monitor, the **State** is **Pending**. When the **State** changes to **Active**, you can access the monitor dashboard to view CloudWatch metrics.

For information on working with the monitor dashboard, see [the section called "Network Monitor dashboards"](#).

Note

It can take several minutes for a newly-added monitor to begin collecting network metrics.

Creating a monitor in Amazon CloudWatch Network Monitor using the AWS Command Line Interface

Instead of using the AWS Management Console, you can use the AWS Command Line Interface or another API to create a monitor.

To create a monitor by using the using the command line

1. Create a monitor by using [create-monitor](#).
2. Create a monitor probe by using [create-probe](#).

Working with monitors and probes in Network Monitor

You can perform any of the following tasks with your monitors and probes by using the Amazon CloudWatch console or the AWS Command Line Interface.

Topics:

- [Edit a monitor](#)
- [Delete a monitor](#)
- [Activating or deactivating a probe](#)
- [Add a probe to a monitor](#)
- [Edit a probe](#)
- [Delete a probe](#)
- [Tag or untag resources](#)

Edit a monitor

You can edit information for a Network Monitor, including change the name, setting a new aggregation period, or adding or removing tags. Changing a monitor's information does not change any of its associated probes. You can edit a monitor by using either the Amazon CloudWatch console or the CLI.


Edit a monitor using the console

Use the CloudWatch console to edit a monitor.

To edit a monitor using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. In the **Network monitors** section, choose the monitor that you want to edit.
3. On the monitor dashboard page, choose **Edit**.

4. For the **Monitor name**, enter the new name for the monitor.
5. For the **Aggregation period**, choose how often you want to send metrics to CloudWatch. Valid periods are:
 - **30 seconds**
 - **60 seconds**

 **Note**

A shorter aggregation period provides faster detection of network issues; however, the aggregation period you choose can affect your billing structure. For more information about pricing, see the [Amazon CloudWatch pricing](#) page.

6. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information. You can also just change the **Value** of any current **Key**.

1. Choose **Add new tag**.
2. Enter a **Key** name and associated **Value**.
3. Choose **Add new tag** to add that new tag.

You can add multiple tags by choosing **Add new tag**, or you can remove any tag by choosing **Remove**.

4. If you want to associate your tags with the monitor, keep **Add tags to probes created by monitor** checked. This adds the tags to the monitor probes, which can be helpful if you're using tag-based authentication or metering.
7. Choose **Save changes**.

Edit a monitor using the CLI

Use the AWS Command Line Interface to edit a monitor in Network Monitor.

To edit a monitor by using the CLI

1. To get a list of your monitors if you don't know the name of the monitor, use [list-monitors](#). Note the name of the monitor that you want to edit.
2. Use [edit-monitor](#), and specify the name of a monitor from the [list-monitors](#) command.

Delete a monitor

Before you can delete a monitor in Network Monitor, you must deactivate or delete all probes associated with the monitor, regardless of the monitor **State**. After a monitor is deactivated or deleted, you are no longer be charged for probes in the monitor. Be aware that you can't restore a deleted monitor. You can delete a monitor using the Amazon CloudWatch console or the AWS Command Line Interface.

If you delete or deactivate a probe, CloudWatch retains the metrics it has published for 15 days.

Delete a monitor using the console

Use the CloudWatch console to delete a monitor in Network Monitor.

To delete a monitor using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. In the **Network monitors** section, choose the monitor that you want to delete.
3. Choose **Actions**, and then choose **Delete**.
4. If you have any active probes for the monitor, you're prompted to deactivate them. Choose **Deactivate probes**.

Note

You can't cancel or undo this action after you choose **Deactivate probes**. Deactivated probes, however, aren't removed from the monitor. If you like, you can reactivate them later. See [the section called "Activating or deactivating a probe"](#).

5. Enter **confirm** in the confirmation field, and then choose **Delete**.

Delete a monitor by using the CLI

Delete a monitor in Network Monitor by using the AWS Command Line Interface.

To delete a monitor by using the CLI

1. To delete a monitor, you need the monitor name. If you don't know the name, get a list of your monitors by running the [list-monitors](#) command. Note the name of the monitor that you want to delete.
2. Verify whether the monitor contains any active probes. Use [get-monitor](#) with the monitor name from the previous step. This returns a list of any probes associated with that monitor.
3. If the monitor contains active probes, you must first either set the probes to inactive or delete them.
 - To set a probe to inactive, use [update-probe](#), and set the state to INACTIVE.
 - To delete a probe, use [delete-probe](#).
4. After the probes are set to INACTIVE or deleted, you can delete the monitor by running the [delete-monitor](#) command. Inactive probes are not deleted when you delete the monitor.

Activating or deactivating a probe

You can activate or deactivate a probe in a monitor in Network Monitor. You might want to deactivate a probe in a scenario where you aren't currently using it but might want to use it again in the future. By deactivating a probe instead of deleting it, you won't need to spend time setting it up again. You are not billed for deactivated probes.

You can change the status of a probe by using either the Amazon CloudWatch console or the AWS Command Line Interface.

Set a probe to active or inactive by using the console

Use the CloudWatch console to set a probe to active or inactive.

To set a probe to active or inactive by using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then, under **Network Monitoring**, choose **Network Monitor**.
2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the probe that you want to activate or deactivate.
4. Choose **Actions**, and then choose **Activate** or **Deactivate**.

Note

When you reactivate a probe, you begin incurring billing charges on the probe again.

Set a probe to active or inactive using the CLI

You can set a probe to active or inactive or deactivate by using the CLI. You can only use this command for a single probe.

To set a probe to active or inactive using the CLI

1. Use [list-monitors](#) to get a list of your monitors if you don't know the name of the monitor. Note the name of the monitor that includes the probe that you want to change the status for.
2. Use [get-monitor](#) with the monitor name from the previous step. This returns a list of any probes associated with that monitor. Note the probe IDs of the probes that you want to change the status for.
3. Use [update-probe](#) to set the probe state to either ACTIVE or INACTIVE.

Add a probe to a monitor

You can add a probe to an existing monitor. Note that when you add probes to a monitor, your billing structure is updated to include the new probe.

Add a probe to a monitor by using the console

To add a probe to a monitor using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. In the **Network monitors** section, do one of the following:
 - Choose the **Name** link of the monitor that you want to add a probe to. Choose the **Monitor details** tab, and then in the **Probes** section, choose **Add probe**.
 - Choose the monitor check box, choose **Actions**, and then choose **Add probe**.
3. On the **Add probe** page, do the following:
 1. Under **AWS network source**, choose a subnet to add to the monitor.

Note

You can only add one probe at a time and up to four probes per monitor.

2. Enter the destination **IP address** of the on-premises network. Both IPv4 and IPv6 addresses are supported.
3. Choose **Advanced settings**.
4. Choose the network **Protocol** for the destination. It can be either **ICMP** or **TCP**.
5. If the **Protocol** is **TCP**, enter the following information. Otherwise, skip to the next step:
 - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
 - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.
4. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information.
 1. Choose **Add new tag**.
 2. Enter a **Key** name and associated **Value**.
 3. Choose **Add new tag** to add the new tag.

You can add multiple tags by choosing **Add new tag**, or you can remove any tag by choosing **Remove**.

5. Choose **Add probe**.

While the probe is being activated, the **State** shows **Pending**. It might take several minutes for the probe to become **Active**.

Add a probe to monitor by using the CLI

Add a probe to a monitor by using the AWS Command Line Interface. You can only use this command for add a single probe at a time.

To add a probe to monitor using the command line or API

1. Use [list-monitors](#) to get a list of your monitors if you don't know the name of the monitor. Note the name of the monitor that you want to add a probe to.

2. Use [create-probe](#) to add a probe to the monitor.

Edit a probe

You can change any information for an existing probe, regardless of whether that probe is active or inactive. You can edit a probe by using either the Amazon CloudWatch console or the AWS Command Line Interface.

Edit a probe by using the console

To edit a probe by using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.

Under **Name**, choose a monitor link to open the monitor dashboard.

2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the link for the probe that you want to edit.
4. On the probe details page, choose **Edit**.
5. On the **Edit probe** page, enter the new destination **IP address** for the probe. IPv4 and IPv6 addresses are both supported.
6. Choose **Advanced settings**.
7. Choose a network **Protocol**, **ICMP** or **TCP**.
8. If the **Protocol** is **TCP**, enter the following information:
 - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
 - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.
9. (Optional) Add, change, or remove Tags for the probe.
10. Choose **Save changes**.

Edit a probe by using the CLI

You can use the AWS Command Line Interface to edit a monitor probe. You can only edit a single probe at once with this command.

To edit a probe by using the CLI

1. Use [list-monitors](#) to get a list of your monitors, and note the name of the monitor with the probe that you want to edit.
2. Use [get-monitor](#) with the monitor name. This returns a list of probes associated with the monitor. Note the probe IDs of the probe or probes that you want to edit.
3. Use [update-probe](#) to change the probe's information.

Delete a probe

You can delete a probe rather than deactivating it if you know that you won't need it again later. You can't recover a deleted probe; instead, you must recreate it. Billing charges end for a probe when the probe is deleted. You can delete a probe by using the Amazon CloudWatch console or the AWS Command Line Interface.

Delete a probe by using the console

To delete a probe using the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. In the **Network monitors** section, under **Name**, choose a monitor link to open the monitor dashboard.
3. Choose the **Monitor details** tab.
4. Choose the monitor check box, choose **Actions**, and then choose **Delete**.
5. On the **Delete probe** dialog box, do the following:
6. Choose **Delete** to confirm that you want to delete the probe.

The **State** of the probe in the **Probes** section shows **Deleting**. After it's deleted, the probe is removed from the **Probes** section.

Delete a probe by using the CLI

Delete a probe by using the AWS Command Line Interface. You can only use this command to delete a single probe.

To delete a probe by using the CLI

1. Use [list-monitors](#) to get a list of your monitors if you don't know the name of the monitor. Note the name of the monitor with the probe you want to delete.
2. Use [get-monitor](#) with the monitor name from the previous step. This returns a list of any probes associated with that monitor. Note the probe ID of any probe you want to delete.
3. Use [delete-probe](#).

Tag or untag resources

You can use the CLI to work with resource tags.

To update monitor tags by using the CLI

- To list resource tags, use [list-tags-for-resources](#).
- To tag a resource, use [tag-resource](#).
- To untag a resource, use [untag-resource](#).

Network Monitor dashboards

You can use the Amazon CloudWatch Network Monitor dashboard to view AWS network health, and probe round-trip time and packet loss. You can view these metrics for both monitors and for individual probes.

Network Monitor dashboards

- [Monitor dashboard](#)
- [Probe dashboard](#)

Probe alarms

You can create Amazon CloudWatch alarms based on Amazon CloudWatch Network Monitor metrics, just as you can for other Amazon CloudWatch metrics. Any alarm that you create will appear in the probe's **Status** column of the **Monitor details** section of the Network Monitor dashboard when the alarm is triggered. The status will either be **OK** or **In Alarm**. If no status displays for a probe, then no alarm was created for that probe.

For example, you can create an alarm based on the Network Monitor metric `PacketLoss`, and configure it to send a notification when the metric is higher than a value that you choose. You configure alarms for Network Monitor metrics following the same guidelines as for other CloudWatch metrics.

The following metrics are available under `AWS/NetworkMonitor` when creating a CloudWatch alarm for Network Monitor.

- **HealthIndicator**
- **PacketLoss**
- **RTT (Round-trip time)**

For the steps to create a Network Monitor alarm, see [the section called "Create an alarm based on a static threshold"](#).

Setting a metrics time frame

Metrics and events on both dashboards use a default time of two hours, calculated from the current time. You can change the default to use one of the following presets:

- **1h** — one hour
- **2h** — two hours
- **1d** — one day
- **1w** — one week

You can also set a custom time frame. Choose **Custom**, choose an **Absolute** or **Relative** time, and then set the time frame to a time of your own choosing. Relative time supports only 15 days back from today's date, per CloudWatch defaults.

Additionally, you can choose the time displayed in the charts based on either the UTC timezone or a Local timezone.

Monitor dashboard

You can use the Amazon CloudWatch Network Monitor dashboard to view AWS network health, and probe round-trip time and packet loss. Network Monitor has dashboards for both monitors and probes.

To access a monitor dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.
2. In the **Network monitors** section, choose the **Name** link to open the monitor dashboard.

Overview

The Overview page displays the following information for your monitor:

- **AWS Network health** — AWS Network health displays the overall health of only the AWS network. The status will be either **Healthy** or **Degraded**. A **Healthy** status indicates Network Monitor did not observe any issue with the AWS network. A **Degraded** status indicates that Network Monitor observed an issue with the AWS network. The status bar in this section shows the status of the network over a default time of one hour. Hover over any point of the status bar to view additional details.
- **Probe traffic summary** — Displays the current state of the traffic between the source AWS subnets in the monitor and the destination IP addresses. **Probe traffic summary** displays the following:
 - **Probes in alarm** — This number indicates how many of your probes are in a degraded state. An alarm is triggered when a metric that you've set up as an alarm is triggered. For information on Network Monitor metric alarms, see [the section called "Probe alarms"](#).
 - **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address. This is represented as a percentage of the total packets sent.
 - **Round-trip time** — The time it takes, in milliseconds, for a packet from the source subnet to reach the destination IP address and then come back again.

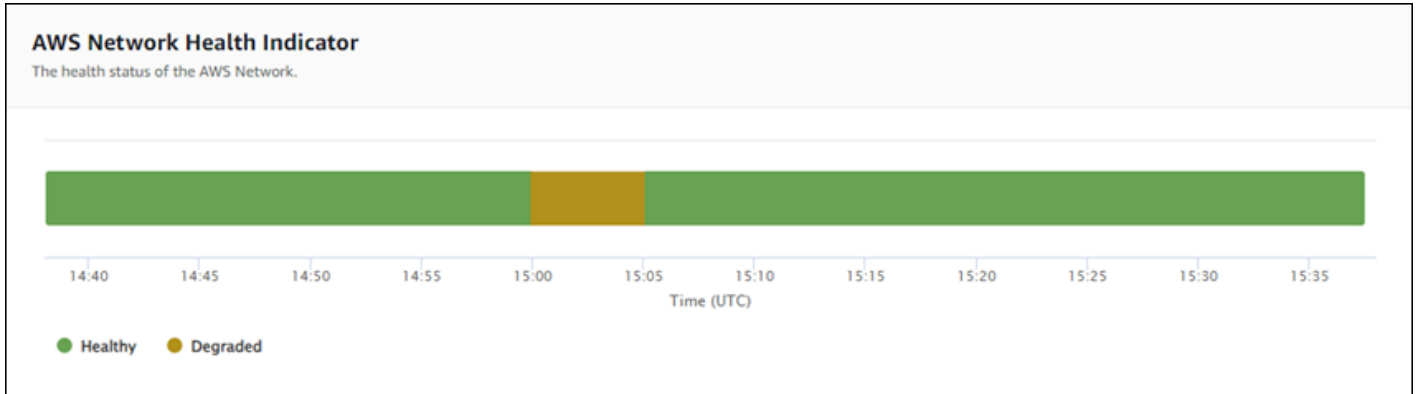
The data is represented by an interactive graph, allowing you to see the details.

By default, data is displayed for a two-hour time frame, calculated from the current date and time. However, you can change the range to fit your requirements. For more information, see [the section called "Setting a metrics time frame"](#).

Tracking metrics

The Network Monitor dashboard displays a graphical representation of your monitors and probes. The following graphs are available:

- **AWS Network Health Indicator** — This represents the health of the AWS network over a specified period. The status will either be **Healthy** or **Degraded**. In the following example, you'll see that from 15:00 UTC until 15:05 UTC, the AWS network was in a degraded state. After 15:05 the network returned to a healthy state. You can hover over any section of the graph to view additional details.



Note

The Network Health Indicator does not indicate the health of the probe but only the AWS network.

- **Packet loss** — This graph displays a unique line showing the percentage of packet loss for each probe in the monitor. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. Hovering over a probe in this chart displays the source subnet, the destination IP, and the percentage of packet loss. In the following example, a packet loss alarm was set up for a probe from a subnet to IP address 127.0.0.1. The alarm was triggered when the packet loss threshold was exceeded for the probe. Hovering over the graph shows the probe source and destination, and shows that there was a 30.97% packet loss for this probe on November 21 at 02:41:30.



- **Round-trip time** — This graph displays a line for each probe, showing the round-trip time for each probe. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. Hovering over a probe in this chart displays the source subnet, the destination IP address, and the round-trip time. The following example shows that on Tuesday, Nov 21 at 21:45:30, the round-trip time for a probe from a subnet to IP address 127.0.0.1 was 0.075 seconds.



Monitor details

The Monitor details page displays the details about your monitor, including probes. On this page you can manage tags or add a probe. This page is divided into the following three sections:

- **Monitor details** — This page provides details about your monitor. Information in this section can't be edited. You can, however, choose the **Role name** link to view details of the Network Monitor service-linked role.
- **Probes** — This section displays a list of all probes associated with the monitor. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe as well, including activating or deactivating it. For more information, see [the section called "Working with monitors and probes"](#).

The **Probes** section displays information about each probe set up for that monitor, including the probe **ID**, the **VPC ID**, the **Subnet ID**, **IP address**, **Protocol**, and whether the probe **State** is **Active** or **Inactive**. If you've set up an alarm for a probe, the current **Status** of that alarm displays. **OK** indicates that there are no metrics events have triggered any alarms; **In alarm** indicates that a metric you set up in CloudWatch triggered an alarm. If no status is displayed for a probe, then no CloudWatch alarm was set up. For information on the types of Network Monitor probe alarms you can create, see [the section called "Probe alarms"](#).

- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see [the section called "Edit a monitor"](#).

Probe dashboard

You can use the Amazon CloudWatch Network Monitor dashboard to view AWS network health, and information about specific round-trip time and packet loss for specific probes. There are two probe dashboards, **Overview** and **Probe details**.

You can create CloudWatch alarms to set packet loss and round-trip time metric thresholds. When a threshold is reached for a metric, a CloudWatch alarm notifies you. For information on creating probe alarms, see [the section called "Probe alarms"](#).

To access a probe dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>, and then under **Network Monitoring**, choose **Network Monitor**.

2. In the **Network monitors** section, choose the **Name** link to open the monitor dashboard.
3. Choose the **ID** link to view the dashboard for that probe.

Overview

The **Overview** page displays the following information for your probe:

- **AWS Network Health Indicator details** — This provides the overall health of only the AWS network. The status will be either **Healthy** or **Degraded**. A **Degraded** status indicates that there's an issue with the AWS network, and does not indicate whether there's an issue with your probe.
- **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address for this probe.
- **Round-trip time** — The time it took, in milliseconds, for a packet from the source subnet to reach the destination IP address and then come back again.

Probe details

The Probe details page displays the details about a probe. On this page you can edit the probe. For more information, see [the section called "Working with monitors and probes"](#).

- **Probe details** — This page provides general information about the probe. Information in this section can't be edited.
- **Probe source and destination** — This section displays details about the probe. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe as well, including activating or deactivating it.
- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see [the section called "Edit a probe"](#).

Network Monitor quotas

The following are the Network Monitor quotas:

Quota	Default	Adjustable
Maximum number of monitors per account per AWS Region	100	Yes
Maximum number of probes per monitor	24	Yes
Maximum number of probes per subnet per monitor	4	Yes

Data security and data protection in Network Monitor

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to Amazon CloudWatch Network Monitor, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using CloudWatch Network Monitor. The following topics show you how to configure CloudWatch Network Monitor to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudWatch Network Monitor resources.

Topics

- [Data protection in Amazon CloudWatch Network Monitor](#)

- [Infrastructure Security in Amazon CloudWatch Network Monitor](#)

Data protection in Amazon CloudWatch Network Monitor

The AWS [shared responsibility model](#) applies to data protection in Amazon CloudWatch Network Monitor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with CloudWatch Network Monitor or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Infrastructure Security in Amazon CloudWatch Network Monitor

As a managed service, Amazon CloudWatch Network Monitor is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access CloudWatch Network Monitor through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Identity and Access management for Amazon CloudWatch Network Monitor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use CloudWatch Network Monitor resources. IAM is an AWS service that you can use with no additional charge. You can use features of IAM to allow other users, services, and applications to use your AWS resources fully or in a limited way, without sharing your security credentials.

By default, IAM users don't have permission to create, view, or modify AWS resources. To allow an IAM user to access resources, such as a global network, and perform tasks, you must:

- Create an IAM policy that grants the user permission to use the specific resources and API actions they need
- Attach the policy to the IAM user or to the group to which the user belongs

When you attach a policy to a user or group of users, it allows or denies the user permissions to perform the specified tasks on the specified resources.

Condition keys

The `Condition` element (or `Condition` block) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. For more information, see [IAM JSON policy elements: Condition operators](#) in the *AWS Identity and Access Management User Guide*.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name.

You can attach tags to CloudWatch Network Monitor resources or pass tags in a request to Cloud WAN. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. See [IAM JSON policy elements: Condition](#) in the *AWS Identity and Access Management User Guide* for more information.

To see all AWS global condition keys, see [AWS global condition context keys](#) in the *AWS Identity and Access Management User Guide*.

Tag core network resources

A tag is a metadata label that either you or AWS assigns to an AWS resource. Each tag consists of a key and a value. For tags that you assign, you define the key and the value. For example, you might define the key as `purpose` and the value as `test` for one resource. Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.
- Control access to your AWS resources. For more information, see [Controlling access to AWS resources using tags](#) in the *AWS Identity and Access Management User Guide*.

How Amazon CloudWatch Network Monitor works with IAM

Before you use IAM to manage access to CloudWatch Network Monitor, learn what IAM features are available to use with CloudWatch Network Monitor.

IAM features you can use with Amazon CloudWatch Network Monitor

IAM feature	CloudWatch Network Monitor support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how CloudWatch Network Monitor and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for Amazon CloudWatch Network Monitor

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for CloudWatch Network Monitor

To view examples of CloudWatch Network Monitor identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

Resource-based policies within CloudWatch Network Monitor

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for CloudWatch Network Monitor

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CloudWatch Network Monitor actions, see [Actions defined by Amazon CloudWatch Network Monitor](#) in the *Service Authorization Reference*.

Policy actions in CloudWatch Network Monitor use the following prefix before the action:

```
networkmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
  "networkmonitor:action1",
  "networkmonitor:action2"
]
```

To view examples of CloudWatch Network Monitor identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

Policy resources for CloudWatch Network Monitor

Supports policy resources

Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*" 
```

To see a list of CloudWatch Network Monitor resource types and their ARNs, see [Resources defined by Amazon CloudWatch Network Monitor](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon CloudWatch Network Monitor](#).

Policy condition keys for CloudWatch Network Monitor

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of CloudWatch Network Monitor condition keys, see [Condition keys for Amazon CloudWatch Network Monitor](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon CloudWatch Network Monitor](#).

ACLs in CloudWatch Network Monitor

Supports ACLs	No
---------------	----

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CloudWatch Network Monitor

Supports ABAC (tags in policies)	Partial
----------------------------------	---------

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using temporary credentials with CloudWatch Network Monitor

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for CloudWatch Network Monitor

Supports forward access sessions (FAS)	Yes
--	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for CloudWatch Network Monitor

Supports service roles	No
------------------------	----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

⚠ Warning

Changing the permissions for a service role might break CloudWatch Network Monitor functionality. Edit service roles only when CloudWatch Network Monitor provides guidance to do so.

Using a service-linked role for CloudWatch Network Monitor

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for CloudWatch Network Monitor

By default, users and roles don't have permission to create or modify CloudWatch Network Monitor resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Creating IAM policies](#) in the *IAM User Guide*.

For details about actions and resource types defined by CloudWatch Network Monitor, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon CloudWatch Network Monitor](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)

- [Using the CloudWatch Network Monitor console](#)
- [Allow users to view their own permissions](#)
- [Troubleshooting CloudWatch Network Monitor identity and access](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CloudWatch Network Monitor resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the CloudWatch Network Monitor console

To access the Amazon CloudWatch Network Monitor console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CloudWatch Network Monitor resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the CloudWatch Network Monitor console, also attach the CloudWatch Network Monitor *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
```

```
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
    ],
    "Resource": "*"
}
]
```

Troubleshooting CloudWatch Network Monitor identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CloudWatch Network Monitor and IAM.

Topics

- [I am not authorized to perform an action in CloudWatch Network Monitor](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my CloudWatch Network Monitor resources](#)

I am not authorized to perform an action in CloudWatch Network Monitor

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional `my-example-widget` resource but doesn't have the fictional `networkmonitor:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
networkmonitor:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *my-example-widget* resource by using the `networkmonitor:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CloudWatch Network Monitor.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CloudWatch Network Monitor. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my CloudWatch Network Monitor resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CloudWatch Network Monitor supports these features, see [How Amazon CloudWatch works with IAM](#).

- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

AWS managed policies for CloudWatch Network Monitor

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

AWS managed policy: `CloudWatchNetworkMonitorServiceRolePolicy`

The `CloudWatchNetworkMonitorServiceRolePolicy` is attached to a service-linked role that allows the service to perform actions on your behalf and access resources associated with CloudWatch Network Monitor. You cannot attach this policy to your IAM identities. For more information, see [the section called "Service-linked roles"](#).

CloudWatch Network Monitoring updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch Network Monitoring since this service began tracking these changes in November 2023.

Change	Description	Date
CloudWatchNetworkMonitorServiceRolePolicy : New policy.	New policy added to CloudWatch Network Monitor.	November 27, 2023
the section called "AWSServiceRoleForNetworkMonitor" . New role.	New role added to CloudWatch Network Monitor.	November 27, 2023

IAM permissions for CloudWatch Network Monitor

To use Amazon CloudWatch Network Monitor users must have the correct permissions.

For more information about security in Amazon CloudWatch, see [Identity and access management for Amazon CloudWatch](#).

Permissions required to view a monitor

To view a monitor for Amazon CloudWatch Network Monitor in the AWS Management Console, you must be signed in as a user or role that has the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "networkmonitor:Get*",
        "networkmonitor:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

```

    }
  ]
}

```

Permissions required to create a monitor

To create a monitor in Amazon CloudWatch Network Monitor, users must have permission to create a service-linked role that is associated with Network Monitor. To learn more about the service-linked role, see [Using a service-linked role for CloudWatch Network Monitor](#).

To create a monitor for Amazon CloudWatch Network Monitor in the AWS Management Console, you must be signed in as a user or role that has the permissions included in the following policy.

Note

If you create an identity-based permissions policy that is more restrictive, users with that policy won't be able to create a monitor.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "networkmonitor:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "networkmonitor.amazonaws.com"
        }
      }
    }
  ],
  {

```

```

        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:GetRole",
            "iam:PutRolePolicy"
        ],
        "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor"
    },
    {
        "Action": [
            "ec2:CreateSecurityGroup",
            "ec2:CreateNetworkInterface",
            "ec2:CreateTags"
        ],
        "Effect": "Allow",
        "Resource": "*"
    }
]
}

```

Using a service-linked role for CloudWatch Network Monitor

Amazon CloudWatch Network Monitor uses the following service-linked role for the permissions that it requires to call other AWS services on your behalf:

- [AWSServiceRoleForNetworkMonitor](#)

AWSServiceRoleForNetworkMonitor

CloudWatch Network Monitoring uses the service-linked role named `AWSServiceRoleForNetworkMonitor` to update and manage CloudWatch network monitors.

The `AWSServiceRoleForNetworkMonitor` service-linked role trusts the following service to assume the role:

- `networkmonitor.amazonaws.com`

The `CloudWatchNetworkMonitorServiceRolePolicy` is attached to the service linked role and grants access for the service to access VPC and EC2 resources in your account, as well as manage the network monitors that were created.

Permission groupings

The policy is grouped into the following sets of permissions:

- **cloudwatch** - This allows the service principal to publish network monitoring metrics to CloudWatch resources.
- **ec2** - This allows the service principal to describe VPCs and subnets in your account to create or update monitors and probes. This also allows the service principal to create, modify, and delete security groups, network interfaces, and their associated permissions to configure the monitor or probe to send monitoring traffic to your endpoints.

For more information about the policy, see [the section called “AWS managed policies”](#).

The following shows the CloudWatchNetworkMonitorServiceRolePolicy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublishCw",
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": "AWS/NetworkMonitor"
        }
      }
    },
    {
      "Sid": "DescribeAny",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeVpcs",
        "ec2:DescribeNetworkInterfacePermissions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    }
  ],
}
```

```

{
  "Sid": "DeleteModifyEc2Resources",
  "Effect": "Allow",
  "Action": [
    "ec2:AuthorizeSecurityGroupEgress",
    "ec2:CreateNetworkInterfacePermission",
    "ec2>DeleteNetworkInterfacePermission",
    "ec2:RevokeSecurityGroupEgress",
    "ec2:ModifyNetworkInterfaceAttribute",
    "ec2>DeleteNetworkInterface",
    "ec2>DeleteSecurityGroup"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:network-interface/*",
    "arn:aws:ec2:*:*:security-group/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/ManagedByCloudWatchNetworkMonitor": "true"
    }
  }
}
]
}

```

Create the service-linked role

`AWSServiceRoleForNetworkMonitor`

You don't need to manually create the or `AWSServiceRoleForNetworkMonitor` role.

- CloudWatch Network Monitor creates the `AWSServiceRoleForNetworkMonitor` role when you create your first network monitor. This role will apply to any subsequent monitors you create.

To create a service-linked role on your behalf, you must have the required permissions. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Edit the service-linked role

You can edit the `AWSServiceRoleForNetworkMonitor` descriptions using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Delete the service-linked role

If you no longer need to use CloudWatch Network Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your network monitor. For information about deleting your network monitor, see [Delete a network monitor](#).

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` CloudWatch Network Monitor will create the role again when you create a new monitor.

Supported Regions for the CloudWatch Network Monitor service-linked role

CloudWatch Network Monitor supports the service-linked role in all of AWS Regions where the service is available. For more information, see [AWS endpoints](#) in the *AWS General Reference*.

Delete the service-linked role

If you no longer need to use CloudWatch Network Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your network monitor. For information about deleting your network monitor, see [Delete a network monitor](#).

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` CloudWatch Network Monitor will create the role again when you create a new monitor.

Pricing for Amazon CloudWatch Network Monitor

With Amazon CloudWatch Network Monitor, there are no upfront costs or long-term commitments. Pricing for Network Monitor has the following two components:

- An hourly per monitored resource fee
- CloudWatch metrics fees

When you create a network monitor, you associate resources with it to be monitored. For Network Monitor, these resources are subnets in Amazon Virtual Private Cloud (VPC). Each monitored resource allows you to create up to four probes from each subnet in your VPCs to four destinations. To help control your bill, you can adjust your subnet coverage and on-premises IP address destination coverage by reducing the number of resources that you monitor.

For more information about pricing, see the [Amazon CloudWatch pricing](#) page.

Infrastructure monitoring

The topics in this section explain CloudWatch features that can help you gain operational visibility into your AWS resources.

Topics

- [Container Insights](#)
- [Lambda Insights](#)
- [Use Contributor Insights to analyze high-cardinality data](#)
- [Amazon CloudWatch Application Insights](#)
- [Using the resource health view in the CloudWatch console](#)

Container Insights

Use CloudWatch Container Insights to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Kubernetes platforms on Amazon EC2. Container Insights supports collecting metrics from clusters deployed on AWS Fargate for both Amazon ECS and Amazon EKS.

CloudWatch automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects.

Container Insights collects data as *performance log events* using [embedded metric format](#). These performance log events are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates aggregated metrics at the cluster, node, pod, task, and service level as CloudWatch metrics. The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and are also viewable in the **Metrics** section of the CloudWatch console. Metrics are not visible until the container tasks have been running for some time.

When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

To help you manage your Container Insights costs, CloudWatch does not automatically create all possible metrics from the log data. However, you can view additional metrics and additional levels of granularity by using CloudWatch Logs Insights to analyze the raw performance log events.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

In Amazon EKS and Kubernetes, Container Insights uses a containerized version of the CloudWatch agent to discover all of the running containers in a cluster. It then collects performance data at every layer of the performance stack.

Container Insights supports encryption with the AWS KMS key for the logs and metrics that it collects. To enable this encryption, you must manually enable AWS KMS encryption for the log group that receives Container Insights data. This causes Container Insights to encrypt this data using the provided KMS key. Only symmetric keys are supported. Do not use asymmetric KMS keys to encrypt your log groups.

For more information, see [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#).

Container Insights with enhanced observability for Amazon EKS

On November 6, 2023, a new version of Container Insights was released. This version supports enhanced observability for Amazon EKS clusters running on Amazon EC2 and can collect more detailed metrics from these clusters. After installation, it automatically collects detailed infrastructure telemetry and container logs for your Amazon EKS clusters. You can then use curated, immediately usable dashboards to drill down into application and infrastructure telemetry.

Container Insights with enhanced observability for Amazon EKS collects granular health, performance, and status metrics up to the container level, and also control plane metrics. For more information about the additional metrics and dimensions collected, see [Amazon EKS and Kubernetes Container Insights metrics](#).

If you installed Container Insights by using the CloudWatch agent on an Amazon EKS cluster on Amazon EC2 after November 6, 2023, you have Container Insights with enhanced observability for Amazon EKS. Otherwise, you can upgrade an Amazon EKS cluster to this new version by following the instructions in [Upgrading to Container Insights with enhanced observability for Amazon EKS](#).

Container Insights supports CloudWatch cross-account observability. You use a single monitoring account to monitor and troubleshoot your applications that span multiple AWS accounts within a single Region. For more information, see [CloudWatch cross-account observability](#).

Container Insights with enhanced observability for Amazon EKS also supports Windows worker nodes.

Container Insights with enhanced observability for Amazon EKS is not supported on Fargate.

Note

You can find whether you have clusters that can be upgraded to Container Insights with enhanced observability for Amazon EKS by navigating to the Container Insights console. To do so, choose **Insights, Container Insights** in the navigation pane of the CloudWatch console. In the Container Insights console, a banner informs you if you have any Amazon EKS clusters that can be upgraded, and links to the upgrade page.

Supported platforms

Container Insights is available for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2 instances.

- For Amazon ECS, Container Insights collects metrics at the cluster, task, and service levels on both Linux and Windows Server instances. It can collect metrics at the instance level only on Linux instances.

For Amazon ECS, network metrics are available only for containers in bridge network mode and awsvpc network mode. They are not available for containers in host network mode.

- For Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2 instances, Container Insights is supported only on Linux instances.

CloudWatch agent container image

Amazon provides a CloudWatch agent container image on Amazon Elastic Container Registry. For more information, see [cloudwatch-agent](#) on Amazon ECR.

Supported Regions

Container Insights for Amazon ECS is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- Canada West (Calgary)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- Middle East (Bahrain)
- Middle East (UAE)

- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- China (Beijing)
- China (Ningxia)

Supported Regions for Amazon EKS and Kubernetes

Container Insights for Amazon EKS and Kubernetes is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)

- AWS GovCloud (US-West)

Setting up Container Insights

The Container Insights setup process is different for Amazon ECS and Amazon EKS and Kubernetes.

Topics

- [Setting up Container Insights on Amazon ECS](#)
- [Setting up Container Insights on Amazon EKS and Kubernetes](#)

Setting up Container Insights on Amazon ECS

You can use one or both of the following options to enable Container Insights on Amazon ECS clusters:

- Use the AWS Management Console or the AWS CLI to start collecting cluster-level, task-level, and service-level metrics.
- Deploy the CloudWatch agent as a daemon service to start collecting of instance-level metrics on clusters that are hosted on Amazon EC2 instances.

Topics

- [Setting up Container Insights on Amazon ECS for cluster- and service-level metrics](#)
- [Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry](#)
- [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS](#)
- [Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters](#)
- [Set up FireLens to send logs to CloudWatch Logs](#)

Setting up Container Insights on Amazon ECS for cluster- and service-level metrics

You can enable Container Insights on new and existing Amazon ECS clusters. Container Insights collects metrics at the cluster, task, and service levels. You can enable Container Insights using either the Amazon ECS console or the AWS CLI.

If you're using Amazon ECS on an Amazon EC2 instance, and you want to collect network and storage metrics from Container Insights, launch that instance using an AMI that includes Amazon

ECS agent version 1.29. For information about updating your agent version, see [Updating the Amazon ECS Container Agent](#)

You can use the AWS CLI to set account-level permission to enable Container Insights for any new Amazon ECS clusters created in your account. To do so, enter the following command.

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

Note

If the customer managed AWS KMS key you use for your Amazon ECS Container Insights metrics is not already configured to work with CloudWatch, you must update the key policy to allow for encrypted logs in CloudWatch Logs. You must also associate your own AWS KMS key with the log group under `/aws/ecs/containerinsights/ClusterName/performance`. For more information, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](#).

Setting up Container Insights on existing Amazon ECS clusters

To enable Container Insights on an existing Amazon ECS cluster, enter the following command. You must be running version 1.16.200 or later of the AWS CLI for the following command to work.

```
aws ecs update-cluster-settings --cluster myCICluster --settings  
name=containerInsights,value=enabled
```

Setting up Container Insights on new Amazon ECS clusters

There are two ways to enable Container Insights on new Amazon ECS clusters. You can configure Amazon ECS so that all new clusters are enabled for Container Insights by default. Otherwise, you can enable a new cluster when you create it.

Using the AWS Management Console

You can turn on Container Insights on all new clusters by default, or on an individual cluster as you create it.

To turn on Container Insights on all new clusters by default

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.

2. In the navigation page, choose **Account Settings**.
3. Choose **Update**.
4. To use CloudWatch Container Insights by default for clusters, under **CloudWatch Container Insights**, select or clear **CloudWatch Container Insights**.
5. Choose **Save changes**.

If you haven't used the preceding procedure to enable Container Insights on all new clusters by default, use the following steps to create a cluster with Container Insights enabled.

To create a cluster with Container Insights turned on

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create cluster**.
4. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.

5. To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.

You can now create task definitions, run tasks, and launch services in the cluster. For more information, see the following:

- [Creating a task definition](#)
- [Running tasks](#)
- [Creating a service](#)

Setting up Container Insights on new Amazon ECS clusters using the AWS CLI

To enable Container Insights on all new clusters by default, enter the following command.

```
aws ecs put-account-setting --name "containerInsights" --value "enabled"
```

If you didn't use the preceding command to enable Container Insights on all new clusters by default, enter the following command to create a new cluster with Container Insights enabled. You must be running version 1.16.200 or later of the AWS CLI for the following command to work.

```
aws ecs create-cluster --cluster-name myCICluster --settings  
"name=containerInsights,value=enabled"
```

Disabling Container Insights on Amazon ECS clusters

To disable Container Insights on an existing Amazon ECS cluster, enter the following command.

```
aws ecs update-cluster-settings --cluster myCICluster --settings  
name=containerInsights,value=disabled
```

Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry

Use this section if you want to use AWS Distro for OpenTelemetry to set up CloudWatch Container Insights on an Amazon ECS cluster. For more information about AWS Distro for Open Telemetry, see [AWS Distro for OpenTelemetry](#).

These steps assume that you already have a cluster running Amazon ECS. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service](#).

Step 1: Create a task role

The first step is creating a task role in the cluster that the AWS OpenTelemetry Collector will use.

To create a task role for AWS Distro for OpenTelemetry

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab and copy in the following policy:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "logs:PutLogEvents",  
        "logs:CreateLogGroup",  
        "logs:CreateLogStream",
```



```

        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "ssm:GetParameters"
    ],
    "Resource": "*"
}
]
}

```

4. Choose **Review policy**.
5. For name, enter **AWSDistroOpenTelemetryPolicy**, and then choose **Create policy**.
6. In the left navigation pane, choose **Roles** and then choose **Create role**.
7. In the list of services, choose **Elastic Container Service**.
8. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
9. In the list of policies, search for **AWSDistroOpenTelemetryPolicy**.
10. Select the check box next to **AWSDistroOpenTelemetryPolicy**.
11. Choose **Next: Tags** and then choose **Next: Review**.
12. For **Role name** enter **AWSOpenTelemetryTaskRole** and then choose **Create role**.

Step 2: Create a task execution role

The next step is creating a task execution role for the AWS OpenTelemetry Collector.

To create a task execution role for AWS Distro for OpenTelemetry

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles** and then choose **Create role**.
3. In the list of services, choose **Elastic Container Service**.
4. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
5. In the list of policies, search for **AmazonECSTaskExecutionRolePolicy** and then select the check box next to **AmazonECSTaskExecutionRolePolicy**.
6. In the list of policies, search for **CloudWatchLogsFullAccess** and then select the check box next to **CloudWatchLogsFullAccess**.
7. In the list of policies, search for **AmazonSSMReadOnlyAccess** and then select the check box next to **AmazonSSMReadOnlyAccess**.

8. Choose **Next: Tags** and then choose **Next: Review**.
9. For **Role name** enter **AWSOpenTelemetryTaskExecutionRole** and then choose **Create role**.

Step 3: Create a task definition

The next step is creating a task definition.

To create a task definition for AWS Distro for OpenTelemetry

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition, Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. Configure your containers, and then choose **Next**.
6. Under **Metrics and logging**, select **Use metric collection**.
7. Choose **Next**.
8. Choose **Create**.

For more information about using the AWS OpenTelemetry collector with Amazon ECS, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service](#).

Step 4: Run the task

The final step is running the task that you've created.

To run the task for AWS Distro for OpenTelemetry

1. Open the console at <https://console.aws.amazon.com/ecs/v2>.
2. In the left navigation pane, choose **Task Definitions** and then select the task that you just created.
3. Choose **Actions, Deploy, Run task**.
4. Choose **Deploy, Run task**.
5. In the **Compute options** section, from **Existing cluster**, choose the cluster.
6. Choose **Create**.
7. Next, you can check for the new metrics in the CloudWatch console.

8. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
9. In the left navigation pane, choose **Metrics**.

You should see a **ECS/ContainerInsights** namespace. Choose that namespace and you should see eight metrics.

Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS

To deploy the CloudWatch agent to collect instance-level metrics from Amazon ECS clusters that are hosted on EC2 instance, use a quick start setup with a default configuration, or install the agent manually to be able to customize it.

Both methods require that you already have at least one Amazon ECS cluster deployed with an EC2 launch type and that the CloudWatch agent container has access to the Amazon EC2 Instance Metadata Service (IMDS). For more information about IMDS, see [Instance metadata and user data](#).

These methods also assume that you have the AWS CLI installed. Additionally, to run the commands in the following procedures, you must be logged on to an account or role that has the **IAMFullAccess** and **AmazonECS_FullAccess** policies.

Topics

- [Quick setup using AWS CloudFormation](#)
- [Manual and custom setup](#)

Quick setup using AWS CloudFormation

To use the quick setup, enter the following command to use AWS CloudFormation to install the agent. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

This command creates the IAM roles **CWAgentECSTaskRole** and **CWAgentECSExecutionRole**. If these roles already exist in your account, use `ParameterKey=CreateIAMRoles,ParameterValue=False` instead of `ParameterKey=CreateIAMRoles,ParameterValue=True` when you enter the command. Otherwise, the command will fail.

```
ClusterName=cluster-name  
Region=cluster-region
```

```
curl -0 https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
  --template-body file://cwagent-ecs-instance-metric-cfn.json \
  --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${Region}
```

(Alternative) Using your own IAM roles

If you want to use your own custom ECS task role and ECS task execution role instead of the **CWAgentECSTaskRole** and **CWAgentECSExecutionRole** roles, first make sure that the role to be used as the ECS task role has **CloudWatchAgentServerPolicy** attached. Also, make sure that the role to be used as the ECS task execution role has both the **CloudWatchAgentServerPolicy** and **AmazonECSTaskExecutionRolePolicy** policies attached. Then enter the following command. In the command, replace *task-role-arn* with the ARN of your custom ECS task role, and replace *execution-role-arn* with the ARN of your custom ECS task execution role.

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
curl -0 https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
  --template-body file://cwagent-ecs-instance-metric-cfn.json \
  --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
                 ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
                 ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${Region}
```

Troubleshooting the quick setup

To check the status of the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
```

```
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --  
region $Region
```

If you see the StackStatus is other than CREATE_COMPLETE or CREATE_IN_PROGRESS, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name  
Region=cluster-region  
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region  
--region $Region
```

To check the status of the cwagent daemon service, enter the following command. In the output, you should see that the runningCount is equal to the desiredCount in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name  
Region=cluster-region  
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --  
region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/ecs/ecs-cwagent-daemon-service** log group.

Deleting the AWS CloudFormation stack for the CloudWatch agent

If you need to delete the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name  
Region=cluster-region  
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --  
region ${Region}
```

Manual and custom setup

Follow the steps in this section to manually deploy the CloudWatch agent to collect instance-level metrics from your Amazon ECS clusters that are hosted on EC2 instances.

Necessary IAM roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see [IAM roles for Tasks](#) and [Amazon ECS Task Execution Role](#).

- An *ECS task role*, which is used by the CloudWatch agent to publish metrics. If this role already exists, you must make sure it has the `CloudWatchAgentServerPolicy` policy attached.
- An *ECS task execution role*, which is used by Amazon ECS agent to launch the CloudWatch agent. If this role already exists, you must make sure it has the `AmazonECSTaskExecutionRolePolicy` and `CloudWatchAgentServerPolicy` policies attached.

If you do not already have these roles, you can use the following commands to create them and attach the necessary policies. This first command creates the ECS task role.

```
aws iam create-role --role-name CWAgentECSTaskRole \  
  --assume-role-policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com\"}, \"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as `"TaskRoleArn"`. You'll need to use it later when you create the task definition. Then enter the following command to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/  
CloudWatchAgentServerPolicy \  
  --role-name CWAgentECSTaskRole
```

This next command creates the ECS task execution role.

```
aws iam create-role --role-name CWAgentECSExecutionRole \  
  --assume-role-policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"ecs-tasks.amazonaws.com\"}, \"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as `"ExecutionRoleArn"`. You'll need to use it later when you create the task definition. Then enter the following commands to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/  
CloudWatchAgentServerPolicy \  
  --role-name CWAgentECSExecutionRole
```

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
  --role-name CWAgentECSExecutionRole
```

Create the task definition and launch the daemon service

Create a task definition and use it to launch the CloudWatch agent as a daemon service. To create the task definition, enter the following command. In the first lines, replace the placeholders with the actual values for your deployment. *logs-region* is the Region where CloudWatch Logs is located, and *cluster-region* is the Region where your cluster is located. *task-role-arn* is the Arn of the ECS task role that you are using, and *execution-role-arn* is the Arn of the ECS task execution role.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cwagent-ecs-instance-metric.json \
  | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
  | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

Then run the following command to launch the daemon service. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

Important

Remove all capacity provider strategies before you run this command. Otherwise, the command won't work.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
  --cluster ${ClusterName} \
  --service-name cwagent-daemon-service \
  --task-definition ecs-cwagent-daemon-service \
  --scheduling-strategy DAEMON \
```

```
--region ${Region}
```

If you see this error message, An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent, you have already created a daemon service named cwagent-daemon-service. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name  
Region=cluster-region  
aws ecs delete-service \  
    --cluster ${ClusterName} \  
    --service cwagent-daemon-service \  
    --region ${Region} \  
    --force
```

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the CloudWatch agent in your Amazon ECS clusters that are hosted on EC2 instances. These options are as follows:

- `metrics_collection_interval` – How often in seconds that the CloudWatch agent collects metrics. The default is 60. The range is 1–172,000.
- `endpoint_override` – (Optional) Specifies a different endpoint to send logs to. You might want to do this if you're publishing from a cluster in a VPC and you want the logs data to go to a VPC endpoint.

The value of `endpoint_override` must be a string that is a URL.

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5 seconds.
- `region` – By default, the agent publishes metrics to the same Region where the Amazon ECS container instance is located. To override this, you can specify a different Region here. For example, `"region" : "us-east-1"`

The following is an example of a customized configuration:

```
{
```



```
"agent": {
  "region": "us-east-1"
},
"logs": {
  "metrics_collected": {
    "ecs": {
      "metrics_collection_interval": 30
    }
  },
  "force_flush_interval": 5
}
}
```

To customize your CloudWatch agent configuration in your Amazon ECS containers

1. Make sure that the **AmazonSSMReadOnlyAccess** policy is attached to your Amazon ECS Task Execution role. You can enter the following command to do so. This example assumes that your Amazon ECS Task Execution role is `CWAgentECSExecutionRole`. If you are using a different role, substitute that role name in the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonSSMReadOnlyAccess \
    --role-name CWAgentECSExecutionRole
```

2. Create the customized configuration file similar to the preceding example. Name this file `/tmp/ecs-cwagent-daemon-config.json`.
3. Run the following command to put this configuration into the Parameter Store. Replace *cluster-region* with the Region of your Amazon ECS cluster. To run this command, you must be logged on to a user or role that has the **AmazonSSMFullAccess** policy.

```
Region=cluster-region
aws ssm put-parameter \
    --name "ecs-cwagent-daemon-service" \
    --type "String" \
    --value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
    --region $Region
```

4. Download the task definition file to a local file, such as `/tmp/cwagent-ecs-instance-metric.json`

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-instance-metric.json
```

5. Modify the task definition file. Remove the following section:

```
"environment": [
  {
    "name": "USE_DEFAULT_CONFIG",
    "value": "True"
  }
],
```

Replace that section with the following:

```
"secrets": [
  {
    "name": "CW_CONFIG_CONTENT",
    "valueFrom": "ecs-cwagent-daemon-service"
  }
],
```

6. Restart the agent as a daemon service by following these steps:

- a. Run the following command.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
  | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
  | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-
json
```

- b. Run the following command to launch the daemon service. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

```
ClusterName=cluster-name
```

```
Region=cluster-region
aws ecs create-service \
  --cluster ${ClusterName} \
  --service-name cwagent-daemon-service \
  --task-definition ecs-cwagent-daemon-service \
  --scheduling-strategy DAEMON \
  --region ${Region}
```

If you see this error message, An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent, you have already created a daemon service named cwagent-daemon-service. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
  --cluster ${ClusterName} \
  --service cwagent-daemon-service \
  --region ${Region} \
  --force
```

Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters

Use the steps in this section to use AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on an Amazon ECS cluster. For more information about the AWS Distro for OpenTelemetry, see [AWS Distro for OpenTelemetry](#).

These steps assume that you already have a cluster running Amazon ECS. This cluster must be deployed with the EC2 launch type. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service for ECS EC2 instance level metrics](#).

Topics

- [Quick setup using AWS CloudFormation](#)
- [Manual and custom setup](#)

Quick setup using AWS CloudFormation

Download the AWS CloudFormation template file for installing the AWS Distro for OpenTelemetry collector for Amazon ECS on EC2. Run the following curl command.

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

After you download the template file, open it and replace *PATH_TO_CloudFormation_TEMPLATE* with the path where you saved the template file. Then export the following parameters and run the AWS CloudFormation command, as shown in the following command.

- **Cluster_Name**– The Amazon ECS cluster name
- **AWS_Region**– The Region where the data will be sent
- **PATH_TO_CloudFormation_TEMPLATE**– The path where you saved the AWS CloudFormation template file.
- **command**– To enable the AWS Distro for OpenTelemetry collector to collect the instance-level metrics for Amazon ECS on Amazon EC2, you must specify `--config=/etc/ecs/otel-instance-metrics-config.yaml` for this parameter.

```
ClusterName=Cluster_Name  
Region=AWS_Region  
command=--config=/etc/ecs/otel-instance-metrics-config.yaml  
aws cloudformation create-stack --stack-name AOCECS-{ClusterName}-{Region} \  
--template-body file://PATH_TO_CloudFormation_TEMPLATE \  
--parameters ParameterKey=ClusterName,ParameterValue={ClusterName} \  
ParameterKey=CreateIAMRoles,ParameterValue=True \  
ParameterKey=command,ParameterValue={command} \  
--capabilities CAPABILITY_NAMED_IAM \  
--region {Region}
```

After running this command, use the Amazon ECS console to see if the task is running.

Troubleshooting the quick setup

To check the status of the AWS CloudFormation stack, enter the following command.

```
ClusterName=cluster-name  
Region=cluster-region
```

```
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region $Region
```

If the value of StackStatus is anything other than CREATE_COMPLETE or CREATE_IN_PROGRESS, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name  
Region=cluster-region  
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --  
region $Region
```

To check the status of the AOCECS daemon service, enter the following command. In the output, you should see that runningCount is equal to the desiredCount in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name  
Region=cluster-region  
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --  
region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/aws/ecs/containerinsights/{ClusterName}/performance** log group.

Manual and custom setup

Follow the steps in this section to manually deploy the AWS Distro for OpenTelemetry to collect instance-level metrics from your Amazon ECS clusters that are hosted on Amazon EC2 instances.

Step 1: Necessary roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see [Create IAM policy](#) and [Create IAM role](#).

Step 2: Create the task definition

Create a task definition and use it to launch the AWS Distro for OpenTelemetry as a daemon service.

To use the task definition template to create the task definition, follow the instructions in [Create ECS EC2 Task Definition for EC2 instance with AWS OTel Collector](#).

To use the Amazon ECS console to create the task definition, follow the instructions in [Install AWS OTel Collector by creating Task Definition through AWS console for Amazon ECS EC2 instance metrics](#).

Step 3: Launch the daemon service

To launch the AWS Distro for OpenTelemetry as a daemon service, follow the instructions in [Run your task on the Amazon Elastic Container Service \(Amazon ECS\) using daemon service](#).

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the AWS Distro for OpenTelemetry in your Amazon ECS clusters that are hosted on Amazon EC2 instances. For more information, about creating a configuration file, see [Custom OpenTelemetry Configuration](#). For more information about the options that you can use in the configuration file, see [AWS Container Insights Receiver](#).

Set up FireLens to send logs to CloudWatch Logs

FireLens for Amazon ECS enables you to use task definition parameters to route logs to Amazon CloudWatch Logs for log storage and analytics. FireLens works with [Fluent Bit](#) and [Fluentd](#). We provide an AWS for Fluent Bit image, or you can use your own Fluent Bit or Fluentd image. Creating Amazon ECS task definitions with a FireLens configuration is supported using the AWS SDKs, AWS CLI, and AWS Management Console. For more information about CloudWatch Logs, see [What is CloudWatch Logs?](#).

There are key considerations when using FireLens for Amazon ECS. For more information, see [Considerations](#).

To find the AWS for Fluent Bit images, see [Using the AWS for Fluent Bit image](#).

To create a task definition that uses a FireLens configuration, see [Creating a task definition that uses a FireLens configuration](#).

Example

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see [What Is Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch Logs User Guide*.

In the log configuration options, specify the log group name and the Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group": "true"`. You can

also specify the task ID as the log stream prefix, which assists in filtering. For more information, see [Fluent Bit Plugin for CloudWatch Logs](#).

```
{
  "family": "firelens-example-cloudwatch",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit"
      },
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "firelens-container",
          "awslogs-region": "us-west-2",
          "awslogs-create-group": "true",
          "awslogs-stream-prefix": "firelens"
        }
      },
      "memoryReservation": 50
    },
    {
      "essential": true,
      "image": "nginx",
      "name": "app",
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
          "Name": "cloudwatch",
          "region": "us-west-2",
          "log_key": "log",
          "log_group_name": "/aws/ecs/containerinsights/
$(ecs_cluster)/application",
          "auto_create_group": "true",
          "log_stream_name": "${ecs_task_id}"
        }
      },
      "memoryReservation": 100
    }
  ]
}
```

```
]
}
```

Setting up Container Insights on Amazon EKS and Kubernetes

Container Insights is supported on Amazon EKS versions 1.23 and later. The quick start method of installation is supported only on versions 1.24 and later.

The overall process for setting up Container Insights on Amazon EKS or Kubernetes is as follows:

1. Verify that you have the necessary prerequisites.
2. Set up the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent, or AWS Distro for OpenTelemetry on your cluster to send metrics to CloudWatch.

Note

To use Container Insights with enhanced observability for Amazon EKS, you must use the Amazon CloudWatch Observability EKS add-on or the CloudWatch agent. For more information about this version of Container Insights, see [Container Insights with enhanced observability for Amazon EKS](#).

To use Container Insights with Fargate, you must use AWS Distro for OpenTelemetry. Container Insights with enhanced observability for Amazon EKS is not supported on Fargate.

Note

Container Insights now supports Windows worker nodes in an Amazon EKS cluster. Container Insights with enhanced observability for Amazon EKS is also supported on Windows. For information about enabling Container Insights on Windows, see [Using the CloudWatch agent with Container Insights enhanced observability enabled](#).

Set up Fluent Bit or Fluentd to send logs to CloudWatch Logs. (This is enabled by default if you install the Amazon CloudWatch Observability EKS add-on.)

You can perform these steps at once as part of the quick start setup if you are using the CloudWatch agent, or do them separately.

3. (Optional) Set up Amazon EKS control plane logging.
4. (Optional) Set up the CloudWatch agent as a StatsD endpoint on the cluster to send StatsD metrics to CloudWatch.
5. (Optional) Enable App Mesh Envoy Access Logs.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Topics

- [Verify prerequisites](#)
- [Using the CloudWatch agent with Container Insights enhanced observability enabled](#)
- [Using AWS Distro for OpenTelemetry](#)
- [Send logs to CloudWatch Logs](#)
- [Updating or deleting Container Insights on Amazon EKS and Kubernetes](#)

Verify prerequisites

Before you install Container Insights on Amazon EKS or Kubernetes, verify the following. These prerequisites apply whether you are using the CloudWatch agent or AWS Distro for OpenTelemetry to set up Container Insights on Amazon EKS clusters.

- You have a functional Amazon EKS or Kubernetes cluster with nodes attached in one of the Regions that supports the Container Insights for Amazon EKS and Kubernetes. For the list of supported Regions, see [Container Insights](#).
- You have `kubectl` installed and running. For more information, see [Installing kubectl](#) in the *Amazon EKS User Guide*.
- If you're using Kubernetes running on AWS instead of using Amazon EKS, the following prerequisites are also necessary:
 - Be sure that your Kubernetes cluster has enabled role-based access control (RBAC). For more information, see [Using RBAC Authorization](#) in the Kubernetes Reference.
 - Your kubelet has enabled Webhook authorization mode. For more information, see [Kubelet authentication/authorization](#) in the Kubernetes Reference.

You must also grant IAM permissions to enable your Amazon EKS worker nodes to send metrics and logs to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and other Kubernetes clusters.
- Use an IAM role for service accounts for the cluster, and attach the policy to this role. This works only for Amazon EKS clusters.

The first option grants permissions to CloudWatch for the entire node, while using an IAM role for the service account gives CloudWatch access to only the appropriate daemonset pods.

Attaching a policy to the IAM role of your worker nodes

Follow these steps to attach the policy to the IAM role of your worker nodes. This works for both Amazon EKS clusters and Kubernetes clusters outside of Amazon EKS.

To attach the necessary policy to the IAM role for your worker nodes

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Select one of the worker node instances and choose the IAM role in the description.
3. On the IAM role page, choose **Attach policies**.
4. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find this policy.
5. Choose **Attach policies**.

If you're running a Kubernetes cluster outside Amazon EKS, you might not already have an IAM role attached to your worker nodes. If not, you must first attach an IAM role to the instance and then add the policy as explained in the previous steps. For more information on attaching a role to an instance, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide*.

If you're running a Kubernetes cluster outside Amazon EKS and you want to collect EBS volume IDs in the metrics, you must add another policy to the IAM role attached to the instance. Add the following as an inline policy. For more information, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
```

```
"Statement": [  
  {  
    "Action": [  
      "ec2:DescribeVolumes"  
    ],  
    "Resource": "*",  
    "Effect": "Allow"  
  }  
]
```

Using an IAM service account role

This method works only on Amazon EKS clusters.

To grant permission to CloudWatch using an IAM service account role

1. If you haven't already, enable IAM roles for service accounts on your cluster. For more information, see [Enabling IAM roles for service accounts on your cluster](#).
2. If you haven't already, configure the service account to use an IAM role. For more information, see [Configuring a Kubernetes service account to assume an IAM role](#).

When you create the role, attach the **CloudWatchAgentServerPolicy** IAM policy to the role in addition to the policy that you create for the role. Also, the associated Kubernetes Service Account that is linked to this role should be created in the `amazon-cloudwatch` namespace, where the CloudWatch and Fluent Bit daemonsets will be deployed in the upcoming steps

3. If you haven't already, associate the IAM role with a service account in your cluster. For more information, see [Configuring a Kubernetes service account to assume an IAM role](#).

Using the CloudWatch agent with Container Insights enhanced observability enabled

Use the instructions in one of the following sections to set up Container Insights on an Amazon EKS cluster or Kubernetes cluster by using the CloudWatch agent. The quick start instructions are supported only on Amazon EKS versions 1.24 and later.

Note

You can install Container Insights by following the instructions in any one of the following sections. You don't need to follow all three sets of instructions.

Topics

- [Install the Amazon CloudWatch Observability EKS add-on](#)
- [Quick Start setup for Container Insights on Amazon EKS and Kubernetes](#)
- [Set up the CloudWatch agent to collect cluster metrics](#)

Install the Amazon CloudWatch Observability EKS add-on

You can use the Amazon EKS add-on to install Container Insights with enhanced observability for Amazon EKS. The add-on installs the CloudWatch agent to send infrastructure metrics from the cluster, installs Fluent Bit to send container logs, and also enables CloudWatch [Application Signals](#) to send application performance telemetry.

When you use the Amazon EKS add-on version 1.5.0 or later, Container Insights is enabled on both Linux and Windows worker nodes in the cluster. Currently, Application Signals is not supported on Windows in Amazon EKS.

The Amazon EKS add-on is not supported for clusters running Kubernetes instead of Amazon EKS.

For more information about the Amazon CloudWatch Observability EKS add-on, see [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart](#).

To install the Amazon CloudWatch Observability EKS add-on

1. First, set up the necessary permissions by attaching the **CloudWatchAgentServerPolicy** IAM policy to your worker nodes. To do so, enter the following command. Replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \  
--role-name my-worker-node-role \  
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

2. Enter the following command to install the add-on:

```
aws eks create-addon --cluster-name my-cluster-name --addon-name amazon-cloudwatch-observability
```

Quick Start setup for Container Insights on Amazon EKS and Kubernetes

Important

If you are installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing networks, you must use the Amazon CloudWatch Observability EKS add-on. For more information and instructions, see [Install the Amazon CloudWatch Observability EKS add-on](#).

To complete the setup of Container Insights, you can follow the quick start instructions in this section. If you are installing in an Amazon EKS cluster and you use the instructions in this section on or after November 6, 2023, you install Container Insights with enhanced observability for Amazon EKS in the cluster.

Important

Before completing the steps in this section, you must have verified the prerequisites including IAM permissions. For more information, see [Verify prerequisites](#).

Alternatively, you can instead follow the instructions in the following two sections, [Set up the CloudWatch agent to collect cluster metrics](#) and [Send logs to CloudWatch Logs](#). Those sections provide more configuration details on how the CloudWatch agent works with Amazon EKS and Kubernetes, but require you to perform more installation steps.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Note

Amazon has now launched Fluent Bit as the default log solution for Container Insights with significant performance gains. We recommend that you use Fluent Bit instead of Fluentd.

Quick Start with the CloudWatch agent operator and Fluent Bit

There are two configurations for Fluent Bit: an optimized version and a version that provides an experience more similar to Fluentd. The Quick Start configuration uses the optimized version. For more details about the Fluentd-compatible configuration, see [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs](#).

The CloudWatch agent operator is an additional container that gets installed to an Amazon EKS cluster. It is modeled after the OpenTelemetry Operator for Kubernetes. The operator manages the lifecycle of Kubernetes resources in a cluster. It installs the CloudWatch Agent, DCGM Exporter (NVIDIA), and the AWS Neuron Monitor on an Amazon EKS cluster and manages them. Fluent Bit and the CloudWatch Agent for Windows are installed directly to an Amazon EKS cluster without the operator managing them.

For a more secure and feature-rich certificate authority solution, the CloudWatch agent operator requires cert-manager, a widely-adopted solution for TLS certificate management in Kubernetes. Using cert-manager simplifies the process of obtaining, renewing, managing and using these certificates. It ensures that certificates are valid and up to date, and attempts to renew certificates at a configured time before expiry. cert-manager also facilitates issuing certificates from a variety of supported sources, including AWS Certificate Manager Private Certificate Authority.

To deploy Container Insights using the quick start

1. Install cert-manager if it is not already installed in the cluster. For more information, see [cert-manager Installation](#).
2. Install the custom resource definitions (CRD) by entering the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl apply --server-side -f -
```

3. Install the operator by entering the following command. Replace *my-cluster-name* with the name of your Amazon EKS or Kubernetes cluster, and replace *my-cluster-region* with the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the AWS outbound data transfer costs.

```
ClusterName=my-cluster-name  
RegionName=my-cluster-region  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/
```

```
{{cluster_name}}/'${ClusterName}']/g;s/{{region_name}}/'${RegionName}']/g' | kubectl  
apply -f -
```

For example, to deploy Container Insights on the cluster named MyCluster and publish the logs and metrics to US West (Oregon), enter the following command.

```
ClusterName='MyCluster'  
RegionName='us-west-2'  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-  
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/  
{{cluster_name}}/'${ClusterName}']/g;s/{{region_name}}/'${RegionName}']/g' | kubectl  
apply -f -
```

Migrating from Container Insights

If you already have Container Insights configured in an Amazon EKS cluster and you want to migrate to Container Insights with enhanced observability for Amazon EKS, see [Upgrading to Container Insights with enhanced observability for Amazon EKS](#)

Deleting Container Insights

If you want to remove Container Insights after using the quick start setup, enter the following commands.

```
ClusterName=my-cluster-name  
RegionName=my-cluster-region  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-  
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/  
{{cluster_name}}/'${ClusterName}']/g;s/{{region_name}}/'${RegionName}']/g' | kubectl  
delete -f -  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-  
insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete  
-f -
```

Set up the CloudWatch agent to collect cluster metrics

Important

If you are installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of

using the instructions in this section. For more information and instructions, see [Install the Amazon CloudWatch Observability EKS add-on](#).

To set up Container Insights to collect metrics, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes](#) or you can follow the steps in this section. In the following steps, you set up the CloudWatch agent to be able to collect metrics from your clusters.

If you are installing in an Amazon EKS cluster and you use the instructions in this section on or after November 6, 2023, you install Container Insights with enhanced observability for Amazon EKS in the cluster.

Step 1: Create a namespace for CloudWatch

Use the following step to create a Kubernetes namespace called `amazon-cloudwatch` for CloudWatch. You can skip this step if you have already created this namespace.

To create a namespace for CloudWatch

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

Step 2: Create a service account in the cluster

Use the following step to create a service account for the CloudWatch agent, if you do not already have one.

To create a service account for the CloudWatch agent

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-serviceaccount.yaml
```


If you didn't follow the previous steps, but you already have a service account for the CloudWatch agent that you want to use, you must ensure that it has the following rules. Additionally, in the rest of the steps in the Container Insights installation, you must use the name of that service account instead of `ccloudwatch-agent`.

```
rules:
- apiGroups: ["" ]
  resources: ["pods", "nodes", "endpoints"]
  verbs: ["list", "watch"]
- apiGroups: [ "" ]
  resources: [ "services" ]
  verbs: [ "list", "watch" ]
- apiGroups: ["apps"]
  resources: ["replicasets", "daemonsets", "deployments", "statefulsets"]
  verbs: ["list", "watch"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["list", "watch"]
- apiGroups: ["" ]
  resources: ["nodes/proxy"]
  verbs: ["get"]
- apiGroups: ["" ]
  resources: ["nodes/stats", "configmaps", "events"]
  verbs: ["create", "get"]
- apiGroups: ["" ]
  resources: ["configmaps"]
  resourceName: ["cwagent-clusterleader"]
  verbs: ["get","update"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get", "list", "watch"]
```

Step 3: Create a ConfigMap for the CloudWatch agent

Use the following steps to create a ConfigMap for the CloudWatch agent.

To create a ConfigMap for the CloudWatch agent

1. Download the ConfigMap YAML to your `kubectl` client host by running the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-configmap-enhanced.yaml
```

2. Edit the downloaded YAML file, as follows:

- **cluster_name** – In the `kubernetes` section, replace `{{cluster_name}}` with the name of your cluster. Remove the `{{}}` characters. Alternatively, if you're using an Amazon EKS cluster, you can delete the `"cluster_name"` field and value. If you do, the CloudWatch agent detects the cluster name from the Amazon EC2 tags.

3. (Optional) Make further changes to the ConfigMap based on your monitoring requirements, as follows:

- **metrics_collection_interval** – In the `kubernetes` section, you can specify how often the agent collects metrics. The default is 60 seconds. The default cadvisor collection interval in kubelet is 15 seconds, so don't set this value to less than 15 seconds.
- **endpoint_override** – In the `logs` section, you can specify the CloudWatch Logs endpoint if you want to override the default endpoint. You might want to do this if you're publishing from a cluster in a VPC and you want the data to go to a VPC endpoint.
- **force_flush_interval** – In the `logs` section, you can specify the interval for batching log events before they are published to CloudWatch Logs. The default is 5 seconds.
- **region** – By default, the agent published metrics to the Region where the worker node is located. To override this, you can add a `region` field in the agent section: for example, `"region": "us-west-2"`.
- **statsd** section – If you want the CloudWatch Logs agent to also run as a StatsD listener in each worker node of your cluster, you can add a `statsd` section to the `metrics` section, as in the following example. For information about other StatsD options for this section, see [Retrieve custom metrics with StatsD](#).

```
"metrics": {
  "metrics_collected": {
    "statsd": {
      "service_address": ":8125"
    }
  }
}
```

A full example of the JSON section is as follows.

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "cluster_name": "MyCluster",
        "metrics_collection_interval": 60,
        "enhanced_container_insights": true
      }
    },
    "force_flush_interval": 5,
    "endpoint_override": "logs.us-east-1.amazonaws.com"
  },
  "metrics": {
    "metrics_collected": {
      "statsd": {
        "service_address": ":8125"
      }
    }
  }
}
```

4. Create the ConfigMap in the cluster by running the following command.

```
kubectl apply -f cwagent-configmap.yaml
```

Step 4: Deploy the CloudWatch agent as a DaemonSet

To finish the installation of the CloudWatch agent and begin collecting container metrics, use the following steps.

To deploy the CloudWatch agent as a DaemonSet

1. • If you do not want to use StatsD on the cluster, enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
```

```
deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-  
daemonset.yaml
```

- If you do want to use StatsD, follow these steps:
 - a. Download the DaemonSet YAML to your `kubectl` client host by running the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-  
container-insights/latest/k8s-deployment-manifest-templates/deployment-  
mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

- b. Uncomment the port section in the `cwagent-daemonset.yaml` file as in the following:

```
ports:  
  - containerPort: 8125  
    hostPort: 8125  
    protocol: UDP
```

- c. Deploy the CloudWatch agent in your cluster by running the following command.

```
kubectl apply -f cwagent-daemonset.yaml
```

- d. Deploy the CloudWatch agent on Windows nodes in your cluster by running the following command. The StatsD listener is not supported on the CloudWatch agent on Windows.

```
kubectl apply -f cwagent-daemonset-windows.yaml
```

2. Validate that the agent is deployed by running the following command.

```
kubectl get pods -n amazon-cloudwatch
```

When complete, the CloudWatch agent creates a log group named `/aws/containerinsights/Cluster_Name/performance` and sends the performance log events to this log group. If you also set up the agent as a StatsD listener, the agent also listens for StatsD metrics on port 8125 with the IP address of the node where the application pod is scheduled.

Troubleshooting

If the agent doesn't deploy correctly, try the following:

- Run the following command to get the list of pods.

```
kubectl get pods -n amazon-cloudwatch
```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Using AWS Distro for OpenTelemetry

You can set up Container Insights to collect metrics from Amazon EKS clusters by using the AWS Distro for OpenTelemetry collector. For more information about the AWS Distro for OpenTelemetry, see [AWS Distro for OpenTelemetry](#).

Important

If you install using AWS Distro for OpenTelemetry, you install Container Insights but do not get Container Insights with enhanced observability for Amazon EKS. You will not collect the detailed metrics supported in Container Insights with enhanced observability for Amazon EKS.

How you set up Container Insights depends on whether the cluster is hosted on Amazon EC2 instances or on AWS Fargate (Fargate).

Amazon EKS clusters hosted on Amazon EC2

If you have not already done so, make sure that you have fulfilled the prerequisites including the necessary IAM roles. For more information, see [Verify prerequisites](#).

Amazon provides a Helm chart that you can use to set up the monitoring of Amazon Elastic Kubernetes Service on Amazon EC2. This monitoring uses the AWS Distro for

OpenTelemetry(ADOT) Collector for metrics and Fluent Bit for logs. Therefore, the Helm chart is useful for customers who use Amazon EKS on Amazon EC2 and want to collect metrics and logs to send to CloudWatch Container Insights. For more information about this Helm chart, see [ADOT Helm chart for EKS on EC2 metrics and logs to Amazon CloudWatch Container Insights](#).

Alternatively, you can also use the instructions in the rest of this section.

First, deploy the AWS Distro for OpenTelemetry collector as a DaemonSet by entering the following command.

```
curl https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/eks/otel-container-insights-infra.yaml |
kubectl apply -f -
```

To confirm that the collector is running, enter the following command.

```
kubectl get pods -l name=aws-otel-eks-ci -n aws-otel-eks
```

If the output of this command includes multiple pods in the Running state, the collector is running and collecting metrics from the cluster. The collector creates a log group named `aws/containerinsights/cluster-name/performance` and sends the performance log events to it.

For information about how to see your Container Insights metrics in CloudWatch, see [Viewing Container Insights metrics](#).

AWS has also provided documentation on GitHub for this scenario. If you want to customize the metrics and logs published by Container Insights, see <https://aws-otel.github.io/docs/getting-started/container-insights/eks-infra>.

Amazon EKS clusters hosted on Fargate

For instructions for how to configure and deploy an ADOT Collector to collect system metrics from workloads deployed to an Amazon EKS cluster on Fargate and send them to CloudWatch Container Insights, see [Container Insights EKS Fargate](#) in the AWS Distro for OpenTelemetry documentation.

Send logs to CloudWatch Logs

To send logs from your containers to Amazon CloudWatch Logs, you can use Fluent Bit or Fluentd. For more information, see [Fluent Bit](#) and [Fluentd](#).

If you are not already using Fluentd, we recommend that you use Fluent Bit for the following reasons:

- Fluent Bit has a smaller resource footprint and is more resource-efficient with memory and CPU usage than Fluentd. For a more detailed comparison, see [Fluent Bit and Fluentd performance comparison](#).
- The Fluent Bit image is developed and maintained by AWS. This gives AWS the ability to adopt new Fluent Bit image features and respond to issues much quicker.

Topics

- [Fluent Bit and Fluentd performance comparison](#)
- [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs](#)
- [\(Optional\) Set up Fluentd as a DaemonSet to send logs to CloudWatch Logs](#)
- [\(Optional\) Set up Amazon EKS control plane logging](#)
- [\(Optional\) Enable App Mesh Envoy access logs](#)
- [\(Optional\) Enable the Use_Kubelet feature for large clusters](#)

Fluent Bit and Fluentd performance comparison

The following tables show the performance advantage that Fluent Bit has over Fluentd in memory and CPU usages. The following numbers are just for reference and might change depending on the environment.

Logs per second	Fluentd CPU usage	Fluent Bit CPU usage with Fluentd-compatible configuration	Fluent Bit CPU usage with optimized configuration
100	0.35 vCPU	0.02 vCPU	0.02 vCPU
1,000	0.32 vCPU	0.14 vCPU	0.11 vCPU
5,000	0.85 vCPU	0.48 vCPU	0.30 vCPU
10,000	0.94 vCPU	0.60 vCPU	0.39 vCPU

Logs per second	Fluentd memory usage	Fluent Bit memory usage with Fluentd-compatible configuration	Fluent Bit memory usage with optimized configuration
100	153 MB	46 MB	37 MB
1,000	270 MB	45 MB	40 MB
5,000	320 MB	55 MB	45 MB
10,000	375 MB	92 MB	75 MB

Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs

The following sections help you deploy Fluent Bit to send logs from containers to CloudWatch Logs.

Topics

- [Differences if you're already using Fluentd](#)
- [Setting up Fluent Bit](#)
- [Multi-line log support](#)
- [\(Optional\) Reducing the log volume from Fluent Bit](#)
- [Troubleshooting](#)
- [Dashboard](#)

Differences if you're already using Fluentd

If you are already using Fluentd to send logs from containers to CloudWatch Logs, read this section to see the differences between Fluentd and Fluent Bit. If you are not already using Fluentd with Container Insights, you can skip to [Setting up Fluent Bit](#).

We provide two default configurations for Fluent Bit:

- **Fluent Bit optimized configuration** — A configuration aligned with Fluent Bit best practices.
- **Fluentd-compatible configuration** — A configuration that is aligned with Fluentd behavior as much as possible.

The following list explains the differences between Fluentd and each Fluent Bit configuration in detail.

- **Differences in log stream names** — If you use the Fluent Bit optimized configuration, the log stream names will be different.

Under `/aws/containerinsights/Cluster_Name/application`

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName-application.var.log.containers.kubernetes-podName_kubernetes-namespace_kubernetes-container-name-kubernetes-containerID`
- Fluentd sends logs to `kubernetes-podName_kubernetes-namespace_kubernetes-containerName_kubernetes-containerID`

Under `/aws/containerinsights/Cluster_Name/host`

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName.host-log-file`
- Fluentd sends logs to `host-log-file-Kubernetes-NodePrivateIp`

Under `/aws/containerinsights/Cluster_Name/dataplane`

- Fluent Bit optimized configuration sends logs to `kubernetes-nodeName.dataplaneServiceLog`
- Fluentd sends logs to `dataplaneServiceLog-Kubernetes-nodeName`
- The kube-proxy and aws-node log files that Container Insights writes are in different locations. In Fluentd configuration, they are in `/aws/containerinsights/Cluster_Name/application`. In the Fluent Bit optimized configuration, they are in `/aws/containerinsights/Cluster_Name/dataplane`.
- Most metadata such as `pod_name` and `namespace_name` are the same in Fluent Bit and Fluentd, but the following are different.
 - The Fluent Bit optimized configuration uses `docker_id` and Fluentd use `Docker.container_id`.
 - Both Fluent Bit configurations do not use the following metadata. They are present only in Fluentd: `container_image_id`, `master_url`, `namespace_id`, and `namespace_labels`.

Setting up Fluent Bit

To set up Fluent Bit to collect logs from your containers, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes](#) or you can follow the steps in this section.

With either method, the IAM role that is attached to the cluster nodes must have sufficient permissions. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the *Amazon EKS User Guide*.

In the following steps, you set up Fluent Bit as a daemonSet to send logs to CloudWatch Logs. When you complete this step, Fluent Bit creates the following log groups if they don't already exist.

Important

If you already have FluentD configured in Container Insights and the FluentD DaemonSet is not running as expected (this can happen if you use the `containerd` runtime), you must uninstall it before installing Fluent Bit to prevent Fluent Bit from processing the FluentD error log messages. Otherwise, you must uninstall FluentD immediately after you have successfully installed Fluent Bit. Uninstalling FluentD after installing Fluent Bit ensures continuity in logging during this migration process. Only one of Fluent Bit or FluentD is needed to send logs to CloudWatch Logs.

Log group name	Log source
<code>/aws/containerinsights/Cluster_N_ame/application</code>	All log files in <code>/var/log/containers</code>
<code>/aws/containerinsights/Cluster_N_ame/host</code>	Logs from <code>/var/log/dmesg</code> , <code>/var/log/secure</code> , and <code>/var/log/messages</code>
<code>/aws/containerinsights/Cluster_N_ame/dataplane</code>	The logs in <code>/var/log/journal</code> for <code>kubelet.service</code> , <code>kubeproxy.service</code> , and <code>docker.service</code> .

To install Fluent Bit to send logs from containers to CloudWatch Logs

1. If you don't already have a namespace called `amazon-cloudwatch`, create one by entering the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

2. Run the following command to create a ConfigMap named `cluster-info` with the cluster name and the Region to send logs to. Replace *cluster-name* and *cluster-region* with your cluster's name and Region.

```
ClusterName=cluster-name
RegionName=cluster-region
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off' ||
  FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||
  FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

In this command, the `FluentBitHttpServer` for monitoring plugin metrics is on by default. To turn it off, change the third line in the command to `FluentBitHttpPort=''` (empty string) in the command.

Also by default, Fluent Bit reads log files from the tail, and will capture only new logs after it is deployed. If you want the opposite, set `FluentBitReadFromHead='On'` and it will collect all logs in the file system.

3. Download and deploy the Fluent Bit daemonset to the cluster by running one of the following commands.
 - If you want the Fluent Bit optimized configuration for Linux computers, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit.yaml
```

- If you want the Fluent Bit optimized configuration for Windows computers, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-windows.yaml
```

- If you are using Linux computers and want the Fluent Bit configuration that is more similar to Fluentd, run this command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml
```

Important

The Fluent Bit daemonset configuration by default sets the log level to INFO, which can result in higher CloudWatch Logs ingestion costs. If you want to reduce log ingestion volume and costs, you can change the log level to ERROR.

For more information about how to reduce the log volume, see [\(Optional\) Reducing the log volume from Fluent Bit](#)

4. Validate the deployment by entering the following command. Each node should have one pod named **fluent-bit-***.

```
kubectl get pods -n amazon-cloudwatch
```

The above steps create the following resources in the cluster:

- A service account named `Fluent-Bit` in the `amazon-cloudwatch` namespace. This service account is used to run the Fluent Bit daemonSet. For more information, see [Managing Service Accounts](#) in the Kubernetes Reference.
- A cluster role named `Fluent-Bit-role` in the `amazon-cloudwatch` namespace. This cluster role grants `get`, `list`, and `watch` permissions on pod logs to the `Fluent-Bit` service account. For more information, see [API Overview](#) in the Kubernetes Reference.
- A ConfigMap named `Fluent-Bit-config` in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by Fluent Bit. For more information, see [Configure a Pod to Use a ConfigMap](#) in the Kubernetes Tasks documentation.

If you want to verify your Fluent Bit setup, follow these steps.

Verify the Fluent Bit setup

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**.
3. Make sure that you're in the Region where you deployed Fluent Bit.
4. Check the list of log groups in the Region. You should see the following:
 - `/aws/containerinsights/Cluster_Name/application`
 - `/aws/containerinsights/Cluster_Name/host`
 - `/aws/containerinsights/Cluster_Name/dataplane`
5. Navigate to one of these log groups and check the **Last Event Time** for the log streams. If it is recent relative to when you deployed Fluent Bit, the setup is verified.

There might be a slight delay in creating the `/dataplane` log group. This is normal as these log groups only get created when Fluent Bit starts sending logs for that log group.

Multi-line log support

For information on how to use Fluent Bit with multi-line logs, see the following sections of the Fluent Bit documentation:

- [Multiline Parsing](#)
- [Multiline and Containers \(v1.8\)](#)
- [Multiline Core \(v1.8\)](#)

- [Always use multiline in the tail input](#)

(Optional) Reducing the log volume from Fluent Bit

By default, we send Fluent Bit application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch. If you have followed the steps on this page to set up Fluent Bit, download the Kubernetes manifest YAML file from the `kubectl apply` command that you previously ran and modify it with your changes, which you can then re-apply to your cluster. Alternatively, if you are using the Amazon CloudWatch Observability EKS add-on or Helm chart, see [\(Optional\) Additional configuration](#) for information about managing the Fluent Bit configuration by using the add-on's advanced config or the Helm chart.

To stop Fluent Bit application logs, remove the following section from the Fluent Bit configuration file.

```
[INPUT]
  Name          tail
  Tag           application.*
  Path          /var/log/containers/fluent-bit*
  Parser       docker
  DB            /fluent-bit/state/flb_log.db
  Mem_Buf_Limit 5MB
  Skip_Long_Lines On
  Refresh_Interval 10
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add the following filters to the `application-log.conf` section in the Fluent Bit configuration. Replace `<Metadata_1>` and the similar fields with with the actual metadata identifiers.

```
application-log.conf: |
  [FILTER]
    Name          nest
    Match         application.*
    Operation     lift
    Nested_under  kubernetes
    Add_prefix    Kube.

  [FILTER]
    Name          modify
```

Match	application.*
Remove	Kube.<Metadata_1>
Remove	Kube.<Metadata_2>
Remove	Kube.<Metadata_3>
[FILTER]	
Name	nest
Match	application.*
Operation	nest
Wildcard	Kube.*
Nested_under	kubernetes
Remove_prefix	Kube.

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the Fluent Bit daemonSet pods to look for the error.

Run the following command and make sure that the status is Running.

```
kubectl get pods -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role that is attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the *Amazon EKS User Guide*.

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

Dashboard

You can create a dashboard to monitor metrics of each running plugin. You can see data for input and output bytes and for record processing rates as well as output errors and retry/failed rates. To view these metrics, you will need to install the CloudWatch agent with Prometheus metrics collection for Amazon EKS and Kubernetes clusters. For more information about how to set up the dashboard, see [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters](#).

Note

Before you can set up this dashboard, you must set up Container Insights for Prometheus metrics. For more information, see [Container Insights Prometheus metrics monitoring](#).

To create a dashboard for the Fluent Bit Prometheus metrics

1. Create environment variables, replacing the values on the right in the following lines to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-west-1
CLUSTER_NAME=your_kubernetes_cluster_name
```

2. Create the dashboard by running the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
| sed "s/{{YOUR_AWS_REGION}}/{{REGION_NAME}}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/{{CLUSTER_NAME}}/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --
dashboard-body
```

(Optional) Set up Fluentd as a DaemonSet to send logs to CloudWatch Logs**Warning**

Container Insights support for Fluentd is now in maintenance mode, which means that AWS will not provide any further updates for Fluentd and that we are planning to deprecate it in near future. Additionally, the current Fluentd configuration for Container Insights is using an old version of the Fluentd Image `fluent/fluentd-kubernetes-daemonset:v1.10.3-debian-cloudwatch-1.0` which does not have the latest improvement and security patches. For the latest Fluentd image supported by the open source community, see [fluentd-kubernetes-daemonset](#).

We strongly recommend that you migrate to use FluentBit with Container Insights whenever possible. Using FluentBit as the log forwarder for Container Insights provides significant performance gains.

For more information, see [Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs](#) and [Differences if you're already using Fluentd](#).

To set up Fluentd to collect logs from your containers, you can follow the steps in [Quick Start setup for Container Insights on Amazon EKS and Kubernetes](#) or you can follow the steps in this section. In the following steps, you set up Fluentd as a DaemonSet to send logs to CloudWatch Logs. When you complete this step, Fluentd creates the following log groups if they don't already exist.

Log group name	Log source
/aws/containerinsights/ <i>Cluster_N</i> <i>ame</i> /application	All log files in /var/log/containers
/aws/containerinsights/ <i>Cluster_N</i> <i>ame</i> /host	Logs from /var/log/dmesg , /var/log/secure , and /var/log/messages
/aws/containerinsights/ <i>Cluster_N</i> <i>ame</i> /dataplane	The logs in /var/log/journal for kubelet.service , kubeproxy.service , and docker.service .

Step 1: Create a namespace for CloudWatch

Use the following step to create a Kubernetes namespace called amazon-cloudwatch for CloudWatch. You can skip this step if you have already created this namespace.

To create a namespace for CloudWatch

- Enter the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

Step 2: Install Fluentd

Start this process by downloading Fluentd. When you finish these steps, the deployment creates the following resources on the cluster:

- A service account named `fluentd` in the `amazon-cloudwatch` namespace. This service account is used to run the Fluentd DaemonSet. For more information, see [Managing Service Accounts](#) in the Kubernetes Reference.
- A cluster role named `fluentd` in the `amazon-cloudwatch` namespace. This cluster role grants `get`, `list`, and `watch` permissions on pod logs to the `fluentd` service account. For more information, see [API Overview](#) in the Kubernetes Reference.
- A ConfigMap named `fluentd-config` in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by Fluentd. For more information, see [Configure a Pod to Use a ConfigMap](#) in the Kubernetes Tasks documentation.

To install Fluentd

1. Create a ConfigMap named `cluster-info` with the cluster name and the AWS Region that the logs will be sent to. Run the following command, updating the placeholders with your cluster and Region names.

```
kubectl create configmap cluster-info \  
--from-literal=cluster.name=cluster_name \  
--from-literal=logs.region=region_name -n amazon-cloudwatch
```

2. Download and deploy the Fluentd DaemonSet to the cluster by running the following command. Make sure that you are using the container image with correct architecture. The example manifest only works on x86 instances and will enter `CrashLoopBackOff` if you have Advanced RISC Machine (ARM) instances in your cluster. The Fluentd daemonSet does not have an official multi-architecture Docker image that enables you to use one tag for multiple underlying images and let the container runtime pull the right one. The Fluentd ARM image uses a different tag with an `arm64` suffix.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-  
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/  
daemonset/container-insights-monitoring/fluentd/fluentd.yaml
```

Note

Because of a recent change to optimize the Fluentd configuration and minimize the impact of Fluentd API requests on Kubernetes API endpoints, the "Watch" option for

Kubernetes filters has been disabled by default. For more details, see [fluent-plugin-kubernetes_metadata_filter](#).

3. Validate the deployment by running the following command. Each node should have one pod named `fluentd-cloudwatch-*`.

```
kubectl get pods -n amazon-cloudwatch
```

Step 3: Verify the Fluentd setup

To verify your Fluentd setup, use the following steps.

To verify the Fluentd setup for Container Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Log groups**. Make sure that you're in the Region where you deployed Fluentd to your containers.

In the list of log groups in the Region, you should see the following:

- `/aws/containerinsights/Cluster_Name/application`
- `/aws/containerinsights/Cluster_Name/host`
- `/aws/containerinsights/Cluster_Name/dataplane`

If you see these log groups, the Fluentd setup is verified.

Multi-line log support

On August 19 2019, we added multi-line log support for the logs collected by Fluentd.

By default, the multi-line log entry starter is any character with no white space. This means that all log lines that start with a character that does not have white space are considered as a new multi-line log entry.

If your own application logs use a different multi-line starter, you can support them by making two changes in the `fluentd.yaml` file.

First, exclude them from the default multi-line support by adding the pathnames of your log files to an `exclude_path` field in the `containers` section of `fluentd.yaml`. The following is an example.

```
<source>
  @type tail
  @id in_tail_container_logs
  @label @containers
  path /var/log/containers/*.log
  exclude_path ["full_pathname_of_log_file*", "full_pathname_of_log_file2*"]
```

Next, add a block for your log files to the `fluentd.yaml` file. The example below is used for the CloudWatch agent's log file, which uses a timestamp regular expression as the multi-line starter. You can copy this block and add it to `fluentd.yaml`. Change the indicated lines to reflect your application log file name and the multi-line starter that you want to use.

```
<source>
  @type tail
  @id in_tail_cwagent_logs
  @label @cwagentlogs
  path /var/log/containers/cloudwatch-agent*
  pos_file /var/log/cloudwatch-agent.log.pos
  tag *
  read_from_head true
<parse>
  @type json
  time_format %Y-%m-%dT%H:%M:%S.%NZ
</parse>
</source>
```

```
<label @cwagentlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_cwagent
  </filter>

  <filter **>
    @type record_transformer
```

```

    @id filter_cwagent_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
    </record>
  </filter>

  <filter **>
    @type concat
    key log
    multiline_start_regexp /\d{4}[-/] \d{1,2}[-/] \d{1,2}/
    separator ""
    flush_interval 5
    timeout_label @NORMAL
  </filter>

  <match **>
    @type relabel
    @label @NORMAL
  </match>
</label>

```

(Optional) Reducing the log volume from Fluentd

By default, we send Fluentd application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch.

To stop Fluentd application logs, remove the following section from the `fluentd.yaml` file.

```

<source>
  @type tail
  @id in_tail_fluentd_logs
  @label @fluentdlogs
  path /var/log/containers/fluentd*
  pos_file /var/log/fluentd.log.pos
  tag *
  read_from_head true
  <parse>
    @type json
    time_format %Y-%m-%dT%H:%M:%S.%NZ
  </parse>
</source>

```

```
<label @fluentdlogs>
  <filter **>
    @type kubernetes_metadata
    @id filter_kube_metadata_fluentd
  </filter>

  <filter **>
    @type record_transformer
    @id filter_fluentd_stream_transformer
    <record>
      stream_name ${tag_parts[3]}
    </record>
  </filter>

  <match **>
    @type relabel
    @label @NORMAL
  </match>
</label>
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add one line to the `record_transformer` section in the `fluentd.yaml` file. In the log source where you want to remove this metadata, add the following line.

```
remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
$.kubernetes.container_image_id, $.kubernetes.namespace_id
```

For example:

```
<filter **>
  @type record_transformer
  @id filter_containers_stream_transformer
  <record>
    stream_name ${tag_parts[3]}
  </record>
  remove_keys $.kubernetes.pod_id, $.kubernetes.master_url,
$.kubernetes.container_image_id, $.kubernetes.namespace_id
</filter>
```

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the Fluentd DaemonSet pods to look for the error.

Run the following command and make sure that the status is `Running`.

```
kubectl get pods -n amazon-cloudwatch
```

In the results of the previous command, note the pod name that starts with `fluentd-cloudwatch`. Use this pod name in the following command.

```
kubectl logs pod_name -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see [Amazon EKS IAM Policies, Roles, and Permissions](#) in the *Amazon EKS User Guide*.

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

If the pod status is `CrashLoopBackOff`, make sure that the architecture of the Fluentd container image is the same as the node when you installed Fluentd. If your cluster has both x86 and ARM64 nodes, you can use a `kubernetes.io/arch` label to place the images on the correct node. For more information, see kubernetes.io/arch.

(Optional) Set up Amazon EKS control plane logging

If you're using Amazon EKS, you can optionally enable Amazon EKS control plane logging, to provide audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs. For more information, see [Amazon EKS Control Plane Logging](#).

(Optional) Enable App Mesh Envoy access logs

You can set up Container Insights Fluentd to send App Mesh Envoy access logs to CloudWatch Logs. For more information, see [Logging](#).

To have Envoy access logs sent to CloudWatch Logs

1. Set up Fluentd in the cluster. For more information, see [\(Optional\) Set up Fluentd as a DaemonSet to send logs to CloudWatch Logs](#).
2. Configure Envoy access logs for your virtual nodes. For instructions, see [Logging](#). Be sure to configure the log path to be `/dev/stdout` in each virtual node.

When you have finished, the envoy access logs are sent to the `/aws/containerinsights/Cluster_Name/application` log group.

(Optional) Enable the `Use_Kubelet` feature for large clusters

By default, the `Use_Kubelet` feature is disabled in the FluentBit Kubernetes plugin. Enabling this feature can reduce traffic to the API server and mitigate the issue of the API Server being a bottleneck. We recommend that you enable this feature for large clusters.

To enable `Use_Kubelet`, first add the `nodes` and `nodes/proxy` permissions to the `clusterRole` config.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
    - /metrics
    verbs:
      - get
  - apiGroups: [""]
    resources:
      - namespaces
      - pods
      - pods/logs
      - nodes
      - nodes/proxy
    verbs: ["get", "list", "watch"]
```

In the `DaemonSet` configuration, this feature needs host network access. The image version for `amazon/aws-for-fluent-bit` should be 2.12.0 or later, or the `fluent-bit` image version should be 1.7.2 or later.

```
apiVersion: apps/v1
```



```
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: fluent-bit
  template:
    metadata:
      labels:
        k8s-app: fluent-bit
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
        - name: fluent-bit
          image: amazon/aws-for-fluent-bit:2.19.0
          imagePullPolicy: Always
          env:
            - name: AWS_REGION
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: logs.region
            - name: CLUSTER_NAME
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: cluster.name
            - name: HTTP_SERVER
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: http.server
            - name: HTTP_PORT
              valueFrom:
                configMapKeyRef:
                  name: fluent-bit-cluster-info
                  key: http.port
```

```
- name: READ_FROM_HEAD
  valueFrom:
    configMapKeyRef:
      name: fluent-bit-cluster-info
      key: read.head
- name: READ_FROM_TAIL
  valueFrom:
    configMapKeyRef:
      name: fluent-bit-cluster-info
      key: read.tail
- name: HOST_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
- name: HOSTNAME
  valueFrom:
    fieldRef:
      apiVersion: v1
      fieldPath: metadata.name
- name: CI_VERSION
  value: "k8s/1.3.8"
resources:
  limits:
    memory: 200Mi
  requests:
    cpu: 500m
    memory: 100Mi
volumeMounts:
# Please don't change below read-only permissions
- name: fluentbitstate
  mountPath: /var/fluent-bit/state
- name: varlog
  mountPath: /var/log
  readOnly: true
- name: varlibdockercontainers
  mountPath: /var/lib/docker/containers
  readOnly: true
- name: fluent-bit-config
  mountPath: /fluent-bit/etc/
- name: runlogjournal
  mountPath: /run/log/journal
  readOnly: true
- name: dmesg
  mountPath: /var/log/dmesg
```

```

    readOnly: true
  terminationGracePeriodSeconds: 10
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  volumes:
  - name: fluentbitstate
    hostPath:
      path: /var/fluent-bit/state
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibdockercontainers
    hostPath:
      path: /var/lib/docker/containers
  - name: fluent-bit-config
    configMap:
      name: fluent-bit-config
  - name: runlogjournal
    hostPath:
      path: /run/log/journal
  - name: dmesg
    hostPath:
      path: /var/log/dmesg
  serviceAccountName: fluent-bit
  tolerations:
  - key: node-role.kubernetes.io/master
    operator: Exists
    effect: NoSchedule
  - operator: "Exists"
    effect: "NoExecute"
  - operator: "Exists"
    effect: "NoSchedule"

```

The Kubernetes Plugin configuration should be similar to the following:

[FILTER]

Name	kubernetes
Match	application.*
Kube_URL	https://kubernetes.default.svc:443
Kube_Tag_Prefix	application.var.log.containers.
Merge_Log	On
Merge_Log_Key	log_processed
K8S-Logging.Parser	On

K8S-Logging.Exclude	Off
Labels	Off
Annotations	Off
Use_Kubelet	On
Kubelet_Port	10250
Buffer_Size	0

Updating or deleting Container Insights on Amazon EKS and Kubernetes

Use the steps in these sections to update your CloudWatch agent container image, or to remove Container Insights from an Amazon EKS or Kubernetes cluster.

Topics

- [Upgrading to Container Insights with enhanced observability for Amazon EKS](#)
- [Updating the CloudWatch agent container image](#)
- [Deleting the CloudWatch agent and Fluent Bit for Container Insights](#)

Upgrading to Container Insights with enhanced observability for Amazon EKS

Important

If you are upgrading or installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing metrics, you must use the Amazon CloudWatch Observability EKS add-on. For more information and instructions, see [Install the Amazon CloudWatch Observability EKS add-on](#).

Container Insights with enhanced observability for Amazon EKS is the newest version of Container Insights. It collects detailed metrics from clusters running Amazon EKS and offers curated, immediately usable dashboards to drill down into application and infrastructure telemetry. For more information about this version of Container Insights, see [Container Insights with enhanced observability for Amazon EKS](#).

If you have installed the original version of Container Insights in an Amazon EKS cluster and you want to upgrade it to the newer version with enhanced observability, follow the instructions in this section.

⚠ Important

Before completing the steps in this section, you must have verified the prerequisites including cert-manager. For more information, see [Quick Start with the CloudWatch agent operator and Fluent Bit](#).

To upgrade an Amazon EKS cluster to Container Insights with enhanced observability for Amazon EKS

1. Install the CloudWatch agent operator by entering the following command. Replace *my-cluster-name* with the name of your Amazon EKS or Kubernetes cluster, and replace *my-cluster-region* with the name of the Region where the logs are published. We recommend that you use the same Region where your cluster is deployed to reduce the AWS outbound data transfer costs.

```
ClusterName=my-cluster-name  
RegionName=my-cluster-region  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-  
container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/  
{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl  
apply -f -
```

If you notice a failure caused by conflicting resources, it is likely because you already have the CloudWatch agent and Fluent Bit with its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster. When the CloudWatch agent operator tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster. We recommend that you delete any existing CloudWatch agent with Container Insights setup that you had previously installed on the cluster, and then install the CloudWatch agent operator.

2. (Optional) To apply an existing custom Fluent Bit configuration, you must update the configmap associated with the Fluent Bit daemonset. The CloudWatch agent operator provides a default configuration for Fluent Bit, and you can override or modify the default configuration as needed. To apply a custom configuration, follow these steps.
 - a. Open the existing configuration by entering the following command.

```
kubectl edit cm fluent-bit-config -n amazon-cloudwatch
```

- b. Make your changes in the file, then enter `:wq` to save the file and exit edit mode.
- c. Restart Fluent Bit by entering the following command.

```
kubectl rollout restart fluent-bit -n amazon-cloudwatch
```

Updating the CloudWatch agent container image

Important

If you are upgrading or installing Container Insights on an Amazon EKS cluster, we recommend that you use the Amazon CloudWatch Observability EKS add-on for the installation, instead of using the instructions in this section. Additionally, to retrieve accelerated computing metrics, you must use the Amazon CloudWatch Observability EKS add-on or the CloudWatch agent operator. For more information and instructions, see [Install the Amazon CloudWatch Observability EKS add-on](#).

If you need to update your container image to the latest version, use the steps in this section.

To update your container image

1. Verify if the `amazoncloudwatchagent` Customer Resource Definition (CRD) already exists by entering the following command.

```
kubectl get crds amazoncloudwatchagents.cloudwatch.aws.amazon.com -n amazon-cloudwatch
```

If this command returns an error that the CRD is missing, the cluster doesn't have Container Insights with enhanced observability for Amazon EKS configured with the CloudWatch agent operator. In this case, see [Upgrading to Container Insights with enhanced observability for Amazon EKS](#).

2. Apply the latest `cwagent-version.yaml` file by entering the following command.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-version.yaml | kubectl apply -f -
```

Deleting the CloudWatch agent and Fluent Bit for Container Insights

If you installed Container Insights by using installing the CloudWatch Observability add-on for Amazon EKS, you can delete Container Insights and the CloudWatch agent by entering the following command:

Note

The Amazon EKS add-on now supports Container Insights on Windows worker nodes. If you delete the Amazon EKS add-on, Container Insights for Windows is also deleted.

```
aws eks delete-addon --cluster-name my-cluster --addon-name amazon-cloudwatch-observability
```

Otherwise, to delete all resources related to the CloudWatch agent and Fluent Bit, enter the following command. In this command, *My_Cluster_Name* is the name of your Amazon EKS or Kubernetes cluster, and *My_Region* is the name of the Region where the logs are published.

```
ClusterName=My_Cluster_Name  
RegionName=My-Region  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-operator-rendered.yaml | sed 's/{{cluster_name}}/'${ClusterName}'/g;s/{{region_name}}/'${RegionName}'/g' | kubectl delete -f -  
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/main/k8s-quickstart/cwagent-custom-resource-definitions.yaml | kubectl delete -f -
```

Viewing Container Insights metrics

After you have Container Insights set up and it is collecting metrics, you can view those metrics in the CloudWatch console.

For Container Insights metrics to appear on your dashboard, you must complete the Container Insights setup. For more information, see [Setting up Container Insights](#).

This procedure explains how to view the metrics that Container Insights automatically generates from the collected log data. The rest of this section explains how to further dive into your data and use CloudWatch Logs Insights to see more metrics at more levels of granularity.

To view Container Insights metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Insights, Container Insights**.
3. In the drop-down box under **Container Insights**, choose **Performance monitoring**.
4. Use the drop-down boxes near the top to select the type of resource to view, as well as the specific resource.

You can set a CloudWatch alarm on any metric that Container Insights collects. For more information, see [Using Amazon CloudWatch alarms](#)

Note

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard. If you have not already enabled Application Insights, you can do so by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

For more information about Application Insights and containerized applications, see [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring](#).

Viewing the top contributors

For some of the views in Container Insights performance monitoring, you can also see the top contributors by memory or CPU, or the most recently active resources. This is available when you select any of the following dashboards in the drop-down box near the top of the page:

- ECS Services
- ECS Tasks

- EKS Namespaces
- EKS Services
- EKS Pods

When you are viewing one of these types of resources, the bottom of the page displays a table sorted initially by CPU usage. You can change it to sort by memory usage or recent activity. To see more about one of the rows in the table, you can select the checkbox next to that row and then choose **Actions** and choose one of the options in the **Actions** menu.

Using CloudWatch Logs Insights to view Container Insights data

Container Insights collects metrics by using performance log events with using [embedded metric format](#). The logs are stored in CloudWatch Logs. CloudWatch generates several metrics automatically from the logs which you can view in the CloudWatch console. You can also do a deeper analysis of the performance data that is collected by using CloudWatch Logs Insights queries.

For more information about CloudWatch Logs Insights, see [Analyze Log Data with CloudWatch Logs Insights](#). For more information about the log fields you can use in queries, see [Container Insights performance log events for Amazon EKS and Kubernetes](#).

To use CloudWatch Logs Insights to query your container metric data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Insights**.

Near the top of the screen is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.

3. In the box above the query editor, select one of the Container Insights log groups to query. For the following example queries to work, the log group name must end with **performance**.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in **Discovered fields** in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not only the events displayed in the table.

4. In the query editor, replace the default query with the following query and choose **Run query**.

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

This query shows a list of nodes, sorted by average node CPU utilization.

5. To try another example, replace that query with another query and choose **Run query**. More sample queries are listed later on this page.

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by
PodName
| SORT avg_number_of_container_restarts DESC
```

This query displays a list of your pods, sorted by average number of container restarts.

6. If you want to try another query, you can use include fields in the list at the right of the screen. For more information about query syntax, see [CloudWatch Logs Insights Query Syntax](#).

To see lists of your resources

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Resources**.
3. The default view is a list of your resources being monitored by Container Insights, and alarms that you have set on these resources. To see a visual map of the resources, choose **Map view**.
4. From the map view, you can pause your pointer over any resource in the map to see basic metrics about that resource. You can choose any resource to see more detailed graphs about the resource.

Use case: Seeing task-level metrics in Amazon ECS containers

The following example illustrates how to use CloudWatch Logs Insights to dive deeper into your Container Insights logs. For more examples, see the blog [Introducing Amazon CloudWatch Container Insights for Amazon ECS](#).

Container Insights does not automatically generate metrics at the Task level of granularity. The following query displays task-level metrics for CPU and memory usage.

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName
```

```
| sort Mem, CPU desc
```

Other sample queries for Container Insights

List of your pods, sorted by average number of container restarts

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName  
| SORT avg_number_of_container_restarts DESC
```

Pods requested vs. pods running

```
fields @timestamp, @message  
| sort @timestamp desc  
| filter Type="Pod"  
| stats min(pod_number_of_containers) as requested,  
min(pod_number_of_running_containers) as running, ceil(avg(pod_number_of_containers-  
pod_number_of_running_containers)) as pods_missing by kubernetes.pod_name  
| sort pods_missing desc
```

Count of cluster node failures

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures  
| filter Type="Cluster"  
| sort @timestamp desc
```

Application log errors by container name

```
stats count() as countoferrors by kubernetes.container_name  
| filter stream="stderr"  
| sort countoferrors desc
```

Metrics collected by Container Insights

Container Insights collects one set of metrics for Amazon ECS and AWS Fargate on Amazon ECS, and a different set for Amazon EKS, AWS Fargate on Amazon EKS, and Kubernetes.

Metrics are not visible until the container tasks have been running for some time.

Topics

- [Amazon ECS Container Insights metrics](#)

- [Amazon EKS and Kubernetes Container Insights metrics](#)

Amazon ECS Container Insights metrics

The following table lists the metrics and dimensions that Container Insights collects for Amazon ECS. These metrics are in the ECS/ContainerInsights namespace. For more information, see [Metrics](#).

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights](#).

The following metrics are available when you complete the steps in [Setting up Container Insights on Amazon ECS for cluster- and service-level metrics](#)

Metric name	Dimensions	Description
ContainerInstanceCount	ClusterName	<p>The number of EC2 instances running the Amazon ECS agent that are registered with a cluster.</p> <p>This metric is collected only for container instances that are running Amazon ECS tasks in the cluster. It is not collected for empty container instances that do not have any Amazon ECS tasks.</p> <p>Unit: Count</p>
CpuUtilized	TaskDefinitionFamily , ClusterName	The CPU units used by tasks in the resource that is specified by the

Metric name	Dimensions	Description
	ServiceName , ClusterName ClusterName	dimension set that you're using. This metric is collected only for tasks that have a defined CPU reservation in their task definition. Unit: None
CpuReserved	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The CPU units reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected only for tasks that have a defined CPU reservation in their task definition. Unit: None
DeploymentCount	ServiceName , ClusterName	The number of deployments in an Amazon ECS service. Unit: Count
DesiredTaskCount	ServiceName , ClusterName	The desired number of tasks for an Amazon ECS service. Unit: Count

Metric name	Dimensions	Description
EBSFilesystemSize	VolumeName , TaskDefinitionFamily ,ClusterName TaskDefinitionFamily ,ClusterName ServiceName , ClusterName	<p>The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is allocated to the resources specified by the dimensions you're using.</p> <p>This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later.</p> <p>Unit: Gigabytes (GB)</p>

Metric name	Dimensions	Description
EBSFilesystemUtilized	VolumeName , TaskDefinitionFamily , ClusterName TaskDefinitionFamily , ClusterName ServiceName , ClusterName	<p>The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is being used by the resources specified by the dimensions that you're using.</p> <p>This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later.</p> <p>For tasks run on Fargate, Fargate reserves space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this additional storage using tools like <code>df</code>.</p> <p>Unit: Gigabytes (GB)</p>

Metric name	Dimensions	Description
EphemeralStorageReserved 1	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The number of bytes reserved from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task.</p> <p>This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later.</p> <p>Unit: Gigabytes (GB)</p>


Metric name	Dimensions	Description
EphemeralStorageUtilized 1	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The number of bytes used from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task.</p> <p>This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later.</p> <p>Unit: Gigabytes (GB)</p>

Metric name	Dimensions	Description
MemoryUtilized	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The memory being used by tasks in the resource that is specified by the dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined memory reservation in their task definition.</p> <p>Unit: Megabytes</p>
MemoryReserved	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using.</p> <p>This metric is collected only for tasks that have a defined memory reservation in their task definition.</p> <p>Unit: Megabytes</p>

Metric name	Dimensions	Description
NetworkRxBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The number of bytes received by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.</p> <p>This metric is available only for containers in tasks using the <code>awsvpc</code> or <code>bridge</code> network modes.</p> <p>Unit: Bytes/Second</p>
NetworkTxBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	<p>The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime.</p> <p>This metric is available only for containers in tasks using the <code>awsvpc</code> or <code>bridge</code> network modes.</p> <p>Unit: Bytes/Second</p>

Metric name	Dimensions	Description
PendingTaskCount	ServiceName , ClusterName	The number of tasks currently in the PENDING state. Unit: Count
RunningTaskCount	ServiceName , ClusterName	The number of tasks currently in the RUNNING state. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count
StorageReadBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The number of bytes read from storage on the instance in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime. Unit: Bytes

Metric name	Dimensions	Description
StorageWriteBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The number of bytes written to storage in the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. Unit: Bytes
TaskCount	ClusterName	The number of tasks running in the cluster. Unit: Count
TaskSetCount	ServiceName , ClusterName	The number of task sets in the service. Unit: Count

 **Note**

The `EphemeralStorageReserved` and `EphemeralStorageUtilized` metrics are only available for tasks that run on Fargate Linux platform version 1.4.0 or later. Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as `df`.

The following metrics are available when you complete the steps in [Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS](#)

Metric name	Dimensions	Description
instance_cpu_limit	ClusterName	The maximum number of CPU

Metric name	Dimensions	Description
		<p>units that can be assigned to a single EC2 Instance in the cluster.</p> <p>Unit: None</p>
instance_cpu_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	<p>The percentage of CPU currently being reserved on a single EC2 instance in the cluster.</p> <p>Unit: Percent</p>
instance_cpu_usage_total	ClusterName	<p>The number of CPU units being used on a Single EC2 instance in the cluster.</p> <p>Unit: None</p>
instance_cpu_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	<p>The total percentage of CPU units being used on a single EC2 instance in the cluster.</p> <p>Unit: Percent</p>

Metric name	Dimensions	Description
instance_filesystem_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	The total percentage of file system capacity being used on a single EC2 instance in the cluster. Unit: Percent
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 Instance in this cluster. Unit: Bytes
instance_memory_reserved_capacity	ClusterName InstanceId , ContainerInstanceId , ClusterName	The percentage of Memory currently being reserved on a single EC2 Instance in the cluster. Unit: Percent
instance_memory_utilization	ClusterName InstanceId , ContainerInstanceId , ClusterName	The total percentage of memory being used on a single EC2 Instance in the cluster. Unit: Percent

Metric name	Dimensions	Description
instance_memory_working_set	ClusterName	The amount of memory, in bytes, being used on a single EC2 Instance in the cluster. Unit: Bytes
instance_network_total_bytes	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 Instance in the cluster. Unit: Bytes/second
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 Instance in the cluster. Unit: Count

Amazon EKS and Kubernetes Container Insights metrics

The following tables list the metrics and dimensions that Container Insights collects for Amazon EKS and Kubernetes. These metrics are in the `ContainerInsights` namespace. For more information, see [Metrics](#).

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights](#).

If you are using version 1.5.0 or later of the Amazon EKS add-on or version 1.300035.0 of the CloudWatch agent, most metrics listed in the following table are collected for both Linux and

Windows nodes. See the **Metric Name** column of the table to see which metrics are not collected for Windows.


With the original version of Container Insights, the metrics are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Note


On Windows, network metrics such as `pod_network_rx_bytes` and `pod_network_tx_bytes` are not collected for host process containers.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<code>cluster_failed_node_count</code>	ClusterName		The number of failed worker nodes in the cluster. A node is considered failed if it is suffering from any <i>node conditions</i> . For more information, see Conditions in the Kubernetes documentation.
<code>cluster_node_count</code>	ClusterName		The total number of worker nodes in the cluster.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
namespace_number_of_running_pods	Namespace ClusterName ClusterName		The number of pods running per namespace in the resource that is specified by the dimensions that you're using.
node_cpu_limit	ClusterName	ClusterName , InstanceId , NodeName	The maximum number of CPU units that can be assigned to a single node in this cluster.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
node_cpu_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName		<p>The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker.</p> <p>Formula: $\text{node_cpu_request} / \text{node_cpu_limit}$</p> <div data-bbox="1187 957 1511 1759" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p> Note</p> <p>node_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.
node_cpu_usage_total	ClusterName	ClusterName , InstanceId , NodeName	The number of CPU units being used on the nodes in the cluster.
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName		The total percentage of CPU units being used on the nodes in the cluster. Formula: $\text{node_cpu_usage_total} / \text{node_cpu_limit}$

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
node_file_system_utilization	NodeName, ClusterName , InstanceId ClusterName		<p>The total percentage of file system capacity being used on nodes in the cluster.</p> <p>Formula: $\text{node_file_system_usage} / \text{node_file_system_capacity}$</p> <div data-bbox="1187 1005 1507 1858" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; background-color: #E6F2FF;"> <p> Note</p> <p>node_file_system_usage and node_file_system_capacity are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			performance log events for Amazon EKS and Kubernetes.
node_memory_limit	ClusterName	ClusterName , InstanceId , NodeName	The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.
node_file_system_inodes This metric is available only with Container Insights with enhanced observability for Amazon EKS. It is not available on Windows.		ClusterName ClusterName , InstanceId , NodeName	The total number of inodes (used and unused) on a node.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_file_system_inodes_free</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS. It is not available on Windows.</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The number of unused inodes on a node.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName		<p>The percentage of memory currently being used on the nodes in the cluster.</p> <p>Formula: $\text{node_memory_request} / \text{node_memory_limit}$</p> <div data-bbox="1187 909 1507 1759" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; background-color: #E6F2FF;"> <p>Note</p> <p>node_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName		<div data-bbox="1187 443 1507 575" style="border: 1px solid #0070C0; border-radius: 10px; padding: 5px; text-align: center;"> EKS and Kubernetes. </div> <p>The percentage of memory currently being used by the node or nodes. It is the percentage of node memory usage divided by the node memory limitation.</p> <p>Formula: <code>node_memory_working_set / node_memory_limit</code> .</p>
node_memory_working_set	ClusterName	ClusterName , InstanceId , NodeName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName		<p>The total number of bytes per second transmitted and received over the network per node in a cluster.</p> <p>Formula: <code>node_network_rx_bytes + node_network_tx_bytes</code></p> <div data-bbox="1187 1003 1507 1856" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p><code>node_network_rx_bytes</code> and <code>node_network_tx_bytes</code> are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			performance log events for Amazon EKS and Kubernetes.
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName		The number of running containers per node in a cluster.
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName		The number of running pods per node in a cluster.
node_status_allocatable_pods This metric is available only with Container Insights with enhanced observability for Amazon EKS		ClusterName ClusterName , InstanceId , NodeName	The number of pods that can be assigned to a node based on its allocatable resources, which is defined as the remainder of a node's capacity after accounting for system daemons reservations and hard eviction thresholds.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_status_capacity_pods</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The number of pods that can be assigned to a node based on its capacity.</p>
<p>node_status_condition_ready</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>Indicates whether the node status condition Ready is true.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_status_condition_memory_pressure</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>Indicates whether the node status condition MemoryPressure is true.</p>
<p>node_status_condition_pid_pressure</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>Indicates whether the node status condition PIDPressure is true.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_status_condition_disk_pressure</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>Indicates whether the node status condition <code>OutOfDisk</code> is true.</p>
<p>node_status_condition_unknown</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>Indicates whether any of the node status conditions are <code>Unknown</code>.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_interface_network_dropped</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The number of packets which were received and subsequently dropped by a network interface on the node.</p>
<p>node_interface_network_dropped</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The number of packets which were due to be transmitted but were dropped by a network interface on the node.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>node_disk_io_io_services_total</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS. It is not available on Windows.</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The total number of bytes transferred by all I/O operations on the node.</p>
<p>node_disk_io_io_serviced_total</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS. It is not available on Windows.</p>		<p>ClusterName</p> <p>ClusterName , InstanceId , NodeName</p>	<p>The total number of I/O operations on the node.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_cpu_reserved_capacity	PodName, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName ClusterName, Namespace, Service	<p>The CPU capacity that is reserved per pod in a cluster.</p> <p>Formula: $\text{pod_cpu_request} / \text{node_cpu_limit}$</p> <div data-bbox="1187 814 1507 1747" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_cpu_utilization	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The percentage of CPU units being used by pods.</p> <p>Formula: $\text{pod_cpu_usage_total} / \text{node_cpu_limit}$</p> <div data-bbox="1187 814 1507 1801" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_cpu_usage_total is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_cpu_utilization_over_pod_limit	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The percentage of CPU units being used by pods relative to the pod limit.</p> <p>Formula: $\text{pod_cpu_usage_total} / \text{pod_cpu_limit}$</p> <div data-bbox="1187 863 1507 1808" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_cpu_usage_total and pod_cpu_limit are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_memory_reserved_capacity	PodName, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName ClusterName, Namespace, Service	<p>The percentage of memory that is reserved for pods.</p> <p>Formula: $\text{pod_memory_request} / \text{node_memory_limit}$</p> <div data-bbox="1187 863 1507 1843" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_memory_utilization	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The percentage of memory currently being used by the pod or pods.</p> <p>Formula: $\text{pod_memory_working_set} / \text{node_memory_limit}$</p> <div data-bbox="1187 957 1511 1808" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p>pod_memory_working_set is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_memory_utilization_over_pod_limit	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The percentage of memory that is being used by pods relative to the pod limit. If any containers in the pod don't have a memory limit defined, this metric doesn't appear.</p> <p>Formula: $\text{pod_memory_working_set} / \text{pod_memory_limit}$</p> <div data-bbox="1187 1199 1507 1856" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; background-color: #E6F2FF;"> <p>Note</p> <p>pod_memory_working_set is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant</p> </div>


Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_network_rx_bytes	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The number of bytes per second being received over the network by the pod.</p> <p>Formula: <code>sum(pod_interface_network_rx_bytes)</code></p> <div data-bbox="1187 909 1507 1808" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p><code>pod_interface_network_rx_bytes</code> is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
pod_network_tx_bytes	PodName, Namespace, ClusterName Namespace, ClusterName Service, Namespace, ClusterName ClusterName	ClusterName, Namespace, PodName, FullPodName	<p>The number of bytes per second being transmitted over the network by the pod.</p> <p>Formula: $\text{sum}(\text{pod_interface_network_tx_bytes})$</p> <div data-bbox="1187 909 1507 1808" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_interface_network_tx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_cpu_request</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>EKS and Kubernetes.</p> <p>The CPU requests for the pod.</p> <p>Formula: <code>sum(container_cpu_request)</code></p> <div data-bbox="1187 913 1507 1852" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_memory_request</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>The memory requests for the pod.</p> <p>Formula: <code>sum(container_memory_request)</code></p> <div data-bbox="1187 766 1507 1749" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p> Note</p> <p>pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_cpu_limit</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>The CPU limit defined for the containers in the pod. If any containers in the pod don't have a CPU limit defined, this metric doesn't appear.</p> <p>Formula: $\text{sum}(\text{container_cpu_limit})$</p> <div data-bbox="1187 1052 1508 1854" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px;"> <p>Note</p> <p>pod_cpu_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_memory_limit</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>The memory limit defined for the containers in the pod. If any containers in the pod don't have a memory limit defined, this metric doesn't appear.</p> <p>Formula: <code>sum(container_memory_limit)</code></p> <div data-bbox="1187 1052 1507 1854" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p>pod_cpu_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_statuses_failed</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>EKS and Kubernetes.</p> <p>Indicates that all containers in the pod have terminated, and at least one container has terminated with a non-zero status or was terminated by the system.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_statuses_ready</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that all containers in the pod are ready, having reached the condition of Container Ready .</p>
<p>pod_statuses_running</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that all containers in the pod are running.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_statuses_scheduled</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that the pod has been scheduled to a node.</p>
<p>pod_statuses_unknown</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that status of the pod can't be obtained.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_statuses_pending</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that the pod has been accepted by the cluster but one or more of the containers has not become ready yet.</p>
<p>pod_statuses_succeeded</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Indicates that all containers in the pod have successfully terminated and will not be restarted.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_number_of_containers</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers defined in the pod specification.</p>
<p>pod_number_of_running_containers</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are currently in the Running state.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_container_status_terminated</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are in the Terminated state.</p>
<p>pod_container_status_running</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are in the Running state.</p>


Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_container_status_waiting</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are in the Waiting state.</p>
<p>pod_container_status_waiting_reason_crash_loop_back_off</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are pending because of a CrashLoopBackOff error, where a container repeatedly fails to start.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_container_status_waiting_reason_create_container_config_error</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are pending with the reason CreateContainerConfigError . This is because of an error while creating the container configuration.</p>
<p>pod_container_status_waiting_reason_create_container_error</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are pending with the reason CreateContainerError because of an error while creating the container.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_container_status_waiting_reason_image_pull_error</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are pending because of ErrImagePull , ImagePullBackOff , or InvalidImageName . These situations are because of an error while pulling the container image.</p>
<p>pod_container_status_waiting_reason_oom_killer</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are in the Terminated state because of running out of memory (OOM killed).</p>


Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_container_status_waiting_reason_start_error</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>Reports the number of containers in the pod which are pending with the reason being StartError because of an error while starting the container.</p>
<p>pod_interface_network_rx_dropped</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>The number of packets which were received and subsequently dropped a network interface for the pod.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>pod_interface_network_tx_dropped</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName</p> <p>Namespace , ClusterName , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p>	<p>The number of packets which were due to be transmitted but were dropped for the pod.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>container_cpu_utilization</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>PodName,</code> <code>Namespace ,</code> <code>ClusterName ,</code> <code>ContainerName</code></p> <p><code>PodName,</code> <code>Namespace ,</code> <code>ClusterName ,</code> <code>ContainerName ,</code> <code>FullPodName</code></p>	<p>The percentage of CPU units being used by the container.</p> <p>Formula: <code>container_cpu_usage_total / node_cpu_limit</code></p> <div data-bbox="1187 863 1507 1843" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px;"> <p> Note</p> <p><code>container_cpu_utilization</code> is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>container_cpu_utilization_over_container_limit</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName , ContainerName</p> <p>PodName, Namespace , ClusterName , ContainerName , FullPodName</p>	<p>The percentage of CPU units being used by the container relative to the container limit. If the container doesn't have a CPU limit defined, this metric doesn't appear.</p> <p>Formula: <code>container_cpu_usage_total / container_cpu_limit</code></p> <div data-bbox="1187 1199 1507 1858" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p><code>container_cpu_utilization_over_container_limit</code> is not reported directly as a metric, but is a field in performance log events.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>container_memory_utilization</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>PodName, Namespace , ClusterName , ContainerName</p> <p>PodName, Namespace , ClusterName , ContainerName , FullPodName</p>	<p>The percentage of memory units being used by the container .</p> <p>Formula: $\text{container_memory_working_set} / \text{node_memory_limit}$</p> <div data-bbox="1187 957 1511 1860" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p> Note</p> <p><code>container_memory_utilization</code> is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			EKS and Kubernetes.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>container_memory_utilization_over_container_limit</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>PodName</code>, <code>Namespace</code> , <code>ClusterName</code> , <code>ContainerName</code></p> <p><code>PodName</code>, <code>Namespace</code> , <code>ClusterName</code> , <code>ContainerName</code> , <code>FullPodName</code></p>	<p>The percentage of memory units being used by the container relative to the container limit. If the container doesn't have a memory limit defined, this metric doesn't appear.</p> <p>Formula: <code>container_memory_working_set / container_memory_limit</code></p> <div data-bbox="1187 1194 1507 1858" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p><code>container_memory_utilization_over_container_limit</code> is not reported directly as a metric, but is a field in performance log events.</p> </div>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
			<p>For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.</p>
<p><code>container_memory_failures_total</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS. It is not available on Windows.</p>		<p><code>ClusterName</code></p> <p><code>PodName</code>, <code>Namespace</code> , <code>ClusterName</code> , <code>ContainerName</code></p> <p><code>PodName</code>, <code>Namespace</code> , <code>ClusterName</code> , <code>ContainerName</code> , <code>FullPodName</code></p>	<p>The number of memory allocation failures experienced by the container.</p>
<p><code>pod_number_of_container_restarts</code></p>	<p><code>PodName</code>, <code>Namespace</code> , <code>ClusterName</code></p>		<p>The total number of container restarts in a pod.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
service_number_of_running_pods	Service, Namespace, ClusterName ClusterName		The number of pods running the service or services in the cluster.
replicas_desired This metric is available only with Container Insights with enhanced observability for Amazon EKS		ClusterName PodName, Namespace, ClusterName	The number of pods desired for a workload as defined in the workload specification.
replicas_ready This metric is available only with Container Insights with enhanced observability for Amazon EKS		ClusterName PodName, Namespace, ClusterName	The number of pods for a workload that have reached the ready status.

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>status_replicas_available</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>PodName,</code> <code>Namespace ,</code> <code>ClusterName</code></p>	<p>The number of pods for a workload which are available. A pod is available when it has been ready for the <code>minReadySeconds</code> defined in the workload specification.</p>
<p><code>status_replicas_unavailable</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>PodName,</code> <code>Namespace ,</code> <code>ClusterName</code></p>	<p>The number of pods for a workload which are unavailable. A pod is available when it has been ready for the <code>minReadySeconds</code> defined in the workload specification. Pods are unavailable if they have not met this criterion.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>apiserver_storage_objects</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , resource</code></p>	<p>The number of objects stored in etcd at the time of the last check.</p>
<p><code>apiserver_request_total</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , code, verb</code></p>	<p>The total number of API requests to the Kubernetes API server.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>apiserver_request_duration_seconds</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , verb</code></p>	<p>Response latency for API requests to the Kubernetes API server.</p>
<p><code>apiserver_admission_controller_admission_duration_seconds</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , operation</code></p>	<p>Admission controller latency in seconds. An admission controller is code which intercepts requests to the Kubernetes API server.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>rest_client_request_duration_seconds</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , operation</code></p>	<p>Response latency experienced by clients calling the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.</p>
<p><code>rest_client_requests_total</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , code, method</code></p>	<p>The total number of API requests to the Kubernetes API server made by clients. This metric is experimental and may change in future releases of Kubernetes.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>etcd_request_duration_seconds</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , operation</p>	<p>Response latency of API calls to Etcd. This metric is experimental and may change in future releases of Kubernetes.</p>
<p>apiserver_storage_size_bytes</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , endpoint</p>	<p>Size of the storage database file physically allocated in bytes. This metric is experimental and may change in future releases of Kubernetes.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>apiserver_longrunning_requests</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , resource</p>	<p>The number of active long-running requests to the Kubernetes API server.</p>
<p>apiserver_current_inflight_requests</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , request_kind</p>	<p>The number of requests that are being processed by Kubernetes API server.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>apiserver_admission_webhook_admission_duration_seconds</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , name</p>	<p>Admission webhook latency in seconds. Admission webhooks are HTTP callbacks that receive admission requests and do something with them.</p>
<p>apiserver_admission_step_admission_duration_seconds</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , operation</p>	<p>Admission sub-step latency in seconds.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>apiserver_request_deprecated_apis</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , group</code></p>	<p>Number of requests to deprecated APIs on the Kubernetes API server.</p>
<p><code>apiserver_request_total_5XX</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , code, verb</code></p>	<p>Number of requests to the Kubernetes API server which were responded to with a 5XX HTTP response code.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p><code>apiserver_storage_list_duration_seconds</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , resource</code></p>	<p>Response latency of listing objects from Etcd. This metric is experimental and may change in future releases of Kubernetes.</p>
<p><code>apiserver_current_inqueue_requests</code></p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p><code>ClusterName</code></p> <p><code>ClusterName , request_kind</code></p>	<p>The number queued requests queued by the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.</p>

Metric name	Dimensions with any version of Container Insights	Additional dimensions with Container Insights with enhanced observability for Amazon EKS	Description
<p>apiserver_flowcontrol_rejected_requests_total</p> <p>This metric is available only with Container Insights with enhanced observability for Amazon EKS</p>		<p>ClusterName</p> <p>ClusterName , reason</p>	<p>Number of requests rejected by API Priority and Fairness subsystem. This metric is experimental and may change in future releases of Kubernetes.</p>

NVIDIA GPU metrics

Beginning with version 1.300034.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects NVIDIA GPU metrics from EKS workloads by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version v1.3.0-eksbuild.1 or later. For more information, see [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart](#). These NVIDIA GPU metrics that are collected are listed in the table in this section.

For Container Insights to collect NVIDIA GPU metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version v1.3.0-eksbuild.1 or later.
- [The NVIDIA device plugin for Kubernetes](#) must be installed in the cluster.
- [The NVIDIA container toolkit](#) must be installed on the nodes of the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

You can opt out of collecting NVIDIA GPU metrics by setting the `accelerated_compute_metrics` option in the beginn CloudWatch agent configuration file to `false`. For more information and an example opt-out configuration, see [\(Optional\) Additional configuration](#).

Metric name	Dimensions	Description
<code>container_gpu_memory_total</code>	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the container.
<code>container_gpu_memory_used</code>	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the container.
<code>container_gpu_memory_utilization</code>	ClusterName ClusterName , Namespace , PodName, ContainerName	The percentage of frame buffer used of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	
container _gpu_powe r_draw	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The power usage in watts of the GPU(s) allocated to the container.
container _gpu_temp erature	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
container_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the GPU(s) allocated to the container.
node_gpu_memory_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the node.
node_gpu_memory_used	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the node.

Metric name	Dimensions	Description
node_gpu_memory_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The percentage of frame buffer used on the GPU(s) allocated to the node.
node_gpu_power_draw	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The power usage in watts of the GPU(s) allocated to the node.
node_gpu_temperature	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the node.
node_gpu_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The percentage utilization of the GPU(s) allocated to the node.

Metric name	Dimensions	Description
pod_gpu_memory_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the pod.
pod_gpu_memory_used	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_memory_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The percentage of frame buffer used of the GPU(s) allocated to the pod.
pod_gpu_power_draw	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The power usage in watts of the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_temperature	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the pod.
pod_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the GPU(s) allocated to the pod.

AWS Neuron metrics for AWS Trainium and AWS Inferentia

Beginning with version 1.300036.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects accelerated computing metrics from AWS Trainium and AWS Inferentia accelerators by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version v1.5.0-eksbuild.1 or later. For more information about the add-on, see [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS](#)

[add-on or the Helm chart](#). For more information about AWS Trainium, see [AWS Trainium](#). For more information about AWS Inferentia, see [AWS Inferentia](#).

For Container Insights to collect AWS Neuron metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.5.0-eksbuild.1` or later.
- The [Neuron driver](#) must be installed on the nodes of the cluster.
- The [Neuron device plugin](#) must be installed on the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

The metrics that are collected are listed in the table in this section. The metrics are collected for AWS Trainium, AWS Inferentia, and AWS Inferentia2.

The CloudWatch agent collects these metrics from the [Neuron monitor](#) and does the necessary Kubernetes resource correlation to deliver metrics at the pod and container levels

Metric name	Dimensions	Description
<code>container_neuroncore_utilization</code>	<p>ClusterName</p> <p>ClusterName , Namespace , PodName, ContainerName</p> <p>ClusterName , Namespace , PodName, FullPodName , ContainerName</p> <p>ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore</p>	<p>NeuronCore utilization, during the captured period of the NeuronCore allocated to the container.</p> <p>Unit: Percent</p>
<code>container_neuroncore_memory_usage_constants</code>	<p>ClusterName</p> <p>ClusterName , Namespace , PodName, ContainerName</p>	<p>The amount of device memory used for constants during training by the NeuronCore that is allocated to the container (or weights during inference).</p>

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDev ice , NeuronCore	Unit: Bytes
container _neuronco re_memory _usage_mo del_code	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDev ice , NeuronCore	The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the container. Unit: Bytes
container _neuronco re_memory _usage_mo del_share d_scratch pad	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDev ice , NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the container. This memory region is reserved for the models. Unit: Bytes

Metric name	Dimensions	Description
container_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the container. Unit: Bytes
container_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for tensors by the NeuronCore allocated to the container. Unit: Bytes

Metric name	Dimensions	Description
container_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the container. Unit: Bytes
container_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

Metric name	Dimensions	Description
pod_neuroncore_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The NeuronCore utilization during the captured period of the NeuronCore allocated to the pod. Unit: Percent
pod_neuroncore_memory_usage_constants	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The amount of device memory used for constants during training by the NeuronCore that is allocated to the pod (or weights during inference). Unit: Bytes

Metric name	Dimensions	Description
<p>pod_neuro ncore_mem ory_usage _model_co de</p>	<p>ClusterName</p> <p>ClusterName , Namespace</p> <p>ClusterName , Namespace , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p> <p>ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore</p>	<p>The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the pod.</p> <p>Unit: Bytes</p>
<p>pod_neuro ncore_mem ory_usage _model_sh ared_scr atchpad</p>	<p>ClusterName</p> <p>ClusterName , Namespace</p> <p>ClusterName , Namespace , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p> <p>ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore</p>	<p>The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the pod. This memory region is reserved for the models.</p> <p>Unit: Bytes</p>

Metric name	Dimensions	Description
<p>pod_neuro ncore_mem ory_usage _runtime_ memory</p>	<p>ClusterName</p> <p>ClusterName , Namespace</p> <p>ClusterName , Namespace , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p> <p>ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore</p>	<p>The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the pod.</p> <p>Unit: Bytes</p>
<p>pod_neuro ncore_mem ory_usage _tensors</p>	<p>ClusterName</p> <p>ClusterName , Namespace</p> <p>ClusterName , Namespace , Service</p> <p>ClusterName , Namespace , PodName, FullPodName</p> <p>ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore</p>	<p>The amount of device memory used for tensors by the NeuronCore allocated to the pod.</p> <p>Unit: Bytes</p>

Metric name	Dimensions	Description
pod_neuro_ncore_memory_usage_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the pod. Unit: Bytes
pod_neuro_ndevice_hw_ecc_events_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device allocated to a pod. Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_utilization	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The NeuronCore utilization during the captured period of the NeuronCore allocated to the node. Unit: Percent
node_neuroncore_memory_usage_constants	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for constants during training by the NeuronCore that is allocated to the node (or weights during inference). Unit: Bytes
node_neuroncore_memory_usage_model_code	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for models' executable code by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the node. This is a memory region reserved for the models. Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_tensors	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The amount of device memory used for tensors by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceType , InstanceId , NodeName, NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuron_execution_errors_total	ClusterName ClusterName , InstanceId , NodeName	The total number of execution errors on the node. This is calculated by the CloudWatch agent by aggregating the errors of the following types: generic, numerical , transient , model, runtime, and hardware Unit: Count

Metric name	Dimensions	Description
node_neuron_device_runtime_memory_used_bytes	ClusterName ClusterName , InstanceId , NodeName	The total Neuron device memory usage in bytes on the node. Unit: Bytes
node_neuron_execution_latency	ClusterName ClusterName , InstanceId , NodeName	In seconds, the latency for an execution on the node as measured by the Neuron runtime. Unit: Seconds
node_neuron_device_hw_ecc_events_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , NodeName, NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

AWS Elastic Fabric Adapter (EFA) metrics

Beginning with version 1.300037.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects AWS Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters on Linux instances. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version v1.5.2-eksbuild.1 or later. For more information about the add-on, see [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart](#). For more information about AWS Elastic Fabric Adapter, see [Elastic Fabric Adapter](#).

For Container Insights to collect AWS Elastic Fabric adapter metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version v1.5.2-eksbuild.1 or later.

- The EFA device plugin must be installed on the cluster. For more information, see [aws-efa-k8s-device-plugin](#) on GitHub.

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description
container_efa_rx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	The number of bytes per second received by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_tx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	The number of bytes per second transmitted by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_rx_dropped	ClusterName ClusterName , Namespace , PodName, ContainerName	The number of packets that were received and then dropped by the EFA device(s) allocated to the container. Unit: Count/Second

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	
container_efa_rdma_read_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_rdma_write_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second

Metric name	Dimensions	Description
container_efa_rdma_write_recv_bytes	<p>ClusterName</p> <p>ClusterName , Namespace , PodName, ContainerName</p> <p>ClusterName , Namespace , PodName, FullPodName , ContainerName</p> <p>ClusterName , Namespace , PodName, FullPodName , ContainerName , EfaDevice</p>	<p>The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the container.</p> <p>Unit: Bytes/Second</p>
pod_efa_rx_bytes	<p>ClusterName</p> <p>ClusterName , Namespace</p> <p>ClusterName , Namespace , Service</p> <p>ClusterName , Namespace , PodName</p> <p>ClusterName , Namespace , PodName, FullPodName</p> <p>ClusterName , Namespace , PodName, FullPodName , EfaDevice</p>	<p>The number of bytes per second received by the EFA device(s) allocated to the pod.</p> <p>Unit: Bytes/Second</p>

Metric name	Dimensions	Description
pod_efa_tx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	The number of bytes per second transmitted by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rx_dropped	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	The number of packets that were received and then dropped by the EFA device(s) allocated to the pod. Unit: Count/Second

Metric name	Dimensions	Description
pod_efa_rdma_read_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rdma_write_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second

Metric name	Dimensions	Description
pod_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , EfaDevice	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of bytes per second received by the EFA device(s) allocated to the node. Unit: Bytes/Second
node_efa_tx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of bytes per second transmitted by the EFA device(s) allocated to the node. Unit: Bytes/Second

Metric name	Dimensions	Description
node_efa_rx_dropped	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of packets that were received and then dropped by the EFA device(s) allocated to the node. Unit: Count/Second
node_efa_rdma_read_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the node. Unit: Bytes/Second
pod_efa_rdma_write_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rdma_write_recv_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, EfaDevice	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the node. Unit: Bytes/Second

Container Insights performance log reference

This section includes reference information about how Container Insights uses performance log events to collect metrics. When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

Topics

- [Container Insights performance log events for Amazon ECS](#)
- [Container Insights performance log events for Amazon EKS and Kubernetes](#)
- [Relevant fields in performance log events for Amazon EKS and Kubernetes](#)

Container Insights performance log events for Amazon ECS

The following are examples of the performance log events that Container Insights collects from Amazon ECS.

These logs are in CloudWatch Logs, in a log group named `/aws/ecs/containerinsights/CLUSTER_NAME/performance`. Within that log group, each container instance will have a log stream named `AgentTelemetry-CONTAINER_INSTANCE_ID`.

You can query these logs using queries such as `{ $.Type = "Container" }` to view all container log events.

Type: Container

```
{
  "Version": "0",
  "Type": "Container",
  "ContainerName": "sleep",
  "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
  "TaskDefinitionFamily": "sleep360",
  "TaskDefinitionRevision": "1",
  "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
  "EC2InstanceId": "i-0c470579dbcd2f3",
  "ClusterName": "MyCluster",
  "Image": "busybox",
  "ContainerKnownStatus": "RUNNING",
  "Timestamp": 1623963900000,
  "CpuUtilized": 0.0,
  "CpuReserved": 10.0,
```



```
"MemoryUtilized":0,  
"MemoryReserved":10,  
"StorageReadBytes":0,  
"StorageWriteBytes":0,  
"NetworkRxBytes":0,  
"NetworkRxDropped":0,  
"NetworkRxErrors":0,  
"NetworkRxPackets":14,  
"NetworkTxBytes":0,  
"NetworkTxDropped":0,  
"NetworkTxErrors":0,  
"NetworkTxPackets":0  
}
```

Type: Task

```
{  
  "Version": "0",  
  "Type": "Task",  
  "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",  
  "TaskDefinitionFamily": "sleep360",  
  "TaskDefinitionRevision": "1",  
  "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",  
  "EC2InstanceId": "i-0c470579dbcdbd2f3",  
  "ClusterName": "MyCluster",  
  "AccountID": "637146863587",  
  "Region": "us-west-2",  
  "AvailabilityZone": "us-west-2b",  
  "KnownStatus": "RUNNING",  
  "LaunchType": "EC2",  
  "PullStartedAt": 1623963608201,  
  "PullStoppedAt": 1623963610065,  
  "CreatedAt": 1623963607094,  
  "StartedAt": 1623963610382,  
  "Timestamp": 1623963900000,  
  "CpuUtilized": 0.0,  
  "CpuReserved": 10.0,  
  "MemoryUtilized": 0,  
  "MemoryReserved": 10,  
  "StorageReadBytes": 0,  
  "StorageWriteBytes": 0,  
  "NetworkRxBytes": 0,  
  "NetworkRxDropped": 0,  
}
```

```
"NetworkRxErrors": 0,
"NetworkRxPackets": 14,
"NetworkTxBytes": 0,
"NetworkTxDropped": 0,
"NetworkTxErrors": 0,
"NetworkTxPackets": 0,
"EBSFilesystemUtilized": 10,
"EBSFilesystemSize": 20,
"CloudWatchMetrics": [
  {
    "Namespace": "ECS/ContainerInsights",
    "Metrics": [
      {
        "Name": "CpuUtilized",
        "Unit": "None"
      },
      {
        "Name": "CpuReserved",
        "Unit": "None"
      },
      {
        "Name": "MemoryUtilized",
        "Unit": "Megabytes"
      },
      {
        "Name": "MemoryReserved",
        "Unit": "Megabytes"
      },
      {
        "Name": "StorageReadBytes",
        "Unit": "Bytes/Second"
      },
      {
        "Name": "StorageWriteBytes",
        "Unit": "Bytes/Second"
      },
      {
        "Name": "NetworkRxBytes",
        "Unit": "Bytes/Second"
      },
      {
        "Name": "NetworkTxBytes",
        "Unit": "Bytes/Second"
      }
    ]
  }
]
```

```

        {
            "Name": "EBSFilesystemSize",
            "Unit": "Gigabytes"
        },
        {
            "Name": "EBSFilesystemUtilized",
            "Unit": "Gigabytes"
        }
    ],
    "Dimensions": [
        ["ClusterName"],
        [
            "ClusterName",
            "TaskDefinitionFamily"
        ]
    ]
}

```

Type: Service

```

{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "RunningTaskCount",

```

```

        "Unit": "Count"
      },
      {
        "Name": "PendingTaskCount",
        "Unit": "Count"
      },
      {
        "Name": "DeploymentCount",
        "Unit": "Count"
      },
      {
        "Name": "TaskSetCount",
        "Unit": "Count"
      }
    ],
    "Dimensions": [
      [
        "ServiceName",
        "ClusterName"
      ]
    ]
  }
]
}

```

Type: Volume

```

{
  "Version": "0",
  "Type": "Volume",
  "TaskDefinitionFamily": "myCITaskDef",
  "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
  "ClusterName": "myCICluster",
  "ServiceName": "myCIService",
  "VolumeId": "vol-1233436545ff708cb",
  "InstanceId": "i-0c470579dbcdbd2f3",
  "LaunchType": "EC2",
  "VolumeName": "MyVolumeName",
  "EBSFilesystemUtilized": 10,
  "EBSFilesystemSize": 20,
  "CloudWatchMetrics": [
    {
      "Namespace": "ECS/ContainerInsights",

```

```

    "Metrics": [
      {
        "Name": "EBSFilesystemSize",
        "Unit": "Gigabytes"
      },
      {
        "Name": "EBSFilesystemUtilized",
        "Unit": "Gigabytes"
      }
    ],
    "Dimensions": [
      ["ClusterName"],
      [
        "VolumeName",
        "TaskDefinitionFamily",
        "ClusterName"
      ],
      [
        "ServiceName",
        "ClusterName"
      ]
    ]
  }
]
}

```

Type: Cluster

```

{
  "Version": "0",
  "Type": "Cluster",
  "ClusterName": "myCICluster",
  "Timestamp": 1561587300000,
  "TaskCount": 5,
  "ContainerInstanceCount": 5,
  "ServiceCount": 2,
  "CloudWatchMetrics": [
    {
      "Namespace": "ECS/ContainerInsights",
      "Metrics": [
        {
          "Name": "TaskCount",
          "Unit": "Count"
        }
      ]
    }
  ]
}

```

```

    },
    {
      "Name": "ContainerInstanceCount",
      "Unit": "Count"
    },
    {
      "Name": "ServiceCount",
      "Unit": "Count"
    }
  ],
  "Dimensions": [
    [
      "ClusterName"
    ]
  ]
}
]
}

```

Container Insights performance log events for Amazon EKS and Kubernetes

The following are examples of the performance log events that Container Insights collects from Amazon EKS and Kubernetes clusters.

Type: Node

```

{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        }
      ]
    }
  ]
}

```

```
    },
    {
      "Unit": "Percent",
      "Name": "node_cpu_reserved_capacity"
    },
    {
      "Unit": "Percent",
      "Name": "node_memory_reserved_capacity"
    },
    {
      "Unit": "Count",
      "Name": "node_number_of_running_pods"
    },
    {
      "Unit": "Count",
      "Name": "node_number_of_running_containers"
    }
  ],
  "Dimensions": [
    [
      "NodeName",
      "InstanceId",
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
},
{
  "Metrics": [
    {
      "Unit": "Percent",
      "Name": "node_cpu_utilization"
    },
    {
      "Unit": "Percent",
      "Name": "node_memory_utilization"
    },
    {
      "Unit": "Bytes/Second",
      "Name": "node_network_total_bytes"
    },
    {
      "Unit": "Percent",
      "Name": "node_cpu_reserved_capacity"
    }
  ]
}
```

```
    },
    {
      "Unit": "Percent",
      "Name": "node_memory_reserved_capacity"
    },
    {
      "Unit": "Count",
      "Name": "node_number_of_running_pods"
    },
    {
      "Unit": "Count",
      "Name": "node_number_of_running_containers"
    },
    {
      "Name": "node_cpu_usage_total"
    },
    {
      "Name": "node_cpu_limit"
    },
    {
      "Unit": "Bytes",
      "Name": "node_memory_working_set"
    },
    {
      "Unit": "Bytes",
      "Name": "node_memory_limit"
    }
  ],
  "Dimensions": [
    [
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
}
],
"ClusterName": "myCICluster",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"Sources": [
  "cadvisor",
  "/proc",
  "pod",
```



```
    "calculated"
  ],
  "Timestamp": "1567096682364",
  "Type": "Node",
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_cpu_limit": 4000,
  "node_cpu_request": 1130,
  "node_cpu_reserved_capacity": 28.249999999999996,
  "node_cpu_usage_system": 33.794636630852764,
  "node_cpu_usage_total": 136.47852169244098,
  "node_cpu_usage_user": 71.67075111567326,
  "node_cpu_utilization": 3.4119630423110245,
  "node_memory_cache": 3103297536,
  "node_memory_failcnt": 0,
  "node_memory_hierarchical_pgfault": 0,
  "node_memory_hierarchical_pgmajfault": 0,
  "node_memory_limit": 16624865280,
  "node_memory_mapped_file": 406646784,
  "node_memory_max_usage": 4230746112,
  "node_memory_pgfault": 0,
  "node_memory_pgmajfault": 0,
  "node_memory_request": 1115684864,
  "node_memory_reserved_capacity": 6.7109407818311055,
  "node_memory_rss": 798146560,
  "node_memory_swap": 0,
  "node_memory_usage": 3901444096,
  "node_memory_utilization": 6.601302600149552,
  "node_memory_working_set": 1097457664,
  "node_network_rx_bytes": 35918.392817386324,
  "node_network_rx_dropped": 0,
  "node_network_rx_errors": 0,
  "node_network_rx_packets": 157.67565245448117,
  "node_network_total_bytes": 68264.20276554905,
  "node_network_tx_bytes": 32345.80994816272,
  "node_network_tx_dropped": 0,
  "node_network_tx_errors": 0,
  "node_network_tx_packets": 154.21455923431654,
  "node_number_of_running_containers": 16,
  "node_number_of_running_pods": 13
}
```

Type: NodeFS

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_filesystem_utilization"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097939726",
  "Type": "NodeFS",
  "Version": "0",
  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_filesystem_available": 17298395136,
```

```

"node_filesystem_capacity": 21462233088,
"node_filesystem_inodes": 10484720,
"node_filesystem_inodes_free": 10367158,
"node_filesystem_usage": 4163837952,
"node_filesystem_utilization": 19.400767547940255
}

```

Type: NodeDiskIO

```

{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeDiskIO",
  "Version": "0",
  "device": "/dev/nvme0n1",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_diskio_io_service_bytes_async": 9750.505814277016,
  "node_diskio_io_service_bytes_read": 0,
  "node_diskio_io_service_bytes_sync": 230.6174506688036,
  "node_diskio_io_service_bytes_total": 9981.123264945818,
  "node_diskio_io_service_bytes_write": 9981.123264945818,
  "node_diskio_io_serviced_async": 1.153087253344018,
  "node_diskio_io_serviced_read": 0,
  "node_diskio_io_serviced_sync": 0.03603397666700056,
  "node_diskio_io_serviced_total": 1.1891212300110185,
  "node_diskio_io_serviced_write": 1.1891212300110185
}

```

Type: NodeNet

```
{
```

```

"AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
"ClusterName": "myCICluster",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"Sources": [
  "cadvisor",
  "calculated"
],
"Timestamp": "1567096928131",
"Type": "NodeNet",
"Version": "0",
"interface": "eni972f6bfa9a0",
"kubernetes": {
  "host": "ip-192-168-75-26.us-west-2.compute.internal"
},
"node_interface_network_rx_bytes": 3163.008420864309,
"node_interface_network_rx_dropped": 0,
"node_interface_network_rx_errors": 0,
"node_interface_network_rx_packets": 16.575629266820258,
"node_interface_network_total_bytes": 3518.3935157426017,
"node_interface_network_tx_bytes": 355.385094878293,
"node_interface_network_tx_dropped": 0,
"node_interface_network_tx_errors": 0,
"node_interface_network_tx_packets": 3.9997714100370625
}

```

Type: Pod

```

{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization"
        }
      ]
    }
  ]
}

```

```
    },
    {
      "Unit": "Bytes/Second",
      "Name": "pod_network_rx_bytes"
    },
    {
      "Unit": "Bytes/Second",
      "Name": "pod_network_tx_bytes"
    },
    {
      "Unit": "Percent",
      "Name": "pod_cpu_utilization_over_pod_limit"
    },
    {
      "Unit": "Percent",
      "Name": "pod_memory_utilization_over_pod_limit"
    }
  ],
  "Dimensions": [
    [
      "PodName",
      "Namespace",
      "ClusterName"
    ],
    [
      "Service",
      "Namespace",
      "ClusterName"
    ],
    [
      "Namespace",
      "ClusterName"
    ],
    [
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
},
{
  "Metrics": [
    {
      "Unit": "Percent",
      "Name": "pod_cpu_reserved_capacity"
```

```
    },
    {
      "Unit": "Percent",
      "Name": "pod_memory_reserved_capacity"
    }
  ],
  "Dimensions": [
    [
      "PodName",
      "Namespace",
      "ClusterName"
    ],
    [
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
},
{
  "Metrics": [
    {
      "Unit": "Count",
      "Name": "pod_number_of_container_restarts"
    }
  ],
  "Dimensions": [
    [
      "PodName",
      "Namespace",
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
}
],
"ClusterName": "myCICluster",
"InstanceId": "i-1234567890123456",
"InstanceType": "t3.xlarge",
"Namespace": "amazon-cloudwatch",
"NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
"PodName": "cloudwatch-agent-statsd",
"Service": "cloudwatch-agent-statsd",
"Sources": [
  "cadvisor",
```

```
"pod",
  "calculated"
],
"Timestamp": "1567097351092",
"Type": "Pod",
"Version": "0",
"kubernetes": {
  "host": "ip-192-168-75-26.us-west-2.compute.internal",
  "labels": {
    "app": "cloudwatch-agent-statsd",
    "pod-template-hash": "df44f855f"
  },
  "namespace_name": "amazon-cloudwatch",
  "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
  "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
  "pod_owners": [
    {
      "owner_kind": "Deployment",
      "owner_name": "cloudwatch-agent-statsd"
    }
  ],
  "service_name": "cloudwatch-agent-statsd"
},
"pod_cpu_limit": 200,
"pod_cpu_request": 200,
"pod_cpu_reserved_capacity": 5,
"pod_cpu_usage_system": 1.4504841104992765,
"pod_cpu_usage_total": 5.817016867430125,
"pod_cpu_usage_user": 1.1281543081661038,
"pod_cpu_utilization": 0.14542542168575312,
"pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
"pod_memory_cache": 8192,
"pod_memory_failcnt": 0,
"pod_memory_hierarchical_pgfault": 0,
"pod_memory_hierarchical_pgmajfault": 0,
"pod_memory_limit": 104857600,
"pod_memory_mapped_file": 0,
"pod_memory_max_usage": 25268224,
"pod_memory_pgfault": 0,
"pod_memory_pgmajfault": 0,
"pod_memory_request": 104857600,
"pod_memory_reserved_capacity": 0.6307275170893897,
"pod_memory_rss": 22777856,
"pod_memory_swap": 0,
```

```

"pod_memory_usage": 25141248,
"pod_memory_utilization": 0.10988455961791709,
"pod_memory_utilization_over_pod_limit": 17.421875,
"pod_memory_working_set": 18268160,
"pod_network_rx_bytes": 9880.697124714186,
"pod_network_rx_dropped": 0,
"pod_network_rx_errors": 0,
"pod_network_rx_packets": 107.80005532263283,
"pod_network_total_bytes": 10158.829201483635,
"pod_network_tx_bytes": 278.13207676944796,
"pod_network_tx_dropped": 0,
"pod_network_tx_errors": 0,
"pod_network_tx_packets": 1.146027574644318,
"pod_number_of_container_restarts": 0,
"pod_number_of_containers": 1,
"pod_number_of_running_containers": 1,
"pod_status": "Running"
}

```

Type: PodNet

```

{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "PodNet",
  "Version": "0",
  "interface": "eth0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",

```



```
    "pod-template-hash": "df44f855f"
  },
  "namespace_name": "amazon-cloudwatch",
  "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
  "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
  "pod_owners": [
    {
      "owner_kind": "Deployment",
      "owner_name": "cloudwatch-agent-statsd"
    }
  ],
  "service_name": "cloudwatch-agent-statsd"
},
"pod_interface_network_rx_bytes": 9880.697124714186,
"pod_interface_network_rx_dropped": 0,
"pod_interface_network_rx_errors": 0,
"pod_interface_network_rx_packets": 107.80005532263283,
"pod_interface_network_total_bytes": 10158.829201483635,
"pod_interface_network_tx_bytes": 278.13207676944796,
"pod_interface_network_tx_dropped": 0,
"pod_interface_network_tx_errors": 0,
"pod_interface_network_tx_packets": 1.146027574644318
}
```

Type: Container

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-sample",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "advisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "Container",
}
```

```
"Version": "0",
"container_cpu_limit": 200,
"container_cpu_request": 200,
"container_cpu_usage_system": 1.87958283771964,
"container_cpu_usage_total": 6.159993652997942,
"container_cpu_usage_user": 1.6707403001952357,
"container_cpu_utilization": 0.15399984132494854,
"container_memory_cache": 8192,
"container_memory_failcnt": 0,
"container_memory_hierarchical_pgfault": 0,
"container_memory_hierarchical_pgmajfault": 0,
"container_memory_limit": 104857600,
"container_memory_mapped_file": 0,
"container_memory_max_usage": 24580096,
"container_memory_pgfault": 0,
"container_memory_pgmajfault": 0,
"container_memory_request": 104857600,
"container_memory_rss": 22736896,
"container_memory_swap": 0,
"container_memory_usage": 24453120,
"container_memory_utilization": 0.10574541028701798,
"container_memory_working_set": 17580032,
"container_status": "Running",
"kubernetes": {
  "container_name": "cloudwatch-agent",
  "docker": {
    "container_id":
"8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
  },
  "host": "ip-192-168-75-26.us-west-2.compute.internal",
  "labels": {
    "app": "cloudwatch-agent-statsd",
    "pod-template-hash": "df44f855f"
  },
  "namespace_name": "amazon-cloudwatch",
  "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
  "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
  "pod_owners": [
    {
      "owner_kind": "Deployment",
      "owner_name": "cloudwatch-agent-statsd"
    }
  ],
  "service_name": "cloudwatch-agent-statsd"
```

```

},
"number_of_container_restarts": 0
}

```

Type: ContainerFS

```

{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "ContainerFS",
  "Version": "0",

  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
"8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {

```

```
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
    }
],
"service_name": "cloudwatch-agent-statsd"
}
}
```

Type: Cluster

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
}
```

Type: ClusterService

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
  },
  "service_number_of_running_pods": 1
}
```

Type: ClusterNamespace

```
{
  "CloudWatchMetrics": [
    {
```

```
  "Metrics": [
    {
      "Unit": "Count",
      "Name": "namespace_number_of_running_pods"
    }
  ],
  "Dimensions": [
    [
      "Namespace",
      "ClusterName"
    ],
    [
      "ClusterName"
    ]
  ],
  "Namespace": "ContainerInsights"
},
"ClusterName": "myCICluster",
"Namespace": "amazon-cloudwatch",
"Sources": [
  "apiserver"
],
"Timestamp": "1567097594160",
"Type": "ClusterNamespace",
"Version": "0",
"kubernetes": {
  "namespace_name": "amazon-cloudwatch"
},
"namespace_number_of_running_pods": 7
}
```

Relevant fields in performance log events for Amazon EKS and Kubernetes

For Amazon EKS and Kubernetes, the containerized CloudWatch agent emits data as performance log events. This enables CloudWatch to ingest and store high-cardinality data. CloudWatch uses the data in the performance log events to create aggregated CloudWatch metrics at the cluster, node, and pod levels without the need to lose granular details.

The following table lists the fields in these performance log events that are relevant to the collection of Container Insights metric data. You can use CloudWatch Logs Insights to query for

any of these fields to collect data or investigate issues. For more information, see [Analyze Log Data With CloudWatch Logs Insights](#).

Type	Log field	Source	Formula or notes
Pod	pod_cpu_utilization	Calculated	Formula: $\text{pod_cpu_usage_total} / \text{node_cpu_limit}$
Pod	pod_cpu_usage_total pod_cpu_usage_total is reported in millicores.	cadvisor	
Pod	pod_cpu_limit	Calculated	Formula: $\text{sum}(\text{container_cpu_limit})$ $\text{sum}(\text{container_cpu_limit})$ includes already-completed pods. If any containers in the pod don't have a CPU limit defined, this field doesn't

Type	Log field	Source	Formula or notes
			appear in the log event. This includes init containers .
Pod	pod_cpu_request	Calculated	Formula: $\text{sum}(\text{container_cpu_request})$ <p>container_cpu_request isn't guaranteed to be set. Only the ones that are set are included in the sum.</p>
Pod	pod_cpu_utilization_over_pod_limit	Calculated	Formula: $\text{pod_cpu_usage_total} / \text{pod_cpu_limit}$
Pod	pod_cpu_reserved_capacity	Calculated	Formula: $\text{pod_cpu_request} / \text{node_cpu_limit}$

Type	Log field	Source	Formula or notes
Pod	pod_memory_utilization	Calculated	<p>Formula:</p> $\text{pod_memory_working_set} / \text{node_memory_limit}$ <p>It is the percentage of pod memory usage over the node memory limitation.</p>
Pod	pod_memory_working_set	cadvisor	
Pod	pod_memory_limit	Calculated	<p>Formula:</p> $\text{sum}(\text{container_memory_limit})$ <p>If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes init containers.</p>

Type	Log field	Source	Formula or notes
Pod	pod_memory_request	Calculated	<p>Formula:</p> $\text{sum}(\text{container_memory_request})$ <p>container_memory_request isn't guaranteed to be set. Only the ones that are set are included in the sum.</p>
Pod	pod_memory_utilization_over_pod_limit	Calculated	<p>Formula:</p> $\frac{\text{pod_memory_working_set}}{\text{pod_memory_limit}}$ <p>If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes init containers.</p>

Type	Log field	Source	Formula or notes
Pod	pod_memory_reserved_capacity	Calculated	Formula: pod_memory_request / node_memory_limit
Pod	pod_network_tx_bytes	Calculated	Formula: sum(pod_interface_network_tx_bytes) This data is available for all the network interfaces per pod. The CloudWatch agent calculates the total and adds metric extraction rules.
Pod	pod_network_rx_bytes	Calculated	Formula: sum(pod_interface_network_rx_bytes)

Type	Log field	Source	Formula or notes
Pod	pod_network_total_bytes	Calculated	Formula: pod_network_rx_bytes + pod_network_tx_bytes
PodNet	pod_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a pod network interface.
PodNet	pod_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a pod network interface.
Container	container_cpu_usage_total	cadvisor	
Container	container_cpu_limit	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_cpu_request	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_working_set	cadvisor	

Type	Log field	Source	Formula or notes
Container	container_memory_limit	pod	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_request	pod	Not guaranteed to be set. It's not emitted if it's not set.
Node	node_cpu_utilization	Calculated	Formula: $\frac{\text{node_cpu_usage_total}}{\text{node_cpu_limit}}$
Node	node_cpu_usage_total	cadvisor	
Node	node_cpu_limit	/proc	

Type	Log field	Source	Formula or notes
Node	node_cpu_request	Calculated	<p>Formula: $\text{sum}(\text{pod_cpu_request})$</p> <p>For cronjobs, node_cpu_request also includes requests from completed pods. This can lead to a high value for node_cpu_reserved_capacity .</p>
Node	node_cpu_reserved_capacity	Calculated	<p>Formula: $\text{node_cpu_request} / \text{node_cpu_limit}$</p>
Node	node_memory_utilization	Calculated	<p>Formula: $\text{node_memory_working_set} / \text{node_memory_limit}$</p>
Node	node_memory_working_set	cadvisor	
Node	node_memory_limit	/proc	

Type	Log field	Source	Formula or notes
Node	node_memory_request	Calculated	Formula: sum(pod_memory_request)
Node	node_memory_reserved_capacity	Calculated	Formula: node_memory_request / node_memory_limit
Node	node_network_rx_bytes	Calculated	Formula: sum(node_interface_network_rx_bytes)
Node	node_network_tx_bytes	Calculated	Formula: sum(node_interface_network_tx_bytes)
Node	node_network_total_bytes	Calculated	Formula: node_network_rx_bytes + node_network_tx_bytes
Node	node_number_of_running_pods	Pod List	

Type	Log field	Source	Formula or notes
Node	node_number_of_running_containers	Pod List	
NodeNet	node_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a worker node network interface.
NodeNet	node_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a worker node network interface.
NodeFS	node_filesystem_capacity	cadvisor	
NodeFS	node_filesystem_usage	cadvisor	
NodeFS	node_filesystem_utilization	Calculated	<p>Formula: $\frac{\text{node_filesystem_usage}}{\text{node_filesystem_capacity}}$ </p> <p>This data is available per device name.</p>

Type	Log field	Source	Formula or notes
Cluster	cluster_failed_node_count	API Server	
Cluster	cluster_node_count	API Server	
Service	service_number_of_running_pods	API Server	
Namespace	namespace_number_of_running_pods	API Server	

Metrics calculation examples

This section includes examples that show how some of the values in the preceding table are calculated.

Suppose that you have a cluster in the following state.

```

Node1
  node_cpu_limit = 4
  node_cpu_usage_total = 3

Pod1
  pod_cpu_usage_total = 2

  Container1
    container_cpu_limit = 1
    container_cpu_request = 1
    container_cpu_usage_total = 0.8

  Container2
    container_cpu_limit = null
    container_cpu_request = null
    container_cpu_usage_total = 1.2

Pod2
  pod_cpu_usage_total = 0.4

  Container3

```

```

container_cpu_limit = 1
container_cpu_request = 0.5
container_cpu_usage_total = 0.4

```

Node2

```

node_cpu_limit = 8
node_cpu_usage_total = 1.5

```

Pod3

```

pod_cpu_usage_total = 1

```

Container4

```

container_cpu_limit = 2
container_cpu_request = 2
container_cpu_usage_total = 1

```

The following table shows how pod CPU metrics are calculated using this data.

Metric	Formula	Pod1	Pod2	Pod3
pod_cpu_utilization	$\text{pod_cpu_usage_total} / \text{node_cpu_limit}$	$2 / 4 = 50\%$	$0.4 / 4 = 10\%$	$1 / 8 = 12.5\%$
pod_cpu_utilization_over_pod_limit	$\text{pod_cpu_usage_total} / \text{sum}(\text{container_cpu_limit})$	N/A because CPU limit for Container 2 isn't defined	$0.4 / 1 = 40\%$	$1 / 2 = 50\%$
pod_cpu_reserved_capacity	$\text{sum}(\text{container_cpu_request}) / \text{node_cpu_limit}$	$(1 + 0) / 4 = 25\%$	$0.5 / 4 = 12.5\%$	$2 / 8 = 25\%$

The following table shows how node CPU metrics are calculated using this data.

Metric	Formula	Node1	Node2
node_cpu_utilization	$\text{node_cpu_usage_total} / \text{node_cpu_limit}$	$3 / 4 = 75\%$	$1.5 / 8 = 18.75\%$
node_cpu_reserved_capacity	$\text{sum}(\text{pod_cpu_request}) / \text{node_cpu_limit}$	$1.5 / 4 = 37.5\%$	$2 / 8 = 25\%$

Container Insights Prometheus metrics monitoring

CloudWatch Container Insights monitoring for Prometheus automates the discovery of Prometheus metrics from containerized systems and workloads. Prometheus is an open-source systems monitoring and alerting toolkit. For more information, see [What is Prometheus?](#) in the Prometheus documentation.

Discovering Prometheus metrics is supported for [Amazon Elastic Container Service](#), [Amazon Elastic Kubernetes Service](#) and [Kubernetes](#) clusters running on Amazon EC2 instances. The Prometheus counter, gauge, and summary metric types are collected. Support for histogram metrics is planned for an upcoming release.

For Amazon ECS and Amazon EKS clusters, both the EC2 and Fargate launch types are supported. Container Insights automatically collects metrics from several workloads, and you can configure it to collect metrics from any workload.

You can adopt Prometheus as an open-source and open-standard method to ingest custom metrics in CloudWatch. The CloudWatch agent with Prometheus support discovers and collects Prometheus metrics to monitor, troubleshoot, and alarm on application performance degradation and failures faster. This also reduces the number of monitoring tools required to improve observability.

Container Insights Prometheus support involves pay-per-use of metrics and logs, including collecting, storing, and analyzing. For more information, see [Amazon CloudWatch Pricing](#).

Pre-built dashboards for some workloads

The Container Insights Prometheus solution includes pre-built dashboards for the popular workloads that are listed in this section. For sample configurations for these workloads, see [\(Optional\) Set up sample containerized Amazon ECS workloads for Prometheus metric testing](#) and [\(Optional\) Set up sample containerized Amazon EKS workloads for Prometheus metric testing](#).

You can also configure Container Insights to collect Prometheus metrics from other containerized services and applications by editing the agent configuration file.

Workloads with pre-built dashboards for Amazon EKS clusters and Kubernetes clusters running on Amazon EC2 instances:

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

Workloads with pre-built dashboards for Amazon ECS clusters:

- AWS App Mesh
- Java/JMX
- NGINX
- NGINX Plus

Set up and configure Prometheus metrics collection on Amazon ECS clusters

To collect Prometheus metrics from Amazon ECS clusters, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see <https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus>.

The following sections explain how to use the CloudWatch agent as the collector to retrieve Prometheus metrics. You install the CloudWatch agent with Prometheus monitoring on clusters running Amazon ECS, and you can optionally configure the agent to scrape additional targets. These sections also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Container Insights on Amazon ECS supports the following launch type and network mode combinations for Prometheus metrics:

Amazon ECS launch type	Network modes supported
EC2 (Linux)	bridge, host, and awsvpc
Fargate	awsvpc

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters](#)
- [Scraping additional Prometheus sources and importing those metrics](#)
- [\(Optional\) Set up sample containerized Amazon ECS workloads for Prometheus metric testing](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon ECS. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- Java/JMX

You can also configure the agent to scrape and import metrics from additional Prometheus workloads and sources.

Set up IAM roles

You need two IAM roles for the CloudWatch agent task definition. If you specify **CreateIAMRoles=True** in the AWS CloudFormation stack to have Container Insights create these roles for you, the roles will be created with the correct permissions. If you want to create them yourself or use existing roles, the following roles and permissions are required.

- **CloudWatch agent ECS task role**— The CloudWatch agent container uses this role. It must include the **CloudWatchAgentServerPolicy** policy and a customer-managed policy which contains the following read-only permissions:
 - `ec2:DescribeInstances`
 - `ecs:ListTasks`
 - `ecs:ListServices`
 - `ecs:DescribeContainerInstances`
 - `ecs:DescribeServices`
 - `ecs:DescribeTasks`
 - `ecs:DescribeTaskDefinition`
- **CloudWatch agent ECS task execution role**— This is the role that Amazon ECS requires to launch and execute your containers. Ensure that your task execution role has the **AmazonSSMReadOnlyAccess**, **AmazonECSTaskExecutionRolePolicy**, and **CloudWatchAgentServerPolicy** policies attached. If you want to store more sensitive data for Amazon ECS to use, see [Specifying sensitive data](#).

Install the CloudWatch agent with Prometheus monitoring by using AWS CloudFormation

You use AWS CloudFormation to install the CloudWatch agent with Prometheus monitoring for Amazon ECS clusters. The following list shows the parameters you will use in the AWS CloudFormation template.

- **ECSClusterName**— Specifies the target Amazon ECS cluster.
- **CreateIAMRoles**— Specify **True** to create new roles for the Amazon ECS task role and Amazon ECS task execution role. Specify **False** to reuse existing roles.
- **TaskRoleName**— If you specified **True** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task role. If you specified **False** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task role.
- **ExecutionRoleName**— If you specified **True** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task execution role. If you specified **False** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task execution role.
- **ECSNetworkMode**— If you are using EC2 launch type, specify the network mode here. It must be either **bridge** or **host**.
- **ECSLaunchType**— Specify either **fargate** or **EC2**.

- **SecurityGroupID**— If the **ECSNetworkMode** is **awsvpc**, specify the security group ID here.
- **SubnetID**— If the **ECSNetworkMode** is **awsvpc**, specify the subnet ID here.

Command samples

This section includes sample AWS CloudFormation commands to install Container Insights with Prometheus monitoring in various scenarios.

Create AWS CloudFormation stack for an Amazon ECS cluster in bridge network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${AWS_DEFAULT_REGION} \
  --profile ${AWS_PROFILE}
```

Create AWS CloudFormation stack for an Amazon ECS cluster in host network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
```

```

export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -0 https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${AWS_DEFAULT_REGION} \
  --profile ${AWS_PROFILE}

```

Create AWS CloudFormation stack for an Amazon ECS cluster in awsvpc network mode

```

export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -0 https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
  --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \

```



```

        ParameterKey=SecurityGroupID,ParameterValue=
    ${ECS_CLUSTER_SECURITY_GROUP} \
        ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
        ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
        ParameterKey=ExecutionRoleName,ParameterValue=
    ${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}

```

Create AWS CloudFormation stack for a Fargate cluster in awsvpc network mode

```

export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
    ${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
        ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
        ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
        ParameterKey=SecurityGroupID,ParameterValue=
    ${ECS_CLUSTER_SECURITY_GROUP} \
        ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
        ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
        ParameterKey=ExecutionRoleName,ParameterValue=
    ${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}

```

AWS resources created by the AWS CloudFormation stack

The following table lists the AWS resources that are created when you use AWS CloudFormation to set up Container Insights with Prometheus monitoring on an Amazon ECS cluster.

Resource type	Resource name	Comments
AWS::SSM: :Parameter	AmazonCloudWatch-CWAgentConfig- <i>\$ECS_CLUSTER_NAME</i> - <i>\$ECS_LAUNCH_TYPE</i> - <i>\$ECS_NETWORK_MODE</i>	This is the CloudWatch agent with the default App Mesh and Java/JMX embedded metric format definition.
AWS::SSM: :Parameter	AmazonCloudWatch-Prometheus ConfigName- <i>\$ECS_CLUSTER_NAME</i> - <i>\$ECS_LAUNCH_TYPE</i> - <i>\$ECS_NETWORK_MODE</i>	This is the Prometheus scraping configuration.
AWS::IAM: :Role	<i>\$ECS_TASK_ROLE_NAME</i> .	The Amazon ECS task role. This is created only if you specified True for <code>CREATE_IAM_ROLES</code> .
AWS::IAM: :Role	<i>#{ECS_EXECUTION_ROLE_NAME}</i>	The Amazon ECS task execution role. This is created only if you specified True for <code>CREATE_IAM_ROLES</code> .
AWS::ECS: :TaskDefinition	cwagent-prometheus- <i>\$ECS_CLUSTER_NAME</i> - <i>\$ECS_LAUNCH_TYPE</i> - <i>\$ECS_NETWORK_MODE</i>	
AWS::ECS: :Service	cwagent-prometheus-replica-service- <i>\$ECS_LAUNCH_TYPE</i> - <i>\$ECS_NETWORK_MODE</i>	

Deleting the AWS CloudFormation stack for the CloudWatch agent with Prometheus monitoring

To delete the CloudWatch agent from an Amazon ECS cluster, enter these commands.

```
export AWS_PROFILE=your_aws_config_profile_eg_default
```

```
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name

aws cloudformation delete-stack \
--stack-name ${CLOUDFORMATION_STACK_NAME} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in [<scrape_config>](#) in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon ECS clusters, the configurations are integrated with the Parameter Store of AWS Systems Manager by the secrets in the Amazon ECS task definition:

- The secret `PROMETHEUS_CONFIG_CONTENT` is for the Prometheus scrape configuration.
- The secret `CW_CONFIG_CONTENT` is for the CloudWatch agent configuration.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in [<scrape_config>](#) in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
```

```
scrape_interval: 1m
scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in the YAML files for Amazon ECS such as `cwagent-ecs-prometheus-metric-for-bridge-host.yaml`, the default scraping jobs are configured in the `ecs_service_discovery` section.

```
"ecs_service_discovery": {
    "sd_frequency": "1m",
    "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
    "docker_label": {
    },
    "task_definition_list": [
        {
            "sd_job_name": "ecs-appmesh-colors",
            "sd_metrics_ports": "9901",
            "sd_task_definition_arn_pattern": ".*:task-definition\\/.*-
ColorTeller-(white):[0-9]+",
            "sd_metrics_path": "/stats/prometheus"
        },
        {
            "sd_job_name": "ecs-appmesh-gateway",
            "sd_metrics_ports": "9901",
            "sd_task_definition_arn_pattern": ".*:task-definition/.*-
ColorGateway:[0-9]+",
            "sd_metrics_path": "/stats/prometheus"
        }
    ]
}
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see [Embedding metrics within logs](#).

Log events from Amazon ECS clusters are stored in the `/aws/ecs/containerinsights/cluster_name/prometheus` log group.

Each scraping job is contained in a different log stream in this log group.

To add a new scraping target, you add a new entry in the `task_definition_list` section under the `ecs_service_discovery` section of the YAML file, and restart the agent. For an example of this process, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon ECS cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses `/aws/ecs/containerinsights/cluster_name/prometheus` for logs from Amazon ECS clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configurations of the Amazon ECS Prometheus target auto-discovery functions. Two modes are supported to discover the Prometheus targets: discovery based on the container's docker label or discovery based on the Amazon ECS task definition ARN regular expression. You can use the two modes together and the CloudWatch agent will de-duplicate the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

The `ecs_service_discovery` section can contain the following fields:

- **sd_frequency** is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
 - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
 - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the agent assumes the default path `/metrics`.
 - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
 - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
 - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.

- `service_name_list_for_tasks` is an optional section that you can use to specify the configuration of service name-based discovery. If you omit this section, service name-based discovery is not used. This section can contain the following fields:
 - `sd_service_name_pattern` is the pattern to use to specify the Amazon ECS service where tasks are to be discovered. This is a regular expression.
 - `sd_metrics_ports` Lists the `containerPort` for the Prometheus metrics. Separate multiple `containerPorts` with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metrics path. If you omit this, the agent assumes that the default path `/metrics`.
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
 - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

If you have multiple labels specified in `source_labels`, we recommend that you do not use `^` or `$` characters in the regular expression for `label_matcher`.

- `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
- `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
- `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
- `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
    "source_labels": [ "Service", "Namespace" ],
    "label_matcher": "(.*node-exporter.*|.kubernetes.*);kube-system$",
    "dimensions": [
      [ "Service", "Namespace" ]
    ],
    "metric_selectors": [
      "^coredns_dns_request_type_count_total$"
    ]
  }
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Name": "coredns_dns_request_type_count_total"
        }
      ],
      "Dimensions": [
        [
          "Namespace",
          "Service"
        ]
      ],
      "Namespace": "ContainerInsights/Prometheus"
    }
  ],
  "Namespace": "kube-system",
```



```
"Service": "kube-dns",
"coredns_dns_request_type_count_total": 2562,
"eks_amazonaws_com_component": "kube-dns",
"instance": "192.168.61.254:9153",
"job": "kubernetes-service-endpoints",
...
}
```

Detailed guide for autodiscovery on Amazon ECS clusters

Prometheus provides dozens of dynamic service-discovery mechanisms as described in [<scrape_config>](#). However there is no built-in service discovery for Amazon ECS. The CloudWatch agent adds this mechanism.

When the Amazon ECS Prometheus service discovery is enabled, the CloudWatch agent periodically makes the following API calls to Amazon ECS and Amazon EC2 frontends to retrieve the metadata of the running ECS tasks in the target ECS cluster.

```
EC2:DescribeInstances
ECS:ListTasks
ECS:ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
ECS:DescribeTaskDefinition
```

The metadata is used by the CloudWatch agent to scan the Prometheus targets within the ECS cluster. The CloudWatch agent supports three service discovery modes:

- Container docker label-based service discovery
- ECS task definition ARN regular expression-based service discovery
- ECS service name regular expression-based service discovery

All modes can be used together. CloudWatch agent de-duplicates the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

All discovered targets are written into a result file specified by the `sd_result_file` configuration field within the CloudWatch agent container. The following is a sample result file:

```
- targets:
```

```
- 10.6.1.95:32785
labels:
  __metrics_path__: /metrics
  ECS_PROMETHEUS_EXPORTER_PORT: "9406"
  ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
  ECS_PROMETHEUS_METRICS_PATH: /metrics
  InstanceType: t3.medium
  LaunchType: EC2
  SubnetId: subnet-123456789012
  TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
  TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
  TaskRevision: "7"
  VpcId: vpc-01234567890
  container_name: demo-jar-ec2-bridge-dynamic-port
  job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
    ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
    TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
    TaskRevision: "12"
    VpcId: vpc-01234567890
    container_name: demo-tomcat-ec2-bridge-mapped-port
    job: demo-tomcat-ec2-bridge-mapped-port
```

You can directly integrate this result file with Prometheus file-based service discovery. For more information about Prometheus file-based service discovery, see [<file_sd_config>](#).

Suppose the result file is written to `/tmp/cwagent_ecs_auto_sd.yaml`. The following Prometheus scrape configuration will consume it.

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
```

```
sample_limit: 10000
file_sd_configs:
  - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

The CloudWatch agent also adds the following additional labels for the discovered targets.

- `container_name`
- `TaskDefinitionFamily`
- `TaskRevision`
- `TaskGroup`
- `StartedBy`
- `LaunchType`
- `job`
- `__metrics_path__`
- Docker labels

When the cluster has the EC2 launch type, the following three labels are added.

- `InstanceType`
- `VpcId`
- `SubnetId`

Note

Docker labels that don't match the regular expression `[a-zA-Z_][a-zA-Z0-9_]*` are filtered out. This matches the Prometheus conventions as listed in `label_name` in [Configuration file](#) in the Prometheus documentation.

ECS service discovery configuration examples

This section includes examples that demonstrate ECS service discovery.

Example 1

```
"ecs_service_discovery": {
```

```
"sd_frequency": "1m",
"sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
"docker_label": {
}
}
```

This example enables docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata once per minute and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

The default value of `sd_port_label` in the `docker_label` section is `ECS_PROMETHEUS_EXPORTER_PORT`. If any running container in the ECS tasks has a `ECS_PROMETHEUS_EXPORTER_PORT` docker label, the CloudWatch agent uses its value as `container_port` to scan all exposed ports of the container. If there is a match, the mapped host port plus the private IP of the container are used to construct the Prometheus exporter target in the following format: `private_ip:host_port`.

The default value of `sd_metrics_path_label` in the `docker_label` section is `ECS_PROMETHEUS_METRICS_PATH`. If the container has this docker label, its value will be used as the `__metrics_path__`. If the container does not have this label, the default value `/metrics` is used.

The default value of `sd_job_name_label` in the `docker_label` section is `job`. If the container has this docker label, its value will be appended as one of the labels for the target to replace the default job name specified in the Prometheus configuration. The value of this docker label is used as the log stream name in the CloudWatch Logs log group.

Example 2

```
"ecs_service_discovery": {
  "sd_frequency": "15s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
    "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
  }
}
```

This example enables docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every 15 seconds and write the discovered targets into the `/tmp/`

`cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container. The containers with a docker label of `ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A` will be scanned. The value of the docker label `ECS_PROMETHEUS_JOB_NAME` is used as the job name.

Example 3

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "task_definition_list": [
    {
      "sd_job_name": "java-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9404; 9406",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
    },
    {
      "sd_job_name": "envoy-prometheus",
      "sd_metrics_path": "/stats/prometheus",
      "sd_container_name_pattern": "^envoy$",
      "sd_metrics_ports": "9901",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
  ]
}
```

This example enables ECS task definition ARN regular expression-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two task definition ARN regular expression sections are defined:

- For the first section, the ECS tasks with `javajmx` in their ECS task definition ARN are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, the scraped metrics are sent to the `java-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

- For the second section, the ECS tasks with appmesh in their ECS task definition ARN and with version of :23 are filtered for the container port scan. For containers with a name of envoy that expose the container port on 9901, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/prometheus`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/prometheus`, and send the scraped metrics to the `envoy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 4

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "service_name_list_for_tasks": [
    {
      "sd_job_name": "nginx-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9113",
      "sd_service_name_pattern": "^nginx-.*"
    },
    {
      "sd_job_name": "haproxy-prometheus",
      "sd_metrics_path": "/stats/metrics",
      "sd_container_name_pattern": "^haproxy$",
      "sd_metrics_ports": "8404",
      "sd_service_name_pattern": ".*haproxy-service.*"
    }
  ]
}
```

This example enables ECS service name regular expression-based service discovery. The CloudWatch agent will query the ECS services' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two service name regular expression sections are defined:

- For the first section, the ECS tasks that are associated with ECS services that have names matching the regular expression `^nginx-.*` are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9113, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, and the scraped metrics are sent to the `nginx-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.
- or the second section, the ECS tasks that are associated with ECS services that have names matching the regular expression `.*haproxy-service.*` are filtered for the container port scan. For containers with a name of `haproxy` expose the container port on 8404, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/metrics`, and the scraped metrics are sent to the `haproxy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 5

```
"ecs_service_discovery": {
  "sd_frequency": "1m30s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
    "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
    "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
  }
}
"task_definition_list": [
  {
    "sd_metrics_ports": "9150",
    "sd_task_definition_arn_pattern": "*memcached.*"
  }
]
}
```

This example enables both ECS service discovery modes. The CloudWatch agent will query the ECS tasks' metadata every 90 seconds and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

For the docker-based service discovery configuration:

- The ECS tasks with docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` will be filtered for Prometheus port scan. The target Prometheus container port is specified by the value of the label `MY_PROMETHEUS_EXPORTER_PORT_LABEL`.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used for `__metrics_path__`. If the container does not have this docker label, the default value `/metrics` is used.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used as the job label. If the container does not have this docker label, the job name defined in the Prometheus configuration is used.

For the ECS task definition ARN regular expression-based service discovery configuration:

- The ECS tasks with `memcached` in the ECS task definition ARN are filtered for container port scan. The target Prometheus container port is 9150 as defined by `sd_metrics_ports`. The default metrics path `/metrics` is used. The job name defined in the Prometheus configuration is used.

(Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see [Prometheus metrics collected by the CloudWatch agent](#).

Topics

- [Sample App Mesh workload for Amazon ECS clusters](#)
- [Sample Java/JMX workload for Amazon ECS clusters](#)
- [Sample NGINX workload for Amazon ECS clusters](#)
- [Sample NGINX Plus workload for Amazon ECS clusters](#)
- [Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS](#)

- [Tutorial for scraping Redis Prometheus metrics on Amazon ECS Fargate](#)

Sample App Mesh workload for Amazon ECS clusters

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS](#).

First, follow this [walkthrough](#) to deploy the sample color app on your Amazon ECS cluster. After you finish, you will have App Mesh Prometheus metrics exposed on port 9901.

Next, follow these steps to install the CloudWatch agent with Prometheus monitoring on the same Amazon ECS cluster where you installed the color app. The steps in this section install the CloudWatch agent in bridge network mode.

The environment variables `ENVIRONMENT_NAME`, `AWS_PROFILE`, and `AWS_DEFAULT_REGION` that you set in the walkthrough will also be used in the following steps.

To install the CloudWatch agent with Prometheus monitoring for testing

1. Download the AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Set the network mode by entering the following commands.

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. Create the AWS CloudFormation stack by entering the following commands.

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=True \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-
TaskRole-${ECS_CLUSTER_NAME} \
```

```
ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-
ExecutionRole-${ECS_CLUSTER_NAME} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

4. (Optional) When the AWS CloudFormation stack is created, you see a `CREATE_COMPLETE` message. If you to check the status before you see that message, enter the following command.

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Troubleshooting

The steps in the walkthrough use `jq` to parse the output result of the AWS CLI. For more information about installing `jq`, see [jq](#). Use the following command to set the default output format of your AWS CLI to JSON so `jq` can parse it correctly.

```
$ aws configure
```

When the response gets to `Default output format`, enter `json`.

Uninstall the CloudWatch agent with Prometheus monitoring

When you are finished testing, enter the following command to uninstall the CloudWatch agent by deleting the AWS CloudFormation stack.

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Sample Java/JMX workload for Amazon ECS clusters

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

The CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the JMX Exporter to expose the metrics on a different port or `metrics_path`. If you do change the port or path, update the default `ecs_service_discovery` section in the CloudWatch agent configuration.

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS](#).

To install the Java/JMX sample workload for Amazon ECS clusters

1. Follow the steps in these sections to create your Docker images.
 - [Example: Java Jar Application Docker image with Prometheus metrics](#)
 - [Example: Apache Tomcat Docker image with Prometheus metrics](#)
2. Specify the following two docker labels in the Amazon ECS task definition file. You can then run the task definition as an Amazon ECS service or Amazon ECS task in the cluster.
 - Set `ECS_PROMETHEUS_EXPORTER_PORT` to point to the containerPort where the Prometheus metrics are exposed.
 - Set `Java_EMF_Metrics` to `true`. The CloudWatch agent uses this flag to generated the embedded metric format in the log event.

The following is an example:

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
        }
      ]
    }
  ]
}
```

```
    }
  ],
  "dockerLabels": {
    "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
    "Java_EMF_Metrics": "true"
  }
}
],
"requiresCompatibilities": [
  "EC2" ],
"cpu": "256",
"memory": "512"
}
```

The default setting of the CloudWatch agent in the AWS CloudFormation template enables both docker label-based service discovery and task definition ARN-based service discovery. To view these default settings, see line 65 of the [CloudWatch agent YAML configuration file](#). The containers with the `ECS_PROMETHEUS_EXPORTER_PORT` label will be auto-discovered based on the specified container port for Prometheus scraping.

The default setting of the CloudWatch agent also has the `metric_declaration` setting for Java/JMX at line 112 of the same file. All docker labels of the target containers will be added as additional labels in the Prometheus metrics and sent to CloudWatch Logs. For the Java/JMX containers with docker label `Java_EMF_Metrics="true"`, the embedded metric format will be generated.

Sample NGINX workload for Amazon ECS clusters

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see [nginx-prometheus-exporter](#) on Github. For more information about the NGINX reverse proxy, see [ecs-nginx-reverse-proxy](#) on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/>.
2. Find the app directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX. First, create a directory with the following two files:
 - A sample Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

- An `nginx.conf` file, modified from <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/>:

```
events {
    worker_connections 768;
}

http {
    # Nginx will handle gzip compression of responses from the app server
    gzip on;
    gzip_proxied any;
    gzip_types text/plain application/json;
    gzip_min_length 1000;

    server{
        listen 8080;
        location /stub_status {
            stub_status on;
        }
    }
}
```

```
server {
    listen 80;

    # Nginx will reject anything not matching /api
    location /api {
        # Reject requests with unsupported HTTP method
        if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
            return 405;
        }

        # Only requests matching the whitelist expectations will
        # get sent to the application server
        proxy_pass http://app:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_cache_bypass $http_upgrade;
    }
}
}
```

Note

`stub_status` must be enabled on the same port that `nginx-prometheus-exporter` is configured to scrape metrics from. In our example task definition, `nginx-prometheus-exporter` is configured to scrape metrics from port 8080.

4. Build an image from files in your new directory:

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`.

The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-image* with the image URI for your customized NGINX image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.scrape-uri",
        "http://nginx:8080/stub_status"
      ]
    }
  ]
}
```

```
    ],
    "links": [
      "nginx"
    ],
    "portMappings": [
      {
        "containerPort": 9113,
        "protocol": "tcp"
      }
    ]
  }
],
"networkMode": "bridge",
"placementConstraints": [],
"family": "nginx-sample-stack"
}
```

2. Register the task definition by entering the following command.

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. Create a service to run the task by entering the following command:

Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-service$`. The next section provides more details.

```
aws ecs create-service \
  --cluster your-cluster-name \
  --service-name nginx-service \
  --task-definition nginx-sample-stack:1 \
  --desired-count 1
```

Configure the CloudWatch agent to scrape NGINX Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the Prometheus metrics for NGINX. With the task discovered and the metrics

available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [  
  {  
    "sd_job_name": "nginx-prometheus-exporter",  
    "sd_metrics_path": "/metrics",  
    "sd_metrics_ports": "9113",  
    "sd_service_name_pattern": "^nginx-service$"  
  }  
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX metrics. Be sure to follow the existing indentation pattern.

```
{  
  "source_labels": ["job"],  
  "label_matcher": ".*nginx.*",  
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],  
  "metric_selectors": [  
    "^nginx_.*$" ]  
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```

ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region $AWS_REGION \
  --change-set-name nginx-scraping-support

```

5. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
6. Review the newly-created changeset **nginx-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
  --desired-count 0 \
  --service cwagent-prometheus-replica-service-EC2-${ECS_NETWORK_MODE} \
  --region $AWS_REGION

```

7. Wait about 10 seconds, and then enter the following command.

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
  --desired-count 1 \
  --service cwagent-prometheus-replica-service-EC2-${ECS_NETWORK_MODE} \
  --region $AWS_REGION

```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```

ECS_CLUSTER_NAME=your_cluster_name

```

```

AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region $AWS_REGION

```

Viewing your NGINX metrics and logs

You can now view the NGINX metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/*your_cluster_name*/prometheus**, in the log stream *nginx-prometheus-exporter*.

Sample NGINX Plus workload for Amazon ECS clusters

NGINX Plus is the commercial version of NGINX. You must have a licence to use it. For more information, see [NGINX Plus](#)

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX Plus reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see [nginx-prometheus-exporter](#) on Github. For more information about the NGINX reverse proxy, see [ecs-nginx-reverse-proxy](#) on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Plus Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX Plus reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX Plus reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/>.
2. Find the app directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX Plus. Before you can build the image for NGINX Plus, you need to obtain the key named `nginx-repo.key` and the SSL certificate `nginx-repo.crt` for your licensed NGINX Plus. Create a directory and store in it your `nginx-repo.key` and `nginx-repo.crt` files.

In the directory that you just created, create the following two files:

- A sample Dockerfile with the following content. This docker file is adopted from a sample file provided at https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image. The important change that we make is that we load a separate file, called `nginx.conf`, which will be created in the next step.

```
FROM debian:buster-slim

LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

# Define NGINX versions for NGINX Plus and NGINX Plus modules
```

```
# Uncomment this block and the versioned nginxPackages block in the main RUN
# instruction to install a specific release
# ENV NGINX_VERSION 21
# ENV NJS_VERSION 0.3.9
# ENV PKG_RELEASE 1~buster

# Download certificate and key from the customer portal (https://cs.nginx.com
# (https://cs.nginx.com/))
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# COPY nginx.conf /etc/ssl/nginx/nginx.conf

RUN set -x \
# Create nginx user/group first, to be consistent throughout Docker variants
&& addgroup --system --gid 101 nginx \
&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /
nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y ca-
certificates gnupg1 \
&& \
NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
hkp://keyserver.ubuntu.com:80 \
hkp://p80.pool.sks-keyservers.net:80 \
pgp.mit.edu (http://pgp.mit.edu/) \
; do \
echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
"$NGINX_GPGKEY" && found=yes && break; \
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" &&
exit 1; \
apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
```

```

# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-
repo.crt\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-
repo.key\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/
apt/sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/
apt/sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]

```

- An `nginx.conf` file, modified from <https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx>.

```
events {
    worker_connections 768;
}

http {
    # Nginx will handle gzip compression of responses from the app server
    gzip on;
    gzip_proxied any;
    gzip_types text/plain application/json;
    gzip_min_length 1000;

    upstream backend {
        zone name 10m;
        server app:3000    weight=2;
        server app2:3000   weight=1;
    }

    server{
        listen 8080;
        location /api {
            api write=on;
        }
    }

    match server_ok {
        status 100-599;
    }

    server {
        listen 80;
        status_zone zone;
        # Nginx will reject anything not matching /api
        location /api {
            # Reject requests with unsupported HTTP method
            if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
                return 405;
            }

            # Only requests matching the whitelist expectations will
            # get sent to the application server
            proxy_pass http://backend;
            health_check uri=/lorem-ipsum match=server_ok;
            proxy_http_version 1.1;
        }
    }
}
```

```
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection 'upgrade';
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_cache_bypass $http_upgrade;
}
}
}
```

4. Build an image from files in your new directory:

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX Plus and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Plus Prometheus metrics.

The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`. The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-plus-image* with the image URI for your customized NGINX Plus image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-plus-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ]
    }
  ]
}
```



```
    }
  ],
  "links": [
    "app",
    "app2"
  ]
},
{
  "name": "app",
  "image": "your-web-server-app-image",
  "memory": 256,
  "cpu": 128,
  "essential": true
},
{
  "name": "app2",
  "image": "your-web-server-app-image",
  "memory": 256,
  "cpu": 128,
  "essential": true
},
{
  "name": "nginx-prometheus-exporter",
  "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
  "memory": 256,
  "cpu": 256,
  "essential": true,
  "command": [
    "-nginx.plus",
    "-nginx.scrape-uri",
    "http://nginx:8080/api"
  ],
  "links": [
    "nginx"
  ],
  "portMappings": [
    {
      "containerPort": 9113,
      "protocol": "tcp"
    }
  ]
}
],
"networkMode": "bridge",
```

```
"placementConstraints": [],
"family": "nginx-plus-sample-stack"
}
```

2. Register the task definition:

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-
definition-json
```

3. Create a service to run the task by entering the following command:

```
aws ecs create-service \
  --cluster your-cluster-name \
  --service-name nginx-plus-service \
  --task-definition nginx-plus-sample-stack:1 \
  --desired-count 1
```

Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-plus-service$`. The next section provides more details.

Configure the CloudWatch agent to scrape NGINX Plus Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task via the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration in the value key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
  }
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX Plus metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginxplus_connections_accepted$",
    "^nginxplus_connections_active$",
    "^nginxplus_connections_dropped$",
    "^nginxplus_connections_idle$",
    "^nginxplus_http_requests_total$",
    "^nginxplus_ssl_handshakes$",
    "^nginxplus_ssl_handshakes_failed$",
    "^nginxplus_up$",
    "^nginxplus_upstream_server_health_checks_fails$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName",
"upstream"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_response_time$"
  ]
},
```

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_responses$",
    "^nginxplus_server_zone_responses$"
  ]
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region $AWS_REGION \
  --change-set-name nginx-plus-scraping-support
```

5. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
6. Review the newly-created changeset **nginx-plus-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
  --desired-count 0 \
```

```
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \  
--region $AWS_REGION
```

7. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \  
--desired-count 1 \  
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \  
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name  
AWS_REGION=your_aws_region  
ECS_NETWORK_MODE=bridge  
CREATE_IAM_ROLES=True  
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name  
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name  
  
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-  
{ECS_CLUSTER_NAME}-EC2-{ECS_NETWORK_MODE} \  
--template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \  
--parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \  
ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \  
ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \  
ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \  
ParameterKey=ExecutionRoleName,ParameterValue=  
$ECS_EXECUTION_ROLE_NAME \  
--capabilities CAPABILITY_NAMED_IAM \  
--region $AWS_REGION
```

Viewing your NGINX Plus metrics and logs

You can now view the NGINX Plus metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.

3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus`, in the log stream `nginx-plus-prometheus-exporter`.

Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Memcached application on an Amazon Amazon ECS cluster with the EC2 launch type. The Memcached Prometheus exporter target will be auto-discovered by the CloudWatch agent by ECS task definition-based service discovery.

Memcached is a general-purpose distributed memory caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. For more information, see [What is Memcached?](#)

The [memcached_exporter](#) (Apache License 2.0) is one of the Prometheus official exporters. By default the memcache_exporter serves on port 0.0.0.0:9150 at `/metrics`.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [Memcached](#)
- [prom/memcached-exporter](#)

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS](#).

Topics

- [Set the Amazon ECS EC2 cluster environment variables](#)
- [Install the sample Memcached workload](#)
- [Configure the CloudWatch agent to scrape Memcached Prometheus metrics](#)
- [Viewing your Memcached metrics](#)

Set the Amazon ECS EC2 cluster environment variables

To set the Amazon ECS EC2 cluster environment variables

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see [Installing the Amazon ECS CLI](#).
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS cluster with the EC2 launch type where you want to install the sample Memcached workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam --size 1 \
--instance-type t3.medium \
--cluster $ECS_CLUSTER_NAME \
--region $AWS_REGION
```

The expected result of this command is as follows:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker
version 19.03.6-ce
INFO[0001] Created cluster                               cluster=ecs-ec2-memcached-
tutorial region=ca-central-1
INFO[0002] Waiting for your cluster resources to be created...
INFO[0002] Cloudformation stack status
stackStatus=CREATE_IN_PROGRESS
INFO[0063] Cloudformation stack status
stackStatus=CREATE_IN_PROGRESS
INFO[0124] Cloudformation stack status
stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxxxxxx
Security Group created: sg-xxxxxxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Install the sample Memcached workload

To install the sample Memcached workload which exposes the Prometheus metrics

1. Download the Memcached AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. Set the IAM role names to be created for Memcached by entering the following commands.

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name  
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Memcached workload by entering the following command. This sample installs the workload in host network mode.

```
MEMCACHED_ECS_NETWORK_MODE=host  
  
aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-  
$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \  
  --template-body file://memcached-traffic-sample.yaml \  
  --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \  
                ParameterKey=ECSNetworkMode,ParameterValue=  
$MEMCACHED_ECS_NETWORK_MODE \  
                ParameterKey=TaskRoleName,ParameterValue=  
$MEMCACHED_ECS_TASK_ROLE_NAME \  
                ParameterKey=ExecutionRoleName,ParameterValue=  
$MEMCACHED_ECS_EXECUTION_ROLE_NAME \  
  --capabilities CAPABILITY_NAMED_IAM \  
  --region $AWS_REGION
```

The AWS CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Memcached task definition
- One Memcached service

In the Memcached task definition, two containers are defined:

- The primary container runs a simple Memcached application and opens port 11211 for access.
- The other container runs the Redis exporter process to expose the Prometheus metrics on port 9150. This is the container to be discovered and scraped by the CloudWatch agent.

Configure the CloudWatch agent to scrape Memcached Prometheus metrics

To configure the CloudWatch agent to scrape Memcached Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

Then, in the `ecs_service_discovery` section, add the following configuration into the `task_definition_list` section.

```
{
  "sd_job_name": "ecs-memcached",
  "sd_metrics_ports": "9150",
  "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-demo.*:[0-9]+"
},
```

For the `metric_declaration` section, the default setting does not allow any Memcached metrics. Add the following section to allow Memcached metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
  "metric_selectors": [
```

```

    "^memcached_current_(bytes|items|connections)$",
    "^memcached_items_(reclaimed|evicted)_total$",
    "^memcached_(written|read)_bytes_total$",
    "^memcached_limit_bytes$",
    "^memcached_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [
    ["ClusterName", "TaskDefinitionFamily", "status", "command"],
    ["ClusterName", "TaskDefinitionFamily", "command"]
  ],
  "metric_selectors": [
    "^memcached_commands_total$"
  ]
},

```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by AWS CloudFormation, you can create a change set by entering the following commands.

```

ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
  --capabilities CAPABILITY_NAMED_IAM \
  --region $AWS_REGION \
  --change-set-name memcached-scraping-support

```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Review the newly created changeset `memcached-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
--parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
--capabilities CAPABILITY_NAMED_IAM \
--region $AWS_REGION
```

Viewing your Memcached metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions	
memcached _current_items	ClusterName , TaskDefinitionFamily	
memcached _current_connections	ClusterName , TaskDefinitionFamily	
memcached _limit_bytes	ClusterName , TaskDefinitionFamily	
memcached _current_bytes	ClusterName , TaskDefinitionFamily	
memcached _written_bytes_total	ClusterName , TaskDefinitionFamily	
memcached _read_bytes_total	ClusterName , TaskDefinitionFamily	
memcached _items_evicted_total	ClusterName , TaskDefinitionFamily	
memcached _items_reclaimed_total	ClusterName , TaskDefinitionFamily	
memcached _commands_total	ClusterName , TaskDefinitionFamily ClusterName , TaskDefinitionFamily, command ClusterName , TaskDefinitionFamily, status, command	

Note

The value of the **command** dimension can be: delete, get, cas, set, decr, touch, incr, or flush.

The value of the **status** dimension can be hit, miss, or badval.

You can also create a CloudWatch dashboard for your Memcached Prometheus metrics.

To create a dashboard for Memcached Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MOD
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/ecs-task-definition-templates/deployment-mode/
replica-service/cwagent-prometheus/sample_cloudwatch_dashboards/memcached/
cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name {DASHBOARD_NAME} --region
$AWS_REGION --dashboard-body
```

Tutorial for scraping Redis Prometheus metrics on Amazon ECS Fargate

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis application in an Amazon ECS Fargate cluster. The Redis Prometheus exporter target will be auto-discovered by the CloudWatch agent with Prometheus metric support based on the container's docker labels.

Redis (<https://redis.io/>) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see [redis](#).

redis_exporter (MIT License licensed) is used to expose the Redis prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see [redis_exporter](#).

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [redis](#)
- [redis_exporter](#)

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see [Setting up Container Insights on Amazon ECS](#).

Topics

- [Set the Amazon ECS Fargate cluster environment variable](#)
- [Set the network environment variables for the Amazon ECS Fargate cluster](#)
- [Install the sample Redis workload](#)
- [Configure the CloudWatch agent to scrape Redis Prometheus metrics](#)
- [Viewing your Redis metrics](#)

Set the Amazon ECS Fargate cluster environment variable

To set the Amazon ECS Fargate cluster environment variable

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see [Installing the Amazon ECS CLI](#).
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial  
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS Fargate cluster where you want to install the sample Redis workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam \
```

```
--cluster $ECS_CLUSTER_NAME \
--launch-type FARGATE \
--region $AWS_DEFAULT_REGION
```

The expected result of this command is as follows:

```
INFO[0000] Created cluster   cluster=ecs-fargate-redis-tutorial region=ca-central-1
INFO[0001] Waiting for your cluster resources to be created...
INFO[0001] Cloudformation stack status   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Set the network environment variables for the Amazon ECS Fargate cluster

To set the network environment variables for the Amazon ECS Fargate cluster

1. Set your VPC and subnet ID of the Amazon ECS cluster. If you created a new cluster in the previous procedure, you'll see these values in the result of the final command. Otherwise, use the IDs of the existing cluster that you are going to use with Redis.

```
ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxxxxxxx
ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxxxxxxx
ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxxxxxxx
```

2. In this tutorial, we are going to install the Redis application and the CloudWatch agent in the default security group of the Amazon ECS cluster's VPC. The default security group allows all network connection within the same security group so the CloudWatch agent can scrape the Prometheus metrics exposed on the Redis containers. In a real production environment, you might want to create dedicated security groups for the Redis application and CloudWatch agent and set customized permissions for them.

Enter the following command to get the default security group ID.

```
aws ec2 describe-security-groups \
--filters Name=vpc-id,Values=$ECS_CLUSTER_VPC \
--region $AWS_DEFAULT_REGION
```

Then set the Fargate cluster default security group variable by entering the following command, replacing *my-default-security-group* with the value you found from the previous command.

```
ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
```

Install the sample Redis workload

To install the sample Redis workload which exposes the Prometheus metrics

1. Download the Redis AWS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. Set the IAM role names to be created for Redis by entering the following commands.

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name  
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Redis workload by entering the following command.

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-  
$ECS_CLUSTER_NAME-fargate-awsipc \  
  --template-body file://redis-traffic-sample.yaml \  
  --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \  
                ParameterKey=SecurityGroupID,ParameterValue=  
$ECS_CLUSTER_SECURITY_GROUP \  
                ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \  
                ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME  
\  
                ParameterKey=ExecutionRoleName,ParameterValue=  
$REDIS_ECS_EXECUTION_ROLE_NAME \  
  --capabilities CAPABILITY_NAMED_IAM \  
  --region $AWS_DEFAULT_REGION
```

The AWS CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Redis task definition
- One Redis service

In the Redis task definition, two containers are defined:

- The primary container runs a simple Redis application and opens port 6379 for access.
- The other container runs the Redis exporter process to expose the Prometheus metrics on port 9121. This is the container to be discovered and scraped by the CloudWatch agent. The following docker label is defined so that the CloudWatch agent can discover this container based on it.

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

Configure the CloudWatch agent to scrape Redis Prometheus metrics

To configure the CloudWatch agent to scrape Redis Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

Then, in the `ecs_service_discovery` section shown here, the `docker_label`-based service discovery is enabled with the default settings which are based on `ECS_PROMETHEUS_EXPORTER_PORT`, which matches the docker label we defined in the Redis ECS task definition. So we do not need to make any changes in this section:

```
ecs_service_discovery": {  
  "sd_frequency": "1m",  
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",  
  * "docker_label": {
```

```
},*
...
```

For the `metric_declaration` section, the default setting does not allow any Redis metrics. Add the following section to allow Redis metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [{"ClusterName", "TaskDefinitionFamily"}],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [{"ClusterName", "TaskDefinitionFamily", "cmd"}],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [{"ClusterName", "TaskDefinitionFamily", "db"}],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
}
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by AWS CloudFormation, you can create a change set by entering the following commands.

```
ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
```

```

ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
  --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
    ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
    ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
    ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
    ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
    ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
    ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${AWS_DEFAULT_REGION} \
  --change-set-name redis-scraping-support

```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Review the newly created changeset `redis-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}

```

6. Wait about 10 seconds, and then enter the following command.

```

aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}

```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```

ECS_LAUNCH_TYPE=FARGATE
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1

```

```

ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
  --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
    ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
    ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
    ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
    ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
    ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
    ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${AWS_DEFAULT_REGION}

```

Viewing your Redis metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric Name	Dimensions
redis_net_input_bytes_total	ClusterName, TaskDefinitionFamily
redis_net_output_bytes_total	ClusterName, TaskDefinitionFamily
redis_expired_keys_total	ClusterName, TaskDefinitionFamily
redis_evicted_keys_total	ClusterName, TaskDefinitionFamily

Metric Name	Dimensions
redis_key space_hits_total	ClusterName, TaskDefinitionFamily
redis_key space_misses_total	ClusterName, TaskDefinitionFamily
redis_memory_used_bytes	ClusterName, TaskDefinitionFamily
redis_connected_clients	ClusterName, TaskDefinitionFamily
redis_commands_total	ClusterName , TaskDefinitionFamily , cmd
redis_db_keys	ClusterName , TaskDefinitionFamily , db

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.
The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis Prometheus metrics.

To create a dashboard for Redis Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo- $\$$ ECS_CLUSTER_NAME-fargate-awsvpc
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Set up and configure Prometheus metrics collection on Amazon EKS and Kubernetes clusters

To collect Prometheus metrics from clusters running Amazon EKS or Kubernetes, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see <https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus>.

The following sections explain how to collect Prometheus metrics using the CloudWatch agent. They explain how to install the CloudWatch agent with Prometheus monitoring on clusters running Amazon EKS or Kubernetes, and how to configure the agent to scrape additional targets. They also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX

- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters](#)
- [Scraping additional Prometheus sources and importing those metrics](#)
- [\(Optional\) Set up sample containerized Amazon EKS workloads for Prometheus metric testing](#)

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Setting up IAM roles](#)
- [Installing the CloudWatch agent to collect Prometheus metrics](#)

Setting up IAM roles

The first step is to set up the necessary IAM role in the cluster. There are two methods:

- Set up an IAM role for a service account, also known as a *service role*. This method works for both the EC2 launch type and the Fargate launch type.
- Add an IAM policy to the IAM role used for the cluster. This works only for the EC2 launch type.

Set up a service role (EC2 launch type and Fargate launch type)

To set up a service role, enter the following command. Replace *MyCluster* with the name of the cluster.

```
eksctl create iamserviceaccount \  
  --name cwagent-prometheus \  
  --namespace amazon-cloudwatch \  
  --cluster MyCluster \  
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \  
  --approve \  
  --override-existing-serviceaccounts
```


Add a policy to the cluster's IAM role (EC2 launch type only)

To set up the IAM policy in a cluster for Prometheus support

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, choose **Instances**.
3. You need to find the prefix of the IAM role name for the cluster. To do this, select the check box next to the name of an instance that is in the cluster, and choose **Actions, Instance Settings, Attach/Replace IAM Role**. Then copy the prefix of the IAM role, such as `eksctl-dev303-workshop-nodegroup`.
4. Open the IAM console at <https://console.aws.amazon.com/iam/>.
5. In the navigation pane, choose **Roles**.
6. Use the search box to find the prefix that you copied earlier in this procedure, and choose that role.
7. Choose **Attach policies**.
8. Use the search box to find **CloudWatchAgentServerPolicy**. Select the check box next to **CloudWatchAgentServerPolicy**, and choose **Attach policy**.

Installing the CloudWatch agent to collect Prometheus metrics

You must install the CloudWatch agent in the cluster to collect the metrics. How to install the agent differs for Amazon EKS clusters and Kubernetes clusters.

Delete previous versions of the CloudWatch agent with Prometheus support

If you have already installed a version of the CloudWatch agent with Prometheus support in your cluster, you must delete that version by entering the following command. This is necessary only for previous versions of the agent with Prometheus support. You do not need to delete the CloudWatch agent that enables Container Insights without Prometheus support.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

Installing the CloudWatch agent on Amazon EKS clusters with the EC2 launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster

1. Enter the following command to check whether the `amazon-cloudwatch` namespace has already been created:

```
kubectl get namespace
```

2. If `amazon-cloudwatch` is not displayed in the results, create it by entering the following command:

```
kubectl create namespace amazon-cloudwatch
```

3. To deploy the agent with the default configuration and have it send data to the AWS Region that it is installed in, enter the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

To have the agent send data to a different Region instead, follow these steps:

- a. Download the YAML file for the agent by entering the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

- b. Open the file with a text editor, and search for the `cwagentconfig.json` block of the file.
- c. Add the highlighted lines, specifying the Region that you want:

```
cwagentconfig.json: |
  {
    "agent": {
      "region": "us-east-2"
    },
    "logs": { ...
```

- d. Save the file and deploy the agent using your updated file.

```
kubectl apply -f prometheus-eks.yaml
```

Installing the CloudWatch agent on Amazon EKS clusters with the Fargate launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type

1. Enter the following command to create a Fargate profile for the CloudWatch agent so that it can run inside the cluster. Replace *MyCluster* with the name of the cluster.

```
eksctl create fargateprofile --cluster MyCluster \  
--name amazon-cloudwatch \  
--namespace amazon-cloudwatch
```

2. To install the CloudWatch agent, enter the following command. Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace *region* with the name of the Region where you want the metrics to be sent. For example, *us-west-1*.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml |  
sed "s/{{cluster_name}}/MyCluster;/s/{{region_name}}/region/" |  
kubectl apply -f -
```

Installing the CloudWatch agent on a Kubernetes cluster

To install the CloudWatch agent with Prometheus support on a cluster running Kubernetes, enter the following command:

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml |
```

```
sed "s/{{cluster_name}}/MyCluster;/s/{{region_name}}/region/" |  
kubectl apply -f -
```

Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace *region* with the name of the AWS Region where you want the metrics to be sent. For example, **us-west-1**.

Verify that the agent is running

On both Amazon EKS and Kubernetes clusters, you can enter the following command to confirm that the agent is running.

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

If the results include a single CloudWatch agent pod in the Running state, the agent is running and collecting Prometheus metrics. By default the CloudWatch agent collects metrics for App Mesh, NGINX, Memcached, Java/JMX, and HAProxy every minute. For more information about those metrics, see [Prometheus metrics collected by the CloudWatch agent](#). For instructions on how to see your Prometheus metrics in CloudWatch, see [Viewing your Prometheus metrics](#)

You can also configure the CloudWatch agent to collect metrics from other Prometheus exporters. For more information, see [Scraping additional Prometheus sources and importing those metrics](#).

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in [<scrape_config>](#) in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon EKS clusters, the configurations are defined in `prometheus-eks.yaml` (for the EC2 launch type) or `prometheus-eks-fargate.yaml` (for the Fargate launch type) as two config maps:

- The name: `prometheus-config` section contains the settings for Prometheus scraping.
- The name: `prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent. You can use this section to configure how the Prometheus metrics are

collected by CloudWatch. For example, you specify which metrics are to be imported into CloudWatch, and define their dimensions.

For Kubernetes clusters running on Amazon EC2 instances, the configurations are defined in the `prometheus-k8s.yaml` YAML file as two config maps:

- The name: `prometheus-config` section contains the settings for Prometheus scraping.
- The name: `prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in [<scrape_config>](#) in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in `prometheus-eks.yaml`, the default scraping jobs are configured in the `job_name` lines in the `scrape_configs` section. In this file, the following default `kubernetes-pod-jmx` section scrapes JMX exporter metrics.

```
- job_name: 'kubernetes-pod-jmx'
  sample_limit: 10000
  metrics_path: /metrics
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels: [__address__]
    action: keep
    regex: '.*:9404$'
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
  - action: replace
    source_labels:
    - __meta_kubernetes_namespace
    target_label: Namespace
  - source_labels: [__meta_kubernetes_pod_name]
    action: replace
    target_label: pod_name
  - action: replace
    source_labels:
    - __meta_kubernetes_pod_container_name
    target_label: container_name
  - action: replace
    source_labels:
    - __meta_kubernetes_pod_controller_name
    target_label: pod_controller_name
  - action: replace
    source_labels:
    - __meta_kubernetes_pod_controller_kind
    target_label: pod_controller_kind
  - action: replace
    source_labels:
    - __meta_kubernetes_pod_phase
    target_label: pod_phase
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see [Embedding metrics within logs](#).

Log events from Amazon EKS and Kubernetes clusters are stored in the `/aws/containerinsights/cluster_name/prometheus` log group in CloudWatch Logs. Log events from Amazon ECS clusters are stored in the `/aws/ecs/containerinsights/cluster_name/prometheus` log group.

Each scraping job is contained in a different log stream in this log group. For example, the Prometheus scraping job `kubernetes-pod-appmesh-envoy` is defined for App Mesh. All App Mesh Prometheus metrics from Amazon EKS and Kubernetes clusters are sent to the log stream named `/aws/containerinsights/cluster_name>prometheus/kubernetes-pod-appmesh-envoy/`.

To add a new scraping target, you add a new `job_name` section to the `scrape_configs` section of the YAML file, and restart the agent. For an example of this process, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon EKS or Kubernetes cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses `/aws/containerinsights/cluster_name/prometheus` for logs from Amazon EKS and Kubernetes clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configuration for Amazon ECS Prometheus service discovery. For more information, see [Detailed guide for autodiscovery on Amazon ECS clusters](#).

The `ecs_service_discovery` section can contain the following fields:

- **sd_frequency** is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for docker label-based service discovery. If you omit this section, docker label-based discovery is not used. This section can contain the following fields:
 - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this docker label, the CloudWatch agent will skip it.
 - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this docker label, the agent assumes the default path `/metrics`.
 - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
 - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
 - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
 - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
 - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
 - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.

- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:

- `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

If you have multiple labels specified in `source_labels`, we recommend that you do not use `^` or `$` characters in the regular expression for `label_matcher`.

- `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
- `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
- `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
- `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [  
  {  
    "source_labels":[ "Service", "Namespace"],  
    "label_matcher": "(.*node-exporter.*|.*kube-dns.*);kube-system",  
    "dimensions":[  
      ["Service", "Namespace"]  
    ],  
    "metric_selectors":[  
      "^coredns_dns_request_type_count_total$"  
    ]  
  }  
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.

- The value of Namespace is kube-system.
- The Prometheus metric coredns_dns_request_type_count_total contains both Service and Namespace labels.

The log event that is sent includes the following highlighted section:

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Name": "coredns_dns_request_type_count_total"
        }
      ],
      "Dimensions": [
        [
          "Namespace",
          "Service"
        ]
      ],
      "Namespace": "ContainerInsights/Prometheus"
    }
  ],
  "Namespace": "kube-system",
  "Service": "kube-dns",
  "coredns_dns_request_type_count_total": 2562,
  "eks_aws_com_component": "kube-dns",
  "instance": "192.168.61.254:9153",
  "job": "kubernetes-service-endpoints",
  ...
}
```

Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics

The Kubernetes API Server exposes Prometheus metrics on endpoints by default. The official example for the Kubernetes API Server scraping configuration is available on [Github](#).

The following tutorial shows how to do the following steps to begin importing Kubernetes API Server metrics into CloudWatch:

- Adding the Prometheus scraping configuration for Kubernetes API Server to the CloudWatch agent YAML file.
- Configuring the embedded metric format metrics definitions in the CloudWatch agent YAML file.
- (Optional) Creating a CloudWatch dashboard for the Kubernetes API Server metrics.

Note

The Kubernetes API Server exposes gauge, counter, histogram, and summary metrics. In this release of Prometheus metrics support, CloudWatch imports only the metrics with gauge, counter, and summary types.

To start collecting Kubernetes API Server Prometheus metrics in CloudWatch

1. Download the latest version of the `prometheus-eks.yaml`, `prometheus-eks-fargate.yaml`, or `prometheus-k8s.yaml` file by entering one of the following commands.

For an Amazon EKS cluster with the EC2 launch type, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, find the `prometheus-config` section, and add the following section inside of that section. Then save the changes:

```

# Scrape config for API servers
- job_name: 'kubernetes-apiservers'
  kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names:
          - default
  scheme: https
  tls_config:
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    insecure_skip_verify: true
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  relabel_configs:
    - source_labels: [__meta_kubernetes_service_name,
__meta_kubernetes_endpoint_port_name]
      action: keep
      regex: kubernetes;https
    - action: replace
      source_labels:
        - __meta_kubernetes_namespace
      target_label: Namespace
    - action: replace
      source_labels:
        - __meta_kubernetes_service_name
      target_label: Service

```

3. While you still have the YAML file open in the text editor, find the `cwagentconfig.json` section. Add the following subsection and save the changes. This section puts the API server metrics onto the CloudWatch agent allow list. Three types of API Server metrics are added to the allow list:

- etcd object counts
- API Server registration controller metrics
- API Server request metrics

```

{"source_labels": ["job", "resource"],
 "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|
deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",
 "dimensions": [["ClusterName", "Service", "resource"]],
 "metric_selectors": [

```

```

    "^etcd_object_counts$"
  ]
},
{"source_labels": ["job", "name"],
  "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",
  "dimensions": [["ClusterName", "Service", "name"]],
  "metric_selectors": [
    "^workqueue_depth$",
    "^workqueue_adds_total$",
    "^workqueue_retries_total$"
  ]
},
{"source_labels": ["job", "code"],
  "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",
  "dimensions": [["ClusterName", "Service", "code"]],
  "metric_selectors": [
    "^apiserver_request_total$"
  ]
},
{"source_labels": ["job"],
  "label_matcher": "^kubernetes-apiservers",
  "dimensions": [["ClusterName", "Service"]],
  "metric_selectors": [
    "^apiserver_request_total$"
  ]
},

```

- If you already have the CloudWatch agent with Prometheus support deployed in the cluster, you must delete it by entering the following command:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

- Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter:

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster;/s/{{region_name}}/region/" \
```

```
| kubectl apply -f -
```

For a Kubernetes cluster, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Once you have done this, you should see a new log stream named **kubernetes-apiservers** in the **/aws/containerinsights/*cluster_name*/prometheus** log group. This log stream should include log events with an embedded metric format definition like the following:

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Name": "apiserver_request_total"
        }
      ],
      "Dimensions": [
        [
          "ClusterName",
          "Service"
        ]
      ],
      "Namespace": "ContainerInsights/Prometheus"
    }
  ],
  "ClusterName": "my-cluster-name",
  "Namespace": "default",
  "Service": "kubernetes",
  "Timestamp": "1592267020339",
  "Version": "0",
  "apiserver_request_count": 0,
  "apiserver_request_total": 0,
  "code": "0",
  "component": "apiserver",
  "contentType": "application/json",
  "instance": "192.0.2.0:443",
```

```
"job":"kubernetes-apiservers",
"prom_metric_type":"counter",
"resource":"pods",
"scope":"namespace",
"verb":"WATCH",
"version":"v1"
}
```

You can view your metrics in the CloudWatch console in the **ContainerInsights/Prometheus** namespace. You can also optionally create a CloudWatch dashboard for your Prometheus Kubernetes API Server metrics.

(Optional) Creating a dashboard for Kubernetes API Server metrics

To see Kubernetes API Server metrics in your dashboard, you must have first completed the steps in the previous sections to start collecting these metrics in CloudWatch.

To create a dashboard for Kubernetes API Server metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Make sure you have the correct AWS Region selected.
3. In the navigation pane, choose **Dashboards**.
4. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
5. In **Add to this dashboard**, choose **Cancel**.
6. Choose **Actions, View/edit source**.
7. Download the following JSON file: [Kubernetes API Dashboard source](#).
8. Open the JSON file that you downloaded with a text editor, and make the following changes:
 - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
 - Replace all the `{{YOUR_AWS_REGION}}` strings with the name of the Region where the metrics are collected. For example `us-west-2`. Be sure not to add whitespaces before or after the text.
9. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
10. Choose **Update, Save dashboard**.

(Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see [Prometheus metrics collected by the CloudWatch agent](#).

Before you can install any of these workloads, you must install Helm 3.x by entering the following commands:

```
brew install helm
```

For more information, see [Helm](#).

Topics

- [Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes](#)
- [Set up NGINX with sample traffic on Amazon EKS and Kubernetes](#)
- [Set up memcached with a metric exporter on Amazon EKS and Kubernetes](#)
- [Set up Java/JMX sample workload on Amazon EKS and Kubernetes](#)
- [Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes](#)
- [Tutorial for adding a new Prometheus scrape target: Redis on Amazon EKS and Kubernetes clusters](#)

Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes

Prometheus support in CloudWatch Container Insights supports AWS App Mesh. The following sections explain how to set up App Mesh.

CloudWatch Container Insights can also collect App Mesh Envoy Access Logs. For more information, see [\(Optional\) Enable App Mesh Envoy access logs](#).

Topics

- [Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster](#)
- [Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type](#)

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the EC2 launch type, or a Kubernetes cluster.

Configure IAM permissions

You must add the **AWSAppMeshFullAccess** policy to the IAM role for your Amazon EKS or Kubernetes node group. On Amazon EKS, this node group name looks similar to `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL`. On Kubernetes, it might look similar to `nodes.integ-test-kops-prometheus.k8s.local`.

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in [App Mesh Controller](#).

Install a sample application

[aws-app-mesh-examples](#) contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that shows how http routes can use headers for matching incoming requests.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: <https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-http-headers>.
2. Launch a curler pod to generate traffic:

```
kubectl -n default run -it curler --image=tutum/curl /bin/bash
```

3. Curl different endpoints by changing HTTP headers. Run the curl command multiple times, as shown:

```
curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/;
echo;

curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/;
echo;

curl -H "color_header: yellow" front.howto-k8s-http-headers.svc.cluster.local:8080/; echo;
```

4. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/  
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the Fargate launch type.

Configure IAM permissions

To set up IAM permissions, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \  
--namespace howto-k8s-fargate \  
--name appmesh-pod \  
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \  
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \  
--attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \  
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \  
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \  

```

```
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \  
--override-existing-serviceaccounts \  
--approve
```

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in [App Mesh Controller](#). Be sure to follow the instructions for Amazon EKS with the Fargate launch type.

Install a sample application

[aws-app-mesh-examples](#) contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that works for Amazon EKS clusters with the Fargate launch type.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: <https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-fargate>.

Those instructions assume that you are creating a new cluster with the correct Fargate profile. If you want to use an Amazon EKS cluster that you've already set up, you can use the following commands to set up that cluster for this demonstration. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \  
  --namespace howto-k8s-fargate \  
  --name appmesh-pod \  
  --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \  
  --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \  
  --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \  
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \  
  --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \  
  --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \  
  --override-existing-serviceaccounts \  
  --approve
```

```
eksctl create fargateprofile --cluster MyCluster \  
  --namespace howto-k8s-fargate --name howto-k8s-fargate
```

2. Port forward the front application deployment:

```
kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
```

3. Curl the front app:

```
while true; do curl -s http://localhost:8080/color; sleep 0.1; echo ; done
```

4. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/  
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up NGINX with sample traffic on Amazon EKS and Kubernetes

NGINX is a web server that can also be used as a load balancer and reverse proxy. For more information about how Kubernetes uses NGINX for ingress , see [kubernetes/ingress-nginx](#).

To install Ingress-NGINX with a sample traffic service to test Container Insights Prometheus support

1. Enter the following command to add the Helm ingress-nginx repo:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

2. Enter the following commands:

```
kubectl create namespace nginx-ingress-sample

helm install my-nginx ingress-nginx/ingress-nginx \
--namespace nginx-ingress-sample \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

3. Check whether the services started correctly by entering the following command:

```
kubectl get service -n nginx-ingress-sample
```

The output of this command should display several columns, including an EXTERNAL - IP column.

4. Set an EXTERNAL - IP variable to the value of the EXTERNAL - IP column in the row of the NGINX ingress controller.

```
EXTERNAL_IP=your-nginx-controller-external-ip
```

5. Start some sample NGINX traffic by entering the following command.

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. Enter the following command to confirm that all three pods are in the Running status.

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

If they are running, you should soon see metrics in the **ContainerInsights/Prometheus** namespace.

To uninstall NGINX and the sample traffic application

1. Delete the sample traffic service by entering the following command:

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. Delete the NGINX egress by the Helm release name.

```
helm uninstall my-nginx --namespace nginx-ingress-sample  
kubectl delete namespace nginx-ingress-sample
```

Set up memcached with a metric exporter on Amazon EKS and Kubernetes

memcached is an open-source memory object caching system. For more information, see [What is Memcached?](#).

If you are running memcached on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \  
--namespace memcached-sample --name memcached-sample
```

To install memcached with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the repo:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace memcached-sample
```

3. Enter the following command to install Memcached

```
helm install my-memcached bitnami/memcached --namespace memcached-sample \  
--set metrics.enabled=true \  
--set-string serviceAnnotations.prometheus\\.io/port="9150" \  
--set-string serviceAnnotations.prometheus\\.io/scrape="true"
```

4. Enter the following command to confirm the annotation of the running service:

```
kubectl describe service my-memcached-metrics -n memcached-sample
```

You should see the following two annotations:

```
Annotations:  prometheus.io/port: 9150
              prometheus.io/scrape: true
```

To uninstall memcached

- Enter the following commands:

```
helm uninstall my-memcached --namespace memcached-sample
kubectl delete namespace memcached-sample
```

Set up Java/JMX sample workload on Amazon EKS and Kubernetes

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

Container Insights can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter.

Default Prometheus scrape configuration

By default, the CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics from `http://CLUSTER_IP:9404/metrics` on each pod in an Amazon EKS or Kubernetes cluster. This is done by `role: pod` discovery of Prometheus `kubernetes_sd_config`. 9404 is the default port allocated for JMX Exporter by Prometheus. For more information about `role: pod` discovery, see [pod](#). You can configure the JMX Exporter to expose the metrics on a different port or `metrics_path`. If you do change the port or path, update the default `jmx scrape_config` in the CloudWatch agent config map. Run the following command to get the current CloudWatch agent Prometheus configuration:

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

The fields to change are the `/metrics` and `regex: '.*:9404$'` fields, as highlighted in the following example.

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
- action: replace
  regex: (.+)
  source_labels:
```

Other Prometheus scrape configuration

If you expose your application running on a set of pods with Java/JMX Prometheus exporters by a Kubernetes Service, you can also switch to use `role: service discovery` or `role: endpoint discovery` of Prometheus `kubernetes_sd_config`. For more information about these discovery methods, see [service](#), [endpoints](#), and [<kubernetes_sd_config>..](#)

More meta labels are provided by these two service discovery modes which could be useful for you to build the CloudWatch metrics dimensions. For example, you can relabel `__meta_kubernetes_service_name` to `Service` and include it into your metrics' dimension. For more information about customizing your CloudWatch metrics and their dimensions, see [CloudWatch agent configuration for Prometheus](#).

Docker image with JMX Exporter

Next, build a Docker image. The following sections provide two example Dockerfiles.

When you have built the image, load it into Amazon EKS or Kubernetes, and then run the following command to verify that Prometheus metrics are exposed by JMX_EXPORTER on port 9404. Replace `$JAR_SAMPLE_TRAFFIC_POD` with the running pod name and replace `$JAR_SAMPLE_TRAFFIC_NAMESPACE` with your application namespace.

If you are running JMX Exporter on a cluster with the Fargate launch type, you also need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace `MyCluster` with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \  
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE\
```



```
--name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl  
http://localhost:9404
```

Example: Apache Tomcat Docker image with Prometheus metrics

Apache Tomcat server exposes JMX mBeans by default. You can integrate JMX Exporter with Tomcat to expose JMX mBeans as Prometheus metrics. The following example Dockerfile shows the steps to build a testing image:

```
# From Tomcat 9.0 JDK8 OpenJDK  
FROM tomcat:9.0-jdk8-openjdk  
  
RUN mkdir -p /opt/jmx_exporter  
  
COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter  
COPY ./config.yaml /opt/jmx_exporter  
COPY ./setenv.sh /usr/local/tomcat/bin  
COPY your web application.war /usr/local/tomcat/webapps/  
  
RUN chmod o+x /usr/local/tomcat/bin/setenv.sh  
  
ENTRYPOINT ["catalina.sh", "run"]
```

The following list explains the four COPY lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true  
lowercaseOutputLabelNames: true  
  
rules:  
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|  
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|  
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'  
  name: java_lang_OperatingSystem_$1
```

```

type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name="\(\w+-\w+)-(\d+)\\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//[(-a-zA-Z0-9+&@#/%=?~_!|:.,;]*[-a-zA-Z0-9+&@#/%=?~_!|:.,;]*[-a-zA-Z0-9+&@#/%=?~_!|.]*), name=(-a-zA-Z0-9+/$%~_!|.)*, J2EEApplication=none, J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="\(\w+-\w+)-(\d+)\\"><>(currentThreadCount|currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=(-a-zA-Z0-9+&@#/%=?~_!|:.,;)*[-a-zA-Z0-9+&@#/%=?~_!|.]*, context=(-a-zA-Z0-9+/$%~_!|.)*><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

```

```
- pattern: ".*"
```

- `setenv.sh` is a Tomcat startup script to start the JMX exporter along with Tomcat and expose Prometheus metrics on port 9404 of the localhost. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/
jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"
```

- `your web application.war` is your web application `war` file to be loaded by Tomcat.

Build a Docker image with this configuration and upload it to an image repository.

Example: Java Jar Application Docker image with Prometheus metrics

The following example Dockerfile shows the steps to build a testing image:

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

The following list explains the four COPY lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

Here is a sample configuration file for Java and Tomcat:

```

lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_operatingsystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\\w+-\\w+)-(\\d+)\"><>(\\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//[(-a-zA-Z0-9+&@#/%=?~_!|:.,;]*[-
a-zA-Z0-9+&@#/%=?~_]|), name=(-a-zA-Z0-9+/$%~_!|.)*, J2EEApplication=none,
J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\\w+-\\w+)-(\\d+)\"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

```

```
- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?~_!|:.,;])*[-a-zA-Z0-9+&@#/%?~_!|:.,;])', context=([-a-zA-Z0-9+/$%~_!|:.,;])*><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` is the script to start the JAR application with the Prometheus metrics exported. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml -cp /opt/jmx_exporter/SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

- `SampleJavaApplication-1.0-SNAPSHOT.jar` is the sample Java application jar file. Replace it with the Java application that you want to monitor.

Build a Docker image with this configuration and upload it to an image repository.

Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes

HAProxy is an open-source proxy application. For more information, see [HAProxy](#).

If you are running HAProxy on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

To install HAProxy with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the Helm incubator repo:

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace haproxy-ingress-sample
```

3. Enter the following commands to install HAProxy:

```
helm install haproxy haproxy-ingress/haproxy-ingress \
--namespace haproxy-ingress-sample \
--set defaultBackend.enabled=true \
--set controller.stats.enabled=true \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

4. Enter the following command to confirm the annotation of the service:

```
kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
```

You should see the following annotations.

```
Annotations:  prometheus.io/port: 9101
              prometheus.io/scrape: true
```

To uninstall HAProxy

- Enter the following commands:

```
helm uninstall haproxy --namespace haproxy-ingress-sample
kubectl delete namespace haproxy-ingress-sample
```

Tutorial for adding a new Prometheus scrape target: Redis on Amazon EKS and Kubernetes clusters

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis application on Amazon EKS and Kubernetes. Redis (<https://redis.io/>) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see [redis](#).

redis_exporter (MIT License licensed) is used to expose the Redis prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see [redis_exporter](#).

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- [redis](#)
- [redis_exporter](#)

To install a sample Redis workload which exposes Prometheus metrics

1. Set the namespace for the sample Redis workload.

```
REDIS_NAMESPACE=redis-sample
```

2. If you are running Redis on a cluster with the Fargate launch type, you need to set up a Fargate profile. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \  
--namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
```

3. Enter the following command to install the sample Redis workload.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-  
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-  
prometheus/sample_traffic/redis/redis-traffic-sample.yaml \  
| sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \  
| kubectl apply -f -
```

4. The installation includes a service named `my-redis-metrics` which exposes the Redis Prometheus metric on port 9121. Enter the following command to get the details of the service:

```
kubectl describe service/my-redis-metrics -n $REDIS_NAMESPACE
```

In the Annotations section of the results, you'll see two annotations which match the Prometheus scrape configuration of the CloudWatch agent, so that it can auto-discover the workloads:

```
prometheus.io/port: 9121
```

```
prometheus.io/scrape: true
```

The related Prometheus scrape configuration can be found in the `job_name: kubernetes-service-endpoints` section of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml`.

To start collecting Redis Prometheus metrics in CloudWatch

1. Download the latest version of the of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml` file by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, and find the `cwagentconfig.json` section. Add the following subsection and save the changes. Be sure that the indentation follows the existing pattern.

```
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [{"Namespace", "ClusterName"}],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
  ]
}
```



```

    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [{"Namespace", "ClusterName", "cmd"}],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [{"Namespace", "ClusterName", "db"}],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
},

```

The section you added puts the Redis metrics onto the CloudWatch agent allow list. For a list of these metrics, see the following section.

3. If you already have the CloudWatch agent with Prometheus support deployed in this cluster, you must delete it by entering the following command.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. Replace *MyCluster* and *region* to match your settings.

For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```

cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -

```

For a Kubernetes cluster, enter this command.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster;/s/{{region_name}}/region/" \
| kubectl apply -f -
```

Viewing your Redis Prometheus metrics

This tutorial sends the following metrics to the **ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions
redis_net_input_bytes_total	ClusterName, Namespace
redis_net_output_bytes_total	ClusterName, Namespace
redis_expired_keys_total	ClusterName, Namespace
redis_evicted_keys_total	ClusterName, Namespace
redis_keyspace_hits_total	ClusterName, Namespace
redis_keyspace_misses_total	ClusterName, Namespace

Metric name	Dimensions
redis_memory_used_bytes	ClusterName, Namespace
redis_connected_clients	ClusterName, Namespace
redis_commands_total	ClusterName, Namespace , cmd
redis_db_keys	ClusterName, Namespace , db

Note

The value of the **cmd** dimension can be: `append`, `client`, `command`, `config`, `dbsize`, `flushall`, `get`, `incr`, `info`, `latency`, or `slowlog`.
The value of the **db** dimension can be `db0` to `db15`.

You can also create a CloudWatch dashboard for your Redis Prometheus metrics.

To create a dashboard for Redis Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/{{REGION_NAME}}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/{{CLUSTER_NAME}}/g" \
| sed "s/{{YOUR_NAMESPACE}}/{{NAMESPACE}}/g" \
```

Prometheus metric type conversion by the CloudWatch Agent

The Prometheus client libraries offer four core metric types:

- Counter
- Gauge
- Summary
- Histogram

The CloudWatch agent supports the counter, gauge, and summary metric types. Support for histogram metrics is planned for an upcoming release.

The Prometheus metrics with the unsupported histogram metric type are dropped by the CloudWatch agent. For more information, see [Logging dropped Prometheus metrics](#).

Gauge metrics

A Prometheus gauge metric is a metric that represents a single numerical value that can arbitrarily go up and down. The CloudWatch agent scrapes gauge metrics and send these values out directly.

Counter metrics

A Prometheus counter metric is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero. The CloudWatch agent calculates a delta from the previous scrape and sends the delta value as the metric value in the log event. So the CloudWatch agent will start to produce one log event from the second scrape and continue with subsequent scrapes, if any.

Summary metrics

A Prometheus summary metric is a complex metric type which is represented by multiple data points. It provides a total count of observations and a sum of all observed values. It calculates configurable quantiles over a sliding time window.

The sum and count of a summary metric are cumulative, but the quantiles are not. The following example shows the variance of quantiles.

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
```

```
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

The CloudWatch agent handles the sum and count of a summary metric in the same way as it handles counter metrics, as described in the previous section. The CloudWatch agent preserves the quantile values as they are originally reported.

Prometheus metrics collected by the CloudWatch agent

The CloudWatch agent with Prometheus support automatically collects metrics from several services and workloads. The metrics that are collected by default are listed in the following sections. You can also configure the agent to collect more metrics from these services, and to collect Prometheus metrics from other applications and services. For more information about collecting additional metrics, see [CloudWatch agent configuration for Prometheus](#).

Prometheus metrics collected from Amazon EKS and Kubernetes clusters are in the **ContainerInsights/Prometheus** namespace. Prometheus metrics collected from Amazon ECS clusters are in the **ECS/ContainerInsights/Prometheus** namespace.

Topics

- [Prometheus metrics for App Mesh](#)
- [Prometheus metrics for NGINX](#)
- [Prometheus metrics for Memcached](#)
- [Prometheus metrics for Java/JMX](#)
- [Prometheus metrics for HAProxy](#)

Prometheus metrics for App Mesh

The following metrics are automatically collected from App Mesh .

CloudWatch Container Insights can also collect App Mesh Envoy Access Logs. For more information, see [\(Optional\) Enable App Mesh Envoy access logs](#).

Prometheus metrics for App Mesh on Amazon EKS and Kubernetes clusters

Metric name	Dimensions
envoy_http_downstream_rq_total	ClusterName, Namespace
envoy_http_downstream_rq_xx	ClusterName, Namespace , envoy_http_conn_manager_prefix, envoy_response_code_class
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, Namespace
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, Namespace
envoy_cluster_membership_healthy	ClusterName, Namespace
envoy_cluster_membership_total	ClusterName, Namespace
envoy_server_memory_heap_size	ClusterName, Namespace
envoy_server_memory_allocated	ClusterName, Namespace

Metric name	Dimensions	
envoy_cluster_upstream_connections_timeout	ClusterName, Namespace	
envoy_cluster_upstream_requests_failed_eject	ClusterName, Namespace	
envoy_cluster_upstream_requests_overflow	ClusterName, Namespace	
envoy_cluster_upstream_request_timeout	ClusterName, Namespace	
envoy_cluster_upstream_requests_per_timeout	ClusterName, Namespace	
envoy_cluster_upstream_request_reset	ClusterName, Namespace	

Metric name	Dimensions	
envoy_cluster_upstream_connections_local_with_active_requests	ClusterName, Namespace	
envoy_cluster_upstream_connections_remote_active_requests	ClusterName, Namespace	
envoy_cluster_upstream_requests_maintenance_mode	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, Namespace	

Metric name	Dimensions
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, Namespace
envoy_cluster_upstream_flow_control_drained_total	ClusterName, Namespace
envoy_cluster_upstream_rq_retry	ClusterName, Namespace
envoy_cluster_upstream_rq_retry_success	ClusterName, Namespace
envoy_cluster_upstream_rq_retry_overflow	ClusterName, Namespace
envoy_server_live	ClusterName, Namespace
envoy_server_uptime	ClusterName, Namespace


Prometheus metrics for App Mesh on Amazon ECS clusters

Metric name	Dimensions	
envoy_http_downstream_rq_total	ClusterName, TaskDefinitionFamily	
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_healthy	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_total	ClusterName, TaskDefinitionFamily	
envoy_server_memory_heap_size	ClusterName, TaskDefinitionFamily	
envoy_server_memory_allocated	ClusterName, TaskDefinitionFamily	
envoy_cluster_upst	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
ream_cx_connect_timeout		
envoy_cluster_upstream_request_failure_eject	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_request_overflow	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_request_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_request_retry_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_request_reset	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_destroy_local_with_active_request	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_cluster_upstream_connections_active_requests	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_requests_maintenance_mode	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions
envoy_cluster_upstream_flow_control_drained_total	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_success	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_overflow	ClusterName, TaskDefinitionFamily
envoy_server_live	ClusterName, TaskDefinitionFamily
envoy_server_uptime	ClusterName, TaskDefinitionFamily
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily, envoy_http_conn_manager_prefix, envoy_response_code_class ClusterName, TaskDefinitionFamily, envoy_response_code_class

 **Note**

TaskDefinitionFamily is the Kubernetes namespace of the mesh.

The value of `envoy_http_conn_manager_prefix` can be `ingress`, `egress`, or `admin`.
 The value of `envoy_response_code_class` can be 1 (stands for 1xx), 2 stands for 2xx), 3 stands for 3xx), 4 stands for 4xx), or 5 stands for 5xx).

Prometheus metrics for NGINX

The following metrics are automatically collected from NGINX on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions
<code>nginx_ingress_controller_nginx_processor_cpu_seconds_total</code>	ClusterName, Namespace , Service
<code>nginx_ingress_controller_success</code>	ClusterName, Namespace , Service
<code>nginx_ingress_controller_requests</code>	ClusterName, Namespace , Service
<code>nginx_ingress_controller_nginx_connections</code>	ClusterName, Namespace , Service
<code>nginx_ingress_controller_nginx_processor</code>	ClusterName, Namespace , Service

Metric name	Dimensions	
ss_connections_total		
nginx_ingress_controllernginx_process_resident_memory_bytes	ClusterName, Namespace , Service	
nginx_ingress_controller_config_last_reload_successful	ClusterName, Namespace , Service	
nginx_ingress_controller_requests	ClusterName, Namespace , Service, status	

Prometheus metrics for Memcached

The following metrics are automatically collected from Memcached on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions	
memcached_current_items	ClusterName, Namespace , Service	

Metric name	Dimensions
memcached _current_ connections	ClusterName, Namespace , Service
memcached _limit_bytes	ClusterName, Namespace , Service
memcached _current_bytes	ClusterName, Namespace , Service
memcached _written_ bytes_total	ClusterName, Namespace , Service
memcached _read_byt es_total	ClusterName, Namespace , Service
memcached _items_ev icted_total	ClusterName, Namespace , Service
memcached _items_re claimed_total	ClusterName, Namespace , Service
memcached _commands _total	ClusterName, Namespace , Service ClusterName, Namespace , Service, command ClusterName, Namespace , Service, status, command

Prometheus metrics for Java/JMX

Metrics collected on Amazon EKS and Kubernetes clusters

On Amazon EKS and Kubernetes clusters, Container Insights can collect the following predefined Prometheus metrics from the Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter. For more information, see [prometheus/jmx_exporter](https://github.com/prometheus/jmx_exporter) on Github.

Java/JMX on Amazon EKS and Kubernetes clusters

Metric name	Dimensions
jvm_classes_loaded	ClusterName , Namespace
jvm_threads_current	ClusterName , Namespace
jvm_threads_daemon	ClusterName , Namespace
java_lang_operating_system_totalswapspace_size	ClusterName , Namespace
java_lang_operating_system_systemcpu_load	ClusterName , Namespace
java_lang_operating_system_processcpu_load	ClusterName , Namespace
java_lang_operating_system_free_swap_space_size	ClusterName , Namespace

Metric name	Dimensions
java_lang_operating_system_total_physical_memory_size	ClusterName , Namespace
java_lang_operating_system_free_physical_memory_size	ClusterName , Namespace
java_lang_operating_system_open_file_descriptor_count	ClusterName , Namespace
java_lang_operating_system_available_processors	ClusterName , Namespace
jvm_memory_bytes_used	ClusterName , Namespace , area
jvm_memory_pool_bytes_used	ClusterName , Namespace , pool

 **Note**

The values of the area dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon EKS and Kubernetes clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload.

Metric name	Dimensions
catalina_manager_activationsessions	ClusterName , Namespace
catalina_manager_rejectedsessions	ClusterName , Namespace
catalina_globalrequestprocessor_bytesreceived	ClusterName , Namespace
catalina_globalrequestprocessor_bytesent	ClusterName , Namespace
catalina_globalrequestprocessor_requestcount	ClusterName , Namespace


Metric name	Dimensions
catalina_globalrequestprocessor_errorcount	ClusterName , Namespace
catalina_globalrequestprocessor_processingtime	ClusterName , Namespace

Java/JMX on Amazon ECS clusters

Metric name	Dimensions
jvm_classes_loaded	ClusterName , TaskDefinitionFamily
jvm_threads_current	ClusterName , TaskDefinitionFamily
jvm_threads_daemon	ClusterName , TaskDefinitionFamily
java_lang_operatingsystem_totalswapspacesize	ClusterName , TaskDefinitionFamily
java_lang_operatingsystem_systemcpuload	ClusterName , TaskDefinitionFamily

Metric name	Dimensions	
java_lang_operating_system_processcpuload	ClusterName ,TaskDefinitionFamily	
java_lang_operating_system_free_swap_space_size	ClusterName ,TaskDefinitionFamily	
java_lang_operating_system_total_physical_memory_size	ClusterName ,TaskDefinitionFamily	
java_lang_operating_system_free_physical_memory_size	ClusterName ,TaskDefinitionFamily	
java_lang_operating_system_open_file_descriptor_count	ClusterName ,TaskDefinitionFamily	
java_lang_operating_system_available_processors	ClusterName ,TaskDefinitionFamily	

Metric name	Dimensions
jvm_memory_bytes_used	ClusterName , TaskDefinitionFamily, area
jvm_memory_pool_bytes_used	ClusterName , TaskDefinitionFamily, pool

 **Note**

The values of the area dimension can be heap or nonheap.
 The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon ECS clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload on Amazon ECS clusters.

Metric name	Dimensions
catalina_manager_active_sessions	ClusterName , TaskDefinitionFamily
catalina_manager_rejected_sessions	ClusterName , TaskDefinitionFamily
catalina_global_request_processor_bytes_received	ClusterName , TaskDefinitionFamily

Metric name	Dimensions
catalina_globalrequestprocessor_bytessent	ClusterName ,TaskDefinitionFamily
catalina_globalrequestprocessor_requestcount	ClusterName ,TaskDefinitionFamily
catalina_globalrequestprocessor_errorcount	ClusterName ,TaskDefinitionFamily
catalina_globalrequestprocessor_processingtime	ClusterName ,TaskDefinitionFamily

Prometheus metrics for HAProxy

The following metrics are automatically collected from HAProxy on Amazon EKS and Kubernetes clusters.

The metrics collected depend on which version of HAProxy Ingress that you are using. For more information about HAProxy Ingress and its versions, see [haproxy-ingress](#).

Metric name	Dimensions	Availability
haproxy_backend_bytes_in_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_bytes_out_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connection_errors_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connections_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_http_responses_total	ClusterName , Namespace , Service, code, backend	All versions of HAProxy Ingress
haproxy_backend_status	ClusterName , Namespace , Service	Only in versions 0.10 or later of HAProxy Ingress
haproxy_backend_up	ClusterName , Namespace , Service	Only in versions of HAProxy Ingress earlier than 0.10

Metric name	Dimensions	Availability
haproxy_frontend_bytes_in_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_bytes_out_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_connections_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_http_requests_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_http_responses_total	ClusterName , Namespace , Service, code, frontend	All versions of HAProxy Ingress
haproxy_frontend_request_errors_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Metric name	Dimensions	Availability
haproxy_frontend_requests_denied_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Note

The values of the code dimension can be 1xx, 2xx, 3xx, 4xx, 5xx, or other.
The values of the backend dimension can be:

- http-default-backend, http-shared-backend, or httpsback-shared-backend for HAProxy Ingress version 0.0.27 or earlier.
- _default_backend for HAProxy Ingress versions later than 0.0.27.

The values of the frontend dimension can be:

- httpfront-default-backend, httpfront-shared-frontend, or httpfronts for HAProxy Ingress version 0.0.27 or earlier.
- _front_http or _front_https for HAProxy Ingress versions later than 0.0.27.

Viewing your Prometheus metrics

You can monitor and alarm on all your Prometheus metrics including the curated pre-aggregated metrics from App Mesh, NGINX, Java/JMX, Memcached, and HAProxy, and any other manually configured Prometheus exporter you may have added. For more information about collecting metrics from other Prometheus exporters, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#).

In the CloudWatch console, Container Insights provides the following pre-built reports:

- For Amazon EKS and Kubernetes clusters, there are pre-built reports for App Mesh, NGINX, HAPROXY, Memcached, and Java/JMX.
- For Amazon ECS clusters, there are pre-built reports for App Mesh and Java/JMX.

Container Insights also provides custom dashboards for each of the workloads that Container Insights collects curated metrics from. You can download these dashboards from GitHub

To see all your Prometheus metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the list of namespaces, choose **ContainerInsights/Prometheus** or **ECS/ContainerInsights/Prometheus**.
4. Choose one of the sets of dimensions in the following list. Then select the checkbox next to the metrics that you want to see.

To see pre-built reports on your Prometheus metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Performance Monitoring**.
3. In the drop-down box near the top of the page, choose any of the Prometheus options.

In the other drop-down box, choose a cluster to view

We have also provided custom dashboards for NGINX, App Mesh, Memcached, HAProxy, and Java/JMX.

To use a custom dashboard that Amazon has provided

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Dashboards**.
3. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
4. In **Add to this dashboard**, choose **Cancel**.
5. Choose **Actions, View/edit source**.
6. Download one of the following JSON files:
 - [NGINX custom dashboard source on Github](#).
 - [App Mesh custom dashboard source on Github](#).
 - [Memcached custom dashboard source on Github](#)

- [HAProxy-Ingress custom dashboard source on Github](#)
 - [Java/JMX custom dashboard source on Github](#).
7. Open the JSON file that you downloaded with a text editor, and make the following changes:
 - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
 - Replace all the `{{YOUR_REGION}}` strings with the AWS Region where your cluster is running. For example, **us-west-1** Make sure not to add whitespaces before or after the text.
 - Replace all the `{{YOUR_NAMESPACE}}` strings with the exact namespace of your workload.
 - Replace all the `{{YOUR_SERVICE_NAME}}` strings with the exact service name of your workload. For example, **haproxy-haproxy-ingress-controller-metrics**
 8. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
 9. Choose **Update, Save dashboard**.

Prometheus metrics troubleshooting

This section provides help for troubleshooting your Prometheus metrics setup.

Topics

- [Prometheus metrics troubleshooting on Amazon ECS](#)
- [Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters](#)

Prometheus metrics troubleshooting on Amazon ECS

This section provides help for troubleshooting your Prometheus metrics setup on Amazon ECS clusters.

I don't see Prometheus metrics sent to CloudWatch Logs

The Prometheus metrics should be ingested as log events in the log group `/aws/ecs/containerinsights/cluster-name/Prometheus`. If the log group is not created or the Prometheus metrics are not sent to the log group, you will need to first check whether the Prometheus targets have been successfully discovered by the CloudWatch agent. And next check the security group and permission settings of the CloudWatch agent. The following steps guide you to do the debugging.

Step 1: Enable the CloudWatch agent debugging mode

First, change the CloudWatch agent to debug mode by adding the following bold lines to your AWS CloudFormation template file, `cwagent-ecs-prometheus-metric-for-bridge-host.yaml` or `cwagent-ecs-prometheus-metric-for-awsvpc.yaml`. Then save the file.

```

cwagentconfig.json: |
  {
    "agent": {
      "debug": true
    },
    "logs": {
      "metrics_collected": {

```

Create a new AWS CloudFormation changeset against the existing stack. Set other parameters in the changeset to the same values as in your existing AWS CloudFormation stack. The following example is for a CloudWatch agent installed in an Amazon ECS cluster using the EC2 launch type and the bridge network mode.

```

ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
  --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
  --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
    ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
    ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
    ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
    ParameterKey=ExecutionRoleName,ParameterValue=${ECS_EXECUTION_ROLE_NAME}
\
  --capabilities CAPABILITY_NAMED_IAM \
  --region $AWS_REGION \
  --change-set-name $NEW_CHANGESET_NAME

```

Go to the AWS CloudFormation console to review the new changeset, `$NEW_CHANGESET_NAME`. There should be one change applied to the **CWAgentConfigSSMParameter** resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

Wait about 10 seconds and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

Step 2: Check the ECS service discovery logs

The ECS task definition of the CloudWatch agent enables the logs by default in the section below. The logs are sent to CloudWatch Logs in the log group **/ecs/ecs-cwagent-prometheus**.

```
LogConfiguration:
  LogDriver: awslogs
  Options:
    awslogs-create-group: 'True'
    awslogs-group: "/ecs/ecs-cwagent-prometheus"
    awslogs-region: !Ref AWS::Region
    awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

Filter the logs by the string `ECS_SD_Stats` to get the metrics related to the ECS service discovery, as shown in the following example.

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

The meaning of each metric for a particular ECS service discovery cycle is as follows:

- **AWSCLI_DescribeContainerInstances** – the number of `ECS::DescribeContainerInstances` API calls made.
- **AWSCLI_DescribeInstancesRequest** – the number of `ECS::DescribeInstancesRequest` API calls made.
- **AWSCLI_DescribeTaskDefinition** – the number of `ECS::DescribeTaskDefinition` API calls made.
- **AWSCLI_DescribeTasks** – the number of `ECS::DescribeTasks` API calls made.
- **AWSCLI_ListTasks** – the number of `ECS::ListTasks` API calls made.
- **ExporterDiscoveredTargetCount** – the number of Prometheus targets that were discovered and successfully exported into the target result file within the container.
- **LRUCache_Get_EC2MetaData** – the number of times that container instances metadata was retrieved from the cache.
- **LRUCache_Get_TaskDefinition** – the number of times that ECS task definition metadata was retrieved from the cache.
- **LRUCache_Size_ContainerInstance** – the number of unique container instance's metadata cached in memory.
- **LRUCache_Size_TaskDefinition** – the number of unique ECS task definitions cached in memory.
- **Latency** – how long the service discovery cycle takes.

Check the value of `ExporterDiscoveredTargetCount` to see whether the discovered Prometheus targets match your expectations. If not, the possible reasons are as follows:

- The configuration of ECS service discovery might not match your application's setting. For the docker label-based service discovery, your target containers may not have the necessary docker label configured in the CloudWatch agent to auto discover them. For the ECS task definition ARN regular expression-based service discovery, the regex setting in the CloudWatch agent may not match your application's task definition.
- The CloudWatch agent's ECS task role might not have permission to retrieve the metadata of ECS tasks. Check that the CloudWatch agent has been granted the following read-only permissions:
 - `ec2:DescribeInstances`
 - `ecs:ListTasks`
 - `ecs:DescribeContainerInstances`
 - `ecs:DescribeTasks`

- `ecs:DescribeTaskDefinition`

Step 3: Check the network connection and the ECS task role policy

If there are still no log events sent to the target CloudWatch Logs log group even though the value of `Exporter_DiscoveredTargetCount` indicates that there are discovered Prometheus targets, this could be caused by one of the following:

- The CloudWatch agent might not be able to connect to the Prometheus target ports. Check the security group setting behind the CloudWatch agent. The private IP should allow the CloudWatch agent to connect to the Prometheus exporter ports.
- The CloudWatch agent's ECS task role might not have the **CloudWatchAgentServerPolicy** managed policy. The CloudWatch agent's ECS task role needs to have this policy to be able to send the Prometheus metrics as log events. If you used the sample AWS CloudFormation template to create the IAM roles automatically, both the ECS task role and the ECS execution role are granted with the least privilege to perform the Prometheus monitoring.

Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters

This section provides help for troubleshooting your Prometheus metrics setup on Amazon EKS and Kubernetes clusters.

General troubleshooting steps on Amazon EKS

To confirm that the CloudWatch agent is running, enter the following command.

```
kubectl get pod -n amazon-cloudwatch
```

The output should include a row with `cwagent-prometheus-id` in the NAME column and Running in the STATUS column.

To display details about the running pod, enter the following command. Replace `pod-name` with the complete name of your pod that has a name that starts with `cw-agent-prometheus`.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

If you have CloudWatch Container Insights installed, you can use CloudWatch Logs Insights to query the logs from the CloudWatch agent collecting the Prometheus metrics.

To query the application logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the log group for the application logs, **/aws/containerinsights/*cluster-name*/application**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,
kubernetes.pod_name, log |
filter haskubernetes_pod_name and kubernetes.pod_name like /cwagent-prometheus
```

You can also confirm that Prometheus metrics and metadata are being ingested as CloudWatch Logs events.

To confirm that Prometheus data is being ingested

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the **/aws/containerinsights/*cluster-name*/prometheus**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

Logging dropped Prometheus metrics

This release does not collect Prometheus metrics of the histogram type. You can use the CloudWatch agent to check whether any Prometheus metrics are being dropped because they are histogram metrics. You can also log a list of the first 500 Prometheus metrics that are dropped and not sent to CloudWatch because they are histogram metrics.

To see whether any metrics are being dropped, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

If any metrics are being dropped, you will see the following lines in the `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` file.

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are supported.
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

If you see those lines and want to know what metrics are being dropped, use the following steps.

To log a list of dropped Prometheus metrics

1. Change the CloudWatch agent to debug mode by adding the following bold lines to your `prometheus-eks.yaml` or `prometheus-k8s.yaml` file, and save the file.

```
{
  "agent": {
    "debug": true
  },
```

This section of the file should then look like this:

```
cwagentconfig.json: |
  {
    "agent": {
      "debug": true
    },
    "logs": {
      "metrics_collected": {
```

2. Reinstall the CloudWatch agent to enable debug mode by entering the following commands:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
kubectl apply -f prometheus.yaml
```

The dropped metrics are logged in the CloudWatch agent pod.

3. To retrieve the logs from the CloudWatch agent pod, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

Or, if you have Container Insights Fluentd logging installed, the logs are also saved in the CloudWatch Logs log group `/aws/containerinsights/cluster_name/application`.

To query these logs, you can follow the steps for querying the application logs in [General troubleshooting steps on Amazon EKS](#).

Where are the Prometheus metrics ingested as CloudWatch Logs log events?

The CloudWatch agent creates a log stream for each Prometheus scrape job configuration. For example, in the `prometheus-eks.yaml` and `prometheus-k8s.yaml` files, the line `job_name: 'kubernetes-pod-appmesh-envoy'` scrapes App Mesh metrics. The Prometheus target is defined as `kubernetes-pod-appmesh-envoy`. So all App Mesh Prometheus metrics are ingested as CloudWatch Logs events in the log stream `kubernetes-pod-appmesh-envoy` under the log group named `/aws/containerinsights/cluster-name/Prometheus`.

I don't see Amazon EKS or Kubernetes Prometheus metrics in CloudWatch metrics

First, make sure that the Prometheus metrics are ingested as log events in the log group `/aws/containerinsights/cluster-name/Prometheus`. Use the information in [Where are the Prometheus metrics ingested as CloudWatch Logs log events?](#) to help you check the target log stream. If the log stream is not created or there are no new log events in the log stream, check the following:

- Check that the Prometheus metrics exporter endpoints are set up correctly
- Check that the Prometheus scraping configurations in the `config map: cwagent-prometheus` section of the CloudWatch agent YAML file is correct. The configuration should be the same as it would be in a Prometheus configuration file. For more information, see [<scrape_config>](#) in the Prometheus documentation.

If the Prometheus metrics are ingested as log events correctly, check that the embedded metric format settings are added into the log events to generate the CloudWatch metrics.

```
"CloudWatchMetrics":[
  {
    "Metrics":[
      {
        "Name":"envoy_http_downstream_cx_destroy_remote_active_rq"
      }
    ],
  },
]
```

```
    "Dimensions": [
      [
        "ClusterName",
        "Namespace"
      ]
    ],
    "Namespace": "ContainerInsights/Prometheus"
  }
],
```

For more information about embedded metric format, see [Specification: Embedded metric format](#).

If there is no embedded metric format in the log events, check that the `metric_declaration` section is configured correctly in the `config map: prometheus-cwagentconfig` section of the CloudWatch agent installation YAML file. For more information, see [Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics](#).

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see [Amazon CloudWatch Application Insights](#).

You can enable Application Insights to gather additional data from your containerized applications and microservices. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard.

For more information about Application Insights and containerized applications, see [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring](#).

See Amazon ECS lifecycle events within Container Insights

You can view Amazon ECS lifecycle events within the Container Insights console. This helps you correlate your container metrics, logs, and events in a single view to give you a more complete operational visibility.

The events include container instance state change events, task state change events, and service action events. They are automatically sent by Amazon ECS to Amazon EventBridge and are also

collected in CloudWatch in event log format. For more information about these events, see [Amazon ECS events](#).

Standard Container Insights pricing applies for Amazon ECS Lifecycle events. For more information, see [Amazon CloudWatch Pricing](#).

To configure the table of lifecycle events and create rules for a cluster, you must have the `events:PutRule`, `events:PutTargets`, and `logs:CreateLogGroup` permissions. You must also make sure that there is a resource policy that enables EventBridge to create the log stream and send logs to CloudWatch Logs. If this resource policy doesn't exist, you can enter the following command to create it:

```
aws --region region logs put-resource-policy --policy-name 'EventBridgeCloudWatchLogs'
--policy-document '{
  "Statement": [
    {
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow",
      "Principal": {
        "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
      },
      "Resource": "arn:aws:logs:region:account-id:log-group:/aws/events/ecs/
containerinsights/*:*'",
      "Sid": "TrustEventBridgeToStoreECSLifecycleLogEvents"
    }
  ],
  "Version": "2012-10-17"
}'
```

You can use the following command to check whether you already have this policy, and to confirm that attaching it worked correctly.

```
aws logs describe-resource-policies --region region --output json
```

To view the table of lifecycle events, you must have the `events:DescribeRule`, `events>ListTargetsByRule`, and `logs:DescribeLogGroups` permissions.

To view Amazon ECS lifecycle events in the CloudWatch Container Insights console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Insights, Container Insights**.
3. Choose **View performance dashboards**.
4. In the next drop-down, choose either **ECS Clusters**, **ECS Services**, or **ECS Tasks**.
5. If you chose **ECS Services** or **ECS Tasks** in the previous step, choose the **Lifecycle events** tab.
6. At the bottom of the page, if you see **Configure lifecycle events**, choose it to create EventBridge rules for your cluster.

The events are displayed below the container insights panes and above the Application Insights section. To run extra analytics and create additional visualizations on these events, choose **View in Logs Insights** in the Lifecycle Events table.

Troubleshooting Container Insights

The following sections can help if you're having trouble issues with Container Insights.

Failed deployment on Amazon EKS or Kubernetes

If the agent doesn't deploy correctly on a Kubernetes cluster, try the following:

- Run the following command to get the list of pods.

```
kubectl get pods -n amazon-cloudwatch
```

- Run the following command and check the events at the bottom of the output.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Unauthorized panic: Cannot retrieve cadvisor data from kubelet

If your deployment fails with the error `Unauthorized panic: Cannot retrieve cadvisor data from kubelet`, your kubelet might not have Webhook authorization mode enabled. This mode is required for Container Insights. For more information, see [Verify prerequisites](#).

Deploying Container Insights on a deleted and re-created cluster on Amazon ECS

If you delete an existing Amazon ECS cluster that does not have Container Insights enabled, and you re-create it with the same name, you can't enable Container Insights on this new cluster at the time you re-create it. You can enable it by re-creating it, and then entering the following command:

```
aws ecs update-cluster-settings --cluster myCIcluster --settings
name=containerInsights,value=enabled
```

Invalid endpoint error

If you see an error message similar to the following, check to make sure that you replaced all the placeholders such as *cluster-name* and *region-name* in the commands that you are using with the correct information for your deployment.

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:
invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://
logs.{{region_name}}.amazonaws.com/\", Err:\"{\\\"}, &awserr.BaseError{code:
\"InvalidEndpointURL\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)
(0xc0008723c0)}}\\n\",
```

Metrics don't appear in the console

If you don't see any Container Insights metrics in the AWS Management Console, be sure that you have completed the setup of Container Insights. Metrics don't appear before Container Insights has been set up completely. For more information, see [Setting up Container Insights](#).

Pod metrics missing on Amazon EKS or Kubernetes after upgrading cluster

This section might be useful if all or some pod metrics are missing after you deploy the CloudWatch agent as a daemonset on a new or upgraded cluster, or you see an error log with the message `W! No pod metric collected`.

These errors can be caused by changes in the container runtime, such as containerd or the docker systemd cgroup driver. You can usually solve this by updating your deployment manifest so that the containerd socket from the host is mounted into the container. See the following example:

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-
insights/blob/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/cwagent/cwagent-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  template:
    spec:
      containers:
        - name: cloudwatch-agent
# ...
        # Don't change the mountPath
        volumeMounts:
# ...
          - name: dockersock
            mountPath: /var/run/docker.sock
            readOnly: true
          - name: varlibdocker
            mountPath: /var/lib/docker
            readOnly: true
          - name: containerdsock # NEW mount
            mountPath: /run/containerd/containerd.sock
            readOnly: true
# ...
      volumes:
# ...
        - name: dockersock
          hostPath:
            path: /var/run/docker.sock
        - name: varlibdocker
          hostPath:
            path: /var/lib/docker
        - name: containerdsock # NEW volume
          hostPath:
            path: /run/containerd/containerd.sock
```


No pod metrics when using Bottlerocket for Amazon EKS

Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers.

Bottlerocket uses a different containerd path on the host, so you need to change the volumes to its location. If you don't, you see an error in the logs that includes `W! No pod metric collected`. See the following example.

```
volumes:
  # ...
  - name: containerdsock
    hostPath:
      # path: /run/containerd/containerd.sock
      # bottlerocket does not mount containerd sock at normal place
      # https://github.com/bottlerocket-os/bottlerocket/
      commit/91810c85b83ff4c3660b496e243ef8b55df0973b
      path: /run/dockerhim.sock
```

No container filesystem metrics when using the containerd runtime for Amazon EKS or Kubernetes

This is a known issue and is being worked on by community contributors. For more information, see [Disk usage metric for containerd](#) and [container file system metrics is not supported by cadvisor for containerd](#) on GitHub.

Unexpected log volume increase from CloudWatch agent when collecting Prometheus metrics

This was a regression introduced in version 1.247347.6b250880 of the CloudWatch agent. This regression has already been fixed in more recent versions of the agent. Its impact was limited to scenarios where customers collected the logs of the CloudWatch agent itself and were also using Prometheus. For more information, see [\[prometheus\] agent is printing all the scraped metrics in log](#) on GitHub.

Latest docker image mentioned in release notes not found from Dockerhub

We update the release note and tag on Github before we start the actual release internally. It usually takes 1-2 weeks to see the latest docker image on registries after we bump the version

number on Github. There is no nightly release for the CloudWatch agent container image. You can build the image directly from source at the following location: <https://github.com/aws/amazon-cloudwatch-agent/tree/main/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile>

CrashLoopBackoff error on the CloudWatch agent

If you see a `CrashLoopBackOff` error for the CloudWatch agent, make sure that your IAM permissions are set correctly. For more information, see [Verify prerequisites](#).

CloudWatch agent or Fluentd pod stuck in pending

If you have a CloudWatch agent or Fluentd pod stuck in `Pending` or with a `FailedScheduling` error, determine if your nodes have enough compute resources based on the number of cores and amount of RAM required by the agents. Enter the following command to describe the pod:

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

Building your own CloudWatch agent Docker image

You can build your own CloudWatch agent Docker image by referring to the Dockerfile located at <https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/latest/cloudwatch-agent-dockerfile/Dockerfile>.

The Dockerfile supports building multi-architecture images directly using `docker buildx`.

Deploying other CloudWatch agent features in your containers

You can deploy additional monitoring features in your containers using the CloudWatch agent. These features include the following:

- **Embedded Metric Format**— For more information, see [Embedding metrics within logs](#).
- **StatsD**— For more information, see [Retrieve custom metrics with StatsD](#).

Instructions and necessary files are located on GitHub at the following locations:

- For Amazon ECS containers, see [Example Amazon ECS task definitions based on deployment modes](#).

- For Amazon EKS and Kubernetes containers, see [Example Kubernetes YAML files based on deployment modes](#).

Lambda Insights

CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. The solution collects, aggregates, and summarizes system-level metrics including CPU time, memory, disk, and network. It also collects, aggregates, and summarizes diagnostic information such as cold starts and Lambda worker shutdowns to help you isolate issues with your Lambda functions and resolve them quickly.

Lambda Insights uses a new CloudWatch Lambda extension, which is provided as a Lambda layer. When you install this extension on a Lambda function, it collects system-level metrics and emits a single performance log event for every invocation of that Lambda function. CloudWatch uses embedded metric formatting to extract metrics from the log events.

For more information about Lambda extensions, see [Using AWS Lambda extensions](#). For more information about embedded metric format, see [Embedding metrics within logs](#).

You can use Lambda Insights with any Lambda function that uses a Lambda runtime that supports Lambda extensions. For a list of these runtimes, see [Lambda Extensions API](#).

Pricing

For each Lambda function enabled for Lambda Insights, you only pay for what you use for metrics and logs. For a pricing example, see [Amazon CloudWatch Pricing](#).

You are charged for the execution time consumed by the Lambda extension, in 1ms increments. For more information about Lambda pricing, see [AWS Lambda Pricing](#).

Get started with Lambda Insights

To enable Lambda Insights on a Lambda function, you can use a one-click toggle in the Lambda console. Alternatively, you can use the AWS CLI, AWS CloudFormation, the AWS Serverless Application Model CLI, or the AWS Cloud Development Kit (AWS CDK).

The following sections provide detailed instructions for completing these steps.

Topics

- [Available versions of the Lambda Insights extension](#)

- [Use the console to enable Lambda Insights on an existing Lambda function](#)
- [Use the AWS CLI to enable Lambda Insights on an existing Lambda function](#)
- [Use the AWS SAM CLI to enable Lambda Insights on an existing Lambda function](#)
- [Use AWS CloudFormation to enable Lambda Insights on an existing Lambda function](#)
- [Use the AWS CDK to enable Lambda Insights on an existing Lambda function](#)
- [Use Serverless Framework to enable Lambda Insights on an existing Lambda function](#)
- [Enable Lambda Insights on a Lambda container image deployment](#)
- [Update the Lambda Insights extension version on a function](#)

Available versions of the Lambda Insights extension

This section lists the versions of the Lambda Insights extension, and the ARNs to use for these extensions in each AWS Region.

Topics

- [x86-64 platforms](#)
- [ARM64 platforms](#)

x86-64 platforms

This section lists the versions of the Lambda Insights extension for x86-64 platforms, and the ARNs to use for these extensions in each AWS Region.

Important

Lambda Insights extensions 1.0.317.0 and later don't support Amazon Linux 1.

1.0.333.0

Version 1.0.333.0 includes bug fixes.

ARNs for version 1.0.333.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:53
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:53
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:53
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:53
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:44
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:44
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:26
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:30
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:21
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:34
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:52

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:53
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:53
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:80
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:52
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:13
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:53
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:53
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:53
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:44
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:52

Region	ARN
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:28</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:50</code>
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:27</code>
Israel (Tel Aviv)	<code>arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:21</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:44</code>
Middle East (UAE)	<code>arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:27</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:52</code>
AWS GovCloud (US-East)	<code>arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:21</code>
AWS GovCloud (US-West)	<code>arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:21</code>

1.0.317.0

Version 1.0.317.0 includes the removal of support for the Amazon Linux 1 platform, and bug fixes. It also includes support for AWS GovCloud (US) Regions.

ARNs for version 1.0.317.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:52</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:52</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:52</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:52</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:43</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:43</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:25</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:29</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:20</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:50</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:33</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:51</code>

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:52
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:52
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:79
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:51
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:12
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:42
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:42
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:52
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:52
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:52
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:43
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:51

Region	ARN
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:27</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:49</code>
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:26</code>
Israel (Tel Aviv)	<code>arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:20</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:43</code>
Middle East (UAE)	<code>arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:26</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:51</code>
AWS GovCloud (US-East)	<code>arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:19</code>
AWS GovCloud (US-West)	<code>arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:19</code>

1.0.295.0

Version 1.0.295.0 includes dependency updates for all the compatible runtimes.

ARNs for version 1.0.295.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:51</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:51</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:51</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:51</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:42</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:42</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:24</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:28</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:19</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:49</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:32</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:50</code>

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:78
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:50
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:11
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:41
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:41
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:51
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:51
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:51
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:42
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:50

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:26
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:48
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:25
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:19
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:42
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:25
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:50

1.0.275.0

Version 1.0.275.0 includes important dependency updates for all the compatible runtimes.

ARNs for version 1.0.275.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:49
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:49

Region	ARN
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:49</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:49</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:40</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:40</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:22</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:26</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:17</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:47</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:30</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:48</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:49</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:49</code>

Region	ARN
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:76</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:48</code>
Canada West (Calgary)	<code>arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:9</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:39</code>
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:39</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:49</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:49</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:49</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:40</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:48</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:24</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:46</code>

Region	ARN
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:23</code>
Israel (Tel Aviv)	<code>arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:17</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:40</code>
Middle East (UAE)	<code>arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:23</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:48</code>

1.0.273.0

Version 1.0.273.0 includes important bug fixes for all compatible runtimes, and adds support for Canada West (Calgary).

ARNs for version 1.0.273.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:45</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:45</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:45</code>

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:45
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:35
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:35
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:21
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:12
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:43
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:26
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:45
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:45
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:72

Region	ARN
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:44</code>
Canada West (Calgary)	<code>arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:4</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:36</code>
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:36</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:45</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:45</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:45</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:35</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:44</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:19</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:42</code>
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:17</code>

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:12
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:35
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:18
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:44

1.0.229.0

Version 1.0.229.0 includes important bug fixes for all compatible runtimes, and adds support for the Israel (Tel Aviv) Region.

ARNs for version 1.0.229.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:38
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:38
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:38
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:38

Region	ARN
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:28</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:28</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:10</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Melbourne)	<code>arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:5</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:36</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:19</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:37</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:38</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:38</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:60</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:37</code>

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:29
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:29
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:38
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:38
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:38
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:28
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:37
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:12
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:35
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:11
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:28

Region	ARN
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:11
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:37

1.0.178.0

Version 1.0.178.0 adds support for the following AWS Regions.

- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Europe (Spain)
- Europe (Zurich)
- Middle East (UAE)

ARNs for version 1.0.178.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:35
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:33
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:33
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:33

Region	ARN
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:25</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:25</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:8</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:31</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:32</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:33</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:33</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:50</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:32</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:26</code>

Region	ARN
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:26</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:35</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:33</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:33</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:25</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:32</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:10</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:30</code>
Europe (Zurich)	<code>arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:7</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:25</code>
Middle East (UAE)	<code>arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:9</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:32</code>

1.0.143.0

Version 1.0.143.0 includes bug fixes in compatibility with Python 3.7 and Go 1.x. The Python 3.6 Lambda runtime is being deprecated. For more information, see [Lambda runtimes](#).

ARNs for version 1.0.143.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:21
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:21
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:20
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:21
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:13
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:13
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:21
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:20

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:21
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:21
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:32
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:20
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:14
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:14
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:21
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:21
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:21
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:13
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:20

Region	ARN
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:13
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:20

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about tmp directory storage quotas, see [Lambda quotas](#).

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

ARNs for version 1.0.135.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:18
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:18
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:18
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:18

Region	ARN
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:1</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:18</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:25</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:18</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:11</code>
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:11</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:18</code>

Region	ARN
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:18</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:18</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:11</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:18</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:18</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:11</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:18</code>

1.0.119.0

ARNs for version 1.0.119.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:16</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:16</code>

Region	ARN
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:16</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:16</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:9</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:9</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:16</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:16</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:16</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:16</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:23</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:16</code>
China (Beijing)	<code>arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:9</code>
China (Ningxia);	<code>arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:9</code>

Region	ARN
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:16</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:16</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:16</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:9</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:16</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:16</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:9</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:16</code>

1.0.98.0

This version removes unnecessary logging and also addresses an issue with the AWS Serverless Application Model CLI local invokes. For more information about this issue, see [Adding LambdaInsightsExtension results in timeout with 'sam local invoke'](#).

ARNs for version 1.0.98.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:14</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:14</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:14</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:8</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:8</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:14</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:14</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:14</code>

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:8
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:8
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:14
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:14
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:14
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:8
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:14
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:8
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:14

1.0.89.0

This version corrects the performance event timestamp to always represent the start of the invocation of the function.

ARNs for version 1.0.89.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:12</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:12</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:12</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:12</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:12</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:12</code>

Region	ARN
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:12</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:12</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:12</code>

1.0.86.0

With version 1.0.54.0 of the extension, memory metrics were sometimes reported incorrectly and sometimes were higher than 100%. Version 1.0.86.0 corrects the memory measurement issue by using the same event data as Lambda platform metrics. This means that you may see a dramatic change in the recorded memory metric values. This is achieved by using the new Lambda Logs API. This provides a more accurate measurement of Lambda sandbox memory usage. However, something to be aware of is that the Lambda Logs API can't deliver platform report events if a function sandbox times out and is subsequently spun down. In this case, Lambda Insights is unable to record the invocation metrics. For more information about Lambda Logs API, see [AWS Lambda Logs API](#).

New features in version 1.0.86.0

- Uses the Lambda Logs API to correct the memory metric. This solves the previous issue where memory statistics were greater than 100%.
- Introduces `Init Duration` as a new CloudWatch metric.
- Uses the invocation ARN to add a **version** dimension for aliases and invoked versions. If you are using Lambda aliases or versions to achieve incremental deployments (such as blue-green deployments), you can view your metrics based on the invoked alias. The **version** dimension is

not applied if the function doesn't use an alias or version. For more information, see [Lambda function aliases](#).

- Adds a `billed_mb_ms` field to the performance events to display the cost per invoke. This does not consider any cost associated with provisioned concurrency.
- Adds `billed_duration` and `duration` fields to the performance events.

ARNs for version 1.0.86.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:11</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:11</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:11</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:11</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:11</code>

Region	ARN
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:11</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:11</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:11</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:11</code>

1.0.54.0

Version 1.0.54.0 was the initial release of the Lambda Insights extension.

ARNs for version 1.0.54.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:2</code>

Region	ARN
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:2</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:2</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:2</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:2</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:2</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:2</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:2</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:2</code>

Region	ARN
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:2</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:2</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:2</code>

ARM64 platforms

This section lists the versions of the Lambda Insights extension for ARM64 platforms, and the ARNs to use for these extensions in each AWS Region.

Important

Lambda Insights extensions 1.0.317.0 and later don't support Amazon Linux 1.

1.0.333.0

Version 1.0.333.0 includes bug fixes.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:22</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>

Region	ARN
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:18</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:18</code>
Asia Pacific (Hyderabad)	<code>arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:6</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:18</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:22</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:17</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:31</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>

Region	ARN
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:18</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:6</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:18</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
AWS GovCloud (US-East)	<code>arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:3</code>
AWS GovCloud (US-West)	<code>arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:3</code>

1.0.317.0

Version 1.0.317.0 includes the removal of support for the Amazon Linux 1 platform, and bug fixes. It also includes support for AWS GovCloud (US) Regions.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:21
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:5
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:19

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:30
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:17
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:17
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:5
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:17
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:17

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:1
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:1

1.0.295.0

Version 1.0.295.0 includes dependency updates for all compatible runtimes.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:4
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:16

Region	ARN
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:15
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:16
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:16

Region	ARN
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:4</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:16</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:16</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16</code>

1.0.275.0

Version 1.0.275.0 includes bug fixes for all compatible runtimes and support for the Europe (Spain) and Asia Pacific (Hyderabad) Regions.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:18</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:14</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:14</code>

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:2
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:13
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:15
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16

Region	ARN
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:14</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:14</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>

1.0.273.0

Version 1.0.273.0 includes bug fixes for all compatible runtimes.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:9</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>

Region	ARN
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:9</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:9</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:9</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:14</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:9</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:11</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:23</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:10</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>

Region	ARN
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12</code>
Europe (Milan)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:9</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:10</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:10</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:9</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:10</code>

1.0.229.0

Version 1.0.229.0 includes bug fixes for all compatible runtimes. It also adds support for the following Regions:

- US West (N. California)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Jakarta)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Canada (Central)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)

- South America (São Paulo)

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:7</code>
US West (N. California)	<code>arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:3</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
Africa (Cape Town)	<code>arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Hong Kong)	<code>arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Jakarta)	<code>arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:7</code>
Asia Pacific (Osaka)	<code>arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Seoul)	<code>arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:4</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>

Region	ARN
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:11</code>
Canada (Central)	<code>arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:3</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5</code>
Europe (Spain)	<code>arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Paris)	<code>arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:3</code>
Europe (Stockholm)	<code>arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:3</code>
Middle East (Bahrain)	<code>arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:2</code>
South America (São Paulo)	<code>arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:3</code>

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about tmp directory storage quotas, see [Lambda quotas](#).

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

Region	ARN
US East (N. Virginia)	<code>arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
US East (Ohio)	<code>arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
US West (Oregon)	<code>arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Mumbai)	<code>arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Singapore)	<code>arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Sydney)	<code>arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Asia Pacific (Tokyo)	<code>arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Frankfurt)	<code>arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (Ireland)	<code>arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>
Europe (London)	<code>arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2</code>

1.0.119.0

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1

Use the console to enable Lambda Insights on an existing Lambda function

Use the following steps in the Lambda Console to enable Lambda Insights on an existing Lambda function.

To enable Lambda Insights on a Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose the name of a function, and then select the **Configuration** tab on the following screen.
3. Under the **Configuration** tab, choose **Monitoring and operations tools** in the left navigation menu, and then choose **Edit**.

You're directed to a screen where you can edit monitoring tools.

4. By **Lambda Insights enhanced monitoring**, choose **Edit**.
5. Under **CloudWatch Lambda Insights**, enable **Enhanced monitoring**, and then choose **Save**.

Use the AWS CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CLI to enable Lambda Insights on an existing Lambda function.

Step 1: Update function permissions

To update the function's permissions

- Attach the **CloudWatchLambdaInsightsExecutionRolePolicy** managed IAM policy to the function's execution role by entering the following command.

```
aws iam attach-role-policy \  
--role-name function-execution-role \  
--policy-arn "arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy"
```

Step 2: Install the Lambda extension

Install the Lambda extension by entering the following command. Replace the ARN value for the layers parameter with the ARN matching your Region and the extension version that you want to use. For more information, see [Available versions of the Lambda Insights extension](#).

```
aws lambda update-function-configuration \  
--function-name function-name \  
--layers "arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14"
```

Step 3: Enable the CloudWatch Logs VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint.

If you need to do this step, enter the following command, replacing the placeholders with information for your VPC.

For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

```
aws ec2 create-vpc-endpoint \  
--vpc-id vpcId \  
--vpc-endpoint-type Interface \  
--service-name com.amazonaws.region.logs \  
--subnet-id subnetId \  
--security-group-id securitygroupId
```

Use the AWS SAM CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS SAM CLI to enable Lambda Insights on an existing Lambda function.

If you don't already have the latest version of the AWS SAM CLI installed, you must first install or upgrade it. For more information, see [Installing the AWS SAM CLI](#).

Step 1: Install the layer

To make the Lambda Insights extension available to all of your Lambda functions, add a `Layers` property to the `Globals` section of your SAM template with the ARN of the Lambda Insights layer. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

```
Globals:  
  Function:  
    Layers:  
      - !Sub "arn:aws:lambda:  
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

To enable this layer for only a single function, add the `Layers` property to the function as shown in this example.

```
Resources:
```



```
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Layers:
      - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

For each function, add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy.

AWS SAM doesn't support global policies, so you must enable those on each function individually, as shown in this example. For more information about globals, see [Globals Section](#).

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Policies:
        - CloudWatchLambdaInsightsExecutionRolePolicy
```

Invoking locally

The AWS SAM CLI supports Lambda extensions. However, every locally executed invocation resets the runtime environment. Lambda Insights data won't be available from local invocations because the runtime is restarted without a shutdown event. For more information, see [Release 1.6.0 - Add support for local testing of AWS Lambda extensions](#).

Troubleshooting

To troubleshoot your Lambda Insights installation, add the following environment variable to your Lambda function to enable debug logging.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          LAMBDA_INSIGHTS_LOG_LEVEL: info
```

Use AWS CloudFormation to enable Lambda Insights on an existing Lambda function

Follow these steps to use AWS CloudFormation to enable Lambda Insights on an existing Lambda function.

Step 1: Install the layer

Add the Lambda Insights layer to the Layers property within the Lambda Insights layer ARN. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

Add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to your function execution role.

```
Resources:
  MyFunctionExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy'
```

Step 3: (Optional) Add VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

```
Resources:
  CloudWatchLogsVpcPrivateEndpoint:
```

```
Type: AWS::EC2::VPCEndpoint
Properties:
  PrivateDnsEnabled: 'true'
  VpcEndpointType: Interface
  VpcId: !Ref: VPC
  ServiceName: !Sub com.amazonaws.${AWS::Region}.logs
  SecurityGroupIds:
    - !Ref InterfaceVpcEndpointSecurityGroup
  SubnetIds:
    - !Ref PublicSubnet01
    - !Ref PublicSubnet02
    - !Ref PublicSubnet03
```

Use the AWS CDK to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CDK to enable Lambda Insights on an existing Lambda function. To use these steps, you must already be using the AWS CDK to manage your resources.

The commands in this section are in TypeScript.

First, update the function permissions.

```
executionRole.addManagedPolicy(
  ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaInsightsExecutionRolePolicy')
);
```

Next, install the extension on the Lambda function. Replace the ARN value for the `layerArn` parameter with the ARN matching your Region and the extension version that you want to use. For more information, see [Available versions of the Lambda Insights extension](#).

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:580247275435:layer:LambdaInsightsExtension:14';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

If necessary, enable the virtual private cloud (VPC) endpoint for CloudWatch Logs. This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs VPC endpoint.

```
const cloudWatchLogsEndpoint = vpc.addInterfaceEndpoint('cwl-gateway', {
  service: InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
```

```
});  
  
cloudWatchLogsEndpoint.connections.allowDefaultPortFromAnyIpv4();
```

Use Serverless Framework to enable Lambda Insights on an existing Lambda function

Follow these steps to use Serverless Framework to enable Lambda Insights on an existing Lambda function. For more information about Serverless Framework, see serverless.com.

This is done through a Lambda Insights plugin for Serverless. For more information, see [serverless-plugin-lambda-insights](#).

If you don't already have the latest version of the Serverless command-line interface installed, you must first install or upgrade it. For more information, see [Get started with Serverless Framework Open Source & AWS](#).

To use Serverless Framework to enable Lambda Insights on a Lambda function

1. Install the Serverless plugin for Lambda Insights by running the following command in your Serverless directory:

```
npm install --save-dev serverless-plugin-lambda-insights
```

2. In your `serverless.yml` file, add the plugin in the `plugins` section as shown:

```
provider:  
  name: aws  
plugins:  
  - serverless-plugin-lambda-insights
```

3. Enable Lambda Insights.
 - You can enable Lambda Insights individually per function by adding the following property to the `serverless.yml` file

```
functions:  
  myLambdaFunction:  
    handler: src/app/index.handler  
    lambdaInsights: true #enables Lambda Insights for this function
```

- You can enable Lambda Insights for all functions within the `serverless.yml` file by adding the following custom section:

```
custom:
  lambdaInsights:
    defaultLambdaInsights: true #enables Lambda Insights for all functions
```

4. Re-deploy the Serverless service by entering the following command:

```
serverless deploy
```

This re-deploys all functions and enables Lambda Insights for those functions that you have specified. It enables Lambda Insights by adding the Lambda Insights layer and attaching the necessary permissions using the `arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy` IAM policy.

Enable Lambda Insights on a Lambda container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image, add lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image. The lines to add are different for x86-64 containers and ARM64 containers.

Note

The Lambda Insights agent is supported only on Lambda runtimes that use Amazon Linux 2.

Topics

- [x86-64 container image deployment](#)
- [ARM64 container image deployment](#)

x86-64 container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image running on an x86-64 container, add the following lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image.

```
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm
```

After you create your Lambda function, assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role, and Lambda Insights is enabled on the container image-based Lambda function.

Note

To use an older version of the Lambda Insights extension, replace the URL in the previous commands with this URL: https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.1.0.111.0.rpm. Currently, only Lambda Insights versions 1.0.111.0 and later are available. For more information, see [Available versions of the Lambda Insights extension](#).

To verify the signature of the Lambda Insights agent package on a Linux server

1. Enter the following command to download the public key.

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
lambda-insights-extension.gpg
```

2. Enter the following command to import the public key into your keyring.

```
shell$ gpg --import lambda-insights-extension.gpg
```

The output will be similar to the following. Make a note of the key value, you will need it in the next step. In this example output, the key value is 848ABDC8.

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

3. Verify the fingerprint by entering the following command. Replace key-value with the value of the key from the preceding step.

```
shell$ gpg --fingerprint key-value
```

The fingerprint string in the output of this command should be E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8. If the string doesn't match, don't install the agent and contact AWS.

4. After you have verified the fingerprint, you can use it to verify the Lambda Insights agent package. Download the package signature file by entering the following command.

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm.sig
```

5. Verify the signature by entering the following command.

```
shell$ gpg --verify lambda-insights-extension.rpm.sig lambda-insights-extension.rpm
```

The output should look like the following:

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8
```

In the expected output, there might be a warning about a trusted signature. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

If the output contains `BAD signature`, check whether you performed the steps correctly. If you continue to get a `BAD signature` response, contact AWS and avoid using the downloaded file.

x86-64 Example

This section includes an example of enabling Lambda Insights on a container image-based Python Lambda function.

An example of enabling Lambda Insights on a Lambda container image

1. Create a Dockerfile that is similar to the following:

```
FROM public.ecr.aws/lambda/python:3.8

// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Create a Python file named `index.py` that is similar to the following:

```
def handler(event, context):
    return {
        'message': 'Hello World!'
    }
```

3. Put the Dockerfile and `index.py` in the same directory. Then, in that directory, run the following steps to build the docker image and upload it to Amazon ECR.

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
"${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. Use that Amazon ECR image that you just created to create the Lambda function.
5. Assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role.

ARM64 container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image running on an AL2_aarch64 container (which uses ARM64 architecture), add the following lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image.

```
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm
```

After you create your Lambda function, assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role, and Lambda Insights is enabled on the container image-based Lambda function.

Note

To use an older version of the Lambda Insights extension, replace the URL in the previous commands with this URL: `https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.1.0.229.0.rpm`. Currently, only Lambda Insights versions 1.0.229.0 and later are available. For more information, see [Available versions of the Lambda Insights extension](#).

To verify the signature of the Lambda Insights agent package on a Linux server

1. Enter the following command to download the public key.

```
shell$ wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/lambda-insights-extension.gpg
```

2. Enter the following command to import the public key into your keyring.

```
shell$ gpg --import lambda-insights-extension.gpg
```

The output will be similar to the following. Make a note of the key value, you will need it in the next step. In this example output, the key value is 848ABDC8.

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
```

```
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

3. Verify the fingerprint by entering the following command. Replace `key-value` with the value of the key from the preceding step.

```
shell$ gpg --fingerprint key-value
```

The fingerprint string in the output of this command should be `E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8`. If the string doesn't match, don't install the agent and contact AWS.

4. After you have verified the fingerprint, you can use it to verify the Lambda Insights agent package. Download the package signature file by entering the following command.

```
shell$ wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm.sig
```

5. Verify the signature by entering the following command.

```
shell$ gpg --verify lambda-insights-extension-arm64.rpm.sig lambda-insights-
extension-arm64.rpm
```

The output should look like the following:

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8
```

In the expected output, there might be a warning about a trusted signature. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

If the output contains `BAD signature`, check whether you performed the steps correctly. If you continue to get a `BAD signature` response, contact AWS and avoid using the downloaded file.

ARM64 Example

This section includes an example of enabling Lambda Insights on a container image-based Python Lambda function.

An example of enabling Lambda Insights on a Lambda container image

1. Create a Dockerfile that is similar to the following:

```
FROM public.ecr.aws/lambda/python:3.8
// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Create a Python file named `index.py` that is similar to the following:

```
def handler(event, context):
    return {
        'message': 'Hello World!'
    }
```

3. Put the Dockerfile and `index.py` in the same directory. Then, in that directory, run the following steps to build the docker image and upload it to Amazon ECR.

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
"${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. Use that Amazon ECR image that you just created to create the Lambda function.
5. Assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role.

Update the Lambda Insights extension version on a function

As a best practice, we recommend that you keep your Lambda Insights extension updated to the latest version. The topics in this page explain how to do so.

Note

This page explains how to update the extension version used by a function that is already using Lambda Insights. For information about how to get started with Lambda Insights, see [Get started with Lambda Insights](#).

Use the Lambda console to update the Lambda Insights extension version

Use the following steps to use the Lambda console to update the Lambda Insights extension version.

To update using the Lambda console

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose the name of your function.
3. In the **Layers** section, choose **Edit**.
4. In the list of layers, search for **LambdaInsightsExtension** and then change the layer version to the latest version listed in [Available versions of the Lambda Insights extension](#).
5. Choose **Save**.

Use the AWS CLI to update the Lambda Insights extension version

To use the AWS CLI to update the Lambda Insights extension version, enter the following command. Replace the ARN value for the layers parameter with the ARN matching your Region and the extension version that you want to use. For information about the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

```
aws lambda update-function-configuration \
--function-name function-name \
--layers "arn:aws:lambda:us-west-1:111122223333:layer:LambdaInsightsExtension:53"
```

Use the AWS SAM CLI to update the Lambda Insights extension on one or more functions

To update the Lambda Insights Extension version for all of your Lambda functions, update the Layers property in the Globals section of your AWS Serverless Application Model (SAM) template with the ARN of the Lambda Insights layer. For information about the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

The following updates all of your Lambda functions.

```
Globals:
  Function:
    Layers:
      - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

The following updates just one function.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

Use AWS CloudFormation to update the Lambda Insights extension on one or more functions

To update the Lambda Insights Extension version by using AWS CloudFormation, update the extension layer in the Layers property within the function's AWS CloudFormation resource, as in the following example. For information about the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

```
Resources:
  MyFunction:
```

```
Type: AWS::Lambda::Function
Properties:
  Layers:
    - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

Use the AWS CDK to update the Lambda Insights extension on one or more functions

You can update the extension version on the Lambda function by replacing the ARN value for the `layerArn` parameter with the ARN matching your Region and the extension version that you want to use. For information about the latest release version of the Lambda Insights extension layer, see [Available versions of the Lambda Insights extension](#).

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:111122223333:layer:LambdaInsightsExtension:53';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

Use Serverless Framework to update the Lambda Insights extension on one or more functions

Follow these steps to use Serverless Framework to update the Lambda Insights extension version on an existing Lambda function. For more information about Serverless Framework, see the [Serverless Framework documentation](#).

This method uses a Lambda Insights plugin for Serverless. For more information, see [serverless-plugin-lambda-insights](#).

If you don't already have the latest version of the Serverless command-line interface installed, you must first install it or upgrade it. For more information, see [Setting Up Serverless Framework With AWS](#).

To update using the Lambda console

1. Update Lambda Insights. If you haven't already done so, add a custom section at the end of the file and specify the Lambda Insights version inside a `lambdaInsightsVersion` property.

```
custom:
  lambdaInsights:
    lambdaInsightsVersion: 53 #specify the Layer Version
```

2. Re-deploy the Serverless service by entering the following command.

serverless deploy

Update the Lambda Insights extension version on a Lambda container image deployment

To update Lambda Insights on a Lambda container image, follow the steps in [Enable Lambda Insights on a Lambda container image deployment](#) to rebuild the image with the latest version of Lambda Insights. Then, use the AWS CLI to [update the function code](#) and provide a container image URI as the value for the `--image-uri` parameter.

View your Lambda Insights metrics

After you have installed the Lambda Insights extension on a Lambda function that has been invoked, you can use the CloudWatch console to see your metrics. You can see a multi-function overview, or focus on a single function.

For a list of Lambda Insights metrics, see [Metrics collected by Lambda Insights](#).

To view the multi-function overview for your Lambda Insights metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Lambda Insights**, choose **Multi-function**.

The top part of the page displays graphs with aggregated metrics of all your Lambda functions in the Region that have Lambda Insights enabled. Lower on the page is a table that lists the functions.

3. To filter by function name to reduce the number of functions displayed, type part of the function name in the box near the top of the page.
4. To add this view to a dashboard as a widget, choose **Add to dashboard**.

To view metrics for a single function

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, under **Lambda Insights**, choose **Single-function**.

The top part of the page displays graphs with metrics for the selected function.

3. If you have X-Ray enabled, you can choose a single trace ID. This opens the X-Ray Trace Map page for that invocation, and from there you can zoom out to see the distributed trace and the

other services involved in handling that specific transaction. For more information about the X-Ray Trace Map, see [Using the X-Ray Trace Map](#).

4. To open CloudWatch Logs Insights and zoom in on a specific error, choose **View logs** by the table at the bottom of the page.
5. To add this view to a dashboard as a widget, choose **Add to dashboard**.

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see [Amazon CloudWatch Application Insights](#).

You can enable Application Insights to gather additional data from your Lambda functions. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** in the **Application Insights** tab below the performance view in the Lambda Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your Lambda functions, the Application Insights dashboard appears below the Lambda Insights dashboard, in the **Application Insights** tab.

Metrics collected by Lambda Insights

Lambda Insights collects several metrics from the Lambda functions where it is installed. Some of these metrics are available as time series aggregated data in CloudWatch Metrics. Other metrics are not aggregated into time series data but can be found in the embedded metric format log entries by using CloudWatch Logs Insights.

The following metrics are available as time series aggregated data in CloudWatch Metrics in the `LambdaInsights` namespace.

Metric name	Dimensions	Description
<code>cpu_total_time</code>	function_name function_name, version	Sum of <code>cpu_system_time</code> and <code>cpu_user_time</code> . Unit: Milliseconds

Metric name	Dimensions	Description
init_duration	function_name function_name, version	The amount of time spent in the <code>init</code> phase of the Lambda execution environment lifecycle. Unit: Milliseconds
memory_utilization	function_name function_name, version	The maximum memory measured as a percentage of the memory allocated to the function. Unit: Percent
rx_bytes	function_name function_name, version	The number of bytes received by the function. Unit: Bytes
tmp_used		The amount of space used in the <code>/tmp</code> directory. Unit: Bytes
tx_bytes	function_name function_name, version	The number of bytes sent by the function. Unit: Bytes

Metric name	Dimensions	Description
total_memory	function_name function_name, version	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes
total_network	function_name function_name, version	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes
used_memory_max	function_name function_name, version	The measured memory of the function sandbox. Unit: Megabytes

The following metrics can be found in the embedded metric format log entries by using CloudWatch Logs Insights. For more information about CloudWatch Logs Insights, see [Analyzing Log Data with CloudWatch Logs Insights](#).

For more information about embedded metric format, see [Embedding metrics within logs](#).

Metric name	Description	
cpu_system_time	The amount of time the CPU spent executing kernel code. Unit: Milliseconds	
cpu_total_time	Sum of <code>cpu_system_time</code> and <code>cpu_user_time</code> . Unit: Milliseconds	
cpu_user_time	The amount of time the CPU spent executing user code. Unit: Milliseconds	
fd_max	The maximum number of file descriptors available. Unit: Count	
fd_use	The maximum number of file descriptors in use. Unit: Count	
memory_utilization	The maximum memory measured as a percentage of the memory allocated to the function. Unit: Percent	
rx_bytes	The number of bytes received by the function. Unit: Bytes	
tx_bytes	The number of bytes sent by the function. Unit: Bytes	

Metric name	Description	
threads_max	The number of threads in use by the function process. As a function author, you don't control the initial number of threads created by the runtime. Unit: Count	
tmp_max	The amount of space available in the /tmp directory. Unit: Bytes	
total_memory	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes	
total_network	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes	
used_memory_max	The measured memory of the function sandbox. Unit: Bytes	

Troubleshooting and known issues

The first step to troubleshooting any issues is to enable debug logging on the Lambda Insights extension. To do this, set the following environment variable on your Lambda function: `LAMBDA_INSIGHTS_LOG_LEVEL=info`. For more information, see [Using AWS Lambda environment variables](#).

The extension emits logs into the same log group as your function (`/aws/lambda/function-name`). Review those logs to see if the error might be related to a setup issue.

I don't see any metrics from Lambda Insights

If you don't see Lambda Insights metrics that you expect to see, check the following possibilities:

- **The metrics might just be delayed**— If the function has not yet been invoked or the data has not been flushed yet, you won't see the metrics in CloudWatch. For more information, see **Known Issues** later in this section.
- **Confirm that the Lambda function has the correct permissions**—Make sure that the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy is assigned to the function's execution role.
- **Check the Lambda runtime**—Lambda Insights supports only certain Lambda runtimes. For a list of supported runtimes, see [Lambda Insights](#).

For example, to use Lambda Insights on Java 8, you must use the `java8.a12` runtime, not the `java8` runtime.

- **Check network access**—The Lambda function might be on a VPC private subnet with no internet access and you don't have a VPC endpoint configured for CloudWatch Logs. To help debug this issue, you can set the environment variable `LAMBDA_INSIGHTS_LOG_LEVEL=info`.

Known issues

Data delay can be as high as 20 minutes. When a function handler completes, Lambda freezes the sandbox, which also freezes the Lambda Insights extension. While the function is running, we use an adaptive batching strategy based on the function TPS to output data. However, if the function stops being invoked for an extended period and there is still event data in the buffer, this data can be delayed until Lambda shuts down the idle sandbox. When Lambda shuts down the sandbox, we flush the buffered data.

Example telemetry event

Each invocation of a Lambda function that has Lambda Insights enabled writes a single log event to the `/aws/lambda-insights` log group. Each log event contains metrics in embedded metric format. For more information about embedded metric format, see [Embedding metrics within logs](#).

To analyze these log events, you can use the following methods:

- The Lambda Insights section of the CloudWatch console, as explained in [View your Lambda Insights metrics](#).
- Log event queries using CloudWatch Logs Insights. For more information, see [Analyzing Log Data with CloudWatch Logs Insights](#).
- Metrics collected in the LambdaInsights namespace, which you graph by using CloudWatch metrics.

The following is an example of a Lambda Insights log event with embedded metric format.

```
{
  "_aws": {
    "Timestamp": 1605034324256,
    "CloudWatchMetrics": [
      {
        "Namespace": "LambdaInsights",
        "Dimensions": [
          [ "function_name" ],
          [ "function_name", "version" ]
        ],
        "Metrics": [
          { "Name": "memory_utilization", "Unit": "Percent" },
          { "Name": "total_memory", "Unit": "Megabytes" },
          { "Name": "used_memory_max", "Unit": "Megabytes" },
          { "Name": "cpu_total_time", "Unit": "Milliseconds" },
          { "Name": "tx_bytes", "Unit": "Bytes" },
          { "Name": "rx_bytes", "Unit": "Bytes" },
          { "Name": "total_network", "Unit": "Bytes" },
          { "Name": "init_duration", "Unit": "Milliseconds" }
        ]
      }
    ],
    "LambdaInsights": {
      "ShareTelemetry": true
    }
  },
  "event_type": "performance",
  "function_name": "cpu-intensive",
  "version": "Blue",
  "request_id": "12345678-8bcc-42f7-b1de-123456789012",
  "trace_id": "1-5faae118-12345678901234567890",
  "duration": 45191,
}
```

```
"billed_duration": 45200,  
"billed_mb_ms": 11571200,  
"cold_start": true,  
"init_duration": 130,  
"tmp_free": 538329088,  
"tmp_max": 551346176,  
"threads_max": 11,  
"used_memory_max": 63,  
"total_memory": 256,  
"memory_utilization": 24,  
"cpu_user_time": 6640,  
"cpu_system_time": 50,  
"cpu_total_time": 6690,  
"fd_use": 416,  
"fd_max": 32642,  
"tx_bytes": 4434,  
"rx_bytes": 6911,  
"timeout": true,  
"shutdown_reason": "Timeout",  
"total_network": 11345,  
"agent_version": "1.0.72.0",  
"agent_memory_avg": 10,  
"agent_memory_max": 10  
}
```

Use Contributor Insights to analyze high-cardinality data

You can use Contributor Insights to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. This helps you find top talkers and understand who or what is impacting system performance. For example, you can find bad hosts, identify the heaviest network users, or find the URLs that generate the most errors.

You can build your rules from scratch, and when you use the AWS Management Console you can also use sample rules that AWS has created. Rules define the log fields that you want to use to define contributors, such as `IpAddress`. You can also filter the log data to find and analyze the behavior of individual contributors.

CloudWatch also provides built-in rules that you can use to analyze metrics from other AWS services.

All rules analyze incoming data in real time.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules in that monitoring account that analyze log groups in source accounts and in the monitoring account. You can also create a single rule that analyzes log groups in multiple accounts. For more information, see [CloudWatch cross-account observability](#).

Note

If you use Contributor Insights, you are charged for each occurrence of a log event that matches a rule. For more information, see [Amazon CloudWatch Pricing](#).

Topics

- [Create a Contributor Insights rule](#)
- [Contributor Insights rule syntax](#)
- [Contributor Insights rule examples](#)
- [Viewing Contributor Insights reports](#)
- [Graphing metrics generated by rules](#)
- [Using Contributor Insights built-in rules](#)

Create a Contributor Insights rule

You can create rules to analyze log data. Any logs in JSON or Common Log Format (CLF) can be evaluated. This includes your custom logs that follow one of these formats and logs from AWS services such as Amazon VPC flow logs, Amazon Route 53 DNS query logs, Amazon ECS container logs, and logs from AWS CloudTrail, Amazon SageMaker, Amazon RDS, AWS AppSync and API Gateway.

In a rule, when you specify field names or values, all matching is case sensitive.

You can use built-in sample rules when you create a rule or you can create your own rule from scratch. Contributor Insights includes sample rules for the following types of logs:

- Amazon API Gateway logs
- Amazon Route 53 public DNS query logs
- Amazon Route 53 resolver query logs

- CloudWatch Container Insights logs
- VPC flow logs

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules for log groups in the source accounts linked to this monitoring account, in addition to creating rules for log groups in the monitoring account. You can also set up a single rule that monitors log groups in different accounts. For more information, see [CloudWatch cross-account observability](#).

Important

When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see [Using condition keys to limit Contributor Insights users' access to log groups](#).

To create a rule using a built-in sample rule

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Insights, Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts, and you can also set up a single rule to analyze log groups in different accounts.
 - (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix. If this is a monitoring account, you can optionally select the accounts to search in, otherwise all accounts are selected.

Note

You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

5. For **Rule type**, choose **Sample rule**. Then choose **Select sample rule** and select the rule.
6. The sample rule has filled out the **Log format**, **Contribution**, **Filters**, and **Aggregate on** fields. You can adjust those values, if you like.
7. Choose **Next**.
8. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, (hyphen), (underscore), and (period).
9. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).

Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.

10. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
11. Choose **Create**.

To create a rule from scratch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts, and you can also set up a single rule to analyze log groups in different accounts.

- (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix.

Note

You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

5. For **Rule type**, choose **Custom rule**.
6. For **Log format**, choose **JSON** or **CLF**.
7. You can finish creating the rule by using the wizard or by choosing the **Syntax** tab and specifying your rule syntax manually.

To continue using the wizard, do the following:

- a. For **Contribution, Key**, enter a contributor type that you want to report on. The report displays the top-N values for this contributor type.

Valid entries are any log field that has values. Examples include **requestId**, **sourceIPAddress**, and **containerID**.

For information about finding the log field names for the logs in a certain log group, see [Finding Log Fields](#).

Keys larger than 1 KB are truncated to 1KB.

- b. (Optional) Choose **Add new key** to add more keys. You can include as many as four keys in a rule. If you enter more than one key, the contributors in the report are defined by unique value combinations of the keys. For example, if you specify three keys, each unique combination of values for the three keys is counted as a unique contributor.
- c. (Optional) If you want to add a filter that narrows the scope of your results, choose **Add filter**. For **Match**, enter the name of the log field that you want to filter on. For **Condition**, choose a comparison operator, and enter a value that you want to filter for.

You can add as many as four filters in a rule. Multiple filters are joined by AND logic, so only log events that match all filters are evaluated.

Note

Arrays that follow comparison operators, such as `In`, `NotIn`, or `StartsWith`, can include as many as 10 string values. For more information about the Contributor Insights rules syntax, see [Contributor Insights rule syntax](#).

- d. For **Aggregate on**, choose **Count** or **Sum**. Choosing **Count** causes the contributor ranking to be based on the number of occurrences. Choosing **Sum** causes the ranking to be based on the aggregated sum of the values of the field that you specify for **Contribution, Value**.
 8. To enter your rule as a JSON object instead of using the wizard, do the following:
 - a. Choose the **Syntax** tab.
 - b. In **Rule body**, enter the JSON object for your rule. For information about rule syntax, see [Contributor Insights rule syntax](#).
 9. Choose **Next**.
 10. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, "-", "_", and ".".
 11. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see [Amazon CloudWatch Pricing](#).
- Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.
12. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see [Tagging your Amazon CloudWatch resources](#).
 13. Choose **Next**.
 14. Confirm the settings that you entered, and choose **Create rule**.

You can disable, enable, or delete rules that you have created.

To enable, disable, or delete a rule in Contributor Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. In the list of rules, select the check box next to a single rule.

Built-in rules are created by AWS services and can't be edited, disabled, or deleted.

4. Choose **Actions**, and then choose the option you want.

Finding log fields

When you create a rule, you need to know the names of fields in the log entries in a log group.

To find the log fields in a log group

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, under **Logs**, choose **Insights**.
3. Above the query editor, select one or more log groups to query.

When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in the right pane in **Discovered fields**.

Contributor Insights rule syntax

This section explains the syntax for Contributor Insights rules. Use this syntax only when you are creating a rule by entering a JSON block. If you use the wizard to create a rule, you don't need to know the syntax. For more information about creating rules using the wizard, see [Create a Contributor Insights rule](#).

All matching of rules to log event field names and values is case sensitive.

The following example illustrates the syntax for JSON logs.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "API-Gateway-Access-Logs*",
    "Log-group-name2"
  ],
  "LogFormat": "JSON",
  "Contribution": {
```

```
    "Keys": [
      "$.ip"
    ],
    "ValueOf": "$.requestBytes",
    "Filters": [
      {
        "Match": "$.httpMethod",
        "In": [
          "PUT"
        ]
      }
    ]
  },
  "AggregateOn": "Sum"
}
```

Fields in Contributor Insights rules

Schema

The value of Schema for a rule that analyzes CloudWatch Logs data must always be {"Name": "CloudWatchLogRule", "Version": 1}

LogGroupNames

An array of strings. For each element in the array, you can optionally use * at the end of a string to include all log groups with names that start with that prefix.

Be careful about using wildcards with log group names. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

LogGroupARNs

If you are creating this rule in a CloudWatch cross-account observability monitoring account, you can use LogGroupARNs to specify log groups in source accounts that are linked to the monitoring account, and to specify log groups in the monitoring account itself. You must specify either LogGroupNames or LogGroupARNs in your rule, but not both.

LogGroupARNs is an array of strings. For each element in the array, you can optionally use * as a wildcard in certain situations. For example you can specify arn:aws:logs:us-west-1:*:log-group/MyLogGroupName2 to specify log groups named MyLogGroupName2

in all source accounts and in the monitoring account, in the US West (N. California) Region. You can also specify `arn:aws:logs:us-west-1:111122223333:log-group/GroupNamePrefix*` to specify all log groups in US West (N. California) in 111122223333 that have names starting with `GroupNamePrefix`.

You can't specify a partial AWS account ID as a prefix with a wild card.

Be careful about using wildcards with log group ARNs. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see [Amazon CloudWatch Pricing](#).

LogFormat

Valid values are JSON and CLF.

Contribution

This object includes a `Keys` array with as many as four members, optionally a single `ValueOf`, and optionally an array of as many as four `Filters`.

Keys

An array of up to four log fields that are used as dimensions to classify contributors. If you enter more than one key, each unique combination of values for the keys is counted as a unique contributor. The fields must be specified using JSON property format notation.

ValueOf

(Optional) Specify this only when you are specifying `Sum` as the value of `AggregateOn`. `ValueOf` specifies a log field with numerical values. In this type of rule, the contributors are ranked by their sum of the value of this field, instead of their number of occurrences in the log entries. For example, if you want to sort contributors by their total `BytesSent` over a period, you would set `ValueOf` to `BytesSent` and specify `Sum` for `AggregateOn`.

Filters

(Optional) Specifies an array of as many as four filters to narrow the log events that are included in the report. If you specify multiple filters, Contributor Insights evaluates them with a logical AND operator. You can use this to filter out irrelevant log events in your search or you can use it to select a single contributor to analyze their behavior.

Each member in the array must include a `Match` field and a field indicating the type of matching operator to use.

The `Match` field specifies a log field to evaluate in the filter. The log field is specified using JSON property format notation.

The matching operator field must be one of the following: `In`, `NotIn`, `StartsWith`, `GreaterThan`, `LessThan`, `EqualTo`, `NotEqualTo`, or `IsPresent`. If the operator field is `In`, `NotIn`, or `StartsWith`, it is followed by an array of string values to check for. Contributor Insights evaluates the array of string values with an OR operator. The array can include as many as 10 string values.

If the operator field is `GreaterThan`, `LessThan`, `EqualTo`, or `NotEqualTo`, it is followed by a single numerical value to compare with.

If the operator field is `IsPresent`, it is followed by either `true` or `false`. This operator matches log events based on whether the specified log field is present in the log event. The `isPresent` works only with values in the leaf node of JSON properties. For example, a filter that looks for matches to `c-count` does not evaluate a log event with a value of `details.c-count.c1`.

See the following four filter examples:

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThan": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

Valid values are `Count` and `Sum`. Specifies whether to aggregate the report based on a count of occurrences or a sum of the values of the field that is specified in the `ValueOf` field.

JSON property format notation

The `Keys`, `ValueOf`, and `Match` fields follow JSON property format with dot notation, where `$` represents the root of the JSON object. This is followed by a period and then an alphanumeric string with the name of the subproperty. Multiple property levels are supported.

The first character of the string can only be A-Z or a-z. The following characters of the string can be A-Z, a-z, or 0-9.

The following list illustrates valid examples of JSON property format:


```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

Additional field in rules for CLF logs

Common Log Format (CLF) log events do not have names for the fields like JSON does. To provide the fields to use for Contributor Insights rules, a CLF log event can be treated as array with an index starting from 1. You can specify the first field as "1", the second field as "2", and so on.

To make a rule for a CLF log easier to read, you can use `Fields`. This enables you to provide a naming alias for CLF field locations. For example, you can specify that the location "4" is an IP address. Once specified, `IpAddress` can be used as property in the `Keys`, `ValueOf`, and `Filters` in the rule.

The following is an example of a rule for a CLF log that uses the `Fields` field.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "API-Gateway-Access-Logs*"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "4": "IpAddress",
    "7": "StatusCode"
  },
  "Contribution": {
    "Keys": [
      "IpAddress"
    ],
    "Filters": [
      {
        "Match": "StatusCode",
        "EqualTo": 200
      }
    ]
  }
},
```

```
"AggregateOn": "Count"
}
```

Contributor Insights rule examples

This section contains examples that illustrate use cases for Contributor Insights rules.

VPC Flow Logs: Byte transfers by source and destination IP address

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "4": "srcaddr",
    "5": "dstaddr",
    "10": "bytes"
  },
  "Contribution": {
    "Keys": [
      "srcaddr",
      "dstaddr"
    ],
    "ValueOf": "bytes",
    "Filters": []
  },
  "AggregateOn": "Sum"
}
```

VPC Flow Logs: Highest number of HTTPS requests

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
```

```

    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "5": "destination address",
    "7": "destination port",
    "9": "packet count"
  },
  "Contribution": {
    "Keys": [
      "destination address"
    ],
    "ValueOf": "packet count",
    "Filters": [
      {
        "Match": "destination port",
        "EqualTo": 443
      }
    ]
  },
  "AggregateOn": "Sum"
}

```

VPC Flow Logs: Rejected TCP connections

```

{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "3": "interfaceID",
    "4": "sourceAddress",
    "8": "protocol",
    "13": "action"
  },
  "Contribution": {
    "Keys": [
      "interfaceID",

```

```

        "sourceAddress"
    ],
    "Filters": [
        {
            "Match": "protocol",
            "EqualTo": 6
        },
        {
            "Match": "action",
            "In": [
                "REJECT"
            ]
        }
    ]
},
"AggregateOn": "Sum"
}

```

Route 53 NXDomain responses by source address

```

{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "$.rcode",
        "StartsWith": [
          "NXDOMAIN"
        ]
      }
    ],
    "Keys": [
      "$.srcaddr"
    ]
  },
  "LogFormat": "JSON",
  "LogGroupNames": [
    "<loggroupname>"
  ]
}

```

```
}
```

Route 53 resolver queries by domain name

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [],
    "Keys": [
      "$.query_name"
    ]
  },
  "LogFormat": "JSON",
  "LogGroupNames": [
    "<loggroupname>"
  ]
}
```

Route 53 resolver queries by query type and source address

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [],
    "Keys": [
      "$.query_type",
      "$.srcaddr"
    ]
  },
  "LogFormat": "JSON",
  "LogGroupNames": [
    "<loggroupname>"
  ]
}
```

Viewing Contributor Insights reports

To view graphs of report data and a ranked list of contributors found by your rules, follow these steps.

To view your rule reports

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Contributor Insights**.
3. In the list of rules, choose the name of a rule.

The graph displays the results of the rule over the last three hours. The table under the graph shows the top 10 contributors.

4. To change the number of contributors shown in the table, choose **Top 10 contributors** at the top of the graph.
5. To filter the graph to show only the results from a single contributor, choose that contributor in the table legend. To again show all contributors, choose that same contributor again in the legend.
6. To change the time range shown in the report, choose **15m, 30m, 1h, 2h, 3h**, or **custom** at the top of the graph.

The maximum time range for the report is 24 hours, but you can choose a 24-hour window that occurred up to 15 days ago. To choose a time window in the past, choose **custom**, **absolute**, and then specify your time window.

7. To change the length of the time period used for the aggregation and ranking of contributors, choose **period** at the top of the graph. Viewing a longer time period generally shows a smoother report with few spikes. Choosing a shorter time period is more likely to display spikes.
8. To add this graph to a CloudWatch dashboard, choose **Add to dashboard**.
9. To open the CloudWatch Logs Insights query window, with the log groups in this report already loaded in the query box, choose **View logs**.
10. To export the report data to your clipboard or a CSV file, choose **Export**.

Graphing metrics generated by rules

Contributor Insights provides a metric math function, `INSIGHT_RULE_METRIC`. You can use this function to add data from a Contributor Insights report to a graph in the **Metrics** tab of the CloudWatch console. You can also set an alarm based on this math function. For more information about metric math functions, see [Use metric math](#).

To use this metric math function, you must be signed in to an account that has both the `cloudwatch:GetMetricData` and `cloudwatch:GetInsightRuleReport` permissions.

The syntax is `INSIGHT_RULE_METRIC(ruleName, metricName)`. *ruleName* is the name of a Contributor Insights rule. *metricName* is one of the values in the following list. The value of *metricName* determines which type of data the math function returns.

- `UniqueContributors` — the number of unique contributors for each data point.
- `MaxContributorValue` — the value of the top contributor for each data point. The identity of the contributor might change for each data point in the graph.

If this rule aggregates by `Count`, the top contributor for each data point is the contributor with the most occurrences in that period. If the rule aggregates by `Sum`, the top contributor is the contributor with the greatest sum in the log field specified by the rule's `Value` during that period.

- `SampleCount` — the number of data points matched by the rule.
- `Sum` — the sum of the values from all contributors during the time period represented by that data point.
- `Minimum` — the minimum value from a single observation during the time period represented by that data point.
- `Maximum` — the maximum value from a single observation during the time period represented by that data point.
- `Average` — the average value from all contributors during the time period represented by that data point.

Setting an alarm on Contributor Insights metric data

Using the function `INSIGHT_RULE_METRIC`, you can set alarms on metrics that Contributor Insights generates. For example, you can create an alarm that's based on the percentage of rejected

transmission control protocol (TCP) connections. To get started with this type of alarm, you can create rules like the ones shown in the following two examples:

Example rule: "RejectedConnectionsRule"

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "3": "interfaceID",
    "4": "sourceAddress",
    "8": "protocol",
    "13": "action"
  },
  "Contribution": {
    "Keys": [
      "interfaceID",
      "sourceAddress"
    ],
    "Filters": [
      {
        "Match": "protocol",
        "EqualTo": 6
      },
      {
        "Match": "action",
        "In": [
          "REJECT"
        ]
      }
    ]
  },
  "AggregateOn": "Sum"
}
```

Example rule: "TotalConnectionsRule"


```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "LogGroupNames": [
    "/aws/containerinsights/sample-cluster-name/flowlogs"
  ],
  "LogFormat": "CLF",
  "Fields": {
    "3": "interfaceID",
    "4": "sourceAddress",
    "8": "protocol",
    "13": "action"
  },
  "Contribution": {
    "Keys": [
      "interfaceID",
      "sourceAddress"
    ],
    "Filters": [{
      "Match": "protocol",
      "EqualTo": 6
    }],
    "AggregateOn": "Sum"
  }
}
```

After you create your rules, you can select the **Metrics** tab in the CloudWatch Console, where you can use the following example metric math expressions to graph the data that Contributor Insights reports:

Example: Metric math expressions

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

In the example, the metric math expression e3 returns all of the rejected TCP connections. If you want to be notified when 20 percent of the TCP connections are rejected, you can modify the expression by changing the threshold from 100 to 20.

Note

You can set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, you can select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon looks like a bell.

For more information about graphing metrics and using metric math functions, see the following section: [Add a math expression to a CloudWatch graph](#).

Using Contributor Insights built-in rules

You can use Contributor Insights built-in rules to analyze metrics from other AWS services. The following services support built-in rules:

- [Contributor Insights for Amazon DynamoDB](#) in the *Amazon DynamoDB Developer Guide*.
- [Use built-in Contributor Insights rules](#) in the *AWS PrivateLink Guide*.

Amazon CloudWatch Application Insights

Amazon CloudWatch Application Insights facilitates observability for your applications and underlying AWS resources. It helps you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications. Application Insights, which is powered by [SageMaker](#) and other AWS technologies, provides automated dashboards that show potential problems with monitored applications, which help you to quickly isolate ongoing issues with your applications and infrastructure. The enhanced visibility into the health of your applications that Application Insights provides helps reduce mean time to repair (MTTR) to troubleshoot your application issues.

When you add your applications to Amazon CloudWatch Application Insights, it scans the resources in the applications and recommends and configures metrics and logs on [CloudWatch](#) for application components. Example application components include SQL Server backend databases and Microsoft IIS/Web tiers. Application Insights analyzes metric patterns using historical data to detect anomalies, and continuously detects errors and exceptions from your application, operating system, and infrastructure logs. It correlates these observations using a combination of classification algorithms and built-in rules. Then, it automatically creates dashboards that show the relevant observations and problem severity information to help you prioritize your actions.

For common problems in .NET and SQL application stacks, such as application latency, SQL Server failed backups, memory leaks, large HTTP requests, and canceled I/O operations, it provides additional insights that point to a possible root cause and steps for resolution. Built-in integration with [AWS SSM OpsCenter](#) allows you to resolve issues by running the relevant Systems Manager Automation document.

Sections

- [What is Amazon CloudWatch Application Insights?](#)
- [How Amazon CloudWatch Application Insights works](#)
- [Get started with Amazon CloudWatch Application Insights](#)
- [Application Insights cross-account observability](#)
- [Work with component configurations](#)
- [Create and configure CloudWatch Application Insights monitoring using CloudFormation templates](#)
- [Tutorial: Set up monitoring for SAP ASE](#)
- [Tutorial: Set up monitoring for SAP HANA](#)
- [Tutorial: Set up monitoring for SAP NetWeaver](#)
- [View and troubleshoot problems detected by Amazon CloudWatch Application Insights](#)
- [Logs and metrics supported by Amazon CloudWatch Application Insights](#)

What is Amazon CloudWatch Application Insights?

CloudWatch Application Insights helps you monitor your applications that use Amazon EC2 instances along with other [application resources](#). It identifies and sets up key metrics, logs, and alarms across your application resources and technology stack (for example, your Microsoft SQL Server database, web (IIS) and application servers, OS, load balancers, and queues). It continuously monitors metrics and logs to detect and correlate anomalies and errors. When errors and anomalies are detected, Application Insights generates [CloudWatch Events](#) that you can use to set up notifications or take actions. To assist with troubleshooting, it creates automated dashboards for detected problems, which include correlated metric anomalies and log errors, along with additional insights to point you to a potential root cause. The automated dashboards help you to take remedial actions to keep your applications healthy and to prevent impact to the end-users of your application. It also creates OpsItems so that you can resolve problems using [AWS SSM OpsCenter](#).

You can configure important counters, such as Mirrored Write Transaction/sec, Recovery Queue Length, and Transaction Delay, as well as Windows Event Logs on CloudWatch. When a failover event or problem occurs with your SQL HA workload, such as a restricted access to query a target database, CloudWatch Application Insights provides automated insights .

CloudWatch Application Insights integrates with [AWS Launch Wizard](#) to provide a one-click monitoring setup experience for deploying SQL Server HA workloads on AWS. When you select the option to set up monitoring and insights with Application Insights on the [Launch Wizard console](#), CloudWatch Application Insights automatically sets up relevant metrics, logs, and alarms on CloudWatch, and starts monitoring newly deployed workloads. You can view automated insights and detected problems, along with the health of your SQL Server HA workloads, on the CloudWatch console.

Contents

- [Features](#)
- [Concepts](#)
- [Pricing](#)
- [Related services](#)
- [Supported application components](#)
- [Supported technology stacks](#)

Features

Application Insights provides the following features.

Automatic set up of monitors for application resources

CloudWatch Application Insights reduces the time it takes to set up monitoring for your applications. It does this by scanning your application resources, providing a customizable list of recommended metrics and logs, and setting them up on CloudWatch to provide necessary visibility into your application resources, such as Amazon EC2 and Elastic Load Balancers (ELB). It also sets up dynamic alarms on monitored metrics. The alarms are automatically updated based on anomalies detected in the previous two weeks.

Problem detection and notification

CloudWatch Application Insights detects signs of potential problems with your application, such as metric anomalies and log errors. It correlates these observations to surface potential problems

with your application. It then generates CloudWatch Events, [which can be configured to receive notifications or take actions](#). This eliminates the need for you to create individual alarms on metrics or log errors.

Troubleshooting

CloudWatch Application Insights creates CloudWatch automatic dashboards for problems that are detected. The dashboards show details about the problem, including the associated metric anomalies and log errors to help you with troubleshooting. They also provide additional insights that point to potential root causes of the anomalies and errors.

Concepts

The following concepts are important for understanding how Application Insights monitors your application.

Component

An auto-grouped, standalone, or custom grouping of similar resources that make up an application. We recommend grouping similar resources into custom components for better monitoring.

Observation

An individual event (metric anomaly, log error, or exception) that is detected with an application or application resource.

Problem

Problems are detected by correlating, classifying, and grouping related observations.

For definitions of other key concepts for CloudWatch Application Insights, see [Amazon CloudWatch Concepts](#).

Pricing

CloudWatch Application Insights sets up recommended metrics and logs for selected application resources using CloudWatch metrics, Logs, and Events for notifications on detected problems. These features are charged to your AWS account according to [CloudWatch pricing](#). For detected problems, [SSM OpsItems](#) are also created by Application Insights to notify you about problems. Additionally, Application Insights creates [SSM Parameter Store parameters](#) to configure the CloudWatch agents on your instances. The Amazon EC2 Systems Manager features are charged

according to [SSM pricing](#). You are not charged for setup assistance, monitoring, data analysis, or problem detection.

Costs for CloudWatch Application Insights

Costs for Amazon EC2 include usage of the following features:

- CloudWatch Agent
 - CloudWatch Agent log groups
 - CloudWatch Agent metrics
 - Prometheus log groups (for JMX workloads)

Costs for all resources include usage of the following features:

- CloudWatch alarms (majority of cost)
- SSM OpsItems (minimal cost)

Example cost calculation

The costs in this example are considered according to the following scenario.

You created a resource group that includes the following:

- An Amazon EC2 instance with SQL Server installed.
- An attached Amazon EBS volume.

When you onboard this resource group with CloudWatch Application Insights, the SQL Server workload installed on the Amazon EC2 instance is detected. CloudWatch Application Insights starts monitoring the following metrics.

The following metrics are monitored for the SQL Server instance:

- CPUUtilization
- StatusCheckFailed
- Memory % Committed Bytes in Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec

- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections
- SQLServer:Locks Number of Deadlocks/sec
- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

The following metrics are monitored for the volumes attached to the SQL Server instance:

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumeIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

For this scenario, the costs are calculated according to the [CloudWatch pricing](#) page and the [SSM pricing](#) page:

- **Custom metrics**

For this scenario, 13 of the above metrics are emitted to CloudWatch using the CloudWatch agent. These metrics are treated as custom metrics. The cost for each custom metric is \$.3/month. The total cost for these custom metrics is $13 * \$.3 = \$3.90/\text{month}$.

- **Alarms**

For this scenario, CloudWatch Application Insights monitors 26 metrics in total, which creates 26 alarms. The cost for each alarm is \$.1/month. The total cost for alarms is $26 * \$.1 = \2.60 /month.

- **Data ingestion and error logs**

The cost of data ingestion is \$.05/GB and storage for the SQL Server error log is \$.03/GB. The total cost for data ingestion and the error log is $\$.05/\text{GB} + \$.03/\text{GB} = \$.08/\text{GB}$.

- **Amazon EC2 Systems Manager OpsItems**

An SSM OpsItem is created for each problem detected by CloudWatch Application Insights. For n number of problems in your application, the total cost is $\$.00267 * n/\text{month}$.

Related services


The following services are used along with CloudWatch Application Insights:

Related AWS services

- **Amazon CloudWatch** provides system-wide visibility into resource utilization, application performance, and operational health. It collects and tracks metrics, sends alarm notifications, automatically updates resources that you are monitoring based on the rules that you define, and allows you to monitor your own custom metrics. CloudWatch Application Insights is initiated through CloudWatch—specifically, within the CloudWatch default operational dashboards. For more information, see the [Amazon CloudWatch User Guide](#).
- **CloudWatch Container Insights** collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. You can use Container Insights to monitor Amazon ECS, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2. When Application Insights is enabled on the Container Insights or Application Insights consoles, Application Insights displays detected problems on your Container Insights dashboard. For more information, see [Container Insights](#).
- **Amazon DynamoDB** is a fully managed NoSQL database service that lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

- **Amazon EC2** provides scalable computing capacity in the AWS Cloud. You can use Amazon EC2 to launch as many or as few virtual servers as you need, to configure security and networking, and to manage storage. You can scale up or down to handle changes in requirements or spikes in popularity, which reduces your need to forecast traffic. For more information, see the [Amazon EC2 User Guide for Linux Instances](#) or [Amazon EC2 Guide for Windows Instances](#).
- **Amazon Elastic Block Store (Amazon EBS)** provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes behave like raw, unformatted block devices. You can mount these volumes as devices on your instances. Amazon EBS volumes that are attached to an instance are exposed as storage volumes that persist independently from the life of the instance. You can create a file system on top of these volumes, or use them in any way you would use a block device (such as a hard drive). You can dynamically change the configuration of a volume attached to an instance. For more information, see the [Amazon EBS User Guide](#).
- **Amazon EC2 Auto Scaling** helps ensure that you have the correct number of EC2 instances available to handle the load for your application. For more information, see the [Amazon EC2 Auto Scaling User Guide](#).
- **Elastic Load Balancing** distributes incoming applications or network traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in multiple Availability Zones. For more information, see the [Elastic Load Balancing User Guide](#).
- **IAM** is a web service that helps you to securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication), and to control the resources they can use and how they can use them (authorization). For more information, see [Authentication and Access Control for Amazon CloudWatch](#).
- **AWS Lambda** lets you build serverless applications composed of functions that are triggered by events and automatically deploy them using CodePipeline and AWS CodeBuild. For more information, see [AWS Lambda Applications](#).
- **AWS Launch Wizard for SQL Server** reduces the time it takes to deploy SQL Server high availability solution to the cloud. You input your application requirements, including performance, number of nodes, and connectivity on the service console, and AWS Launch Wizard identifies the right AWS resources to deploy and run your SQL Server Always On application.
- **AWS Resource Groups** help you to organize the resources that make up your application. With Resource Groups, you can manage and automate tasks on a large number of resources at one time. Only one Resource Group can be registered for a single application. For more information, see the [AWS Resource Groups User Guide](#).

- **Amazon SQS** offers a secure, durable, and available hosted queue that allows you to integrate and decouple distributed software systems and components. For more information, see the [Amazon SQS User Guide](#).
- **AWS Step Functions** is a serverless function composer that allows you to sequence a variety of AWS services and resources, including AWS Lambda functions, into structured, visual workflows. For more information, see the [AWS Step Functions User Guide](#).
- **AWS SSM OpsCenter** aggregates and standardizes OpsItems across services while providing contextual investigation data about each OpsItem, related OpsItems, and related resources. OpsCenter also provides Systems Manager Automation documents (runbooks) that you can use to quickly resolve issues. You can specify searchable, custom data for each OpsItem. You can also view automatically-generated summary reports about OpsItems by status and source. For more information, see the [AWS Systems Manager User Guide](#).
- **Amazon API Gateway** is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. For more information, see the [Amazon API Gateway User Guide](#).

 **Note**

Application Insights supports only REST API protocols (v1 of the API Gateway service).

- **Amazon Elastic Container Service (Amazon ECS)** is a fully managed container orchestration service. You can use Amazon ECS to run your most sensitive and mission-critical applications. For more information, see the [Amazon Elastic Container Service Developer Guide](#).
- **Amazon Elastic Kubernetes Service (Amazon EKS)** is a managed service that you can use to run Kubernetes on AWS without having to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. For more information, see the [Amazon EKS User Guide](#).
- **Kubernetes on Amazon EC2.** Kubernetes is open-source software that helps you deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. With Kubernetes, you can run any type of containerized application with the same toolset on-premises and in the cloud. For more information, see [Kubernetes Documentation: Getting started](#).

- **Amazon FSx** helps you to launch and run popular file systems that are fully managed by AWS. With Amazon FSx, you can leverage the feature sets and performance of common open source and commercially-licensed file systems to avoid time-consuming administrative tasks. For more information, see the [Amazon FSx Documentation](#).
- **Amazon Simple Notification Service (SNS)** is a fully-managed messaging service for both application-to-application and application-to-person communication. You can configure Amazon SNS for monitoring by Application Insights. When Amazon SNS is configured as a resource for monitoring, Application Insights tracks SNS metrics to help determine why SNS messages may encounter issues or fail.
- **Amazon Elastic File System (Amazon EFS)** is a fully-managed elastic NFS file system for use with AWS Cloud services and on-premises resources. It is built to scale to petabytes on demand without disrupting applications. It grows and shrinks automatically as you add and remove files, which eliminates the need to provision and manage capacity to accommodate growth. For more information, see the [Amazon Elastic File System documentation](#).

Related third-party services

- For some workloads and applications monitored in Application Insights, **Prometheus JMX exporter** is installed using AWS Systems Manager Distributor so that CloudWatch Application Insights can retrieve Java-specific metrics. When you choose to monitor a Java application, Application Insights automatically installs the Prometheus JMX exporter for you.

Supported application components

CloudWatch Application Insights scans your resource group to identify application components. Components can be standalone, auto-grouped (such as instances in an Auto Scaling group or behind a load balancer), or custom (by grouping together individual Amazon EC2 instances).

The following components are supported by CloudWatch Application Insights:

AWS components

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing: Application Load Balancer and Classic Load Balancer (all target instances of these load balancers are identified and configured).

- Amazon EC2 Auto Scaling groups: AWS Auto Scaling (Auto Scaling groups are dynamically configured for all target instances; if your application scales up, CloudWatch Application Insights automatically configures the new instances). Auto Scaling groups are not supported for CloudFormation stack-based resource groups.
- AWS Lambda
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB table
- Amazon S3 bucket metrics
- AWS Step Functions
- Amazon API Gateway REST API stages
- Amazon Elastic Container Service (Amazon ECS): cluster, service, and task
- Amazon Elastic Kubernetes Service (Amazon EKS): cluster
- Kubernetes on Amazon EC2: Kubernetes cluster running on EC2
- Amazon SNS topic

Any other component type resources are not currently tracked by CloudWatch Application Insights. If a component type that is supported does not appear in your Application Insights application, the component may already be registered and managed by another application you own that is monitored by Application Insights.

Supported technology stacks

You can use CloudWatch Application Insights to monitor your applications running on Windows Server and Linux operating systems by selecting the application tier dropdown menu option for one of the following technologies:

- Front-end: Microsoft Internet Information Services (IIS) Web Server
- Worker-tier:
 - .NET Framework
 - .NET Core
- Applications:
 - Java
 - SAP NetWeaver standard, distributed, and high availability deployments
- Active Directory

- SharePoint
- Databases:
 - Microsoft SQL Server running on Amazon RDS or Amazon EC2 (including SQL Server High Availability configurations. See, [Component configuration examples](#)).
 - MySQL running on Amazon RDS, Amazon Aurora, or Amazon EC2
 - PostgreSQL running on Amazon RDS or Amazon EC2
 - Amazon DynamoDB table
 - Oracle running on Amazon RDS or Amazon EC2
 - SAP HANA database on a single Amazon EC2 instance and multiple EC2 instances
 - Cross-AZ SAP HANA database high availability setup
 - SAP Sybase ASE database on a single Amazon EC2 instance
 - Cross-AZ SAP Sybase ASE database high availability setup

If none of the technology stacks listed above apply to your application resources, you can monitor your application stack by choosing **Custom** from the application tier dropdown menu on the **Manage monitoring** page.

How Amazon CloudWatch Application Insights works

This section contains information about how CloudWatch Application Insights works, including:

- [How Application Insights monitors applications](#)
- [Data retention](#)
- [Quotas](#)
- [AWS Systems Manager \(SSM\) packages used by CloudWatch Application Insights](#)
- [AWS Systems Manager \(SSM\) Documents used by CloudWatch Application Insights](#)

How Application Insights monitors applications

Application Insights monitors applications as follows.

Application discovery and configuration

The first time an application is added to CloudWatch Application Insights it scans the application components to recommend key metrics, logs, and other data sources to monitor for your application. You can then configure your application based on these recommendations.

Data preprocessing

CloudWatch Application Insights continuously analyzes the data sources being monitored across the application resources to discover metric anomalies and log errors (observations).

Intelligent problem detection

The CloudWatch Application Insights engine detects problems in your application by correlating observations using classification algorithms and built-in rules. To assist in troubleshooting, it creates automated CloudWatch dashboards, which include contextual information about the problems.

Alert and action

When CloudWatch Application Insights detects a problem with your application, it generates CloudWatch Events to notify you of the problem. See [Application Insights CloudWatch Events and notifications for detected problems](#) for more information about how to set up these Events.

Example scenario

You have an ASP .NET application that is backed by a SQL Server database. Suddenly, your database begins to malfunction because of high memory pressure. This leads to application performance degradation and possibly HTTP 500 errors in your web servers and load balancer.

With CloudWatch Application Insights and its intelligent analytics, you can identify the application layer that is causing the problem by checking the dynamically created dashboard that shows the related metrics and log file snippets. In this case, the problem might be at the SQL database layer.

Data retention

CloudWatch Application Insights retains problems for 55 days and observations for 60 days.

Quotas

For default quotas for CloudWatch Application Insights, see [Amazon CloudWatch Application Insights endpoints and quotas](#). Unless otherwise noted, each quota is per AWS Region. Contact [AWS Support](#) to request an increase in your service quota. Many services contain quotas that cannot be changed. For more information about the quotas for a specific service, see the documentation for that service.

AWS Systems Manager (SSM) packages used by CloudWatch Application Insights

The packages listed in this section are used by Application Insights, and can be independently managed and deployed with AWS Systems Manager Distributor. For more information about SSM Distributor, see [AWS Systems Manager Distributor](#) in the *AWS Systems Manager User Guide*.

Packages:

- [AWSObservabilityExporter-JMXExporterInstallAndConfigure](#)
- [AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure](#)
- [AWSObservabilityExporter-HAClusterExporterInstallAndConfigure](#)
- [AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure](#)
- [AWSObservabilityExporter-SQLExporterInstallAndConfigure](#)

AWSObservabilityExporter-JMXExporterInstallAndConfigure

You can retrieve workload-specific Java metrics from [Prometheus JMX exporter](#) for Application Insights to configure and monitor alarms. In the Application Insights console, on the **Manage monitoring** page, select **JAVA application** from the **Application tier** dropdown. Then under **JAVA Prometheus exporter configuration**, select your **Collection method** and **JMX port number**.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus JMX exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus JMX exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- The JAVA_HOME environment variable is set

Install and configure the AWSObservabilityExporter-JMXExporterInstallAndConfigure package

The AWSObservabilityExporter-JMXExporterInstallAndConfigure package is an SSM Distributor package that you can use to install and configure [Prometheus JMX Exporter](#). When Java metrics are sent by the Prometheus JMX exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the [Prometheus JMX exporter YAML configuration file](#) located in the Prometheus GitHub repository. Use the example configuration and option descriptions to guide you.
2. Copy the Prometheus JMX exporter YAML configuration file encoded as Base64 to a new SSM parameter in [SSM Parameter Store](#).
3. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-JMXExporterInstallAndConfigure** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIGURATION": "{{s3:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_EXPOSITION_PORT": "9404"
}
```

Note

Port 9404 is the default port used to send Prometheus JMX metrics. You can update this port.

Example: Configure CloudWatch agent to retrieve Java metrics

1. Install the Prometheus JMX exporter, as described in the previous procedure. Then verify that it is correctly installed on your instance by checking the port status.

Successful installation on Windows instance example

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)
StatusCode : 200
StatusDescription : OK
Content : # HELP jvm_info JVM version info
```

Successful installation on Linux instance example

```
$ curl localhost:9404
# HELP jmx_config_reload_failure_total Number of times configuration have failed to be reloaded.
```



```
# TYPE jmx_config_reload_failure_total counter
jmx_config_reload_failure_total 0.0
```

2. Create the Prometheus service discovery YAML file. The following example service discovery file performs the following:

- Specifies the Prometheus JMX exporter host port as `localhost: 9404`.
- Attaches labels (`Application`, `ComponentName`, and `InstanceId`) to the metrics, which can be set as CloudWatch metric dimensions.

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn:aws:elasticloadbalancing:us-
east-1:123456789012:loadbalancer/app/sampl-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. Create the Prometheus JMX exporter configuration YAML file. The following example configuration file specifies the following:

- The metrics retrieval job interval and timeout period.
- The metrics retrieval jobs (`jmx` and `sap`), also known as scraping, which include the job name, maximum time series returned at a time, and service discovery file path.

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. Verify that the CloudWatch agent is installed on your Amazon EC2 instance and that the version is 1.247346.1b249759 or later. To install the CloudWatch agent on your EC2 instance, see [Installing the CloudWatch Agent](#). To verify the version, see [Finding information about CloudWatch agent versions](#).
5. Configure the CloudWatch agent. For more information about how to configure the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](#). The following example CloudWatch agent configuration file performs the following:
 - Specifies the Prometheus JMX exporter configuration file path.
 - Specifies the target log group to which to publish EMF metric logs.
 - Specifies two sets of dimensions for each metric name.
 - Sends 8 (4 metric names * 2 sets of dimensions per metric name) CloudWatch metrics.

```
{
  "logs":{
    "logs_collected":{
      ....
    },
    "metrics_collected":{
      "prometheus":{
        "cluster_name":"prometheus-test-cluster",
        "log_group_name":"prometheus-test",
        "prometheus_config_path":"/tmp/prometheus.yaml",
        "emf_processor":{
          "metric_declaration_dedup":true,
          "metric_namespace":"CWAgent",
          "metric_unit":{
            "jvm_threads_current":"Count",
            "jvm_gc_collection_seconds_sum":"Second",
            "jvm_memory_bytes_used":"Bytes"
          },
          "metric_declaration":[
            {
              "source_labels":[
                "job"
              ],
              "label_matcher":"^jmx$",
              "dimensions":[
                [
```

```

        "InstanceId",
        "ComponentName"
    ],
    [
        "ComponentName"
    ]
],
"metric_selectors":[
    "^java_lang_threading_threadcount$",
    "^java_lang_memory_heapmemoryusage_used$",
    "^java_lang_memory_heapmemoryusage_committed$"
]
}
]
}
}
},
"metrics":{
    ....
}
}
}

```

AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure

You can retrieve workload-specific SAP HANA metrics from [Prometheus HANA database exporter](#) for Application Insights to configure and monitor alarms. For more information, see [Set up your SAP HANA database for monitoring](#) in this guide.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus HANA database exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HANA database exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP HANA database
- Linux operating system (SUSE Linux, RedHat Linux)
- A secret with SAP HANA database monitoring credentials, using AWS Secrets Manager. Create a secret using the key/value pairs format, specify the key username, and enter the database user

for the value. Add a second key password, and then enter the password for the value. For more information about how to create secrets, see [Create a secret](#) in the *AWS Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
  "username": "<database_user>",
  "password": "<database_password>"
}
```

Install and configure the AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure package

The AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure package is an SSM Distributor package that you can use to install and configure [Prometheus HANA database Exporter](#). When HANA database metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter configurations. The following is an example parameter value.

```
{\"exposition_port\":9668,\"multi_tenant\":true,\"timeout\":600,\"hana\":{\"host\": \"localhost\", \"port\":30013,\"aws_secret_name\": \"HANA_DB_CREDS\", \"scale_out_mode \":true}}
```

Note

In this example, the export runs only on the Amazon EC2 instance with the active SYSTEM database, and it will remain idle on the other EC2 instances in order to avoid duplicate metrics. The exporter can retrieve all of the database tenant information from the SYSTEM database.

2. Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter metrics queries. The package can accept more than one metrics parameter. Each parameter must have a valid JSON object format. The following is an example parameter value:

```
{\"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE, SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION, 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait
```

```
Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0,
TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0))
IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor'
GROUP BY HOST, MEASURED_ELEMENT_NAME;\":{\\"enabled\\":true,\\"metrics\\":[{\\"name\\":
\\"hanadb_cpu_user\\",\\"description\\":\\"Percentage of CPU time spent by HANA DB in user
space, over the last minute (in seconds)\\",\\"labels\\":[\\"HOST\\",\\"CORE\\"],\\"value\\":
\\"USER_PCT\\",\\"unit\\":\\"percent\\",\\"type\\":\\"gauge\\"},{\\"name\\":\\"hanadb_cpu_system
\\",\\"description\\":\\"Percentage of CPU time spent by HANA DB in Kernel space,
over the last minute (in seconds)\\",\\"labels\\":[\\"HOST\\",\\"CORE\\"],\\"value\\":
\\"SYSTEM_PCT\\",\\"unit\\":\\"percent\\",\\"type\\":\\"gauge\\"},{\\"name\\":\\"hanadb_cpu_waitio
\\",\\"description\\":\\"Percentage of CPU time spent by HANA DB in IO mode, over the
last minute (in seconds)\\",\\"labels\\":[\\"HOST\\",\\"CORE\\"],\\"value\\":\\"WAITIO_PCT\\",
\\"unit\\":\\"percent\\",\\"type\\":\\"gauge\\"},{\\"name\\":\\"hanadb_cpu_busy\\",\\"description
\\":\\"Percentage of CPU time spent by HANA DB, over the last minute (in seconds)\\",
\\"labels\\":[\\"HOST\\",\\"CORE\\"],\\"value\\":\\"BUSY_PCT\\",\\"unit\\":\\"percent\\",\\"type\\":
\\"gauge\\"},{\\"name\\":\\"hanadb_cpu_idle\\",\\"description\\":\\"Percentage of CPU time not
spent by HANA DB, over the last minute (in seconds)\\",\\"labels\\":[\\"HOST\\",\\"CORE
\\"],\\"value\\":\\"IDLE_PCT\\",\\"unit\\":\\"percent\\",\\"type\\":\\"gauge\\"}}}]}
```

For more information about metrics queries, see the [SUSE / hanadb_exporter](#) repo on GitHub.

3. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure*** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{srm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_EXPORTER_METRICS_1": "{srm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}",
  "SSM_EXPORTER_METRICS_2": "{srm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}"
}
```

5. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

AWSObservabilityExporter-HAClusterExporterInstallAndConfigure

You can retrieve workload-specific High Availability (HA) cluster metrics from [Prometheus HANA cluster exporter](#) for Application Insights to configure and monitor alarms for an SAP HANA

database High Availability setup. For more information, see [Set up your SAP HANA database for monitoring](#) in this guide.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the AWS-provided Prometheus HA cluster exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HA cluster exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- HA cluster for Pacemaker, Corosync, SBD, and DRBD
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the AWSObservabilityExporter-HAClusterExporterInstallAndConfigure package

The AWSObservabilityExporter-HAClusterExporterInstallAndConfigure package is an SSM Distributor package that you can use to install and configure Prometheus HA Cluster Exporter. When cluster metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter configurations in JSON format. The following is an example parameter value.

```
{\"port\": \"9664\", \"address\": \"0.0.0.0\", \"log-level\": \"info\", \"crm-mon-path\": \"/usr/sbin/crm_mon\", \"cibadmin-path\": \"/usr/sbin/cibadmin\", \"corosync-cfgtool-path\": \"/usr/sbin/corosync-cfgtool\", \"corosync-quorumtool-path\": \"/usr/sbin/corosync-quorumtool\", \"sbd-path\": \"/usr/sbin/sbd\", \"sbd-config-path\": \"/etc/sysconfig/sbd\", \"drbdsetup-path\": \"/sbin/drbdsetup\", \"enable-timestamps\": false}
```

For more information about the exporter configurations, see the [ClusterLabs / ha_cluster_exporter](#) repo on GitHub.

2. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure*** and choose **Install one time**.
3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"

```

4. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure

You can retrieve workload-specific SAP NetWeaver metrics from [Prometheus SAP host exporter](#) for Application Insights to configure and monitor alarms for SAP NetWeaver Distributed and High Availability deployments. For more information, see [Get started with Amazon CloudWatch Application Insights](#).

To use [AWS Systems Manager Distributor](#) to package, install, and configure the SAP host exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SAP host exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP NetWeaver application servers
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure package

The `AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure SAP NetWeaver Prometheus metrics exporter. When SAP NetWeaver metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in [SSM Parameter Store](#) to store the Exporter configurations in JSON format. The following is an example parameter value.

```
{\"address\": \"0.0.0.0\", \"port\": \"9680\", \"log-level\": \"info\", \"is-HA\": false}
```

- **address**

The target address to which to send the Prometheus metrics. The default value is `localhost`.

- **port**

The target port to which to send the Prometheus metrics. The default value is 9680.

- **is-HA**

true for SAP NetWeaver High Availability deployments. For all other deployments the value is false.

2. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure** and choose **Install one time**.
3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_INSTANCES_NUM": "<instances_number seperated by comma>"
}
```

Example

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:exporter_config_paramter}}",
  "SSM_INSTANCES_NUM": "11,12,10",
  "SSM_SID": "PR1"
}
```

4. Select the Amazon EC2 instances with SAP NetWeaver applications, and choose **Run**.

Note

The Prometheus exporter services the SAP NetWeaver metrics on a local endpoint. The local endpoint can be accessed by only the operating system users on the Amazon EC2 instance. Therefore, after the exporter package is installed, the metrics are available to all of the operating system users. The default local endpoint is localhost:9680/metrics.

AWSObservabilityExporter-SQLExporterInstallAndConfigure

You can retrieve workload-specific SQL Server metrics from [Prometheus SQL exporter](#) for Application Insights to monitor key metrics.

To use [AWS Systems Manager Distributor](#) to package, install, and configure the SQL exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SQL exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- Amazon EC2 instance running SQL Server on Windows with SQL Server user authentication enabled.
- A SQL Server user with the following permissions:

```
GRANT VIEW ANY DEFINITION TO
```

```
GRANT VIEW SERVER STATE TO
```

- A secret containing the database connection string using AWS Secrets Manager. For more information about how to create secrets, see [Create a secret](#) in the *AWS Secrets Manager User Guide*. The secret must be formatted as follows:

```
{  
  "data_source_name": "sqlserver://<username>:<password>@localhost:1433"  
}
```

Note

If the password or username contains special characters, you must percent encode the special characters to ensure a successful connection to the database.

Install and configure the AWSObservabilityExporter-SQLExporterInstallAndConfigure package

The AWSObservabilityExporter-SQLExporterInstallAndConfigure package is an SSM Distributor package that you can use to install and configure SQL Prometheus metrics exporter.

When metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the SQL Exporter YAML configuration. The following sample configuration has a single metric configured. Use the [example configuration](#) to update the configuration with additional metrics or create your own configuration.

```
---
global:
  scrape_timeout_offset: 500ms
  min_interval: 0s
  max_connections: 3
  max_idle_connections: 3
target:
  aws_secret_name: <SECRET_NAME>
collectors:
  - mssql_standard
collectors:
  - collector_name: mssql_standard
    metrics:
      - metric_name: mssql_batch_requests
        type: counter
        help: 'Number of command batches received.'
        values: [cntr_value]
        query: |
          SELECT cntr_value
          FROM sys.dm_os_performance_counters WITH (NOLOCK)
          WHERE counter_name = 'Batch Requests/sec'
```

2. Copy the Prometheus SQL exporter YAML configuration file encoded as Base64 to a new SSM parameter in [SSM Parameter Store](#).
3. Navigate to the [SSM Distributor](#) console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SQLExporterInstallAndConfigure** and choose **Install one time**.
4. Replace the "Additional Arguments" with the following information. The SSM_PARAMETER_NAME is the name of the parameter you created in Step 2.

```
{
  "SSM_EXPORTER_CONFIGURATION":
    "{{srm:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_PROMETHEUS_PORT": "9399",
```

```
"SSM_WORKLOAD_NAME": "SQL"
}
```

5. Select the Amazon EC2 instance with the SQL Server database, then choose run.

AWS Systems Manager (SSM) Documents used by CloudWatch Application Insights

Application Insights uses the SSM Documents listed in this section to define the actions that AWS Systems Manager performs on your managed instances. These documents use the Run Command capability of Systems Manager to automate the tasks necessary for carrying out Application Insights monitoring capabilities. The run schedules for these documents are maintained by Application Insights and can't be altered.

For more information about SSM Documents, see [AWS Systems Manager Documents](#) in the *AWS Systems Manager User Guide*.

Documents managed by CloudWatch Application Insights

The following table lists the SSM documents that are managed by Application Insights.

Document name	Description	Run schedule
AWSEC2-DetectWorkload	Auto detects applications running in your application environment that can be set up to be monitored by Application Insights.	This document runs hourly in your application environment to get up-to-date application details.
AWSEC2-CheckPerformanceCounterSets	Checks whether Performance Counter namespaces are enabled on your Amazon EC2 Windows instances.	This document runs hourly in your application environment and only monitors Performance Counter metrics if the corresponding namespaces are enabled.
AWSEC2-ApplicationInsightsCloudwatch	Installs and configures CloudWatch Agent based	This document runs every 30 minutes to ensure that the

Document name	Description	Run schedule
AgentInstallAndConfigure	on the monitoring configuration of your application components.	CloudWatch Agent configuration is always accurate and up-to-date. The document also runs immediately after a change is made to your application monitoring setup such as adding or removing metrics or updating log configurations.

Documents managed by AWS Systems Manager

The following documents are used by CloudWatch Application Insights and managed by Systems Manager.

AWS-ConfigureAWSPackage

Application Insights uses this document to install and uninstall Prometheus exporter distributor packages, to collect workload specific metrics, and to enable comprehensive monitoring of workloads on customer Amazon EC2 instances. CloudWatch Application Insights installs the Prometheus exporter distributor packages only if the correlated target workload is running on your instance.

The following table lists the Prometheus exporter distributor packages and the correlated target workloads.

Prometheus exporter distributor package name	Target workload
AWSObservabilityExporter-HA ClusterExporterInstallAndConfigure	SAP HANA HA
AWSObservabilityExporter-JMX ExporterInstallAndConfigure	Java/JMX

Prometheus exporter distributor package name	Target workload
AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure	SAP HANA
AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure	NetWeaver
AWSObservabilityExporter-SQL-ExporterInstallAndConfigure	SQL Server (Windows) and SAP ASE (Linux)

AmazonCloudWatch-ManagedAgent

Application Insights uses this document to manage the status and configuration of CloudWatch Agent on your instances and to collect internal system level metrics and logs from Amazon EC2 instances across operating systems.

Get started with Amazon CloudWatch Application Insights

To get started with CloudWatch Application Insights, verify that you have met the following prerequisites and have created an IAM policy. Then, you can get started using the console link to enable CloudWatch Application Insights. To configure your application resources, follow the steps under [Set up, configure, and manage your application for monitoring](#).

Contents

- [Access CloudWatch Application Insights](#)
- [Prerequisites](#)
- [IAM policy](#)
- [IAM role permissions for account-based application onboarding](#)
- [Set up, configure, and manage your application for monitoring](#)

Access CloudWatch Application Insights

You can access and manage CloudWatch Application Insights through one of the following interfaces:

- **CloudWatch console.** To add monitors for your application, choose **Application Insights** under **Insights** in the left navigation pane of the [CloudWatch console](#). After your application is configured, you can use the [CloudWatch console](#) to view and analyze problems that are detected.
- **AWS Command Line Interface (AWS CLI).** You can use the AWS CLI to access AWS API operations. For more information, see [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*. For **Application Insights** API information, see the [Amazon CloudWatch Application Insights API Reference](#).

Prerequisites

You must complete the following prerequisites to configure an application with CloudWatch Application Insights:

- **AWS Systems Manager enablement** – Install Systems Manager Agent (SSM Agent) on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see [Setting up AWS Systems Manager](#) in the *AWS Systems Manager User Guide*.
- **EC2 instance role** – You must attach the following Amazon EC2 instance roles to enable Systems Manager
 - You must attach the AmazonSSMManagedInstanceCore role to enable Systems Manager. For more information, see [AWS Systems Manager identity-based policy examples](#).
 - You must attach the CloudWatchAgentServerPolicy policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).
- **AWS resource groups** – To onboard your applications to CloudWatch Application Insights, create a resource group that includes all of the associated AWS resources used by your application stack. This includes application load balancers, Amazon EC2 instances running IIS and web front-end, .NET worker tiers, and SQL Server databases. For more information about application components and technology stacks supported by Application Insights, see [Supported application components](#). CloudWatch Application Insights automatically includes Auto Scaling groups using the same tags or CloudFormation stacks as your resource group, because Auto Scaling groups are

not supported by CloudFormation resource groups. For more information, see [Getting Started with AWS Resource Groups](#).

- **IAM permissions** – For users who don't have administrative access, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see [IAM policy](#).
- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).
- **Performance Counter metrics support for EC2 Windows instances** – To monitor Performance Counter metrics on your Amazon EC2 Windows instances, Performance Counters must be installed on the instances. For Performance Counter metrics and corresponding Performance Counter set names, see [Performance Counter metrics](#). For more information about Performance Counters, see [Performance Counters](#).
- **Amazon CloudWatch agent** – Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application Insights from the existing CloudWatch agent configuration file. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

IAM policy

To use CloudWatch Application Insights, you must create an [AWS Identity and Access Management \(IAM\) policy](#) and attach it to your user, group, or role. For more information about users, groups, and roles, see [IAM Identities \(users, user groups, and roles\)](#). The IAM policy defines the user permissions.

To create an IAM policy using the console

To create an IAM policy using the IAM console, perform the following steps.

1. Go to the [IAM console](#). In the left navigation pane, select **Policies**.
2. At the top of the page, select **Create policy**.
3. Select the **JSON** tab.
4. Copy and paste the following JSON document under the **JSON** tab.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "applicationinsights:*",
        "iam:CreateServiceLinkedRole",
        "iam:ListRoles",
        "resource-groups:ListGroup"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

5. Select **Review Policy**.
6. Enter a **Name** for the policy, for example, "AppInsightsPolicy." Optionally, enter a **Description**.
7. Select **Create Policy**.
8. In the left navigation pane, select **User groups**, **Users**, or **Roles**.
9. Select the name of the user group, user, or role to which you would like to attach the policy.
10. Select **Add permissions**.
11. Select **Attach existing policies directly**.
12. Search for the policy that you just created, and select the check box to the left of the policy name.
13. Select **Next: Review**.
14. Make sure that the correct policy is listed, and select **Add permissions**.
15. Make sure that you log in with the user associated with the policy that you just created when you use CloudWatch Application Insights.

To create an IAM policy using the AWS CLI

To create an IAM policy using the AWS CLI, run the [create-policy](#) operation from the command line using the JSON document above as a file in your current folder.

To create an IAM policy using AWS Tools for Windows PowerShell

To create an IAM policy using the AWS Tools for Windows PowerShell, run the [New-IAMPolicy cmdlet](#) using the JSON document above as a file in your current folder.

IAM role permissions for account-based application onboarding

If you want to onboard all of the resources in your account, and you choose not to use the [Application Insights managed policy](#) for full access to Application Insights functionality, you must attach the following permissions to your IAM role so that Application Insights can discover all of the resources in your account:

```
"ec2:DescribeInstances"  
"ec2:DescribeNatGateways"  
"ec2:DescribeVolumes"  
"ec2:DescribeVPCs"  
"rds:DescribeDBInstances"  
"rds:DescribeDBClusters"  
"sqs:ListQueues"  
"elasticloadbalancing:DescribeLoadBalancers"  
"autoscaling:DescribeAutoScalingGroups"  
"lambda:ListFunctions"  
"dynamodb:ListTables"  
"s3:ListAllMyBuckets"  
"sns:ListTopics"  
"states:ListStateMachines"  
"apigateway:GET"  
"ecs:ListClusters"  
"ecs:DescribeTaskDefinition"  
"ecs:ListServices"  
"ecs:ListTasks"  
"eks:ListClusters"  
"eks:ListNodegroups"  
"fsx:DescribeFileSystems"  
"route53:ListHealthChecks"  
"route53:ListHostedZones"  
"route53:ListQueryLoggingConfigs"  
"route53resolver:ListFirewallRuleGroups"  
"route53resolver:ListFirewallRuleGroupAssociations"  
"route53resolver:ListResolverEndpoints"  
"route53resolver:ListResolverQueryLogConfigs"  
"route53resolver:ListResolverQueryLogConfigAssociations"  
"logs:DescribeLogGroups"  
"resource-explorer:ListResources"
```

Set up, configure, and manage your application for monitoring

This section provides steps to set up, configure, and manage your CloudWatch Application Insights application using the console, the AWS CLI, and AWS Tools for Windows PowerShell.

Topics

- [Set up, configure, and manage your application for monitoring from the CloudWatch console](#)
- [Set up, configure, and manage your application for monitoring using the command line](#)
- [Application Insights CloudWatch Events and notifications for detected problems](#)

Set up, configure, and manage your application for monitoring from the CloudWatch console

This section provides steps to set up, configure, and manage your application for monitoring from the CloudWatch console.

Console procedures

- [Add and configure an application](#)
- [Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring](#)
- [Disable monitoring for an application component](#)
- [Delete an application](#)


Add and configure an application

Add and configure an application from the CloudWatch console

To get started with CloudWatch Application Insights from the CloudWatch console, perform the following steps.

1. **Start.** Open the [CloudWatch console landing page](#). From the left navigation pane, under **Insights**, choose **Application Insights**. The page that opens shows the list of applications that are monitored with CloudWatch Application Insights, along with their monitoring status.
2. **Add an application.** To set up monitoring for your application, choose **Add an application**. When you choose **Add an application**, you are prompted to **Choose Application Type**.
 - **Resource group-based application.** When you select this option, you can choose which resource groups in this account to monitor. To use multiple applications on a component, you must use resource group-based monitoring.

- **Account-based application.** When you select this option, you can monitor all of the resources in this account. If you want to monitor all of the resources in an account, we recommend this option over the resource group-based option because the application onboarding process is faster.

 **Note**

You can't combine resource group-based monitoring with account-based monitoring using Application Insights. In order to change the application type, you must delete all of the applications that are being monitored, and **Choose Application Type**.

When you add your first application for monitoring, CloudWatch Application Insights creates a service-linked role in your account, which gives Application Insights permissions to call other AWS services on your behalf. For more information about the service-linked role created in your account by Application Insights, see [Using service-linked roles for CloudWatch Application Insights](#).

3. Resource-based application monitoring

1. **Select resource group.** On the **Specify application details** page, select the AWS resource group that contains your application resources from the dropdown list. These resources include front-end servers, load balancers, auto scaling groups, and database servers.

If you have not created a resource group for your application, you can create one by choosing **Create new resource group**. For more information about creating resource groups, see the [AWS Resource Groups User Guide](#).

2. **Monitor CloudWatch Events.** Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
3. **Integrate with AWS Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the SNS topic ARN.

4. **Tags — optional.** CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see [Working with Tag Editor](#).
5. Choose **Next**.

An [ARN](#) is generated for the application in the following format.

```
arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
```

Example

```
arn:aws:applicationinsights:us-east-1:123456789012:application/resource-group/my-resource-group
```

6. On the **Review detected components** page, under **Review components for monitoring**, the table lists the detected components and their associated detected workloads.

Note

For components that support multiple customized workloads, you can monitor up to five workloads for each component. These workloads will be monitored separately from the component.

Review detected components [Info](#)


▼ Selected application

Application
test-MW-W19

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/test-MW-W19

Review components for monitoring (1) [Info](#) Edit component

Components and their workloads detected by Application Insights.

Detected components	Monitoring	Associated workloads
<input type="radio"/> EC2 instance group i-0a0858a7fd11cd51c: windows 2019	 Enabled	<ul style="list-style-type: none"> • DN_CORE (.NET Core tier) • JAVA1 (JAVA application)

Cancel Previous Next

Under **Associated workloads**, there are several possible messages that appear if a workload is not listed.

- **Couldn't detect workloads** – An issue occurred when trying to detect workloads. Make sure that you have completed the [Prerequisites](#). If you need to add workloads, choose **Edit component**.
 - **No workloads detected** – We didn't detect any workloads. You may need to add workloads. To do so, choose **Edit component**.
 - **Not applicable** – The component doesn't support customized workloads and will be monitored with default metrics, alarms, and logs. You can't add workloads to these components.
7. To edit a component, select a component, and then choose **Edit component**. A side panel opens with workloads detected on the component. In this panel, you can edit the component details and add new workloads.

Review detected components [info](#)

▼ Selected application

Application
test-MW-W19

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/test-MW-W19

Review components for monitoring (1/1) [info](#) Edit component

Components and their workloads detected by Application Insights.

< 1 > ⚙

Detected components	Monitoring	Associated workloads
<ul style="list-style-type: none"> EC2 instance group i-0a0858a7fd11cd51c; windows 2019 	<ul style="list-style-type: none"> Enabled 	<ul style="list-style-type: none"> DN_CORE (.NET Core tier) JAVA1 (JAVA application)

Cancel Previous **Next**

- To edit the workload type or name, use the dropdown list.

Review detected components [Info](#)

▼ **Selected application**

Application
test-MW-W19

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/test-MW-W19

Review components for monitoring (1/1) [Info](#) Edit component

Components and their workloads detected by Application Insights.

Find components

Detected components	Monitoring	Associate...
EC2 instance group i-0a0858a7fd11cd51c: windows 2019	Enabled	<ul style="list-style-type: none"> DN_CORE (.NET) JAVA1 (JAVA ap)

Cancel Previous Next

Edit component ×

Component type
Amazon EC2 instance

Component name
i-0a0858a7fd11cd51c: windows 2019

Monitoring
 Enabled
Monitoring includes key metrics, logs, and alarms.

Associated workloads

Some workload types support adding only one workload of that type on a component. For more information about workload types supported by Application Insights, see [Documentation](#).

Workload type	Workload name	
.NET Core tier	DN_CORE	Remove
JAVA application	JAVA1	Remove

Add new workload

You can add up to 5 workloads

Cancel Save changes

- To add a workload to the component, choose **Add new workload**.

Review detected components [Info](#)

▼ **Selected application**

Application
test-MW-W19

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/test-MW-W19

Review components for monitoring (1/1) [Info](#) Edit component

Components and their workloads detected by Application Insights.

Find components

Detected components	Monitoring	Associate...
EC2 instance group i-0a0858a7fd11cd51c: windows 2019	Enabled	<ul style="list-style-type: none"> DN_CORE (.NET) JAVA1 (JAVA ap)

Cancel Previous Next

Edit component ×

Component type
Amazon EC2 instance

Component name
i-0a0858a7fd11cd51c: windows 2019

Monitoring
 Enabled
Monitoring includes key metrics, logs, and alarms.

Associated workloads

Some workload types support adding only one workload of that type on a component. For more information about workload types supported by Application Insights, see [Documentation](#).

Workload type	Workload name	
.NET Core tier	DN_CORE	Remove
JAVA application	JAVA1	Remove

Add new workload

You can add up to 5 workloads

Cancel Save changes

- If **Add new workload** doesn't appear, this component doesn't support multiple workloads.
- If the **Associated workloads** heading doesn't appear, this component doesn't support customized workloads.
- To remove a workload, choose **Remove** next to the workload that you want to remove from monitoring.

The screenshot shows two panels side-by-side. The left panel, titled 'Review detected components', shows a table with one component: 'EC2 instance group' with monitoring status 'Enabled'. The right panel, titled 'Edit component', shows details for the selected component. In the 'Monitoring' section, the 'Enabled' checkbox is checked and circled in red. Below it, there are two 'Associated workloads' listed: '.NET Core tier' and 'JAVA application', each with a 'Remove' button.

- To disable monitoring for the entire component, clear the **Monitoring** check box.

This screenshot is similar to the previous one, but in the 'Edit component' panel, the 'Monitoring' checkbox is now unchecked and circled in red. The 'Associated workloads' section remains the same.

- When you are done editing the component, choose **Save changes** in the lower right corner. Any changes to workloads for a component are visible on the **Review components for monitoring** table under **Associated workloads**.
8. On the **Review detected components** page, choose **Next**.
 9. The **Specify component details** page includes all components with customizable associated workloads from the previous step.

Note

If a component header has an *optional* tag, additional details for the workloads in that component are optional.

If a component doesn't appear on this page, the component doesn't have any additional details that can be specified in this step.

10. Choose **Next**.

11. On the **Review and submit** page, review all monitored component and workload details.

12. Choose **Submit**.

Account-based application monitoring

1. **Application name.** Enter a name for your account-based application.
2. **Automated monitoring of new resources.** By default, Application Insights uses recommended settings to configure monitoring for resource components that are added to your account after you onboard the application. You can exclude monitoring for resources added after onboarding your application by clearing the check box.
3. **Monitor CloudWatch Events.** Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
4. **Integrate with AWS Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the SNS topic ARN.
5. **Tags — optional.** CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see [Working with Tag Editor](#).
6. **Discovered resources.** All of the resources discovered in your account are added to this list. If Application Insights is unable to discover all of the resources in your account, an error message appears at the top of the page. This message includes a link to the [documentation for how to add the required permissions](#).
7. Choose **Next**.

An [ARN](#) is generated for the application in the following format.


```
arn:partition:applicationinsights:region:account-id:application/  
TBD/application-name
```

Example

```
arn:aws:applicationinsights:us-east-1:123456789012:application/TBD/my-  
application
```

4. After you submit your application monitoring configuration, you will be taken to the details page for the application, where you can view the **Application summary**, the list of **Monitored components** and **Unmonitored components**, and, by selecting the tabs next to **Components**, the **Configuration history**, **Log patterns**, and any **Tags** that you have applied.

To view insights for the application, choose **View Insights**.

You can update your selections for CloudWatch Events monitoring and integration with AWS Systems Manager OpsCenter by choosing **Edit**.

Under **Components**, you can select the **Actions** menu to Create, Modify, or Ungroup an instance group.

You can manage monitoring for components, including application tier, log groups, event logs, metrics, and custom alarms, by selecting the bullet next to a component and choosing **Manage monitoring**.

Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring

You can enable Application Insights to monitor containerized applications and microservices from the Container Insights console. Application Insights supports monitoring for the following resources:

- Amazon ECS clusters
- Amazon ECS services
- Amazon ECS tasks
- Amazon EKS clusters

When Application Insights is enabled, it provides recommended metrics and logs, detects potential problems, generates CloudWatch Events, and creates automatic dashboards for your containerized applications and microservices.

You can enable Application Insights for containerized resources from the Container Insights or Application Insights consoles.

Enable Application Insights from the Container Insights console

From the Container Insights console, on the Container Insights **Performance monitoring** dashboard, choose **Auto-configure Application Insights**. When Application Insights is enabled, it displays details about detected problems.

Enable Application Insights from the Application Insights console

When ECS clusters appear in the component list, Application Insights automatically enables additional container monitoring with Container Insights.

For EKS clusters, you can enable additional monitoring with Container Insights to provide diagnostics information, such as container restart failures, to help you isolate and resolve problems. Additional steps are required to set up Container Insights for EKS. For information, see [Setting up Container Insights on Amazon EKS and Kubernetes](#) for steps to set up Container Insights on EKS.

Additional monitoring for EKS with Container Insights is supported on Linux instances with EKS.

For more information about Container Insights support for ECS and EKS clusters, see [Container Insights](#).

Disable monitoring for an application component

To disable monitoring for an application component, from the application details page, select the component for which you want to disable monitoring. Choose **Actions**, and then **Remove from monitoring**.

Delete an application

To delete an application, from the CloudWatch dashboard, on the left navigation pane, choose **Application Insights** under **Insights**. Select the application that you want to delete. Under **Actions**, choose **Delete application**. This deletes monitoring and deletes all of the saved monitors for application components. The application resources are not deleted.

Set up, configure, and manage your application for monitoring using the command line

This section provides steps for setting up, configuring, and managing your application for monitoring using the AWS CLI and AWS Tools for Windows PowerShell.

Command line procedures

- [Add and manage an application](#)
- [Manage and update monitoring](#)
- [Configure monitoring for SQL Always On Availability Groups](#)
- [Configure monitoring for MySQL RDS](#)
- [Configure monitoring for MySQL EC2](#)
- [Configure monitoring for PostgreSQL RDS](#)
- [Configure monitoring for PostgreSQL EC2](#)
- [Configure monitoring for Oracle RDS](#)
- [Configure monitoring for Oracle EC2](#)

Add and manage an application

You can add, get information about, manage, and configure your Application Insights application using the command line.

Topics

- [Add an application](#)
- [Describe an application](#)
- [List components in an application](#)
- [Describe a component](#)
- [Group similar resources into a custom component](#)
- [Ungroup a custom component](#)
- [Update an application](#)
- [Update a custom component](#)

Add an application

Add an application using the AWS CLI

To use the AWS CLI to add an application for your resource group called `my-resource-group`, with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
aws application-insights create-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Add an application using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to add an application for your resource group called `my-resource-group` with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
New-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Describe an application

Describe an application using the AWS CLI

To use the AWS CLI to describe an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights describe-application --resource-group-name my-resource-group
```

Describe an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an application created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIApplication -ResourceGroupName my-resource-group
```

List components in an application

List components in an application using the AWS CLI

To use the AWS CLI to list the components created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-components --resource-group-name my-resource-group
```

List components in an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list the components created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAComponentList -ResourceGroupName my-resource-group
```

Describe a component

Describe a component using the AWS CLI

You can use the following AWS CLI command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
aws application-insights describe-component --resource-group-name my-resource-group --  
component-name my-component
```

Describe a component using AWS Tools for Windows PowerShell

You can use the following AWS Tools for Windows PowerShell command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
Get-CWAComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Group similar resources into a custom component

We recommend grouping similar resources, such as .NET web server instances, into custom components for easier onboarding and better monitoring and insights. Currently, CloudWatch Application Insights supports custom groups for EC2 instances.

To group resources into a custom component using the AWS CLI

To use the AWS CLI to group three instances (`arn:aws:ec2:us-east-1:123456789012:instance/i-11111`, `arn:aws:ec2:us-east-1:123456789012:instance/i-22222`, and `arn:aws:ec2:us-east-1:123456789012:instance/i-33333`) together into a custom component called `my-component` for an application created for the resource group called `my-resource-group`, use the following command.

```
aws application-insights create-component --resource-group-name my-  
resource-group --component-name my-component --resource-list arn:aws:ec2:us-  
east-1:123456789012:instance/i-11111 arn:aws:ec2:us-east-1:123456789012:instance/  
i-22222 arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

To group resources into a custom component using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to group three instances (arn:aws:ec2:us-east-1:123456789012:instance/i-11111, arn:aws:ec2:us-east-1:123456789012:instance/i-22222, and arn:aws:ec2:us-east-1:123456789012:instance/i-33333) together into a custom component called my-component, for an application created for the resource group called my-resource-group, use the following command.

```
New-CWAComponent -ResourceGroupName my-resource-group -ComponentName my-component  
-ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-11111,arn:aws:ec2:us-  
east-1:123456789012:instance/i-22222,arn:aws:ec2:us-east-1:123456789012:instance/  
i-33333
```

Ungroup a custom component

To ungroup a custom component using the AWS CLI

To use the AWS CLI to ungroup a custom component named my-component in an application created on the resource group, my-resource-group, use the following command.

```
aws application-insights delete-component --resource-group-name my-resource-group --  
component-name my-new-component
```

To ungroup a custom component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to ungroup a custom component named my-component in an application created on the resource group, my-resource-group, use the following command.

```
Remove-CWAComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Update an application

Update an application using the AWS CLI

You can use the AWS CLI to update an application to generate AWS Systems Manager OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic`, using the following command.

```
aws application-insights update-application --resource-group-name my-resource-group --ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update an application using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update an application to generate AWS SSM OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic`, using the following command.

```
Update-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update a custom component

Update a custom component using the AWS CLI

You can use the AWS CLI to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
aws application-insights update-component --resource-group-name my-resource-group --component-name my-component --new-component-name my-new-component --resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-44444 arn:aws:ec2:us-east-1:123456789012:instance/i-55555
```

Update a custom component using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-component -ResourceGroupName my-resource-group -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-44444,arn:aws:ec2:us-east-1:123456789012:instance/i-55555
```

Manage and update monitoring

You can manage and update monitoring for your Application Insights application using the command line.

Topics

- [List problems with your application](#)
- [Describe an application problem](#)
- [Describe the anomalies or errors associated with a problem](#)
- [Describe an anomaly or error with the application](#)
- [Describe the monitoring configurations of a component](#)
- [Describe the recommended monitoring configuration of a component](#)
- [Update the monitoring configurations for a component](#)
- [Remove a specified resource group from Application Insights monitoring](#)

List problems with your application

List problems with your application using the AWS CLI

To use the AWS CLI to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-problems --resource-group-name my-resource-group --start-time 1000 --end-time 10000
```

List problems with your application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
$startDate = "8/6/2019 3:33:00"  
$endDate = "8/6/2019 3:34:00"  
Get-CWAIProblemList -ResourceGroupName my-resource-group -StartTime $startDate -  
EndTime $endDate
```


Describe an application problem

Describe an application problem using the AWS CLI

To use the AWS CLI to describe a problem with problem id p-1234567890, use the following command.

```
aws application-insights describe-problem --problem-id p-1234567890
```

Describe an application problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe a problem with problem id p-1234567890, use the following command.

```
Get-CWAIPProblem -ProblemId p-1234567890
```

Describe the anomalies or errors associated with a problem

Describe the anomalies or errors associated with a problem using the AWS CLI

To use the AWS CLI to describe the anomalies or errors associated with a problem with problem id p-1234567890, use the following command.

```
aws application-insights describe-problem-observations --problem-id p-1234567890
```

Describe the anomalies or errors associated with a problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the anomalies or errors associated with a problem with problem id p-1234567890, use the following command.

```
Get-CWAIPProblemObservation -ProblemId p-1234567890
```

Describe an anomaly or error with the application

Describe an anomaly or error with the application using the AWS CLI

To use the AWS CLI to describe an anomaly or error with the application with the observation id o-1234567890, use the following command.

```
aws application-insights describe-observation --observation-id o-1234567890
```

Describe an anomaly or error with the application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an anomaly or error with the application with the observation id `o-1234567890`, use the following command.

```
Get-CWAI0bservation -ObservationId o-1234567890
```

Describe the monitoring configurations of a component

Describe the monitoring configurations of a component using the AWS CLI

To use the AWS CLI to describe the monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, use the following command.

```
aws application-insights describe-component-configuration --resource-group-name my-resource-group --component-name my-component
```

Describe the monitoring configurations of a component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the monitoring configuration of a component called `my-component`, in an application created on the resource group `my-resource-group`, use the following command.

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-resource-group
```

For more information about component configuration and for example JSON files, see [Work with component configurations](#).

Describe the recommended monitoring configuration of a component

Describe the recommended monitoring configuration of a component using the AWS CLI

When the component is part of a .NET Worker application, you can use the AWS CLI to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
aws application-insights describe-component-configuration-recommendation --resource-group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

Describe the recommended monitoring configuration of a component using AWS Tools for Windows PowerShell

When the component is part of a .NET Worker application, you can use the AWS Tools for Windows PowerShell to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
Get-CWAComponentConfigurationRecommendation -ComponentName my-component -  
ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

For more information about component configuration and for example JSON files, see [Work with component configurations](#).

Update the monitoring configurations for a component

Update the monitoring configurations for a component using the AWS CLI

To use the AWS CLI to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
aws application-insights update-component-configuration --resource-group-name my-  
resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --  
component-configuration "file://configuration.txt"
```

Update the monitoring configurations for a component using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.

2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
[string]$config = Get-Content -Path configuration.txt  
Update-CWAIDComponentConfiguration -ComponentName my-component -ResourceGroupName my-  
resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

For more information about component configuration and for example JSON files, see [Work with component configurations](#).

Remove a specified resource group from Application Insights monitoring

Remove a specified resource group from Application Insights monitoring using the AWS CLI

To use the AWS CLI to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
aws application-insights delete-application --resource-group-name my-resource-group
```

Remove a specified resource group from Application Insights monitoring using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
Remove-CWAIAApplication -ResourceGroupName my-resource-group
```

Configure monitoring for SQL Always On Availability Groups

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application --region <REGION> --resource-group-name  
<RESOURCE_GROUP_NAME>
```

2. Define the EC2 instances that represent the SQL HA cluster by creating a new application component.

```
aws application-insights create-component --resource-group-name  
"<RESOURCE_GROUP_NAME>" --component-name SQL_HA_CLUSTER --resource-list
```

```
"arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>"
"arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>"
```

3. Configure the SQL HA component.

```
aws application-insights update-component-configuration --resource-group-name
"<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "SQL_HA_CLUSTER" --
monitor --tier SQL_SERVER_ALWAYS_ON_AVAILABILITY_GROUP --monitor --component-
configuration '{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }
  ]
}
```

```
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/
sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    }
  ],
```

```

"windowsEvents" : [ {
  "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
  "eventName" : "Application",
  "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
  "monitor" : true
}, {
  "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
  "eventName" : "System",
  "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
  "monitor" : true
}, {
  "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
  "eventName" : "Security",
  "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
  "monitor" : true
} ],
"logs" : [ {
  "logGroupName" : "SQL_SERVER_ALWAYS_ON_AVAILABILITY_GROUP-
<RESOURCE_GROUP_NAME>",
  "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**\\MSSQLSERVER\\
\\MSSQL\\Log\\ERRORLOG",
  "logType" : "SQL_SERVER",
  "monitor" : true,
  "encoding" : "utf-8"
} ]
}, {
  "subComponentType" : "AWS::EC2::Volume",
  "alarmMetrics" : [ {
    "alarmMetricName" : "VolumeReadBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeReadOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeQueueLength",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeThroughputPercentage",

```

```

    "monitor" : true
  }, {
    "alarmMetricName" : "BurstBalance",
    "monitor" : true
  } ]
} ]
}'

```

Note

Application Insights must ingest Application Event logs (information level) to detect cluster activities such as failover.

Configure monitoring for MySQL RDS

1. Create an application for the resource group with the RDS MySQL database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
<RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see [Accessing the MySQL Slow Query and General Logs](#).

- set slow_query_log = 1
- set log_output = FILE

3. Export the logs to be monitored to CloudWatch logs. For more information, see [Publishing MySQL Logs to CloudWatch Logs](#).

4. Configure the MySQL RDS component.

```
aws application-insights update-component-configuration --resource-group-name
"<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
--monitor --tier DEFAULT --monitor --component-configuration "{\"alarmMetrics\":
[{\\"alarmMetricName\\":\\"CPUUtilization\\",\\"monitor\\":true}],\\"logs\\":[{\\"logType\\":
\\"MYSQL\\",\\"monitor\\":true},{\\"logType\\": \\"MYSQL_SLOW_QUERY\\",\\"monitor\\":false}]}"
```

Configure monitoring for MySQL EC2

1. Create an application for the resource group with the SQL HA EC2 instances.


```
aws application-insights create-application --region <REGION> --resource-group-name
<RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see [Accessing the MySQL Slow Query and General Logs](#).

- set `slow_query_log = 1`
- set `log_output = FILE`

3. Configure the MySQL EC2 component.

```
aws application-insights update-component-configuration --resource-group-name
"<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
--monitor --tier MYSQL --monitor --component-configuration "{\"alarmMetrics\":
[{\\"alarmMetricName\\":\\"CPUUtilization\\",\\"monitor\\":true}],\\"logs\\":[{\\"logGroupName
\\":\\"<UNIQUE_LOG_GROUP_NAME>\\",\\"logPath\\":\\"C:\\\\ProgramData\\\\MySQL\\\\MySQL
Server *\\\\Data\\\\<FILE_NAME>.err\\",\\"logType\\":\\"MYSQL\\",\\"monitor\\":true,
\\"encoding\\":\\"utf-8\\"}]}"
```

Configure monitoring for PostgreSQL RDS

1. Create an application for the resource group with the PostgreSQL RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
<RESOURCE_GROUP_NAME>
```

2. Publishing PostgreSQL logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check box labeled **PostgreSQL log**. Choose **Continue** to save this setting.

3. Your PostgreSQL logs are exported to CloudWatch.

4. Configure the PostgreSQL RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
  \"alarmMetrics\": [
    {
      \"alarmMetricName\": \"CPUUtilization\",
      \"monitor\": true
    }
  ]
}
```

```

    }
  ],
  \"logs\":[
    {
      \"logType\": \"POSTGRESQL\",
      \"monitor\": true
    }
  ]
}]"

```

Configure monitoring for PostgreSQL EC2

1. Create an application for the resource group with the PostgreSQL EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
<RESOURCE_GROUP_NAME>
```

2. Configure the PostgreSQL EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier POSTGRESQL --component-configuration
"{
  \"alarmMetrics\":[
    {
      \"alarmMetricName\": \"CPUUtilization\",
      \"monitor\": true
    }
  ],
  \"logs\":[
    {
      \"logGroupName\": \"<UNIQUE_LOG_GROUP_NAME>\",
      \"logPath\": \"var/lib/pgsql/data/log/\",
      \"logType\": \"POSTGRESQL\",
      \"monitor\": true,
      \"encoding\": \"utf-8\"
    }
  ]
}]"

```

Configure monitoring for Oracle RDS

1. Create an application for the resource group with the Oracle RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name <RESOURCE_GROUP_NAME>
```

2. Publishing Oracle logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check boxes labeled **Alert** log and **Listener** log. Choose **Continue** to save this setting.
3. Your Oracle logs are exported to CloudWatch.
4. Configure the Oracle RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --tier DEFAULT --component-configuration '{
  \"alarmMetrics\": [
    {
      \"alarmMetricName\": \"CPUUtilization\",
      \"monitor\": true
    }
  ],
  \"logs\": [
    {
      \"logType\": \"ORACLE_ALERT\",
      \"monitor\": true
    },
    {
      \"logType\": \"ORACLE_LISTENER\",
      \"monitor\": true
    }
  ]
}'
```

Configure monitoring for Oracle EC2

1. Create an application for the resource group with the Oracle EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
<RESOURCE_GROUP_NAME>
```

2. Configure the Oracle EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier ORACLE --component-configuration
"{
  \"alarmMetrics\": [
    {
      \"alarmMetricName\": \"CPUUtilization\",
      \"monitor\": true
    }
  ],
  \"logs\": [
    {
      \"logGroupName\": \"<UNIQUE_LOG_GROUP_NAME>\",
      \"logPath\": \"/opt/oracle/diag/rdbms/*/*/trace\",
      \"logType\": \"ORACLE_ALERT\",
      \"monitor\": true,
    },
    {
      \"logGroupName\": \"<UNIQUE_LOG_GROUP_NAME>\",
      \"logPath\": \"/opt/oracle/diag/tnslnsr/$HOSTNAME/listener/trace/\",
      \"logType\": \"ORACLE_ALERT\",
      \"monitor\": true,
    }
  ]
}"
```

Application Insights CloudWatch Events and notifications for detected problems

For each application that is added to CloudWatch Application Insights, a CloudWatch event is published for the following events on a best effort basis:

- **Problem creation.** Emitted when CloudWatch Application Insights detects a new problem.
 - Detail Type: **"Application Insights Problem Detected"**
 - Detail:
 - `problemId`: The detected problem ID.

- **region:** The AWS Region where the problem was created.
 - **resourceGroupName:** The Resource Group for the registered application for which the problem was detected.
 - **status:** The status of the problem. Possible status and definitions are as follows:
 - **In progress:** A new problem has been identified. The problem is still receiving observations.
 - **Recovering:** The problem is stabilizing. You can manually resolve the problem when it is in this state.
 - **Resolved:** The problem is resolved. There are no new observations about this problem.
 - **Recurring:** The problem was resolved within the past 24 hours. It has reopened as a result of additional observations.
 - **severity:** The severity of the problem.
 - **problemUrl:** The console URL for the problem.
- **Problem update.** Emitted when the problem is updated with a new observation or when an existing observation is updated and the problem is subsequently updated; updates include a resolution or closure of the problem.
 - **Detail Type: "Application Insights Problem Updated"**
 - **Detail:**
 - **problemId:** The created problem ID.
 - **region:** The AWS Region where the problem was created.
 - **resourceGroupName:** The Resource Group for the registered application for which the problem was detected.
 - **status:** The status of the problem.
 - **severity:** The severity of the problem.
 - **problemUrl:** The console URL for the problem.

How to receive notification for problem events generated by an application

From the CloudWatch console, select **Rules** under **Events** in the left navigation pane. From the **Rules** page, select **Create rule**. Choose **Amazon CloudWatch Application Insights** from the **Service Name** dropdown list and choose the **Event Type**. Then, choose **Add target** and select the target and parameters, for example, an **SNS topic** or **Lambda function**.

Actions through AWS Systems Manager. CloudWatch Application Insights provides built-in integration with Systems Manager OpsCenter. If you choose to use this integration for your application, an OpsItem is created on the OpsCenter console for every problem detected with the application. From the OpsCenter console, you can view summarized information about the problem detected by CloudWatch Application Insights and pick a Systems Manager Automation runbook to take remedial actions or further identify Windows processes that are causing resource issues in your application.

Application Insights cross-account observability

With CloudWatch Application Insights cross-account observability, you can monitor and troubleshoot your applications that span multiple AWS accounts within a single Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your AWS accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a sink in your monitoring account. You use the sink to create a link from your source account to your monitoring account. For more information, see [CloudWatch cross-account observability](#).

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, ensure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Applications in CloudWatch Application Insights
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs
- Traces in [AWS X-Ray](#)

Work with component configurations

A component configuration is a text file in JSON format that describes the configuration settings of the component. This section provides an example template fragment, descriptions of component configuration sections, and example component configurations.

Topics

- [Component configuration template fragment](#)
- [Component configuration sections](#)
- [Component configuration examples](#)

Component configuration template fragment

The following example shows a template fragment in JSON format.

```
{
  "alarmMetrics" : [
    list of alarm metrics
  ],
  "logs" : [
    list of logs
  ],
  "processes" : [
    list of processes
  ],
  "windowsEvents" : [
    list of windows events channels configurations
  ],
  "alarms" : [
    list of CloudWatch alarms
  ],
  "jmxPrometheusExporter": {
    JMX Prometheus Exporter configuration
  },
  "hanaPrometheusExporter": {
    SAP HANA Prometheus Exporter configuration
  },
  "haClusterPrometheusExporter": {
    HA Cluster Prometheus Exporter configuration
  },
  "netWeaverPrometheusExporter": {
    SAP NetWeaver Prometheus Exporter configuration
  },
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance" ...
      component nested instances configuration
    },
    {
```

```
    "subComponentType" : "AWS::EC2::Volume" ...
    component nested volumes configuration
  }
]
}
```

Component configuration sections

A component configuration includes several major sections. Sections in a component configuration can be listed in any order.

- **alarmMetrics (optional)**

A list of [metrics](#) to monitor for the component. All component types can have an alarmMetrics section.

- **logs (optional)**

A list of [logs](#) to monitor for the component. Only EC2 instances can have a logs section.

- **processes (optional)**

A list of [processes](#) to monitor for the component. Only EC2 instances can have a processes section.

- **subComponents (optional)**

Nested instance and volume subComponent configuration for the component. The following types of components can have nested instances and a subComponents section: ELB, ASG, custom-grouped EC2 instances , and EC2 instances.

- **alarms (optional)**

A list of [alarms](#) to monitor for the component. All component types can have an alarm section.

- **windowsEvents (optional)**

A list of [windows events](#) to monitor for the component. Only Windows on EC2 instances have a windowsEvents section.

- **JMXPrometheusExporter (optional)**

JMXPrometheus Exporter configuration.

- **hanaPrometheusExporter (optional)**

SAP HANA Prometheus Exporter configuration.

- **haClusterPrometheusExporter (optional)**

HA Cluster Prometheus Exporter configuration.

- **netWeaverPrometheusExporter (optional)**

SAP NetWeaver Prometheus Exporter configuration.

- **sapAsePrometheusExporter (optional)**

SAP ASE Prometheus Exporter configuration.

The following example shows the syntax for the **subComponents** section fragment in JSON format.

```
[
  {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [
      list of alarm metrics
    ],
    "logs" : [
      list of logs
    ],
    "processes": [
      list of processes
    ],
    "windowsEvents" : [
      list of windows events channels configurations
    ]
  },
  {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      list of alarm metrics
    ]
  }
]
```

Component configuration section properties

This section describes the properties of each component configuration section.

Sections

- [Metric](#)
- [Log](#)
- [Process](#)
- [JMX Prometheus Exporter](#)
- [HANA Prometheus Exporter](#)
- [HA Cluster Prometheus Exporter](#)
- [NetWeaver Prometheus Exporter](#)
- [SAP ASE Prometheus Exporter](#)
- [Windows Events](#)
- [Alarm](#)

Metric

Defines a metric to be monitored for the component.

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

Properties

- **alarmMetricName (required)**

The name of the metric to be monitored for the component. For metrics supported by Application Insights, see [Logs and metrics supported by Amazon CloudWatch Application Insights](#).

- **monitor (optional)**

Boolean to indicate whether to monitor the metric. The default value is `true`.

Log

Defines a log to be monitored for the component.

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see [CreateLogGroup](#).

- **logPath (required for EC2 instance components; not required for components that do not use CloudWatch Agent, such as AWS Lambda)**

The path of the logs to be monitored. The log path must be an absolute Windows system file path. For more information, see [CloudWatch Agent Configuration File: Logs Section](#).

- **logType (required)**

The log type decides the log patterns against which Application Insights analyzes the log. The log type is selected from the following:

- SQL_SERVER
- MYSQL
- MYSQL_SLOW_QUERY
- POSTGRESQL
- ORACLE_ALERT
- ORACLE_LISTENER
- IIS
- APPLICATION
- WINDOWS_EVENTS

- WINDOWS_EVENTS_ACTIVE_DIRECTORY
- WINDOWS_EVENTS_DNS
- WINDOWS_EVENTS_IIS
- WINDOWS_EVENTS_SHAREPOINT
- SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP
- SQL_SERVER_FAILOVER_CLUSTER_INSTANCE
- DEFAULT
- CUSTOM
- STEP_FUNCTION
- API_GATEWAY_ACCESS
- API_GATEWAY_EXECUTION
- SAP_HANA_LOGS
- SAP_HANA_TRACE
- SAP_HANA_HIGH_AVAILABILITY
- SAP_NETWEAVER_DEV_TRACE_LOGS
- PACEMAKER_HIGH_AVAILABILITY
- **encoding (optional)**

The type of encoding of the logs to be monitored. The specified encoding should be included in the list of [CloudWatch agent supported encodings](#). If not provided, CloudWatch Application Insights uses the default encoding of type utf-8, except for:

- SQL_SERVER: utf-16 encoding
- IIS: ascii encoding
- **monitor (optional)**

Boolean that indicates whether to monitor the logs. The default value is true.

Process

Defines a process to be monitored for the component.

JSON

```
{
```

```
"processName" : "monitoredProcessName",
"alarmMetrics" : [
  list of alarm metrics
]
}
```

Properties

- **processName (required)**

The name of the process to be monitored for the component. The process name must not contain a process stem, such as `sqlservr` or `sqlservr.exe`.

- **alarmMetrics (required)**

A list of [metrics](#) to monitor for this process. To view process metrics supported by CloudWatch Application Insights, see [Amazon Elastic Compute Cloud \(EC2\)](#).

JMX Prometheus Exporter

Defines the JMX Prometheus Exporter settings.

JSON

```
"JMXPrometheusExporter": {
  "jmxURL" : "JMX URL",
  "hostPort" : "The host and port",
  "prometheusPort" : "Target port to emit Prometheus metrics"
}
```

Properties

- **jmxURL (optional)**

A complete JMX URL to connect to.

- **hostPort (optional)**

The host and port to connect to through remote JMX. Only one of `jmxURL` and `hostPort` can be specified.

- **prometheusPort (optional)**

The target port to send Prometheus metrics to. If not specified, the default port 9404 is used.

HANA Prometheus Exporter

Defines the HANA Prometheus Exporter settings.

JSON

```
"hanaPrometheusExporter": {
  "hanaSid": "SAP HANA SID",
  "hanaPort": "HANA database port",
  "hanaSecretName": "HANA secret name",
  "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **hanaSid**

The three-character SAP system ID (SID) of the SAP HANA system.

- **hanaPort**

The HANA database port by which the exporter will query HANA metrics.

- **hanaSecretName**

The AWS Secrets Manager secret that stores HANA monitoring user credentials. The HANA Prometheus exporter uses these credentials to connect to the database and query HANA metrics.

- **prometheusPort (optional)**

The target port to which Prometheus sends metrics. If not specified, the default port 9668 is used.

HA Cluster Prometheus Exporter

Defines the HA Cluster Prometheus Exporter settings.

JSON

```
"haClusterPrometheusExporter": {
  "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **prometheusPort (optional)**

The target port to which Prometheus sends metrics. If not specified, the default port 9664 is used.

NetWeaver Prometheus Exporter

Defines the NetWeaver Prometheus Exporter settings.

JSON

```
"netWeaverPrometheusExporter": {
  "sapSid": "SAP NetWeaver SID",
  "instanceNumbers": [ "Array of instance Numbers of SAP NetWeaver system "],
  "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **sapSid**

The 3 character SAP system ID (SID) of the SAP NetWeaver system.

- **instanceNumbers**

Array of the instance Numbers of SAP NetWeaver system.

Example: "instanceNumbers": ["00", "01"]

- **prometheusPort (optional)**

The target port to which to send Prometheus metrics. If not specified, the default port 9680 is used.

SAP ASE Prometheus Exporter

Defines the SAP ASE Prometheus Exporter settings.

JSON

```
"sapASEPrometheusExporter": {
  "sapAseSid": "SAP ASE SID",
  "sapAsePort": "SAP ASE database port",
}
```

```
"sapAseSecretName": "SAP ASE secret name",
"prometheusPort": "Target port to emit Prometheus metrics",
"agreeToEnableASEMonitoring": true
}
```

Properties

- **sapAseSid**

The three-character SAP system ID (SID) of the SAP ASE system.

- **sapAsePort**

The SAP ASE database port by which the exporter will query ASE metrics.

- **sapAseSecretName**

The AWS Secrets Manager secret that stores ASE monitoring user credentials. The SAP ASE Prometheus exporter uses these credentials to connect to the database and query ASE metrics.

- **prometheusPort (optional)**

The target port to which Prometheus sends metrics. If not specified, the default port 9399 is used. If there is another ASE DB that is using the default port, then we use 9499.

Windows Events

Defines Windows Events to log.

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR", "WARNING", "CRITICAL", "INFORMATION", "VERBOSE"],
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see [CreateLogGroup](#).

- **eventName (required)**

The type of Windows Events to log. It is equivalent to the Windows Event log channel name. For example, System, Security, CustomEventName, etc. This field is required for each type of Windows event to log.

- **eventLevels (required)**

The levels of event to log. You must specify each level to log. Possible values include INFORMATION, WARNING, ERROR, CRITICAL, and VERBOSE. This field is required for each type of Windows Event to log.

- **monitor (optional)**

Boolean that indicates whether to monitor the logs. The default value is true.

Alarm

Defines a CloudWatch alarm to be monitored for the component.

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

Properties

- **alarmName (required)**

The name of the CloudWatch alarm to be monitored for the component.

- **severity (optional)**

Indicates the degree of outage when the alarm goes off.

Component configuration examples

The following examples show component configurations in JSON format for relevant services.

Example component configurations

- [Amazon DynamoDB table](#)
- [Amazon EC2 Auto Scaling \(ASG\)](#)
- [Amazon EKS cluster](#)
- [Amazon Elastic Compute Cloud \(EC2\) instance](#)
- [Amazon Elastic Container Service \(Amazon ECS\)](#)
- [Amazon ECS services](#)
- [Amazon ECS tasks](#)
- [Amazon Elastic File System \(Amazon EFS\)](#)
- [Amazon FSx](#)
- [Amazon Relational Database Service \(RDS\) Aurora MySQL](#)
- [Amazon Relational Database Service \(RDS\) instance](#)
- [Amazon Route 53 health check](#)
- [Amazon Route 53 hosted zone](#)
- [Amazon Route 53 Resolver endpoint](#)
- [Amazon Route 53 Resolver query logging configuration](#)
- [Amazon S3 bucket](#)
- [Amazon Simple Queue Service \(SQS\)](#)
- [Amazon SNS topic](#)
- [Amazon Virtual Private Cloud \(Amazon VPC\)](#)
- [Amazon VPC Network Address Translation \(NAT\) gateways](#)
- [API Gateway REST API stages](#)
- [Application Elastic Load Balancing](#)
- [AWS Lambda Function](#)
- [AWS Network Firewall rule group](#)
- [AWS Network Firewall rule group association](#)
- [AWS Step Functions](#)
- [Customer-grouped Amazon EC2 instances](#)
- [Elastic Load Balancing](#)
- [Java](#)
- [Kubernetes on Amazon EC2](#)

- [RDS MariaDB and RDS MySQL](#)
- [RDS Oracle](#)
- [RDS PostgreSQL](#)
- [SAP ASE on Amazon EC2](#)
- [SAP ASE High Availability on Amazon EC2](#)
- [SAP HANA on Amazon EC2](#)
- [SAP HANA High Availability on Amazon EC2](#)
- [SAP NetWeaver on Amazon EC2](#)
- [SAP NetWeaver High Availability on Amazon EC2](#)
- [SQL Always On Availability Group](#)
- [SQL failover cluster instance](#)

Amazon DynamoDB table

The following example shows a component configuration in JSON format for Amazon DynamoDB table.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "SystemErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "UserErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedReadCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedWriteCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ReadThrottleEvents",
      "monitor": false
    }
  ]
}
```

```
    },
    {
      "alarmMetricName": "WriteThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "ConditionalCheckFailedRequests",
      "monitor": false
    },
    {
      "alarmMetricName": "TransactionConflict",
      "monitor": false
    }
  ],
  "logs": []
}
```

Amazon EC2 Auto Scaling (ASG)

The following example shows a component configuration in JSON format for Amazon EC2 Auto Scaling (ASG).

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "CPUCreditBalance"
    }, {
      "alarmMetricName" : "EBSIOBalance%"
    }
  ],
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance",
      "alarmMetrics" : [
        {
          "alarmMetricName" : "CPUUtilization"
        }, {
          "alarmMetricName" : "StatusCheckFailed"
        }
      ],
      "logs" : [
        {
          "logGroupName" : "my_log_group",

```

```
        "logPath" : "C:\\\\LogFolder\\\\*",
        "logType" : "APPLICATION"
    }
],
"processes" : [
    {
        "processName" : "my_process",
        "alarmMetrics" : [
            {
                "alarmMetricName" : "procstat cpu_usage",
                "monitor" : true
            }, {
                "alarmMetricName" : "procstat memory_rss",
                "monitor" : true
            }
        ]
    }
],
"windowsEvents" : [
    {
        "logGroupName" : "my_log_group_2",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
    }
], {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
        {
            "alarmMetricName" : "VolumeQueueLength"
        }, {
            "alarmMetricName" : "BurstBalance"
        }
    ]
}
],
"alarms" : [
    {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
    }
]
}
```

Amazon EKS cluster

The following example shows a component configuration in JSON format for Amazon EKS cluster.

```
{
  "alarmMetrics":[
    {
      "alarmMetricName": "cluster_failed_node_count",
      "monitor":true
    },
    {
      "alarmMetricName": "node_cpu_reserved_capacity",
      "monitor":true
    },
    {
      "alarmMetricName": "node_cpu_utilization",
      "monitor":true
    },
    {
      "alarmMetricName": "node_filesystem_utilization",
      "monitor":true
    },
    {
      "alarmMetricName": "node_memory_reserved_capacity",
      "monitor":true
    },
    {
      "alarmMetricName": "node_memory_utilization",
      "monitor":true
    },
    {
      "alarmMetricName": "node_network_total_bytes",
      "monitor":true
    },
    {
      "alarmMetricName": "pod_cpu_reserved_capacity",
      "monitor":true
    },
    {
      "alarmMetricName": "pod_cpu_utilization",
      "monitor":true
    },
    {
      "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
```

```
    "monitor":true
  },
  {
    "alarmMetricName": "pod_memory_reserved_capacity",
    "monitor":true
  },
  {
    "alarmMetricName": "pod_memory_utilization",
    "monitor":true
  },
  {
    "alarmMetricName": "pod_memory_utilization_over_pod_limit",
    "monitor":true
  },
  {
    "alarmMetricName": "pod_network_rx_bytes",
    "monitor":true
  },
  {
    "alarmMetricName": "pod_network_tx_bytes",
    "monitor":true
  }
],
"logs":[
  {
    "logGroupName": "/aws/containerinsights/kubernetes/application",
    "logType":"APPLICATION",
    "monitor":true,
    "encoding":"utf-8"
  }
],
"subComponents":[
  {
    "subComponentType":"AWS::EC2::Instance",
    "alarmMetrics":[
      {
        "alarmMetricName":"CPUUtilization",
        "monitor":true
      },
      {
        "alarmMetricName":"StatusCheckFailed",
        "monitor":true
      }
    ]
  }
]
```

```
        "alarmMetricName": "disk_used_percent",
        "monitor": true
    },
    {
        "alarmMetricName": "mem_used_percent",
        "monitor": true
    }
],
"logs": [
    {
        "logGroupName": "APPLICATION-KubernetesClusterOnEC2-IAD",
        "logPath": "",
        "logType": "APPLICATION",
        "monitor": true,
        "encoding": "utf-8"
    }
],
"processes" : [
    {
        "processName" : "my_process",
        "alarmMetrics" : [
            {
                "alarmMetricName" : "procstat cpu_usage",
                "monitor" : true
            }, {
                "alarmMetricName" : "procstat memory_rss",
                "monitor" : true
            }
        ]
    }
],
"windowsEvents": [
    {
        "logGroupName": "my_log_group_2",
        "eventName": "Application",
        "eventLevels": [
            "ERROR",
            "WARNING",
            "CRITICAL"
        ],
        "monitor": true
    }
],
},
```



```
{
  "subComponentType": "AWS::AutoScaling::AutoScalingGroup",
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUCreditBalance",
      "monitor": true
    },
    {
      "alarmMetricName": "EBSIOBalance%",
      "monitor": true
    }
  ]
},
{
  "subComponentType": "AWS::EC2::Volume",
  "alarmMetrics": [
    {
      "alarmMetricName": "VolumeReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "VolumeWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "VolumeReadOps",
      "monitor": true
    },
    {
      "alarmMetricName": "VolumeWriteOps",
      "monitor": true
    },
    {
      "alarmMetricName": "VolumeQueueLength",
      "monitor": true
    },
    {
      "alarmMetricName": "BurstBalance",
      "monitor": true
    }
  ]
}
]
```

```
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance`, `AWS::EC2::Volume`, and `AWS::AutoScaling::AutoScalingGroup` applies only to Amazon EKS cluster running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon Elastic Compute Cloud (EC2) instance

The following example shows a component configuration in JSON format for an Amazon EC2 instance.

Important

When an Amazon EC2 instance enters a stopped state, it is removed from monitoring. When it returns to a running state, it is added to the list of **Unmonitored components** on the **Application details** page of the CloudWatch Application Insights console. If automatic monitoring of new resources is enabled for the application, the instance is added to the list of **Monitored components**. However, the logs and metrics are set to the default for the workload. The previous log and metrics configuration is not saved.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed"
    }
  ],
  "logs" : [
    {
      "logGroupName" : "my_log_group",
      "logPath" : "C:\\\\LogFolder\\\\"
    }
  ]
}
```

```
    "logType" : "APPLICATION",
    "monitor" : true
  },
  {
    "logGroupName" : "my_log_group_2",
    "logPath" : "C:\\\\LogFolder2\\\\*",
    "logType" : "IIS",
    "encoding" : "utf-8"
  }
],
"processes" : [
  {
    "processName" : "my_process",
    "alarmMetrics" : [
      {
        "alarmMetricName" : "procstat cpu_usage",
        "monitor" : true
      }, {
        "alarmMetricName" : "procstat memory_rss",
        "monitor" : true
      }
    ]
  }
],
"windowsEvents" : [
  {
    "logGroupName" : "my_log_group_3",
    "eventName" : "Application",
    "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
    "monitor" : true
  }, {
    "logGroupName" : "my_log_group_4",
    "eventName" : "System",
    "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
    "monitor" : true
  }
],
"alarms" : [
  {
    "alarmName" : "my_instance_alarm_1",
    "severity" : "HIGH"
  },
  {
    "alarmName" : "my_instance_alarm_2",
    "severity" : "LOW"
  }
]
```

```
    }
  ],
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Volume",
      "alarmMetrics" : [
        {
          "alarmMetricName" : "VolumeQueueLength",
          "monitor" : "true"
        },
        {
          "alarmMetricName" : "VolumeThroughputPercentage",
          "monitor" : "true"
        },
        {
          "alarmMetricName" : "BurstBalance",
          "monitor" : "true"
        }
      ]
    }
  ]
}
```

Amazon Elastic Container Service (Amazon ECS)

The following example shows a component configuration in JSON format for Amazon Elastic Container Service (Amazon ECS).

```
{
  "alarmMetrics":[
    {
      "alarmMetricName":"CpuUtilized",
      "monitor":true
    },
    {
      "alarmMetricName":"MemoryUtilized",
      "monitor":true
    },
    {
      "alarmMetricName":"NetworkRxBytes",
      "monitor":true
    },
    {
      "alarmMetricName":"NetworkTxBytes",
      "monitor":true
    }
  ]
}
```

```
    },
    {
      "alarmMetricName": "RunningTaskCount",
      "monitor": true
    },
    {
      "alarmMetricName": "PendingTaskCount",
      "monitor": true
    },
    {
      "alarmMetricName": "StorageReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "StorageWriteBytes",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName": "/ecs/my-task-definition",
      "logType": "APPLICATION",
      "monitor": true
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::ElasticLoadBalancing::LoadBalancer",
      "alarmMetrics": [
        {
          "alarmMetricName": "HTTPCode_Backend_4XX",
          "monitor": true
        },
        {
          "alarmMetricName": "HTTPCode_Backend_5XX",
          "monitor": true
        },
        {
          "alarmMetricName": "Latency",
          "monitor": true
        },
        {
          "alarmMetricName": "SurgeQueueLength",
          "monitor": true
        }
      ]
    }
  ]
}
```

```
    },
    {
      "alarmMetricName": "UnHealthyHostCount",
      "monitor": true
    }
  ]
},
{
  "subComponentType": "AWS::ElasticLoadBalancingV2::LoadBalancer",
  "alarmMetrics": [
    {
      "alarmMetricName": "HTTPCode_Target_4XX_Count",
      "monitor": true
    },
    {
      "alarmMetricName": "HTTPCode_Target_5XX_Count",
      "monitor": true
    },
    {
      "alarmMetricName": "TargetResponseTime",
      "monitor": true
    },
    {
      "alarmMetricName": "UnHealthyHostCount",
      "monitor": true
    }
  ]
},
{
  "subComponentType": "AWS::EC2::Instance",
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "StatusCheckFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "disk_used_percent",
      "monitor": true
    },
    {
```

```
        "alarmMetricName": "mem_used_percent",
        "monitor": true
    }
],
"logs": [
    {
        "logGroupName": "my_log_group",
        "logPath": "/mylog/path",
        "logType": "APPLICATION",
        "monitor": true
    }
],
"processes" : [
    {
        "processName" : "my_process",
        "alarmMetrics" : [
            {
                "alarmMetricName" : "procstat cpu_usage",
                "monitor" : true
            }, {
                "alarmMetricName" : "procstat memory_rss",
                "monitor" : true
            }
        ]
    }
],
"windowsEvents": [
    {
        "logGroupName": "my_log_group_2",
        "eventName": "Application",
        "eventLevels": [
            "ERROR",
            "WARNING",
            "CRITICAL"
        ],
        "monitor": true
    }
]
},
{
    "subComponentType": "AWS::EC2::Volume",
    "alarmMetrics": [
        {
            "alarmMetricName": "VolumeQueueLength",
```

```

        "monitor": "true"
    },
    {
        "alarmMetricName": "VolumeThroughputPercentage",
        "monitor": "true"
    },
    {
        "alarmMetricName": "BurstBalance",
        "monitor": "true"
    }
]
}
]
}

```

Note

- The subComponents section of AWS::EC2::Instance and AWS::EC2::Volume applies only to Amazon ECS clusters with ECS service or ECS task running on the EC2 launch type.
- The windowsEvents section of AWS::EC2::Instance in subComponents applies only to Windows running on Amazon EC2 instances.

Amazon ECS services

The following example shows a component configuration in JSON format for an Amazon ECS service.

```

{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "MemoryUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CpuUtilized",

```



```
    "monitor":true
  },
  {
    "alarmMetricName":"MemoryUtilized",
    "monitor":true
  },
  {
    "alarmMetricName":"NetworkRxBytes",
    "monitor":true
  },
  {
    "alarmMetricName":"NetworkTxBytes",
    "monitor":true
  },
  {
    "alarmMetricName":"RunningTaskCount",
    "monitor":true
  },
  {
    "alarmMetricName":"PendingTaskCount",
    "monitor":true
  },
  {
    "alarmMetricName":"StorageReadBytes",
    "monitor":true
  },
  {
    "alarmMetricName":"StorageWriteBytes",
    "monitor":true
  }
],
"logs":[
  {
    "logGroupName":"/ecs/my-task-definition",
    "logType":"APPLICATION",
    "monitor":true
  }
],
"subComponents":[
  {
    "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
    "alarmMetrics":[
      {
        "alarmMetricName":"HTTPCode_Backend_4XX",
```

```

        "monitor":true
    },
    {
        "alarmMetricName":"HTTPCode_Backend_5XX",
        "monitor":true
    },
    {
        "alarmMetricName":"Latency",
        "monitor":true
    },
    {
        "alarmMetricName":"SurgeQueueLength",
        "monitor":true
    },
    {
        "alarmMetricName":"UnHealthyHostCount",
        "monitor":true
    }
]
},
{
    "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
    "alarmMetrics":[
        {
            "alarmMetricName":"HTTPCode_Target_4XX_Count",
            "monitor":true
        },
        {
            "alarmMetricName":"HTTPCode_Target_5XX_Count",
            "monitor":true
        },
        {
            "alarmMetricName":"TargetResponseTime",
            "monitor":true
        },
        {
            "alarmMetricName":"UnHealthyHostCount",
            "monitor":true
        }
    ]
},
{
    "subComponentType":"AWS::EC2::Instance",
    "alarmMetrics":[

```

```

    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "StatusCheckFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "disk_used_percent",
      "monitor": true
    },
    {
      "alarmMetricName": "mem_used_percent",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName": "my_log_group",
      "logPath": "/mylog/path",
      "logType": "APPLICATION",
      "monitor": true
    }
  ],
  "processes" : [
    {
      "processName" : "my_process",
      "alarmMetrics" : [
        {
          "alarmMetricName" : "procstat cpu_usage",
          "monitor" : true
        }, {
          "alarmMetricName" : "procstat memory_rss",
          "monitor" : true
        }
      ]
    }
  ]
},
"windowsEvents": [
  {
    "logGroupName": "my_log_group_2",
    "eventName": "Application",
    "eventLevels": [

```

```

        "ERROR",
        "WARNING",
        "CRITICAL"
    ],
    "monitor":true
  }
]
},
{
  "subComponentType":"AWS::EC2::Volume",
  "alarmMetrics":[
    {
      "alarmMetricName":"VolumeQueueLength",
      "monitor":"true"
    },
    {
      "alarmMetricName":"VolumeThroughputPercentage",
      "monitor":"true"
    },
    {
      "alarmMetricName":"BurstBalance",
      "monitor":"true"
    }
  ]
}
]
}

```

Note

- The subComponents section of AWS::EC2::Instance and AWS::EC2::Volume applies only to Amazon ECS running on the EC2 launch type.
- The windowsEvents section of AWS::EC2::Instance in subComponents applies only to Windows running on Amazon EC2 instances.

Amazon ECS tasks

The following example shows a component configuration in JSON format for an Amazon ECS task.

```
{
```

```
"logs":[
  {
    "logGroupName":"/ecs/my-task-definition",
    "logType":"APPLICATION",
    "monitor":true
  }
],
"processes" : [
  {
    "processName" : "my_process",
    "alarmMetrics" : [
      {
        "alarmMetricName" : "procstat cpu_usage",
        "monitor" : true
      }, {
        "alarmMetricName" : "procstat memory_rss",
        "monitor" : true
      }
    ]
  }
]
}
```

Amazon Elastic File System (Amazon EFS)

The following example shows a component configuration in JSON format for Amazon EFS.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "BurstCreditBalance",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentIOLimit",
      "monitor": true
    },
    {
      "alarmMetricName": "PermittedThroughput",
      "monitor": true
    },
    {
      "alarmMetricName": "MeteredIOBytes",
      "monitor": true
    }
  ]
}
```

```
},
{
  "alarmMetricName": "TotalIOBytes",
  "monitor": true
},
{
  "alarmMetricName": "DataWriteIOBytes",
  "monitor": true
},
{
  "alarmMetricName": "DataReadIOBytes",
  "monitor": true
},
{
  "alarmMetricName": "MetadataIOBytes",
  "monitor": true
},
{
  "alarmMetricName": "ClientConnections",
  "monitor": true
},
{
  "alarmMetricName": "TimeSinceLastSync",
  "monitor": true
},
{
  "alarmMetricName": "Throughput",
  "monitor": true
},
{
  "alarmMetricName": "PercentageOfPermittedThroughputUtilization",
  "monitor": true
},
{
  "alarmMetricName": "ThroughputIOPS",
  "monitor": true
},
{
  "alarmMetricName": "PercentThroughputDataReadIOBytes",
  "monitor": true
},
{
  "alarmMetricName": "PercentThroughputDataWriteIOBytes",
  "monitor": true
}
```

```

    },
    {
      "alarmMetricName": "PercentageOfIOPSDataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfIOPSDataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "AverageDataReadIOBytesSize",
      "monitor": true
    },
    {
      "alarmMetricName": "AverageDataWriteIOBytesSize",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName": "/aws/efs/utils",
      "logType": "EFS_MOUNT_STATUS",
      "monitor": true,
    }
  ]
}

```

Amazon FSx

The following example shows a component configuration in JSON format for Amazon FSx.

```

{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
    }
  ]
}

```

```
    },
    {
      "alarmMetricName": "DataWriteOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

The following example shows a component configuration in JSON format for Amazon RDS Aurora MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```


Amazon Relational Database Service (RDS) instance

The following example shows a component configuration in JSON format for an Amazon RDS instance.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    }, {
      "alarmMetricName" : "WriteThroughput",
      "monitor" : false
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Route 53 health check

The following example shows a component configuration in JSON format for Amazon Route 53 health check.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ChildHealthCheckHealthyCount",
      "monitor": true
    },
    {
      "alarmMetricName": "ConnectionTime",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckPercentageHealthy",
      "monitor": true
    },
  ],
}
```

```
{
  "alarmMetricName": "HealthCheckStatus",
  "monitor": true
},
{
  "alarmMetricName": "SSLHandshakeTime",
  "monitor": true
},
{
  "alarmMetricName": "TimeToFirstByte",
  "monitor": true
}
]
}
```

Amazon Route 53 hosted zone

The following example shows a component configuration in JSON format for Amazon Route 53 hosted zone.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DNSQueries",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECInternalFailure",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeysNeedingAction",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyMaxNeedingActionAge",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyAge",
      "monitor": true
    }
  ],
}
```

```
"logs": [  
  {  
    "logGroupName": "/hosted-zone/logs",  
    "logType": "ROUTE53_DNS_PUBLIC_QUERY_LOGS",  
    "monitor": true  
  }  
]  
}
```

Amazon Route 53 Resolver endpoint

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver endpoint.

```
{  
  "alarmMetrics": [  
    {  
      "alarmMetricName": "EndpointHealthyENICount",  
      "monitor": true  
    },  
    {  
      "alarmMetricName": "EndpointUnHealthyENICount",  
      "monitor": true  
    },  
    {  
      "alarmMetricName": "InboundQueryVolume",  
      "monitor": true  
    },  
    {  
      "alarmMetricName": "OutboundQueryVolume",  
      "monitor": true  
    },  
    {  
      "alarmMetricName": "OutboundQueryAggregateVolume",  
      "monitor": true  
    }  
  ]  
}
```

Amazon Route 53 Resolver query logging configuration

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver query logging configuration.

```
{
  "logs": [
    {
      "logGroupName": "/resolver-query-log-config/logs",
      "logType": "ROUTE53_RESOLVER_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon S3 bucket

The following example shows a component configurations in JSON format for Amazon S3 bucket.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ReplicationLatency",
      "monitor" : true
    }, {
      "alarmMetricName" : "5xxErrors",
      "monitor" : true
    }, {
      "alarmMetricName" : "BytesDownloaded"
      "monitor" : true
    }
  ]
}
```

Amazon Simple Queue Service (SQS)

The following example shows a component configuration in JSON format for Amazon Simple Queue Service.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ApproximateAgeOfOldestMessage"
    }, {
      "alarmMetricName" : "NumberOfEmptyReceives"
    }
  ],
}
```

```
"alarms" : [
  {
    "alarmName" : "my_sqs_alarm",
    "severity" : "MEDIUM"
  }
]
```

Amazon SNS topic

The following example shows a component configuration in JSON format for Amazon SNS topic.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

Amazon Virtual Private Cloud (Amazon VPC)

The following example shows a component configuration in JSON format for Amazon VPC.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NetworkAddressUsage",
      "monitor": true
    },
  ],
}
```

```
{
  "alarmMetricName": "NetworkAddressUsagePeered",
  "monitor": true
},
{
  "alarmMetricName": "VPCFirewallQueryVolume",
  "monitor": true
}
]
```

Amazon VPC Network Address Translation (NAT) gateways

The following example shows a component configuration in JSON format for NAT gateways.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ErrorPortAllocation",
      "monitor": true
    },
    {
      "alarmMetricName": "IdleTimeoutCount",
      "monitor": true
    }
  ]
}
```

API Gateway REST API stages

The following example shows a component configuration in JSON format for API Gateway REST API stages.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "4XXError",
      "monitor" : true
    },
    {
      "alarmMetricName" : "5XXError",
      "monitor" : true
    }
  ]
}
```

```
    ],
    "logs" : [
      {
        "logType" : "API_GATEWAY_EXECUTION",
        "monitor" : true
      },
      {
        "logType" : "API_GATEWAY_ACCESS",
        "monitor" : true
      }
    ]
  }
}
```

Application Elastic Load Balancing

The following example shows a component configuration in JSON format for Application Elastic Load Balancing.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ActiveConnectionCount",
    }, {
      "alarmMetricName": "TargetResponseTime"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ]
    },
  ],
  "logs": [
    {
      "logGroupName": "my_log_group",
      "logPath": "C:\\\\LogFolder\\*",
      "logType": "APPLICATION",
    }
  ],
}
```

```
    "windowsEvents": [
      {
        "logGroupName": "my_log_group_2",
        "eventName": "Application",
        "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
      }
    ]
  }, {
    "subComponentType": "AWS::EC2::Volume",
    "alarmMetrics": [
      {
        "alarmMetricName": "VolumeQueueLength",
      }, {
        "alarmMetricName": "BurstBalance"
      }
    ]
  }
],
"alarms": [
  {
    "alarmName": "my_alb_alarm",
    "severity": "LOW"
  }
]
}
```

AWS Lambda Function

The following example shows a component configuration in JSON format for AWS Lambda Function.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
```



```
    "alarmMetricName": "IteratorAge",
    "monitor": true
  },
  {
    "alarmMetricName": "Duration",
    "monitor": true
  }
],
"logs": [
  {
    "logType": "DEFAULT",
    "monitor": true
  }
]
}
```

AWS Network Firewall rule group

The following example shows a component configuration in JSON format for AWS Network Firewall rule group.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall rule group association

The following example shows a component configuration in JSON format for AWS Network Firewall rule group association.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

```
}
```

AWS Step Functions

The following example shows a component configurations in JSON format for AWS Step Functions.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ExecutionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "LambdaFunctionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "ProvisionedRefillRate",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName": "/aws/states/HelloWorld-Logs",
      "logType": "STEP_FUNCTION",
      "monitor": true,
    }
  ]
}
```

Customer-grouped Amazon EC2 instances

The following example shows a component configuration in JSON format for customer-grouped Amazon EC2 instances.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        },
      ],
    }
  ]
}
```

```
    {
      "alarmMetricName": "StatusCheckFailed"
    }
  ],
  "logs": [
    {
      "logGroupName": "my_log_group",
      "logPath": "C:\\\\LogFolder\\*",
      "logType": "APPLICATION",
    }
  ],
  "processes": [
    {
      "processName": "my_process",
      "alarmMetrics": [
        {
          "alarmMetricName": "procstat cpu_usage",
          "monitor": true
        }, {
          "alarmMetricName": "procstat memory_rss",
          "monitor": true
        }
      ]
    }
  ],
  "windowsEvents": [
    {
      "logGroupName": "my_log_group_2",
      "eventName": "Application",
      "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
    }
  ]
}, {
  "subComponentType": "AWS::EC2::Volume",
  "alarmMetrics": [
    {
      "alarmMetricName": "VolumeQueueLength",
    }, {
      "alarmMetricName": "BurstBalance"
    }
  ]
}
],
"alarms": [
```

```
{
  "alarmName": "my_alarm",
  "severity": "MEDIUM"
}
]
```

Elastic Load Balancing

The following example shows a component configuration in JSON format for Elastic Load Balancing.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "EstimatedALBActiveConnectionCount"
    }, {
      "alarmMetricName": "HTTPCode_Backend_5XX"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization"
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ]
    },
    {
      "logGroupName": "my_log_group",
      "logPath": "C:\\\\LogFolder\\*",
      "logType": "APPLICATION"
    }
  ],
  "processes": [
    {
      "processName": "my_process",
      "alarmMetrics": [
        {
          "alarmMetricName": "procstat cpu_usage",

```

```
        "monitor": true
      }, {
        "alarmMetricName": "procstat memory_rss",
        "monitor": true
      }
    ]
  }
],
"windowsEvents": [
  {
    "logGroupName": "my_log_group_2",
    "eventName": "Application",
    "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ],
    "monitor": true
  }
]
}, {
  "subComponentType": "AWS::EC2::Volume",
  "alarmMetrics": [
    {
      "alarmMetricName": "VolumeQueueLength"
    }, {
      "alarmMetricName": "BurstBalance"
    }
  ]
}
],
"alarms": [
  {
    "alarmName": "my_elb_alarm",
    "severity": "HIGH"
  }
]
}
```

Java

The following example shows a component configuration in JSON format for Java.

```
{
  "alarmMetrics": [ {
    "alarmMetricName": "java_lang_threading_threadcount",
    "monitor": true
  }
```

```
},
{
  "alarmMetricName": "java_lang_memory_heapmemoryusage_used",
  "monitor": true
},
{
  "alarmMetricName": "java_lang_memory_heapmemoryusage_committed",
  "monitor": true
}],
"logs": [ ],
"JMXPrometheusExporter": {
  "hostPort": "8686",
  "prometheusPort": "9404"
}
}
```

Note

Application Insights does not support configuring authentication for Prometheus JMX exporter. For information about how to set up authentication, see the [Prometheus JMX exporter example configuration](#).

Kubernetes on Amazon EC2

The following example shows a component configuration in JSON format for Kubernetes on Amazon EC2.

```
{
  "alarmMetrics":[
    {
      "alarmMetricName":"cluster_failed_node_count",
      "monitor":true
    },
    {
      "alarmMetricName":"node_cpu_reserved_capacity",
      "monitor":true
    },
    {
      "alarmMetricName":"node_cpu_utilization",
      "monitor":true
    },
  ],
}
```

```
{
  "alarmMetricName": "node_filesystem_utilization",
  "monitor": true
},
{
  "alarmMetricName": "node_memory_reserved_capacity",
  "monitor": true
},
{
  "alarmMetricName": "node_memory_utilization",
  "monitor": true
},
{
  "alarmMetricName": "node_network_total_bytes",
  "monitor": true
},
{
  "alarmMetricName": "pod_cpu_reserved_capacity",
  "monitor": true
},
{
  "alarmMetricName": "pod_cpu_utilization",
  "monitor": true
},
{
  "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
  "monitor": true
},
{
  "alarmMetricName": "pod_memory_reserved_capacity",
  "monitor": true
},
{
  "alarmMetricName": "pod_memory_utilization",
  "monitor": true
},
{
  "alarmMetricName": "pod_memory_utilization_over_pod_limit",
  "monitor": true
},
{
  "alarmMetricName": "pod_network_rx_bytes",
  "monitor": true
},
},
```

```
{
  "alarmMetricName": "pod_network_tx_bytes",
  "monitor": true
},
"logs": [
  {
    "logGroupName": "/aws/containerinsights/kubernetes/application",
    "logType": "APPLICATION",
    "monitor": true,
    "encoding": "utf-8"
  }
],
"subComponents": [
  {
    "subComponentType": "AWS::EC2::Instance",
    "alarmMetrics": [
      {
        "alarmMetricName": "CPUUtilization",
        "monitor": true
      },
      {
        "alarmMetricName": "StatusCheckFailed",
        "monitor": true
      },
      {
        "alarmMetricName": "disk_used_percent",
        "monitor": true
      },
      {
        "alarmMetricName": "mem_used_percent",
        "monitor": true
      }
    ],
    "logs": [
      {
        "logGroupName": "APPLICATION-KubernetesClusterOnEC2-IAD",
        "logPath": "",
        "logType": "APPLICATION",
        "monitor": true,
        "encoding": "utf-8"
      }
    ],
    "processes" : [
```



```
    {
      "processName" : "my_process",
      "alarmMetrics" : [
        {
          "alarmMetricName" : "procstat cpu_usage",
          "monitor" : true
        }, {
          "alarmMetricName" : "procstat memory_rss",
          "monitor" : true
        }
      ]
    }
  ],
},
{
  "subComponentType":"AWS::EC2::Volume",
  "alarmMetrics":[
    {
      "alarmMetricName":"VolumeReadBytes",
      "monitor":true
    },
    {
      "alarmMetricName":"VolumeWriteBytes",
      "monitor":true
    },
    {
      "alarmMetricName":"VolumeReadOps",
      "monitor":true
    },
    {
      "alarmMetricName":"VolumeWriteOps",
      "monitor":true
    },
    {
      "alarmMetricName":"VolumeQueueLength",
      "monitor":true
    },
    {
      "alarmMetricName":"BurstBalance",
      "monitor":true
    }
  ]
}
]
```

```
}
```

RDS MariaDB and RDS MySQL

The following example shows a component configuration in JSON format for RDS MariaDB and RDS MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS Oracle

The following example shows a component configuration in JSON format for RDS Oracle.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "ORACLE_ALERT",
      "monitor": true,
    },
    {
```

```
    "logType": "ORACLE_LISTENER",
    "monitor": false
  }
]
```

RDS PostgreSQL

The following example shows a component configurations in JSON format for RDS PostgreSQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

SAP ASE on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        }
      ]
    }
  ]
}
```

```
{
  "alarmMetricName": "asedb_last_db_backup_age_in_days",
  "monitor": true
},
{
  "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
  "monitor": true
},
{
  "alarmMetricName": "asedb_suspected_database",
  "monitor": true
},
{
  "alarmMetricName": "asedb_db_space_usage_percent",
  "monitor": true
},
{
  "alarmMetricName": "asedb_db_log_space_usage_percent",
  "monitor": true
},
{
  "alarmMetricName": "asedb_locked_login",
  "monitor": true
},
{
  "alarmMetricName": "asedb_data_cache_hit_ratio",
  "monitor": true
}
],
"logs": [
  {
    "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
    "logType": "SAP_ASE_SERVER_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  },
  {
    "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
    "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  }
]
```

```
],
"sapAsePrometheusExporter": {
  "sapAseSid": "ASE",
  "sapAsePort": "4901",
  "sapAseSecretName": "ASE_DB_CREDS",
  "prometheusPort": "9399",
  "agreeToEnableASEMonitoring": true
}
```

SAP ASE High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        }
      ]
    }
  ]
}
```

```
{
  "alarmMetricName": "asedb_ha_replication_state",
  "monitor": true
},
{
  "alarmMetricName": "asedb_ha_replication_mode",
  "monitor": true
},
{
  "alarmMetricName": "asedb_ha_replication_latency_in_minutes",
  "monitor": true
}
],
"logs": [
  {
    "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
    "logType": "SAP_ASE_SERVER_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  },
  {
    "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
    "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  },
  {
    "logGroupName": "SAP_ASE_REP_SERVER_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/DM/repservername/repservername.log",
    "logType": "SAP_ASE_REP_SERVER_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  },
  {
    "logGroupName": "SAP_ASE_RMA_AGENT_LOGS-my-resource-group",
    "logPath": "/sybase/SY2/DM/RMA-*/instances/AgentContainer/logs/",
    "logType": "SAP_ASE_RMA_AGENT_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  },
  {
    "logGroupName": "SAP_ASE_FAULT_MANAGER_LOGS-my-resource-group",
```

```
        "logPath": "/opt/sap/FaultManager/dev_sybdbfm",
        "logType": "SAP_ASE_FAULT_MANAGER_LOGS",
        "monitor": true,
        "encoding": "utf-8"
    }
],
"sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

SAP HANA on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",
          "monitor": true
        }
      ]
    }
  ]
}
```

```
    },
    {
      "alarmMetricName": "hanadb_table_allocation_limit_used_percent",
      "monitor": true
    },
    {
      "alarmMetricName": "hanadb_cpu_usage_percent",
      "monitor": true
    },
    {
      "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",
      "monitor": true
    },
    {
      "alarmMetricName": "hanadb_last_data_backup_age_days",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName": "SAP_HANA_TRACE-my-resource-group",
      "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
      "logType": "SAP_HANA_TRACE",
      "monitor": true,
      "encoding": "utf-8"
    },
    {
      "logGroupName": "SAP_HANA_LOGS-my-resource-group",
      "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",
      "logType": "SAP_HANA_LOGS",
      "monitor": true,
      "encoding": "utf-8"
    }
  ]
}
],
"hanaPrometheusExporter": {
  "hanaSid": "HDB",
  "hanaPort": "30013",
  "hanaSecretName": "HANA_DB_CREDS",
  "prometheusPort": "9668"
}
}
```


SAP HANA High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resource-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
          "logPath": "/var/log/pacemaker/pacemaker.log",
          "logType": "SAP_HANA_HIGH_AVAILABILITY",
          "monitor": true,

```

```
        "encoding": "utf-8"
      }
    ]
  }
],
"hanaPrometheusExporter": {
  "hanaSid": "HDB",
  "hanaPort": "30013",
  "hanaSecretName": "HANA_DB_CREDS",
  "prometheusPort": "9668"
},
"haClusterPrometheusExporter": {
  "prometheusPort": "9664"
}
}
```

SAP NetWeaver on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
          "monitor": true
        },
        {
          "alarmMetricName": "StatusCheckFailed",
          "monitor": true
        },
        {
          "alarmMetricName": "disk_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "mem_used_percent",
          "monitor": true
        },
        {
```

```
    "alarmMetricName": "sap_alerts_ResponseTime",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_ResponseTimeDialog",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_DBRequestTime",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_LongRunners",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_AbortedJobs",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_BasisSystem",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_Database",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_Security",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_System",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_alerts_QueueTime",
    "monitor": true
  },
  {

```

```
    "alarmMetricName": "sap_alerts_Availability",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_start_service_processes",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_dispatcher_queue_now",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_dispatcher_queue_max",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_enqueue_server_locks_max",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_enqueue_server_locks_now",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_enqueue_server_locks_state",
    "monitor": true
  },
  {
    "alarmMetricName": "sap_enqueue_server_replication_state",
    "monitor": true
  }
],
"logs": [
  {
    "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-ML4",
    "logPath": "/usr/sap/ML4/*/work/dev_w*",
    "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  }
]
}
],
"netWeaverPrometheusExporter": {
```

```
"sapSid": "ML4",
"instanceNumbers": [
  "00",
  "11"
],
"prometheusPort": "9680"
}
}
```

SAP NetWeaver High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "ha_cluster_corosync_ring_errors",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_fail_count",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_check_failover_config_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_get_failover_config_HAActive",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        }
      ]
    }
  ]
}
```

```
{
  "alarmMetricName": "sap_alerts_BasisSystem",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_DBRequestTime",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_Database",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_FrontendResponseTime",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_LongRunners",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_QueueTime",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_ResponseTime",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_ResponseTimeDialog",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_Security",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_Shortdumps",
  "monitor": true
},
},
```

```
{
  "alarmMetricName": "sap_alerts_SqlError",
  "monitor": true
},
{
  "alarmMetricName": "sap_alerts_System",
  "monitor": true
},
{
  "alarmMetricName": "sap_enqueue_server_replication_state",
  "monitor": true
},
{
  "alarmMetricName": "sap_start_service_processes",
  "monitor": true
}
],
"logs": [
  {
    "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-PR1",
    "logPath": "/usr/sap/<SID>/D*/work/dev_w*",
    "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
    "monitor": true,
    "encoding": "utf-8"
  }
]
}
],
"haClusterPrometheusExporter": {
  "prometheusPort": "9664"
},
"netWeaverPrometheusExporter": {
  "sapSid": "PR1",
  "instanceNumbers": [
    "11",
    "12"
  ],
  "prometheusPort": "9680"
}
}
```

SQL Always On Availability Group

The following example shows a component configuration in JSON format for SQL Always On Availability Group.

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }
  ]
}
```



```

}, {
  "alarmMetricName" : "SQLServer:General Statistics User Connections",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
  "monitor" : true
}, {
  "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
  "monitor" : true
} ],
"windowsEvents" : [ {
  "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
  "eventName" : "Application",
  "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],

```

```

    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
    "eventName" : "System",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
    "eventName" : "Security",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
  } ],
  "logs" : [ {
    "logGroupName" : "SQL_SERVER_ALWAYS_ON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
    "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\\
\Log\\ERRORLOG",
    "logType" : "SQL_SERVER",
    "monitor" : true,
    "encoding" : "utf-8"
  } ]
}, {
  "subComponentType" : "AWS::EC2::Volume",
  "alarmMetrics" : [ {
    "alarmMetricName" : "VolumeReadBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeReadOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeQueueLength",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
    "monitor" : true
  }, {
    "alarmMetricName" : "BurstBalance",
    "monitor" : true
  } ]
} ]

```

```
} ]
}
```

SQL failover cluster instance

The following example shows a component configuration in JSON format for SQL failover cluster instance.

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Normal Messages Queue Length/sec",
```

```
    "monitor" : true
  }, {
    "alarmMetricName" : "Urgent Message Queue Length/se",
    "monitor" : true
  }, {
    "alarmMetricName" : "Reconnect Count",
    "monitor" : true
  }, {
    "alarmMetricName" : "Unacknowledged Message Queue Length/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Messages Outstanding",
    "monitor" : true
  }, {
    "alarmMetricName" : "Messages Sent/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Database Update Messages/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Update Messages/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Flushes/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Crypto Checkpoints Saved/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Crypto Checkpoints Restored/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Registry Checkpoints Restored/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Registry Checkpoints Saved/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Cluster API Calls/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Resource API Calls/sec",
    "monitor" : true
  }, {
```

```

    "alarmMetricName" : "Cluster Handles/sec",
    "monitor" : true
  }, {
    "alarmMetricName" : "Resource Handles/sec",
    "monitor" : true
  } ],
  "windowsEvents" : [ {
    "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
    "eventName" : "Application",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL"],
    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
    "eventName" : "System",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
    "monitor" : true
  }, {
    "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
    "eventName" : "Security",
    "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
    "monitor" : true
  } ],
  "logs" : [ {
    "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
    "logPath" : "\\\\"amznfsxjzbykwn.mydomain.aws\\SQLDB\\MSSQL**.*MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
    "logType" : "SQL_SERVER",
    "monitor" : true,
    "encoding" : "utf-8"
  } ]
}, {
  "subComponentType" : "AWS::EC2::Volume",
  "alarmMetrics" : [ {
    "alarmMetricName" : "VolumeReadBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteBytes",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeReadOps",
    "monitor" : true
  }, {
    "alarmMetricName" : "VolumeWriteOps",
    "monitor" : true
  } ]
} ]

```

```
    }, {
      "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
      "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

Create and configure CloudWatch Application Insights monitoring using CloudFormation templates

You can add Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from AWS CloudFormation templates.

This section provides sample AWS CloudFormation templates in both JSON and YAML formats to help you create and configure Application Insights monitoring.

To view the Application Insights resource and property reference in the *AWS CloudFormation User Guide*, see [ApplicationInsights resource type reference](#).

Sample templates

- [Create an Application Insights application for the entire AWS CloudFormation stack](#)
- [Create an Application Insights application with detailed settings](#)
- [Create an Application Insights application with CUSTOM mode component configuration](#)
- [Create an Application Insights application with DEFAULT mode component configuration](#)
- [Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration](#)

Create an Application Insights application for the entire AWS CloudFormation stack

To apply the following template, you must create AWS resources and one or more resource groups from which to create Application Insights applications to monitor those resources. For more information, see [Getting started with AWS Resource Groups](#).

The first two parts of the following template specify a resource and a resource group. The last part of the template creates an Application Insights application for the resource group, but does not configure the application or apply monitoring. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Test Resource Group stack",
  "Resources": {
    "EC2Instance": {
      "Type": "AWS::EC2::Instance",
      "Properties": {
        "ImageId" : "ami-abcd1234efgh5678i",
        "SecurityGroupIds" : ["sg-abcd1234"]
      }
    },
    ...
    "ResourceGroup": {
      "Type": "AWS::ResourceGroups::Group",
      "Properties": {
        "Name": "my_resource_group"
      }
    },
    "AppInsightsApp": {
      "Type": "AWS::ApplicationInsights::Application",
      "Properties": {
        "ResourceGroupName": "my_resource_group"
      },
      "DependsOn" : "ResourceGroup"
    }
  }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-abcd1234efgh5678i
      SecurityGroupIds:
        - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
    DependsOn: ResourceGroup
```

The following template section applies the default monitoring configuration to the Application Insights application. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.

When `AutoConfigurationEnabled` is set to `true`, all components of the application are configured with the recommended monitoring settings for the DEFAULT application tier. For more information about these settings and tiers, see [DescribeComponentConfigurationRecommendation](#) and [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Test Application Insights Application stack",
  "Resources": {
    "AppInsightsApp": {
      "Type": "AWS::ApplicationInsights::Application",
      "Properties": {
        "ResourceGroupName": "my_resource_group",
```



```
        "AutoConfigurationEnabled": true
      }
    }
  }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

Create an Application Insights application with detailed settings

The following template performs these actions:

- Creates an Application Insights application with CloudWatch Events notification and OpsCenter enabled. For more information, see the [CreateApplication](#) command details in the *Amazon CloudWatch Application Insights API Reference*.
- Tags the application with two tags, one of which has no tag values. For more information, see [TagResource](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two custom instance group components. For more information, see [CreateComponent](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two log pattern sets. For more information, see [CreateLogPattern](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Sets `AutoConfigurationEnabled` to `true`, which configures all components of the application with the recommended monitoring settings for the DEFAULT tier. For more information, see [DescribeComponentConfigurationRecommendation](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
```

```
"Type": "AWS::ApplicationInsights::Application",
"Properties": {
  "ResourceGroupName": "my_resource_group",
  "CWEMonitorEnabled": true,
  "OpsCenterEnabled": true,
  "OpsItemSNSTopicArn": "arn:aws:sns:us-east-1:123456789012:my_topic",
  "AutoConfigurationEnabled": true,
  "Tags": [
    {
      "Key": "key1",
      "Value": "value1"
    },
    {
      "Key": "key2",
      "Value": ""
    }
  ],
  "CustomComponents": [
    {
      "ComponentName": "test_component_1",
      "ResourceList": [
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
      ]
    },
    {
      "ComponentName": "test_component_2",
      "ResourceList": [
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i",
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
      ]
    }
  ],
  "LogPatternSets": [
    {
      "PatternSetName": "pattern_set_1",
      "LogPatterns": [
        {
          "PatternName": "deadlock_pattern",
          "Pattern": ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))",
          "Rank": 1
        }
      ]
    }
  ],
  {
```

```

    "PatternSetName": "pattern_set_2",
    "LogPatterns": [
      {
        "PatternName": "error_pattern",
        "Pattern": ".*[\\s\\[]ERROR[\\s\\]].*",
        "Rank": 1
      },
      {
        "PatternName": "warning_pattern",
        "Pattern": ".*[\\s\\[]WARN(ING)?[\\s\\]].*",
        "Rank": 10
      }
    ]
  }
]
}
}

```

Template in YAML format

```

---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  CWEMonitorEnabled: true
  OpsCenterEnabled: true
  OpsItemSNSTopicArn: arn:aws:sns:us-east-1:123456789012:my_topic
  AutoConfigurationEnabled: true
  Tags:
    - Key: key1
      Value: value1
    - Key: key2
      Value: ''
  CustomComponents:
    - ComponentName: test_component_1
      ResourceList:
        - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
    - ComponentName: test_component_2
      ResourceList:
        - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
        - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  LogPatternSets:
    - PatternSetName: pattern_set_1

```

```

LogPatterns:
- PatternName: deadlock_pattern
  Pattern: ".*\\sDeadlocked\\sSchedulers(([^\w].*)|($))"
  Rank: 1
- PatternSetName: pattern_set_2
  LogPatterns:
  - PatternName: error_pattern
    Pattern: ".*[\\s\\[[]ERROR[\\s\\]].*"
    Rank: 1
  - PatternName: warning_pattern
    Pattern: ".*[\\s\\[[]WARN(ING)?[\\s\\]].*"
    Rank: 10

```

Create an Application Insights application with CUSTOM mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `CUSTOM`, which causes this component to be configured with the configuration specified in `CustomComponentConfiguration`. For more information, see [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```

{
  "Type": "AWS::ApplicationInsights::Application",
  "Properties": {
    "ResourceGroupName": "my_resource_group",
    "ComponentMonitoringSettings": [
      {
        "ComponentARN": "my_component",
        "Tier": "SQL_SERVER",
        "ComponentConfigurationMode": "CUSTOM",
        "CustomComponentConfiguration": {
          "ConfigurationDetails": {
            "AlarmMetrics": [
              {
                "AlarmMetricName": "StatusCheckFailed"

```

```
    },
    ...
  ],
  "Logs": [
    {
      "LogGroupName": "my_log_group_1",
      "LogPath": "C:\\\\LogFolder_1\\\\*",
      "LogType": "DOT_NET_CORE",
      "Encoding": "utf-8",
      "PatternSet": "my_pattern_set_1"
    },
    ...
  ],
  "WindowsEvents": [
    {
      "LogGroupName": "my_windows_event_log_group_1",
      "EventName": "Application",
      "EventLevels": [
        "ERROR",
        "WARNING",
        ...
      ],
      "Encoding": "utf-8",
      "PatternSet": "my_pattern_set_2"
    },
    ...
  ],
  "Alarms": [
    {
      "AlarmName": "my_alarm_name",
      "Severity": "HIGH"
    },
    ...
  ]
},
"SubComponentTypeConfigurations": [
  {
    "SubComponentType": "EC2_INSTANCE",
    "SubComponentConfigurationDetails": {
      "AlarmMetrics": [
        {
          "AlarmMetricName": "DiskReadOps"
        },
        ...
      ]
    }
  }
]
```

```
    ],
    "Logs": [
      {
        "LogGroupName": "my_log_group_2",
        "LogPath": "C:\\\\LogFolder_2\\\\*",
        "LogType": "IIS",
        "Encoding": "utf-8",
        "PatternSet": "my_pattern_set_3"
      },
      ...
    ],
    "processes" : [
      {
        "processName" : "my_process",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "procstat cpu_usage",
            "monitor" : true
          }, {
            "alarmMetricName" : "procstat memory_rss",
            "monitor" : true
          }
        ]
      }
    ]
  },
  ],
  "WindowsEvents": [
    {
      "LogGroupName": "my_windows_event_log_group_2",
      "EventName": "Application",
      "EventLevels": [
        "ERROR",
        "WARNING",
        ...
      ],
      "Encoding": "utf-8",
      "PatternSet": "my_pattern_set_4"
    },
    ...
  ]
}
}
```

```
    ]  
  }  
}
```

Template in YAML format

```
---  
Type: AWS::ApplicationInsights::Application  
Properties:  
  ResourceGroupName: my_resource_group  
  ComponentMonitoringSettings:  
    - ComponentARN: my_component  
      Tier: SQL_SERVER  
      ComponentConfigurationMode: CUSTOM  
      CustomComponentConfiguration:  
        ConfigurationDetails:  
          AlarmMetrics:  
            - AlarmMetricName: StatusCheckFailed  
            ...  
          Logs:  
            - LogGroupName: my_log_group_1  
              LogPath: C:\LogFolder_1\  
              LogType: DOT_NET_CORE  
              Encoding: utf-8  
              PatternSet: my_pattern_set_1  
            ...  
          WindowsEvents:  
            - LogGroupName: my_windows_event_log_group_1  
              EventName: Application  
              EventLevels:  
                - ERROR  
                - WARNING  
              ...  
              Encoding: utf-8  
              PatternSet: my_pattern_set_2  
            ...  
          Alarms:  
            - AlarmName: my_alarm_name  
              Severity: HIGH  
            ...  
  SubComponentTypeConfigurations:  
    - SubComponentType: EC2_INSTANCE  
      SubComponentConfigurationDetails:
```

```
AlarmMetrics:
- AlarmMetricName: DiskReadOps
...
Logs:
- LogGroupName: my_log_group_2
  LogPath: C:\LogFolder_2\*
  LogType: IIS
  Encoding: utf-8
  PatternSet: my_pattern_set_3
...
Processes:
- ProcessName: my_process
  AlarmMetrics:
  - AlarmMetricName: procstat cpu_usage
    ...
    ...
WindowsEvents:
- LogGroupName: my_windows_event_log_group_2
  EventName: Application
  EventLevels:
  - ERROR
  - WARNING
  ...
  Encoding: utf-8
  PatternSet: my_pattern_set_4
...
```

Create an Application Insights application with DEFAULT mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT` and `Tier` to `SQL_SERVER`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `SQL_Server` tier. For more information, see [DescribeComponentConfiguration](#) and [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format


```
{
  "Type": "AWS::ApplicationInsights::Application",
  "Properties": {
    "ResourceGroupName": "my_resource_group",
    "ComponentMonitoringSettings": [
      {
        "ComponentARN": "my_component",
        "Tier": "SQL_SERVER",
        "ComponentConfigurationMode": "DEFAULT"
      }
    ]
  }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: DEFAULT
```

Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see [CreateApplication](#) in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT_WITH_OVERWRITE` and `tier` to `DOT_NET_CORE`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `DOT_NET_CORE` tier. Overwritten configuration settings are specified in the `DefaultOverwriteComponentConfiguration`:
 - At the component level `AlarmMetrics` settings are overwritten.

- At the sub-component level, for the EC2_Instance type sub-components, Logs settings are overwritten.

For more information, see [UpdateComponentConfiguration](#) in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
  "Type": "AWS::ApplicationInsights::Application",
  "Properties": {
    "ResourceGroupName": "my_resource_group",
    "ComponentMonitoringSettings": [
      {
        "ComponentName": "my_component",
        "Tier": "DOT_NET_CORE",
        "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
        "DefaultOverwriteComponentConfiguration": {
          "ConfigurationDetails": {
            "AlarmMetrics": [
              {
                "AlarmMetricName": "StatusCheckFailed"
              }
            ]
          },
          "SubComponentTypeConfigurations": [
            {
              "SubComponentType": "EC2_INSTANCE",
              "SubComponentConfigurationDetails": {
                "Logs": [
                  {
                    "LogGroupName": "my_log_group",
                    "LogPath": "C:\\\\LogFolder\\*",
                    "LogType": "IIS",
                    "Encoding": "utf-8",
                    "PatternSet": "my_pattern_set"
                  }
                ]
              }
            }
          ]
        }
      ]
    }
  }
}
```

```

    }
  ]
}
}

```

Template in YAML format

```

---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
    - ComponentName: my_component
      Tier: DOT_NET_CORE
      ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
      DefaultOverwriteComponentConfiguration:
        ConfigurationDetails:
          AlarmMetrics:
            - AlarmMetricName: StatusCheckFailed
          SubComponentTypeConfigurations:
            - SubComponentType: EC2_INSTANCE
              SubComponentConfigurationDetails:
                Logs:
                  - LogGroupName: my_log_group
                    LogPath: C:\LogFolder\*
                    LogType: IIS
                    Encoding: utf-8
                    PatternSet: my_pattern_set

```

Tutorial: Set up monitoring for SAP ASE

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP ASE databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP ASE databases.

Application Insights for SAP ASE topics

- [Supported environments](#)
- [Supported operating systems](#)
- [Features](#)

- [Prerequisites](#)
- [Set up monitoring on your SAP ASE database](#)
- [Manage monitoring of your SAP ASE database](#)
- [Configure the alarm threshold](#)
- [View and troubleshoot SAP ASE problems detected by Application Insights](#)
- [Troubleshooting Application Insights for SAP ASE](#)

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns. You provide and install SAP ASE database software and supported SAP application software.

- **One or more SAP ASE databases on a single Amazon EC2 instance** – SAP ASE in a single-node, scale-up architecture.
- **Cross-AZ SAP ASE database high availability setup** – SAP ASE with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Note

CloudWatch Application Insights supports only single SAP system ID (SID) ASE HA environments. If multiple ASE HA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP ASE supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4
- SuSE Linux 12 SP5
- SuSE Linux 15
- SuSE Linux 15 SP1

- SuSE Linux 15 SP2
- SuSE Linux 15 SP3
- SuSE Linux 15 SP4
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- RedHat Linux 7.6
- RedHat Linux 7.7
- RedHat Linux 7.9
- RedHat Linux 8.1
- RedHat Linux 8.4
- RedHat Linux 8.6

Features

CloudWatch Application Insights for SAP ASE provides the following features:

- Automatic SAP ASE workload detection
- Automatic SAP ASE alarm creation based on static threshold
- Automatic SAP ASE alarm creation based on anomaly detection
- Automatic SAP ASE log pattern recognition
- Health dashboard for SAP ASE
- Problem dashboard for SAP ASE

Prerequisites

You must perform the following prerequisites to configure an SAP ASE database with CloudWatch Application Insights:

- **SAP ASE configuration parameters** – The following configuration parameters must be enabled on your ASE DB: "enable monitoring", "sql text pipe max messages", "sql text pipe active". This allows CloudWatch Application Insights to provide full monitoring capabilities for your DB. If these settings aren't enabled on your ASE database, Application Insights will automatically enable them to collect the necessary metrics to allow monitoring.
- **SAP ASE database user** – The database user provided during Application Insights onboarding must have permission to access the following:
 - System tables in the master database and user (tenant) databases
 - Monitoring tables
- **SAPHostCtrl** – Install and set up SAPHostCtrl on your Amazon EC2 instance.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).
- **AWS Systems Manager enablement** – Install SSM Agent on your instances, and enable the instances enabled for SSM. For information about how to install the SSM agent, see [Working with SSM Agent](#) in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your database.
 - You must attach the AmazonSSMManagedInstanceCore role to enable Systems Manager. For more information, see [AWS Systems Manager identity-based policy examples](#).
 - You must attach the CloudWatchAgentServerPolicy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM roles and users for use with Amazon CloudWatch agent](#).
 - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in AWS Secrets Manager. For more information about inline policies, see [Inline policies](#) in the *AWS Identity and Access Management User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
```

```

        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}

```

- **AWS Resource Groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP ASE database. If there are multiple databases per account, we recommend that you create one resource group that includes the AWS resources for each SAP ASE database system.
- **IAM permissions** – For non-admin users:
 - You must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see [IAM policy](#).
 - The user must have permission to create a secret in AWS Secrets Manager to store the database user credentials. For more information, see [Example: Permission to create secrets](#).

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}


```

- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).

Set up monitoring on your SAP ASE database

Use the following steps to set up monitoring for your SAP ASE database

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group that contains your SAP ASE database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the [AWS Resource Groups User Guide](#).
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
6. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and AWS CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see [Tag Editor](#) in the *AWS Resource Groups and Tags User Guide*.
8. Choose **Next** to continue to set up monitoring.
9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

 **Note**

Components that contain a detected SAP ASE High Availability workload support only one workload on a component. Components that contain a detected SAP ASE single node workload support multiple workloads, but you can't add or remove workloads. All automatically detected workloads will be monitored.

10. Choose **Next**.

11. On the **Specify component details** page, enter the username and password of your SAP ASE databases.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**. If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP ASE system, which can take up to 2 hours.

Manage monitoring of your SAP ASE database

You can manage user credentials, metrics, and log paths for your SAP ASE database by performing the following steps:

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP ASE metrics according to your requirements. SAP ASE metric names are prefixed with `asedb`. You can add up to 60 metrics per component.
7. Under **ASE configuration**, enter the username and password for the SAP ASE database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP ASE database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Configure the alarm threshold

CloudWatch Application Insights automatically creates a Amazon CloudWatch metric for the alarm to watch, along with the threshold for that metric. The alarm changes to the **ALARM** state when the metric surpasses the threshold for a specified number of evaluation periods. Note that these settings are not retained by Application Insights.

To edit an alarm for a single metric, perform the following steps:

1. Open the [CloudWatch console](#).
2. In the left navigation pane, choose **Alarms>All alarms**.
3. Select the radio button next to the alarm that was automatically created by CloudWatch Application Insights. Then choose **Actions**, and select **Edit** from the dropdown menu.
4. Edit the following parameters under **Metric**.
 - a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile. For example, p95 . 45.
 - b. Under **Period**, choose the evaluations period for the alarm. When you evaluate the alarm, each period is aggregated into one data point.
5. Edit the following parameters under **Conditions**.
 - a. Choose whether the metric must be greater than, less than, or equal to the threshold.
 - b. Specify the threshold value.
6. Under **Additional configuration** edit the following parameters.
 - a. Under **Datapoints to alarm**, specify the number of data points, or evaluation periods, that must be in the **ALARM** state to initiate the alarm. When the two values match, an alarm is created that enters **ALARM** state if the designated number of consecutive periods are exceeded. To create an m out of n alarm, specify a lower value for the first data point than for the second. For more information about evaluating alarms, see [Evaluating an alarm](#).
 - b. Under **Missing data treatment**, choose the behavior of the alarm when some data points are missing. For more information about missing data treatment, see [Configuring how CloudWatch alarms treat missing data](#).
 - c. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when

the sample size is too low. For more information about percentiles with low samples, see [Percentile-based CloudWatch alarms and low data samples](#).

7. Choose **Next**.
8. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.
9. Choose **Update alarm**.

View and troubleshoot SAP ASE problems detected by Application Insights

This section helps you resolve common troubleshooting problems that occur when you configure monitoring for SAP ASE on Application Insights.

SAP ASE Backup Server errors

You can identify the error message by checking the dynamically created dashboard. The dashboard shows the error message reported in the SAP ASE Backup Server. For more details about SAP ASE Backup Server logs, see [SAP Documentation Backup Server Error Logging](#).

SAP ASE long running transactions

Identify the long running transaction and confirm whether it can be stopped or if the running time is intentional. For more details, see [2180410 — How to display transaction log records for long running transactions? — SAP ASE](#).

SAP ASE User connections

Review whether your SAP ASE database is sized accordingly for the workload you intend to run on the database. For more details, see [Configuring User Connections](#) in the SAP documentation.

SAP ASE disk space

You can identify the database layer that is causing the problem by checking the dynamically created dashboard. The dashboard shows the related metrics and log file snippets. It is important to understand the cause of the disk growth and when applicable, increase the physical disk size, the allocated disk space, or both. For more details, see [SAP Documentation disk resize](#) in the SAP documentation.

Troubleshooting Application Insights for SAP ASE

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Error	Error returned	Root cause	Resolution
Unable to add more than 60 monitor metrics.	Component cannot have more than 60 monitored metric	The current metric limit is 60 monitored metrics per component.	Remove unnecessary metrics to adhere to the limit.
No SAP metrics or alarms appear after the onboarding process	The run command on the AWS-ConfigureAWSPackage failed in AWS Systems Manager. The output shows the error: CT-LIBRARY error:ct_connect(): protocol specific layer: external error: The attempt to connect to the server failed	The username and password might be incorrect.	Verify that the username and password are valid, then rerun the onboarding process.

Tutorial: Set up monitoring for SAP HANA

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP HANA databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP HANA databases.

Application Insights for SAP HANA topics

- [Supported environments](#)

- [Supported operating systems](#)
- [Features](#)
- [Prerequisites](#)
- [Set up your SAP HANA database for monitoring](#)
- [Manage monitoring of your SAP HANA database](#)
- [View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights](#)
- [Anomaly detection for SAP HANA](#)
- [Troubleshooting Application Insights for SAP HANA](#)

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns. You provide and install SAP HANA database software and supported SAP application software.

- **SAP HANA database on a single Amazon EC2 instance** — SAP HANA in a single-node, scale-up architecture, with up to 24TB of memory.
- **SAP HANA database on multiple Amazon EC2 instances** — SAP HANA in a multi-node, scale-out architecture.
- **Cross-AZ SAP HANA database high availability setup** — SAP HANA with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Note

CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP HANA supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP

- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 15 SP5 For SAP
- RedHat Linux 8.6 For SAP With High Availability and Update Services
- RedHat Linux 8.5 For SAP With High Availability and Update Services
- RedHat Linux 8.4 For SAP With High Availability and Update Services
- RedHat Linux 8.3 For SAP With High Availability and Update Services
- RedHat Linux 8.2 For SAP With High Availability and Update Services
- RedHat Linux 8.1 For SAP With High Availability and Update Services
- RedHat Linux 7.9 For SAP With High Availability and Update Services

Features

CloudWatch Application Insights for SAP HANA provides the following features:

- Automatic SAP HANA workload detection
- Automatic SAP HANA alarm creation based on static threshold
- Automatic SAP HANA alarm creation based on anomaly detection
- Automatic SAP HANA log pattern recognition
- Health dashboard for SAP HANA
- Problem dashboard for SAP HANA

Prerequisites

You must perform the following prerequisites to configure an SAP HANA database with CloudWatch Application Insights:

- **SAP HANA** – Install a running and reachable SAP HANA database 2.0 SPS05 on an Amazon EC2 instance.
- **SAP HANA database user** – A database user with monitoring roles must be created in the SYSTEM database and all tenants.

Example

The following SQL commands create a user with monitoring roles.

```
su - <sid>adm
hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
FORCE_FIRST_PASSWORD_CHANGE;
CREATE ROLE CW_HANADB_EXPORTER_ROLE;
GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
```

- **Python 3.8** – Install Python 3.8 or later versions on your operating system. Use the latest release of Python. If Python3 is not detected on your operating system, Python 3.6 will be installed.

For more information, see the [installation example](#).

Note

Manual installation of Python 3.8 or higher is required for SuSE Linux 15 SP4, RedHat Linux 8.6, and later operating systems.

- **Pip3** – Install the installer program, pip3, on your operating system. If pip3 is not detected on your operating system, it will be installed.
- **hdbclient** – CloudWatch Application Insights uses the python driver to connect to the SAP HANA database. If the client is not installed under python3, ensure that you have hdbclient tar file version 2.10 or later under /hana/shared/SID/hdbclient/.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

- **AWS Systems Manager enablement** – Install SSM Agent on your instances, and the instances must be enabled for SSM. For information about how to install the SSM Agent, see [Working with SSM Agent](#) in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your database.
 - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see [AWS Systems Manager identity-based policy examples](#).
 - You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).
 - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in AWS Secrets Manager. For more information about inline policies, see [Inline policies](#) in the *AWS Identity and Access Management User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **AWS resource groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP HANA database. If there are multiple databases per account, we recommend that you create one resource group that includes the AWS resources for each SAP HANA database system.
- **IAM permissions** – For non-admin users:
 - You must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see [IAM policy](#).

- The user must have permission to create a secret in AWS Secrets Manager to store the database user credentials. For more information, see [Example: Permission to create secrets](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).


Set up your SAP HANA database for monitoring

Use the following steps to set up monitoring for your SAP HANA database

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group that contains your SAP HANA database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the [AWS Resource Groups User Guide](#).
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS

CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.

6. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and AWS CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see [Tag Editor](#) in the *AWS Resource Groups and Tags User Guide*.
8. Choose **Next** to continue to set up monitoring.
9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.
 - a. To add workloads to a component that contains a detected SAP HANA single node workload, select the component, then choose **Edit component**.

 **Note**

Components that contain a detected SAP HANA multi node or HANA High Availability workload support only one workload on a component.

Review detected components [Info](#)

▼ Selected application

Application
NWHANA_QE9

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/NWHANA_QE9

Review components for monitoring (1/2) [Info](#) Edit component

Components and their workloads detected by Application Insights.

< 1 > ⚙

Detected components	Monitoring	Associated workloads
<input checked="" type="radio"/> HANA database HANA-QE7-00	✔ Enabled	• HANA_SN (HANA single node)
<input type="radio"/> SAP NetWeaver SAP-NW-QE7	✔ Enabled	• SAP_NWD (NetWeaver Distributed)

Hana database client agreement

Install the HANA database client in my environment

▶ [SAP HANA client license agreement](#)

Cancel Previous Next

b. To add a new workload, choose **Add new workload.**

CloudWatch > Application Insights > Add an application

Step 2 of 4

Review detected components [Info](#)

▼ Selected application

Application
NWHANA_QE9

Resource group ARN
arn:aws:resource-groups:us-east-1:856960489879:group/NWHANA_QE9

Review components for monitoring (1/2) [Info](#) Edit component

Components and their workloads detected by Application Insights.

< 1 > ⚙

Detected components	Monitoring	Associa..
<input checked="" type="radio"/> HANA database HANA-QE7-00	✔ Enabled	• HANA...
<input type="radio"/> SAP NetWeaver SAP-NW-QE7	✔ Enabled	• SAP_N...

Edit component ✕

Component type
HANA database

Component name
HANA-QE7-00

Associated workloads

Some workload types support adding only one workload of that type on a component. For more information about workload types supported by Application Insights, see [Documentation](#) 🔗

Workload type Workload name

Add new workload

You can add up to 5 workloads

Cancel Save changes

- c. When you are finished editing workloads, choose **Save changes**.
10. Choose **Next**.
11. On the **Specify component details** page, enter the username and password.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**. If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Manage monitoring of your SAP HANA database

You can manage user credentials, metrics, and log paths for your SAP HANA database by performing the following steps:

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP HANA metrics according to your requirements. SAP HANA metric names are prefixed with hanadb. You can add up to 40 metrics per component.
7. Under **HANA configuration**, enter the password and user name for the SAP HANA database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP HANA database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP HANA on Application Insights.

Troubleshooting topics

- [SAP HANA database reaches memory allocation limit](#)
- [Disk full event](#)
- [SAP HANA backup stopped running](#)

SAP HANA database reaches memory allocation limit

Description

Your SAP application that is backed by an SAP HANA database malfunctions because of high memory pressure, leading to application performance degradation.

Resolution

You can identify the application layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be because of a large data load in the SAP HANA system.

CloudWatch: Application Insights
Problem Id: p-91974e9c-e31b-4f35-8577-0ca00fabff84 [Edit configuration](#)

1h 3h 12h 1d 3d 1w custom (4d) Actions Refresh

Problem summary

Severity	Problem summary	Source	Start-time	Status	Resource group	SSM OpsItem
High	SAP HANA: Allocation limit used (%) exceeded the threshold	saphanacomponent-DM4-00-79ec8266-5692-49c3-8dd8-38163d420087	2021-11-03T14:01:21Z	In progress	AI-SUSE-1-Node-DM4	oi-902e0d35c005

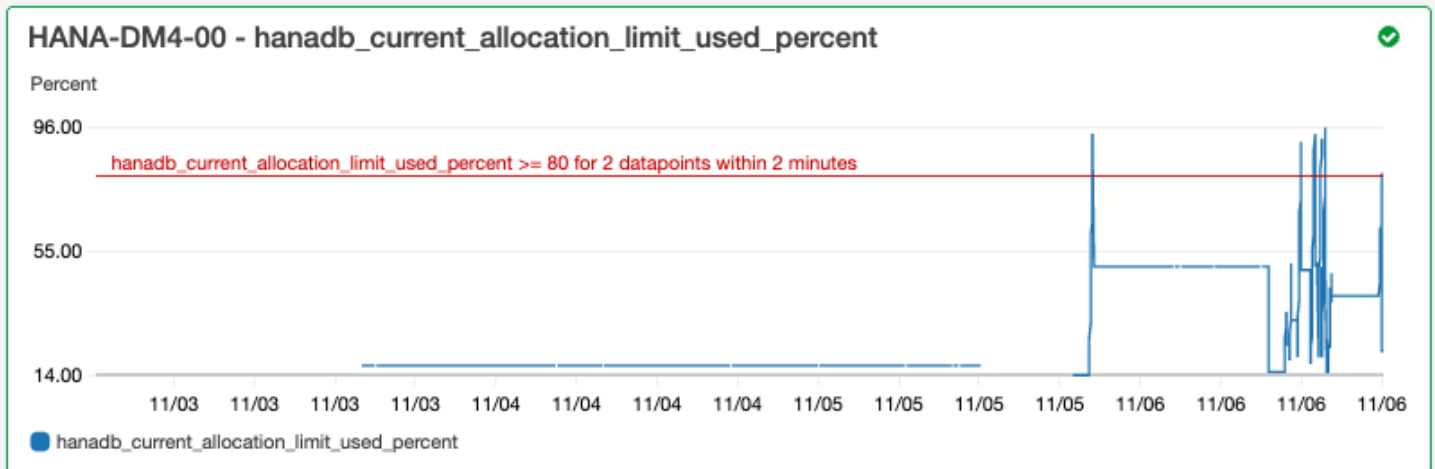
Insight

Check the current memory utilization. Identify and resolve reasons which are responsible for the used memory coming close to the allocation limit. In addition, examine the CloudWatch Log Insights widget in the problem dashboard below. If your investigation indicates a requirement to have more memory capacity, you can resize your instances to a different EC2 instance type. See <https://aws.amazon.com/sap/instance-types/> for all the SAP certified EC2 instances for SAP HANA.

Help us improve our models: This insight is useful This insight is not useful [Submit feedback](#)

The used memory allocation exceeds the threshold of 80 percent of the total memory allocation limit.

EC2 instance group - HANA-DM4-00



The log group shows the scheme BNR-DATA and table IMDBMASTER_30003 ran out of memory. In addition, the log group shows the exact time of the issue, current global location limit, shared memory, code size, and OOM reservation allocation size.

Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM4, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.OutOfMemory

```
#      :@timestamp      :@message
# 1 2021-11-06T13:31:23.317Z GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 2 [2867][311260][22/963854] 2021-11-06 13:00:44.99570 e OOM.Notification Statement.cc(94580) : oom exception occurred at 'imdbmaster:30003': conn_id=311260, stmt_id=1336853818011966, stmt_hash=17e1ccc2b5f460604ce0e8c98690fd01, sql=CAL
# 3 [3033][311513][22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
# 4 Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad
# 5 [2822][-1][-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
# 6 Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad
# 7 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 8 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 9 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 10 [3033][311513][22/967162] 2021-11-06 13:31:17.180223 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/PersistenceManager/PersistentSpace/DefaultLTPA/DataPage, size 167772168, alignment=40968, flags 0x0, reason GLOBAL_ALLO
# 11 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 12 [3033][311513][22/967162] 2021-11-06 13:31:17.163640 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
# 13 Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad
# 14 [2822][-1][-1/-1] 2021-11-06 13:31:17.170707 w Memory mmPoolAllocator.cpp(01212) : Out of memory for Pool/malloc/libhdbbase.ment.so, size 422808, alignment=88, flags 0x0, reason GLOBAL_ALLOCATION_LIMIT
# 15 [2822][-1][-1/-1] 2021-11-06 13:31:17.175597 e Memory mmReportMemoryProblems.cpp(01805) : OUT OF MEMORY occurred.
# 16 Current callstack: 1: 0x00007f824538dd35 in MemoryManager::PoolAllocator::notifyOOMImpl(unsigned long, unsigned long, bool, ltt::allocation_failure_type, bool)+0x1b1 at mmPoolAllocator.cpp:2284 (libhdbbasis.so) 2: 0x00007f824524a7ad
# 17 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 18 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
# 19 GLOBAL_ALLOCATION_LIMIT (GAL) = 55.78gb (S9901001728b), SHARED_MEMORY = 567.77mb (S95357696b), CODE_SIZE = 2.94gb (3162550272b), OOM_RESERVATION_ALLOCATOR_SIZE = 96.14mb (100810752b)
```

Disk full event

Description

Your SAP application that is backed by an SAP HANA database stops responding, which leads to an inability to access the database.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be that the administrator failed to enable automatic log backup, which caused the sap/hana/log directory to fill up.

Problem summary

Severity	Problem summary	Source	Start-time	Status	Resource group	SSM OpsItem
Medium	SAP HANA: DISK FULL error has been detected	i-043851dc9a2ab15cc	2021-11-05T18:07:29Z	In progress	AI-SUSE-1-Node-DM2	oi-88f4cb8fcf8

Insight

If the HANA database does not accept any of the new requests due to log volume is full. We strongly advise against remove either data files or log files using operating system tools as this will corrupt the database. The recommendation is to follow SAP Note 1679938 to temporarily free up space in the log volume, this way you should be able to start up the database for root cause analysis and problem resolution.

Help us improve our models: This insight is **useful** This insight is **not useful**

The log group widget in the problem dashboard shows the DISKFULL event.

Log Group: SAP_HANA_TRACE-AI-SUSE-1-Node-DM2, Log Type: SAP_HANA_TRACE, AWS::SAPHANA.DiskFull

```
#      :@timestamp      :@message
▼ 1   2021-11-06T18:00:20.072Z [26768][-1][-1/-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
      @ingestionTime      1636221622489
      @log                  [REDACTED]:SAP_HANA_TRACE-AI-SUSE-1-Node-DM2
      @logStream            i-[REDACTED]
      @message              [26768][-1][-1/-1] 2021-11-06 18:00:16.556583 i EventHandler LocalFileCallback.cpp(00517) : [DISKFULL] restarting queue with 1 requests
      @timestamp           1636221620072
```

SAP HANA backup stopped running

Description

Your SAP application that is backed by an SAP HANA database has stopped working.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets.

The log group widget in the problem dashboard shows the ACCESS DENIED event. This includes additional information, such as the S3 bucket, the S3 bucket folder, and the S3 bucket Region.

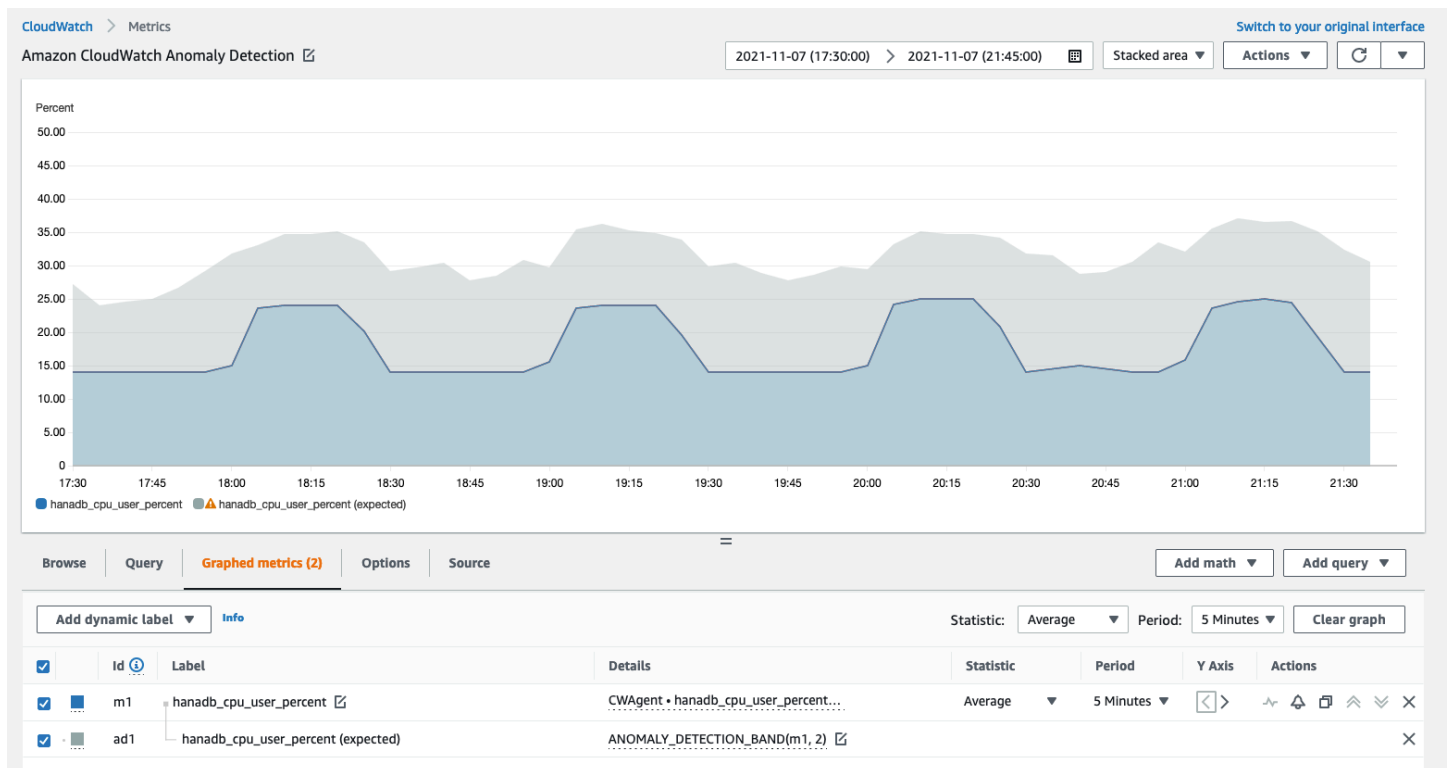
Log Group: SAP_HANA_LOGS-AI-SUSE-1-Node-DM3, Log Type: SAP_HANA_LOGS, AWS::SAPHANA.BackupErrorAccessDenied

```
#      :@timestamp      :@message
▼ 1   2021-11-06T20:28:34.502Z 2021-11-06 20:28:34.493 backint terminated: pid: 21196 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
      @ingestionTime      1636238519523
      @log                  784391381160:SAP_HANA_LOGS-AI-SUSE-1-Node-DM3
      @logStream            i-00164aade25f3231b
      @message              2021-11-06 20:28:34.493 backint terminated:
                          pid: 21196
                          exit code: 1
                          output:
                          exception:
                          exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243)
                          Backint exited with exit code 1 instead of 0. console output: time="2021-11-06T20:28:34Z" level=info msg="Starting execution." time="2021-11-06T20:28:34Z" level=info msg="Loading configuration file /usr/sap/DM3/SYS/global/hdb/opt/hdbconfi
      @timestamp           1636238514502
  2   2021-11-06T20:27:46.035Z 2021-11-06 20:27:41.418 backint terminated: pid: 21080 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
  3   2021-11-06T20:27:22.974Z 2021-11-06 20:27:22.959 backint terminated: pid: 21089 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
  4   2021-11-06T20:26:46.035Z 2021-11-06 20:26:41.277 backint terminated: pid: 20947 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
  5   2021-11-06T20:26:39.035Z 2021-11-06 20:26:34.218 backint terminated: pid: 20931 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
  6   2021-11-06T20:26:22.940Z 2021-11-06 20:26:22.823 backint terminated: pid: 20876 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
  7   2021-11-06T20:25:41.183Z 2021-11-06 20:25:41.136 backint terminated: pid: 20814 exit code: 1 output: exception: exception 1: no.110507 (Backup/Destination/Backint/impl/BackupDestBackint_Executor.cpp:243) Backint exited with exit code 1 instead of 0. console _
```

Anomaly detection for SAP HANA

For specific SAP HANA metrics, such as the number of thread count, CloudWatch applies statistical and machine learning algorithms to define the threshold. These algorithms continuously analyze the metrics of the SAP HANA database, determine normal baselines, and surface anomalies with minimal user intervention. The algorithms generate an anomaly detection model, which generates a range of expected values that represent normal metric behavior.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes can be hourly, daily, or weekly, as shown in the following examples of the SAP HANA CPU usage.



After you create a model, CloudWatch anomaly detection continuously evaluates the model and makes adjustments to it to ensure that it is as accurate as possible. This includes retraining the model to adjust if the metric values evolve over time or experience sudden changes. It also includes predictors to improve the models for metrics that are seasonal, spiky, or sparse.

Troubleshooting Application Insights for SAP HANA

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitored metrics

The output shows the following error.

```
Component cannot have more than 60 monitored metrics
```

Root cause – The current metric limit is 60 monitored metrics per component.

Resolution – To stay under the limit, remove metrics that are not necessary.

No SAP metrics appear after the onboarding process

Use the following information to find out why SAP metrics don't appear on the dashboard after the onboarding process. The first step is to troubleshoot why the SAP metrics don't appear using the AWS Management Console or Exporter logs from an Amazon EC2 instance. Next, review the error output to find a resolution.

Troubleshoot why SAP metrics don't appear after onboarding

You can use the AWS Management Console or exporter logs from an Amazon EC2 instance for troubleshooting.

AWS Management Console

Troubleshoot no SAP metrics appear after onboarding using the console

1. Open the AWS Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the left navigation pane, choose **State Manager**.
3. Under **Associations**, check the status of the document `AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.
4. Under **Associations**, check the status of the document `AWS-ConfigureAWSPackage`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.


Exporter logs from Amazon EC2 instance

Troubleshoot no SAP metrics appear after onboarding using exporter logs

1. Connect to the Amazon EC2 instance where your SAP HANA database is running.

2. Find the correct naming convention for `WORKLOAD_SHORT_NAME` using the following command. You will use this short name in the following two steps.

```
sudo systemctl | grep exporter
```

 **Note**

Application Insights adds a suffix, `WORKLOAD_SHORT_NAME` to the service name depending on the workload that is running. The short names for SAP HANA single node, multiple nodes, and high availability deployments are `HANA_SN`, `HANA_MN`, and `HANA_HA`.

3. To check for errors in the exporter manager service logs, run the following command replacing `WORKLOAD_SHORT_NAME` with the short name you found in [Step 2](#).

```
sudo journalctl -e --unit=prometheus-  
hanadb_exporter_manager_WORKLOAD_SHORT_NAME.service
```

4. If the exporter manager service logs do not show an error, check for errors in the exporter service logs by running the following command.

```
sudo journalctl -e --unit=prometheus-hanadb_exporter_WORKLOAD_SHORT_NAME.service
```

Resolving the common root causes for SAP metrics not appearing after onboarding

The following examples describe how to resolve the common root causes of SAP metrics not appearing after onboarding.

- The output shows the following error.

```
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-  
cloudwatch-agent.d/default ...  
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/  
amazon-cloudwatch-agent.d/ssm_AmazonCloudWatch-ApplicationInsights-  
SSMParameterForTESTCWE2INSTANCEi0d88867f1f3e36285.tmp ...  
2023/11/30 22:25:17 Failed to merge multiple json config files.  
2023/11/30 22:25:17 Failed to merge multiple json config files.  
2023/11/30 22:25:17 Under path : /metrics/append_dimensions | Error : Different  
values are specified for append_dimensions
```

```
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/disk | Error : Different
values are specified for disk
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/mem | Error : Different
values are specified for mem
2023/11/30 22:25:17 Configuration validation first phase failed. Agent version: 1.0.
Verify the JSON input is only using features supported by this version.
```

Resolution – Application Insights is trying to configure the same metrics that are pre-configured as part of the existing CloudWatch agent configuration file. Remove the existing files under `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/` or remove the metrics that are causing the conflict from the existing CloudWatch agent configuration file.

- The output shows the following error.

```
Unable to find a host with system database, for more info rerun using -v
```

Resolution – The username, password, or database port may be incorrect. Verify that the username, password, and port are valid, then re-run the onboarding process.

- The output shows the following error.

```
This hdbcli installer is not compatible with your Python interpreter
```

Resolution – Upgrade pip3 and wheel as shown in the following example for Python 3.6.

```
python3.6 -m pip install --upgrade pip setuptools wheel
```

- The output shows the following error.

```
Unable to install hdbcli using pip3. Please try to install it
```

Resolution – Ensure that you have followed the `hdbcli` prerequisites or install `hdbcli` manually under pip3.

- The output shows the following error.

```
Package 'boto3' requires a different Python: 3.6.15 not in '>= 3.7'
```

Resolution – Python 3.8 or higher is required for this operating system version. Check the Python 3.8 prerequisites and install it.

- The output shows one of the following installation errors.

```
Can not execute `setup.py` since setuptools is not available in the build environment
```

or

```
[SSL: CERTIFICATE_VERIFY_FAILED]
```

Resolution – Install Python using SUSE Linux commands as shown in the following example. The following example installs the latest version of [Python 3.8](#).

```
wget https://www.python.org/ftp/python/3.8.<LATEST_RELEASE>/
Python-3.8.<LATEST_RELEASE>.tgz
tar xf Python-3.*
cd Python-3.*
sudo zypper install make gcc-c++ gcc automake autoconf libtool
sudo zypper install zlib-devel
sudo zypper install libopenssl-devel libffi-devel
./configure --with-ensurepip=install
sudo make
sudo make install
sudo su
python3.8 -m pip install --upgrade pip setuptools wheel
```

Tutorial: Set up monitoring for SAP NetWeaver

This tutorial demonstrates how to configure Amazon CloudWatch Application Insights to set up monitoring for SAP NetWeaver. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and reduce mean time to resolution (MTTR) for your SAP NetWeaver application servers.

CloudWatch Application Insights for SAP NetWeaver topics

- [Supported environments](#)
- [Supported operating systems](#)
- [Features](#)
- [Prerequisites](#)

- [Set up your SAP NetWeaver application servers for monitoring](#)
- [Manage monitoring of your SAP NetWeaver application servers](#)
- [View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights](#)
- [Troubleshooting Application Insights for SAP NetWeaver](#)

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns.

- **SAP NetWeaver Standard System Deployment.**
- **SAP NetWeaver Distributed deployments on multiple Amazon EC2 instances.**
- **Cross-AZ SAP NetWeaver high availability setup** – SAP NetWeaver with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Supported operating systems

CloudWatch Application Insights for SAP NetWeaver is supported on the following operating systems:

- Oracle Linux 8
- Red Hat Enterprise Linux 7.6
- Red Hat Enterprise Linux 7.7
- Red Hat Enterprise Linux 7.9
- Red Hat Enterprise Linux 8.1
- Red Hat Enterprise Linux 8.2
- Red Hat Enterprise Linux 8.4
- Red Hat Enterprise Linux 8.6
- SUSE Linux Enterprise Server 15 for SAP
- SUSE Linux Enterprise Server 15 SP1 for SAP
- SUSE Linux Enterprise Server 15 SP2 for SAP
- SUSE Linux Enterprise Server 15 SP3 for SAP
- SUSE Linux Enterprise Server 15 SP4 for SAP

- SUSE Linux Enterprise Server 12 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP5 for SAP
- SUSE Linux Enterprise Server 15 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP1 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP2 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP3 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP5 except High Availability patterns

Features

CloudWatch Application Insights for SAP NetWeaver 7.0x–7.5x (including ABAP Platform) provides the following features:

- Automatic SAP NetWeaver workload detection
- Automatic SAP NetWeaver alarm creation based on static thresholds
- Automatic SAP NetWeaver log pattern recognition
- Health dashboard for SAP NetWeaver
- Problem dashboard for SAP NetWeaver

Prerequisites

You must perform the following prerequisites to configure SAP NetWeaver with CloudWatch Application Insights:

- **AWS Systems Manager enablement** – Install SSM Agent on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see [Setting up AWS Systems Manager](#) in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your SAP NetWeaver monitoring.
 - You must attach the AmazonSSMManagedInstanceCore role to enable Systems Manager. For more information, see [AWS Systems Manager identity-based policy examples](#).

- You must attach the `CloudWatchAgentServerPolicy` policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).
- **AWS resource groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances, Amazon EFS, and Amazon EBS volumes running your SAP NetWeaver application servers. If there are multiple SAP NetWeaver systems per account, we recommend that you create one resource group that includes the AWS resources for each SAP NetWeaver system. For more information about creating resource groups, see the [AWS Resource Groups and Tags User Guide](#).
- **IAM permissions** – For users who don't have administrative access, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see [IAM policy](#).
- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).
- **Amazon CloudWatch agent** – Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application Insights from the existing CloudWatch agent configuration file. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

Set up your SAP NetWeaver application servers for monitoring

Use the following steps to set up monitoring for your SAP NetWeaver application servers.

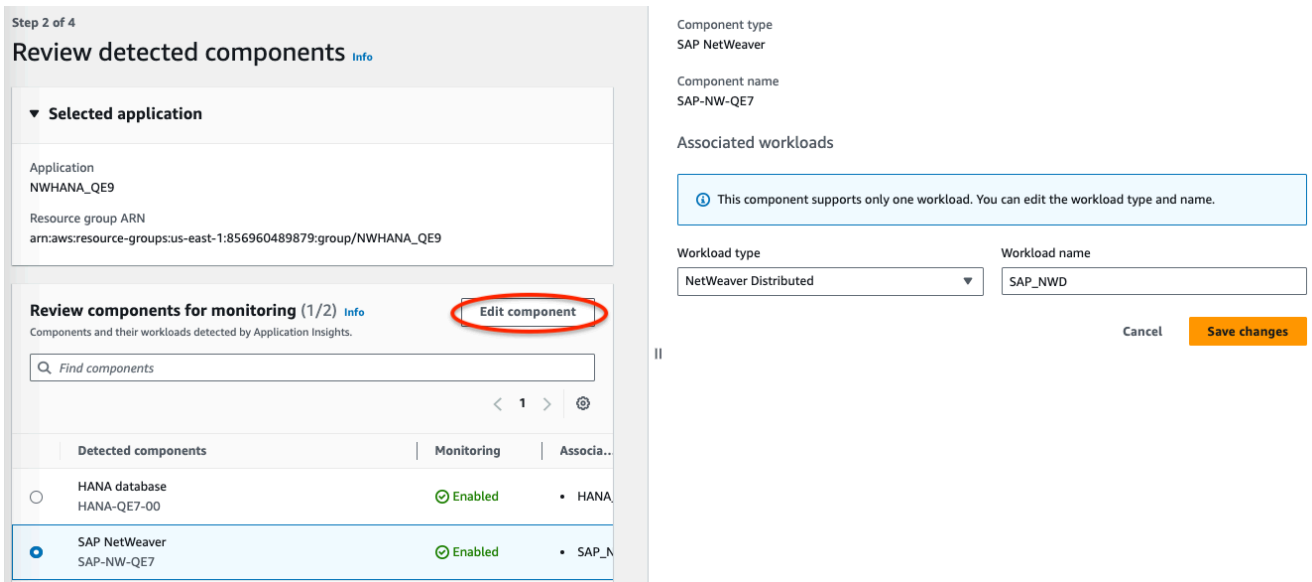
To set up monitoring

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, select **Add an application**.

4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group you created that contains your SAP NetWeaver resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown list.
5. Under **Automatic monitoring of new resources**, select the check box to allow Application Insights to automatically monitor the resources that are added to the application's resource group after onboarding.
6. Under **Monitor EventBridge events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
7. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called [OpsItems](#), that are related to your AWS resources, provide an SNS topic ARN.
8. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and AWS CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see [Tag Editor](#) in the *AWS Resource Groups and Tags User Guide*.
9. To review detected components, choose **Next**.
10. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.
 - To edit the workload type and name, choose **Edit component**.

 **Note**

Components that contain a detected NetWeaver Distributed or NetWeaver High Availability workload support only one workload on a component.



11. Choose **Next**.
12. On the **Specify component details** page, choose **Next**.
13. Review your application monitoring configuration, then choose **Submit**.
14. The application details page opens, where you can view the **Application summary**, **Dashboard**, **Components**, and **Workloads**. You can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. After you submit your application, CloudWatch Application Insights deploys all of the metrics and alarms for your SAP NetWeaver system, which can take up to an hour.

Manage monitoring of your SAP NetWeaver application servers

Use the following steps to manage monitoring of your SAP NetWeaver application servers.

To manage monitoring

1. Open the [CloudWatch console](#).
2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. Choose the **List view** tab.
4. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
5. Select your application.

6. Choose the **Components** tab.
7. Under **Monitored components**, select the radio button next to the component name. Then, select **Manage monitoring**.
8. Under **Instance logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
9. Under **Metrics**, you can select the SAP NetWeaver metrics according to your requirements. SAP NetWeaver metric names are prefixed with `sap`. You can add up to 40 metrics per component.
10. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
11. Review your application monitoring configuration and choose **Save**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP NetWeaver systems.

View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP NetWeaver on Application Insights.

Troubleshooting topics

- [SAP NetWeaver database connectivity issues](#)
- [SAP NetWeaver application availability issues](#)

SAP NetWeaver database connectivity issues

Description

Your SAP NetWeaver application experiences database connectivity issues.

Cause

You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.

Detected problems summary [Info](#) Last 7 days ▾

1 Problems

■ Resolved ■ Unresolved

Detected problems (1) Find problems Last 7 days ▾ < 1 > ⚙️

Severity	Problem summary	Source	Start time	Status
High	SAP: Availability	netweavercomponent-HE4-9da46bcb-f...	2022-12-09T18:56:40Z	In progress

In the following example, under **Problem summary**, SAP: Availability is the issue.

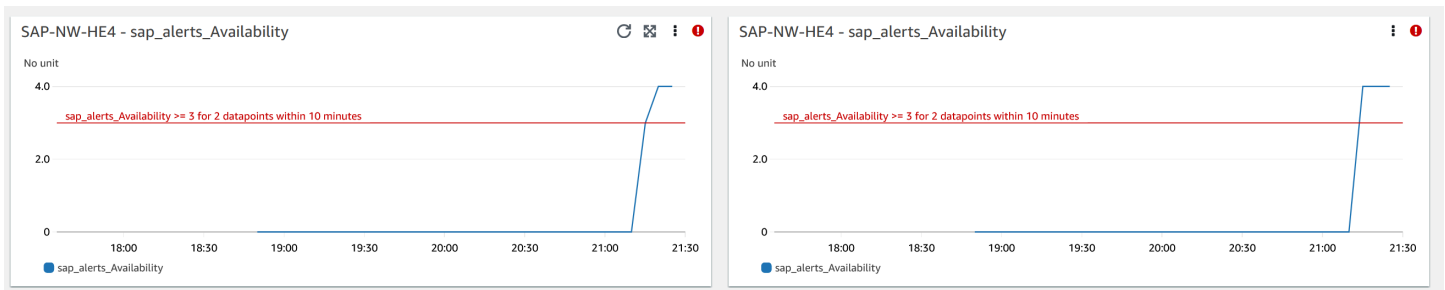
Problem summary Problem ID p-61324679-dc66-4524-aa5a-6fadfc588d37 Severity High Problem summary SAP: Availability Resolution Method Info -	Source netweavercomponent-HE4-9da46bcb-f49c-4dc5-a0cd-7a46965de8bb First occurrence time 2022-12-09T18:56:40Z Last recurrence time - Resolution time -	Status In progress Number of recurrences 0 Resource group HA_HE4 SSM OpsItem oi-657ee61effbd Info
---	--	---

Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.

Insight [Info](#)

An availability issue with your SAP application server instance has been detected. Check SM21, SM50, SM51, SM66 and CCMS (RZ20) > InstanceAsTask > Availability.

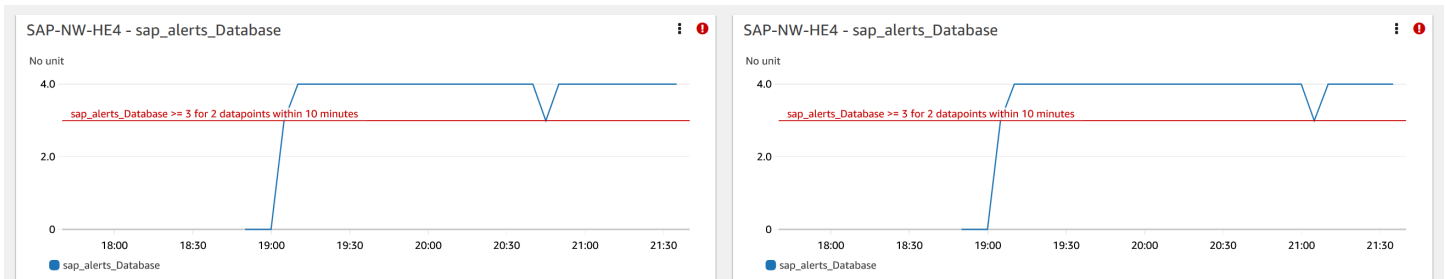
On the same problem dashboard, you can view related logs and metrics that problem detection has grouped together to help you isolate the cause of the error. The `sap_alerts_Availability` metric tracks the availability of the SAP NetWeaver system over time. You can use historical tracking to correlate when the metric initiated an error state or breached the alarm threshold. In the following example, there is an availability issue with the SAP NetWeaver system. The example shows two alarms because there are two SAP application server instances and an alarm was created for each instance.



For more information about each alarm, hover over the `sap_alerts_Availability` metric name.

CWAgent sap_alerts_Availability	
Application:	HA_HE4
ComponentName:	SAP-NW-HE4
instance_hostname:	sapapp
instance_number:	0
object:	InstanceAsTask
SID:	HE4
Region:	us-east-1
Threshold:	sap_alerts_Availability >= 3 for 2 datapoints within 10 minutes
Period:	5 minutes
Statistic:	Maximum
Unit:	None
Min:	0
Max:	4
Average:	0.657143
Sum:	23
Last value:	4
Last time:	2022-12-09 21:40:00 UTC

In the following example, the `sap_alerts_Database` metric shows that the database layer has an issue or a failure. This alarm indicates that SAP NetWeaver had issues connecting to or communicating with its database.



Since the database is a key resource for SAP NetWeaver, you may get many related alarms when the database has an issue or failure. In the following example, the `sap_alerts_FrontendResponseTime` and `sap_alerts_LongRunners` metrics are initiated because the database is not available.



Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to the CloudWatch alarms. After the error conditions are fixed, the alarm is resolved when the alarms and logs are recovered. When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard.

In the following example, the SAP Availability issue is resolved.

Detected problems (1)					
Severity	Problem summary	Source	Start time	Status	
High	SAP: Availability	netweavercomponent-HE4-9da46bcb-f...	2022-12-09T18:56:40Z	Resolved	

SAP NetWeaver application availability issues

Description


Your SAP NetWeaver High Availability Enqueue replication stopped working.

Cause

You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.

Dashboard Components **Detected problems** Configuration history Log patterns Tags

Detected problems summary [Info](#) Last 7 days ▾



2 Problems

■ Resolved ■ Unresolved

Top recurrent problems [↗](#)

There are no recurrent problems

Detected problems (2) [↻](#)

Last 7 days ▾ < 1 > ⚙

Severity	Problem summary	Source	Start time	Status
High	SAP Performance: Response Time RFC	netweavercomponent-HE4-9da46bcb-f49c-...	2022-12-13T01:00:55Z	In progress
High	SAP: Availability	netweavercomponent-HE4-9da46bcb-f49c-...	2022-12-09T18:56:40Z	Resolved

In the following example, under **Problem summary**, High Availability Enqueue Replication is the issue.

Problem summary

Problem ID

p-e296f993-864d-4e92-8b6a-7507c954ad74

Severity

⚠ High

Problem summary

SAP Availability: Enqueue Replication

Resolution Method [Info](#)

-

Source

netweavercomponent-HE2-2b8c0d84-a867-42e6-a6fe-3841183533cb

First occurrence time

2022-11-17T20:31:53Z

Last recurrence time

-

Resolution time

Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.

Insight [Info](#)

An issue with your SAP enqueue replication (ERS) state has been detected. Check that your enqueue replication is working with SAP transactions, such as SMENQ or the `ensmon` command.

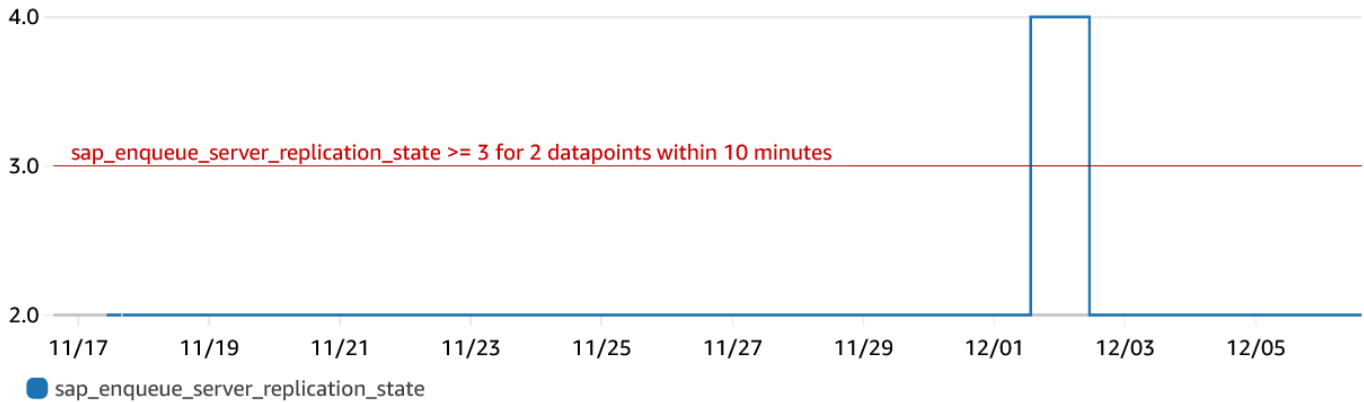
The following example shows the problem dashboard where you view logs and metrics which are grouped to help you isolate the causes of the error. The `sap_enqueue_server_replication_state` metric tracks the value over time. You can use

historical tracking to correlate when the metric initiated an error state or breached the alarm threshold.

SAP-NW-HE2 - sap_enqueue_server_replication_state



No unit



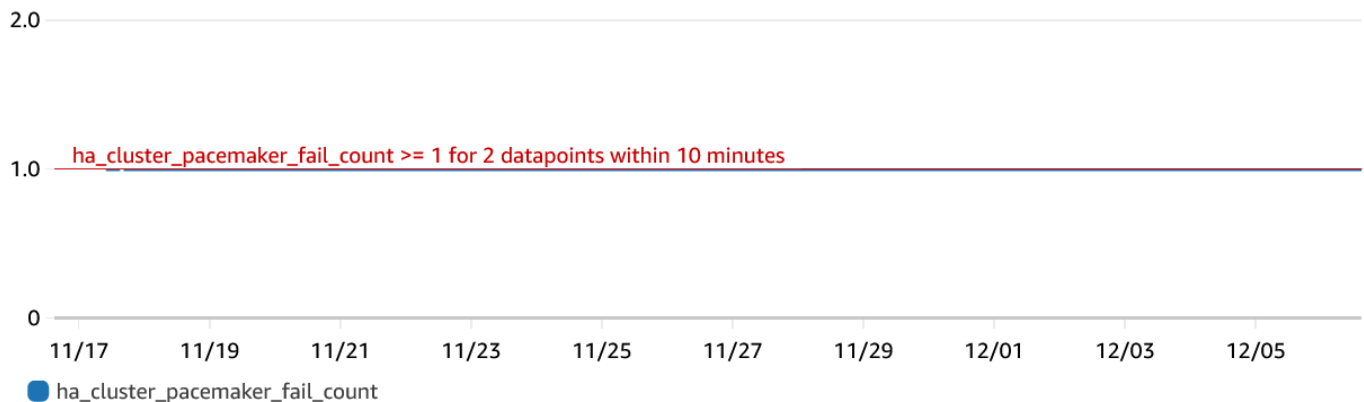
In the following example, the `ha_cluster_pacemaker_fail_count` metric shows that the high availability pacemaker cluster experienced a resource failure. The specific pacemaker resources that had a fail count greater than or equal to one are identified in the component dashboard.

EC2 instance group - SAP-NW-HE2

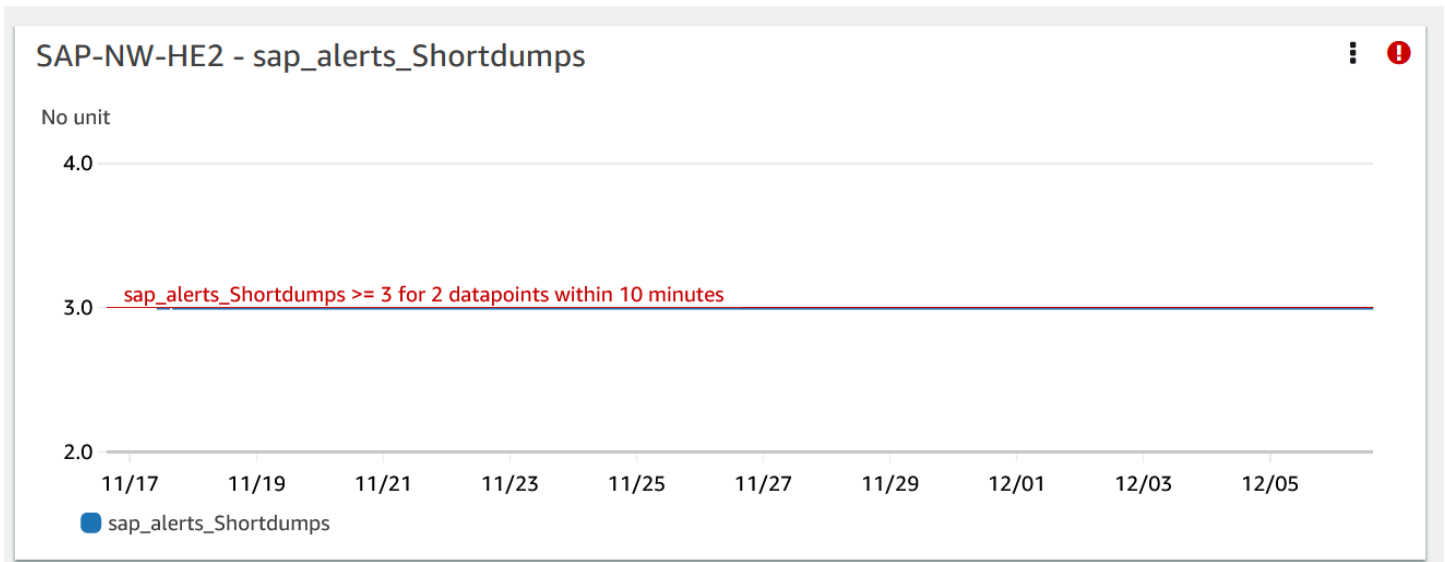
SAP-NW-HE2 - ha_cluster_pacemaker_fail_count



Count



The following example shows the `sap_alerts_Shortdumps` metric, which indicates that the SAP application performance was reduced when the problem was detected.



Logs

The log entries are helpful to get a better understanding of issues that occurred at the SAP NetWeaver layer when the problem was detected. The log group widget in the problem dashboard shows the specific time of the issue.

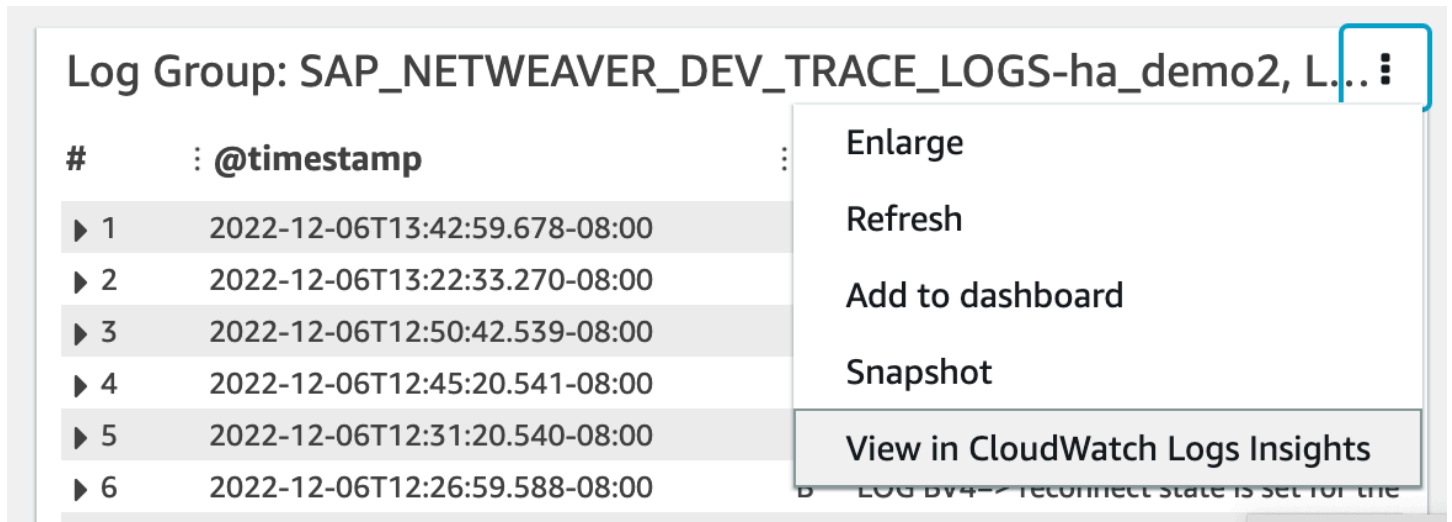
Log Group: SAP_NETWEAVER_DEV_TRACE_LOGS-ha_demo2, Log Type: SAP_NETWEAVER_DE... ⋮

#	: @timestamp	: @message
▶ 1	2022-11-30T19:46:15.481-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
▶ 2	2022-11-30T19:46:15.481-08:00	B ***LOG BY0=> Connect failed (connect timeout expired) (Socket connect timeout (60000 n
▶ 3	2022-11-30T19:46:15.481-08:00	A P4: Connect failed (connect timeout expired) (Socket connect timeout (60000 ms) {10.0.2f
▶ 4	2022-11-17T11:34:50.594-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
▶ 5	2022-11-17T10:28:50.144-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
▶ 6	2022-11-17T10:18:50.143-08:00	C SQLERRTEXT : Connect failed (connect timeout expired) (Socket connect timeout (60000 n
▶ 7	2022-11-17T10:18:50.143-08:00	B ***LOG BY0=> Connect failed (connect timeout expired) (Socket connect timeout (60000 n

< >

< >

To see detailed information about the logs, select the three vertical dots in the upper-right corner, and select **View in CloudWatch Logs Insights**.

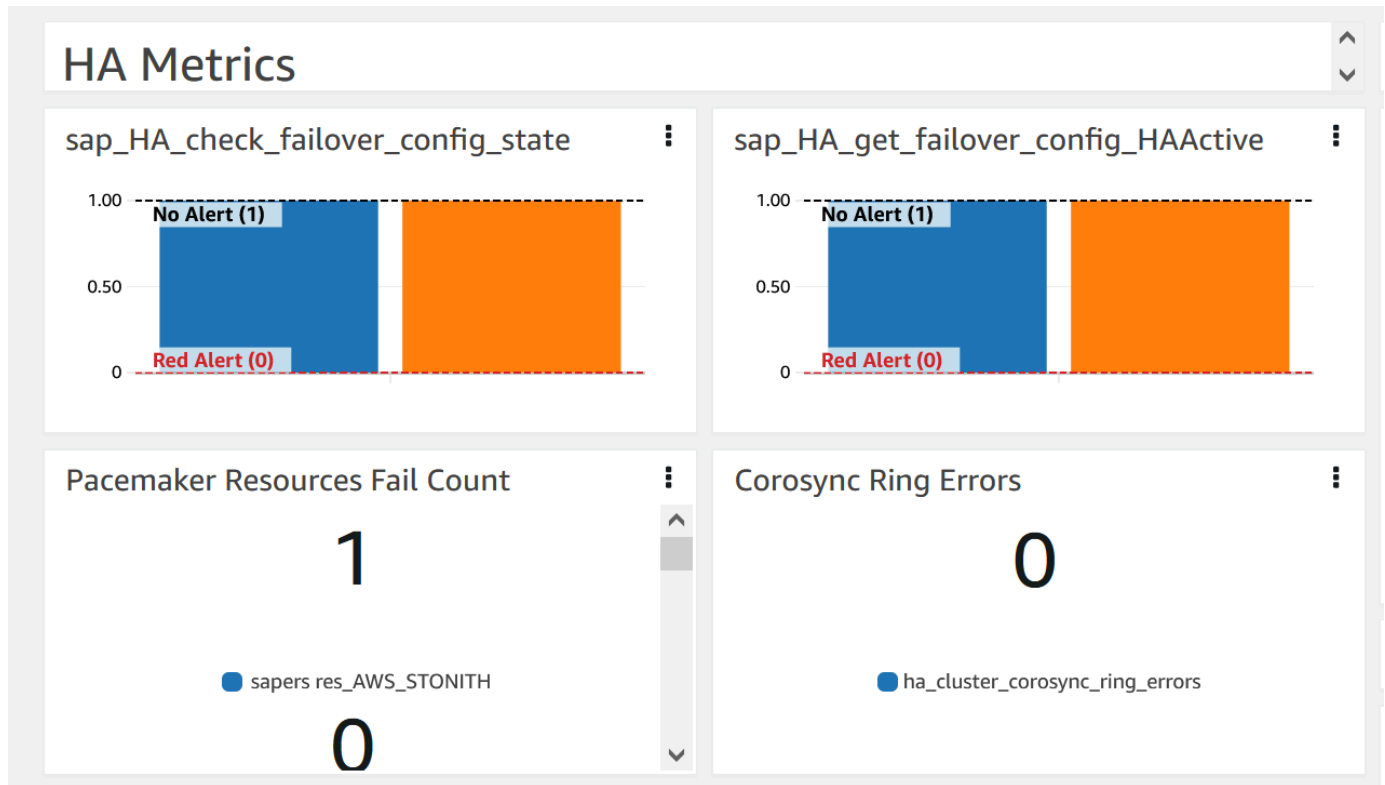


Use the following steps to get more information about the metrics and alarms displayed in the problem dashboard.

To get more information about metrics and alarms

1. Open the [CloudWatch console](#).
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Select the **Components** tab. Then, select the SAP NetWeaver component about which you want to get more information.

The following example shows the **HA Metrics** section with the `ha_cluster_pacemaker_fail_count` metric that was displayed in the problem dashboard.



Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to this problem.

For the `sap_alerts_Shortdumps` alarm, you must resolve the alert in the SAP NetWeaver system by using transaction code `RZ20 # R3Abap # Shortdumps` to navigate to the CCMS alert. For more information about CCMS alerts, see the [SAP website](#). Resolve all of the CCMS alerts in the Shortdumps tree. After all of the alerts are resolved in the SAP NetWeaver system, CloudWatch no longer reports the metric in an alarm state.

When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard. In the following example, the SAP Netweaver High Availability Enqueue Replication problem is resolved.

Severity	Problem summary	Source	Start time	Status
High	SAP Availability: Enqueue Replication	netweavercomponent-HE2-2b8c0...	2022-12-08T20:01:43Z	Resolved

Troubleshooting Application Insights for SAP NetWeaver

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitor metrics

Error returned: Component cannot have more than 60 monitored metrics.

Root cause: The current metric limit is 60 monitor metrics per component.

Resolution: Remove metrics that are not necessary to adhere to the limit.

SAP metrics do not appear on the dashboard after the onboarding process

Root cause: Component Dashboard uses a five minute metric period to aggregate the data points.

Resolution: All metrics should show up on the dashboard after five minutes.

SAP metrics and alarms don't appear on the dashboard

Use the following steps to identify why SAP metrics and alarms don't appear on the dashboard after the onboarding process.


To identify the issue with metrics and alarms

1. Open the [CloudWatch console](#).
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Choose the **Configuration history** tab.
4. If you see missing metrics datapoints, check for errors related to `prometheus-sap_host_exporter`.
5. If you don't find an error in the previous step, [Connect to your Linux instance](#). For High Availability deployments, connect to the primary cluster Amazon EC2 instance.
6. In your instance, verify that the exporter is running by using the following command. The default port is 9680. If you are using a different port, replace 9680 with the port you are using.

```
curl localhost:9680/metrics
```

If no data is returned, then the exporter failed to start.

7. To find the correct naming convention to use for `WORKLOAD_SHORT_NAME` in the next two steps, run the following command.

 **Note**

Application Insights adds a suffix, `WORKLOAD_SHORT_NAME`, to the service name depending on the workload that is running. The short names for NetWeaver Distributed, Standard, and High Availability deployments are `SAP_NWD`, `SAP_NWS`, and `SAP_NWH`.


```
sudo systemctl | grep exporter
```

8. To check for errors in the exporter service logs, run the following command:

```
sudo journalctl -e --unit=prometheus-sap_host_exporter_WORKLOAD_SHORT_NAME.service
```

9. To check for errors in the exporter manager service logs, run the following command:

```
sudo journalctl -e --unit=prometheus-  
sap_host_exporter_manager_WORKLOAD_SHORT_NAME.service
```

 **Note**

This service should be up and running at all times.

If this command does not return an error, continue to the next step.

10. To manually start the exporter, run the following command. Then, check the exporter output.

```
sudo /opt/aws/sap_host_exporter/sap_host_exporter
```

You can exit the exporter process after you check for errors.

Root cause: There are several possible causes for this issue. A common cause is that the exporter is not able to connect to one of the application server instances.

Resolution

Use the following steps to connect the exporter to the application server instances. You will verify that the SAP application instance is running and use SAPControl to connect to the instance.

To connect the exporter to the application server instances

1. In your Amazon EC2 instance, run the following command to verify that the SAP application is running.

```
sapcontrol -nr <App_InstNo> -function GetProcessList
```

2. You must establish a working SAPControl connection. If the SAPControl connection doesn't work, find the root cause of the issue on the relevant SAP application instance.
3. To manually start the exporter after you fix the SAP Control connection issue, run the following command:

```
sudo systemctl start prometheus-sap_host_exporter.service
```

4. If you can't resolve the SAPControl connection issue, use the following procedure as a temporary fix.
 - a. Open the [AWS Systems Manager console](#).
 - b. From the left navigation pane, choose **State Manager**.
 - c. Under **Associations** search for the SAP NetWeaver system's association.

```
Association Name: Equal: AWS-ApplicationInsights-SSMSAPHostExporterAssociationForCUSTOMSAPNW<SID>-1
```

- d. Select the **Association id**.
- e. Choose the **Parameters** tab and remove the application server number from **additionalArguments**.
- f. Choose **Apply Association Now**.

Note

This is a temporary fix. If updates are made to the component's monitoring configurations, the instance will be added back.

View and troubleshoot problems detected by Amazon CloudWatch Application Insights

The topics in this section provide detailed information about the detected problems and insights displayed by Application Insights. It also provides suggested resolutions for detected issues with your account or your configuration.

Troubleshooting topics

- [CloudWatch console overview](#)
- [Application Insights problem summary page](#)
- [CloudWatch agent merge conflict failures](#)
- [Alarms are not created](#)
- [Feedback](#)
- [Configuration errors](#)

CloudWatch console overview

An overview of problems that impact your monitored applications can be found under the CloudWatch Application Insights pane in the overview page of the [CloudWatch console](#). For more information, see [Get started with Amazon CloudWatch Application Insights](#).

The CloudWatch Application Insights overview pane displays the following:

- The severity of the problems detected: High/Medium/Low
- A short summary of the problem
- The problem source
- The time the problem started
- The resolution status of the problem
- The affected resource group

To view the details of a specific problem, under **Problem Summary**, select the description of the problem. A detailed dashboard displays insights into the problem and related metric anomalies and snippets of log errors. You can provide feedback on the relevance of the insight by selecting whether it is useful.

If a new resource is detected that is not configured, the problem summary description takes you to the **Edit configuration** wizard to configure your new resource. You can view or edit your resource group configuration by choosing **View/edit configuration** in the upper right-hand corner of the detailed dashboard.

To return to the overview, choose **Back to overview**, which is next to the CloudWatch Application Insights detailed dashboard header.

Application Insights problem summary page

Application Insights problem summary page

CloudWatch Application Insights provides the following information about detected problems on the problem summary page:

- A short summary of the problem
- The start time and date of the problem
- The problem severity: High/Medium/Low
- The status of the detected problem: In-progress/Resolved
- Insights: Automatically generated insights on the detected problem and possible root cause
- Feedback on insights: Feedback you have provided about the usefulness of the insights generated by CloudWatch Application Insights
- Related observations: A detailed view of the metric anomalies and error snippets of relevant logs related to the problem across various application components

CloudWatch agent merge conflict failures

CloudWatch Application Insights installs and configures the CloudWatch agent on customer instances. This includes creation of a CloudWatch agent configuration file with configurations for metrics or logs. A merge conflict can occur if a customer's instance already has a CloudWatch agent configuration file with different configurations defined for the same metrics or logs. To resolve the merge conflict, use the following steps:

1. Identify the CloudWatch agent configuration files on your system. For more information about the file locations, see [CloudWatch agent files and locations](#).

2. Remove the resource configurations that you want to use in Application Insights from the existing CloudWatch agent configuration file. If you want to only use Application Insights configurations, delete the existing CloudWatch agent configuration files.

Alarms are not created

For some metrics, Application Insights predicts the alarm threshold based on previous data points for the metric. To enable this prediction, the following criteria must be met.

- **Recent data points** – There must be a minimum of 100 data points from the last 24 hours. The data points don't need to be continuous and can be scattered throughout the 24 hour time frame.
- **Historical data** – There must be a minimum of 100 data points spanning the time frame from 15 days before the current date to 1 day before the current date. The data points don't need to be continuous and can be scattered throughout the 15 day time frame.

Note

For some metrics, Application Insights delays the creation of alarms until the preceding conditions are met. In this case, you get a configuration history event that the metric lacks sufficient data points to establish the alarm threshold.

Feedback

Feedback

You can provide feedback on the automatically generated insights on detected problems by designating them useful or not useful. Your feedback on the insights, along with your application diagnostics (metric anomalies and log exceptions), are used to improve the future detection of similar problems.

Configuration errors

CloudWatch Application Insights uses your configuration to create monitoring telemetries for the components. When Application Insights detects an issue with your account or your configuration, information is provided in the **Remarks** field of the Application summary about how to resolve the configuration issue for your application.

The following table shows suggested resolutions for specific remarks.

Remarks	Suggested resolution	Additional notes
The quota for CloudFormation has already been reached.	Application Insights creates one CloudFormation stack for each application to manage CloudWatch agent installation and configuration for all application components. By default, each AWS account can have 2000 stacks. See AWS CloudFormation Limits . To resolve this, raise the limit for CloudFormation stacks.	n/a
No SSM instance role on the following instances.	For Application Insights to be able to install and configure CloudWatch agent on application instances, AmazonSSMManagedInstanceCore and CloudWatchAgentServerPolicy policies must be attached to the instance role.	Application Insights calls the SSM DescribeInstanceInformation API to get the list of instances with SSM permission. After the role is attached to the instance, it takes time for SSM to include the instance in the DescribeInstanceInformation result. Until SSM includes the instance in the result, NO_SSM_INSTANCE_ROLE error remains present for the application.
New components may need configuration.	Application Insights detects that there are new components in the application Resource Group. To resolve this, configure the new components accordingly.	n/a

Logs and metrics supported by Amazon CloudWatch Application Insights

The following lists show the supported logs and metrics for Amazon CloudWatch Application Insights.

CloudWatch Application Insights supports the following logs:

- Microsoft Internet Information Services (IIS) logs
- Error log for SQL Server on EC2
- Custom .NET application logs, such as Log4Net
- Windows Event logs, including Windows logs (System, Application, and Security) and Applications and Services log
- Amazon CloudWatch Logs for AWS Lambda
- Error log and slow log for RDS MySQL, Aurora MySQL, and MySQL on EC2
- Postgresql log for PostgreSQL RDS and PostgreSQL on EC2
- Amazon CloudWatch Logs for AWS Step Functions
- Execution logs and access logs (JSON, CSV, and XML, but not CLF) for API Gateway REST API stages
- Prometheus JMX exporter logs (EMF)
- Alert logs and listener logs for Oracle on Amazon RDS and Oracle on Amazon EC2
- Container logs routing from Amazon ECS containers to CloudWatch using [awslogs log driver](#).
- Container logs routing from Amazon ECS containers to CloudWatch using [FireLens container log router](#).
- Container logs routing from Amazon EKS or Kubernetes running on Amazon EC2 to CloudWatch using [Fluent Bit or Fluentd log processor](#) with Container Insights.
- SAP HANA trace and error logs
- HA Pacemaker logs
- SAP ASE server logs
- SAP ASE backup server logs
- SAP ASE Replication server logs
- SAP ASE RMA agent logs

- SAP ASE Fault Manager logs
- SAP NetWeaver dev trace logs
- Process metrics for Windows processes using [proctstat plugin for CloudWatch agent](#)
- Public DNS query logs for hosted zone
- Amazon Route 53 Resolver DNS query logs

CloudWatch Application Insights supports the following log classes:

- **Standard** – Amazon CloudWatch Application Insights requires that log groups are configured with the [CloudWatch Logs Standard log class](#) to enable monitoring.

CloudWatch Application Insights supports metrics for the following application components:

- [Amazon Elastic Compute Cloud \(EC2\)](#)
 - [CloudWatch built-in metrics](#)
 - [CloudWatch agent metrics \(Windows server\)](#)
 - [CloudWatch agent process metrics \(Windows server\)](#)
 - [CloudWatch agent metrics \(Linux server\)](#)
- [Elastic Block Store \(EBS\)](#)
- [Amazon Elastic File System \(Amazon EFS\)](#)
- [Elastic Load Balancer \(ELB\)](#)
- [Application ELB](#)
- [Amazon EC2 Auto Scaling groups](#)
- [Amazon Simple Queue Server \(SQS\)](#)
- [Amazon Relational Database Service \(RDS\)](#)
 - [RDS Database instances](#)
 - [RDS Database clusters](#)
- [AWS Lambda function](#)
- [Amazon DynamoDB table](#)
- [Amazon S3 bucket](#)
- [AWS Step Functions](#)
 - [Execution-level](#)

- [Activity](#)
- [Lambda function](#)
- [Service integration](#)
- [Step Functions API](#)
- [API Gateway REST API stages](#)
- [SAP HANA](#)
- [SAP ASE](#)
- [SAP ASE High Availability on Amazon EC2](#)
- [SAP NetWeaver](#)
- [HA Cluster](#)
- [Java](#)
- [Amazon Elastic Container Service \(Amazon ECS\)](#)
 - [CloudWatch built-in metrics](#)
 - [Container Insights metrics](#)
 - [Container Insights Prometheus metrics](#)
- [Kubernetes on AWS](#)
 - [Container Insights metrics](#)
 - [Container Insights Prometheus metrics](#)
- [Amazon FSx](#)
- [Amazon VPC](#)
- [Amazon VPC NAT gateways](#)
- [Amazon Route 53 health check](#)
- [Amazon Route 53 hosted zone](#)
- [Amazon Route 53 Resolver endpoint](#)
- [AWS Network Firewall rule group](#)
- [AWS Network Firewall rule group association](#)
- [Metrics with data points requirements](#)
 - [AWS/ApplicationELB](#)
 - [AWS/AutoScaling](#)

- [AWS/EC2](#)
- [Elastic Block Store \(EBS\)](#)
- [AWS/ELB](#)
- [AWS/RDS](#)
- [AWS/Lambda](#)
- [AWS/SQS](#)
- [AWS/CWAgent](#)
- [AWS/DynamoDB](#)
- [AWS/S3](#)
- [AWS/States](#)
- [AWS/ApiGateway](#)
- [AWS/SNS](#)
- [Recommended metrics](#)
- [Performance Counter metrics](#)

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights supports the following metrics:

Metrics

- [CloudWatch built-in metrics](#)
- [CloudWatch agent metrics \(Windows server\)](#)
- [CloudWatch agent process metrics \(Windows server\)](#)
- [CloudWatch agent metrics \(Linux server\)](#)

CloudWatch built-in metrics

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch agent metrics (Windows server)

.NET CLR Exceptions # of Exceps Thrown

.NET CLR Exceptions # of Exceps Thrown/Sec

.NET CLR Exceptions # of Filters/sec

.NET CLR Exceptions # of Finallys/sec

.NET CLR Exceptions Throw to Catch Depth/sec

.NET CLR Interop # of CCWs

.NET CLR Interop # of Stubs

.NET CLR Interop # of TLB exports/sec

.NET CLR Interop # of TLB imports/sec

.NET CLR Interop # of marshaling

.NET CLR Jit % Time in Jit

.NET CLR Jit Standard Jit Failures

.NET CLR Loading % Time Loading

.NET CLR Loading Rate of Load Failures

.NET CLR LocksAndThreads Contention Rate/sec

.NET CLR LocksAndThreads Queue Length/sec

.NET CLR Memory # Total Committed Bytes

.NET CLR Memory % Time in GC

.NET CLR Networking 4.0.0.0 HttpRequest Average Queue Time

.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec

APP_POOL_WAS Total Worker Process Ping Failures

ASP.NET Application Restarts

ASP.NET Applications % Managed Processor Time (estimated)

ASP.NET Applications Errors Total/Sec

ASP.NET Applications Errors Unhandled During Execution/sec

ASP.NET Applications Requests in Application Queue

ASP.NET Applications Requests/Sec

ASP.NET Request Wait Time

ASP.NET Requests Queued

HTTP Service Request Queues CurrentQueueSize

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Memory Pages/sec

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk Queue Length

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Full Scans/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:General Statistics Processes blocked

SQLServer:General Statistics User Connections

SQLServer:Latches Average Latch Wait Time (ms)

SQLServer:Locks Average Wait Time (ms)

SQLServer:Locks Lock Timeouts/sec

SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec

SQLServer:Memory Manager Memory Grants Pending

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

W3SVC_W3WP File Cache Flushes

W3SVC_W3WP File Cache Misses

W3SVC_W3WP Requests/Sec

W3SVC_W3WP URI Cache Flushes

W3SVC_W3WP URI Cache Misses

Web Service Bytes Received/Sec

Web Service Bytes Sent/Sec

Web Service Connection attempts/sec

Web Service Current Connections

Web Service Get Requests/sec

Web Service Post Requests/sec

Bytes Received/sec

Normal Messages Queue Length/sec

Urgent Message Queue Length/sec

Reconnect Count

Unacknowledged Message Queue Length/sec

Messages Outstanding

Messages Sent/sec

Database Update Messages/sec

Update Messages/sec

Flushes/sec

Crypto Checkpoints Saved/sec

Crypto Checkpoints Restored/sec

Registry Checkpoints Restored/sec

Registry Checkpoints Saved/sec

Cluster API Calls/sec

Resource API Calls/sec

Cluster Handles/sec

Resource Handles/sec

CloudWatch agent process metrics (Windows server)

Process metrics are collected using the [CloudWatch agent procstat plugin](#). Only Amazon EC2 instances running Windows workloads support process metrics.

procstat cpu_time_system

procstat cpu_time_user

procstat cpu_usage

procstat memory_rss

procstat memory_vms

procstat read_bytes

procstat write_bytes

.procstat read_count

procstat write_count

CloudWatch agent metrics (Linux server)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used

disk_used_percent

diskio_io_time

diskio_iops_in_progress

diskio_read_bytes

diskio_read_time

diskio_reads

diskio_write_bytes

diskio_write_time

diskio_writes

mem_active

mem_available

mem_available_percent

mem_buffered

mem_cached

mem_free

mem_inactive

mem_used

mem_used_percent

net_bytes_rcv

net_bytes_sent

net_drop_in

net_drop_out

net_err_in

net_err_out

net_packets_rcv

net_packets_sent

netstat_tcp_close

netstat_tcp_close_wait

netstat_tcp_closing

netstat_tcp_established

netstat_tcp_fin_wait1

netstat_tcp_fin_wait2

netstat_tcp_last_ack

netstat_tcp_listen

netstat_tcp_none

netstat_tcp_syn_recv

netstat_tcp_syn_sent

netstat_tcp_time_wait

netstat_udp_socket

processes_blocked

processes_dead

processes_idle

processes_paging

processes_running

processes_sleeping

processes_stopped

processes_total

processes_total_threads

processes_wait

processes_zombies

swap_free

swap_used

swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights supports the following metrics:

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

Amazon Elastic File System (Amazon EFS)

CloudWatch Application Insights supports the following metrics:

BurstCreditBalance

PercentIOLimit

PermittedThroughput

MeteredIOBytes

TotalIOBytes

DataWriteIOBytes

DataReadIOBytes

MetadataIOBytes

ClientConnections

TimeSinceLastSync

StorageBytes

Throughput

PercentageOfPermittedThroughputUtilization

ThroughputIOPS

PercentThroughputDataReadIOByte

PercentThroughputDataWriteIOBytes

PercentageOfIOPSDataReadIOBytes

PercentageOfIOPSDataWriteIOBytes

AverageDataReadIOBytesSize

AverageDataWriteIOBytesSize

Elastic Load Balancer (ELB)

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

RequestCount

UnHealthyHostCount

Application ELB

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

Amazon EC2 Auto Scaling groups

CloudWatch Application Insights supports the following metrics:

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights supports the following metrics:

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights supports the following metrics:

Metrics

- [RDS Database instances](#)
- [RDS Database clusters](#)

RDS Database instances

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

ReadIOPS

ReadLatency

ReadThroughput

WriteIOPS

WriteLatency

WriteThroughput

RDS Database clusters

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization

CommitLatency

CommitThroughput

DDLlatency

DDLThroughput

DMLlatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

EngineUptime

FreeLocalStorage

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ResultSetCacheHitRatio

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

AWS Lambda function

CloudWatch Application Insights supports the following metrics:

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB table

CloudWatch Application Insights supports the following metrics:

SystemErrors

UserErrors

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 bucket

CloudWatch Application Insights supports the following metrics:

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS Step Functions

CloudWatch Application Insights supports the following metrics:

Metrics

- [Execution-level](#)
- [Activity](#)
- [Lambda function](#)
- [Service integration](#)
- [Step Functions API](#)

Execution-level

ExecutionTime

ExecutionThrottled

ExecutionsFailed

ExecutionsTimedOut

ExecutionsAborted

ExecutionsSucceeded

ExecutionsStarted

Activity

ActivityRunTime

ActivityScheduleTime

ActivityTime

ActivitiesFailed

ActivitiesHeartbeatTimedOut

ActivitiesTimedOut

ActivitiesScheduled

ActivitiesSucceeded

ActivitiesStarted

Lambda function

LambdaFunctionRunTime

LambdaFunctionScheduleTime

LambdaFunctionTime

LambdaFunctionsFailed

LambdaFunctionsTimedOut

LambdaFunctionsScheduled

LambdaFunctionsSucceeded

LambdaFunctionsStarted

Service integration

ServiceIntegrationRunTime

ServiceIntegrationScheduleTime

ServiceIntegrationTime

ServiceIntegrationsFailed

ServiceIntegrationsTimedOut

ServiceIntegrationsScheduled

ServiceIntegrationsSucceeded

ServiceIntegrationsStarted

Step Functions API

ThrottledEvents

ProvisionedBucketSize

ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API stages

CloudWatch Application Insights supports the following metrics:

4XXError

5XXError

IntegrationLatency

Latency

CacheHitCount

CacheMissCount

SAP HANA

Note

CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

CloudWatch Application Insights supports the following metrics:

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write_ratio_percent

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count

hanadb_failed_io_writes_count

hanadb_disk_data_unused_percent

hanadb_current_allocation_limit_used_percent

hanadb_table_allocation_limit_used_percent

hanadb_host_total_physical_memory_mb

hanadb_host_physical_memory_used_mb

hanadb_host_physical_memory_free_mb

hanadb_swap_memory_free_mb

hanadb_swap_memory_used_mb

hanadb_host_allocation_limit_mb

hanadb_host_total_memory_used_mb

hanadb_host_total_peak_memory_used_mb

hanadb_host_total_allocation_limit_mb

hanadb_host_code_size_mb

hanadb_host_shared_memory_allocation_mb

hanadb_cpu_usage_percent

hanadb_cpu_user_percent

hanadb_cpu_system_percent

hanadb_cpu_waitio_percent

hanadb_cpu_busy_percent

hanadb_cpu_idle_percent

hanadb_long_delta_merge_count

hanadb_unsuccessful_delta_merge_count

hanadb_successful_delta_merge_count

hanadb_row_store_allocated_size_mb

hanadb_row_store_free_size_mb

hanadb_row_store_used_size_mb

hanadb_temporary_tables_count

hanadb_large_non_compressed_tables_count

hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds

hanadb_average_commit_time_milliseconds

hanadb_suspended_sql_statements_count

hanadb_plan_cache_hit_ratio_percent

hanadb_plan_cache_lookup_count

hanadb_plan_cache_hit_count

hanadb_plan_cache_total_execution_microseconds

hanadb_plan_cache_cursor_duration_microseconds

hanadb_plan_cache_preparation_microseconds

hanadb_plan_cache_evicted_count

hanadb_plan_cache_evicted_microseconds

hanadb_plan_cache_evicted_preparation_count

hanadb_plan_cache_evicted_execution_count

hanadb_plan_cache_evicted_preparation_microseconds

hanadb_plan_cache_evicted_cursor_duration_microseconds

hanadb_plan_cache_evicted_total_execution_microseconds

hanadb_plan_cache_evicted_plan_size_mb

hanadb_plan_cache_count

hanadb_plan_cache_preparation_count

hanadb_plan_cache_execution_count

hanadb_network_collision_rate

hanadb_network_receive_rate

hanadb_network_transmit_rate

hanadb_network_packet_receive_rate

hanadb_network_packet_transmit_rate

hanadb_network_transmit_error_rate

hanadb_network_receive_error_rate

hanadb_time_until_license_expires_days

hanadb_is_license_valid_status

hanadb_local_running_connections_count

hanadb_local_idle_connections_count

hanadb_remote_running_connections_count

hanadb_remote_idle_connections_count

hanadb_last_full_data_backup_age_days

hanadb_last_data_backup_age_days

hanadb_last_log_backup_age_hours

hanadb_failed_data_backup_past_7_days_count

hanadb_failed_log_backup_past_7_days_count

hanadb_oldest_backup_in_catalog_age_days

hanadb_backup_catalog_size_mb

hanadb_hsr_replication_status

hanadb_hsr_log_shipping_delay_seconds

hanadb_hsr_secondary_failover_count

hanadb_hsr_secondary_reconnect_count

hanadb_hsr_async_buffer_used_mb

hanadb_hsr_secondary_active_status

hanadb_handle_count

hanadb_ping_time_milliseconds

hanadb_connection_count

hanadb_internal_connection_count

hanadb_external_connection_count

hanadb_idle_connection_count

hanadb_transaction_count

hanadb_internal_transaction_count

hanadb_external_transaction_count

hanadb_user_transaction_count

hanadb_blocked_transaction_count

hanadb_statement_count

hanadb_active_commit_id_range_count

hanadb_mvcc_version_count

hanadb_pending_session_count

hanadb_record_lock_count

hanadb_read_count

hanadb_write_count

hanadb_merge_count

hanadb_unload_count

hanadb_active_thread_count

hanadb_waiting_thread_count

hanadb_total_thread_count

hanadb_active_sql_executor_count

hanadb_waiting_sql_executor_count

hanadb_total_sql_executor_count

hanadb_data_write_size_mb

hanadb_data_write_time_milliseconds

hanadb_log_write_size_mb

hanadb_log_write_time_milliseconds

hanadb_data_read_size_mb

hanadb_data_read_time_milliseconds

hanadb_log_read_size_mb

hanadb_log_read_time_milliseconds

hanadb_data_backup_write_size_mb

hanadb_data_backup_write_time_milliseconds

hanadb_log_backup_write_size_mb

hanadb_log_backup_write_time_milliseconds

hanadb_mutex_collision_count

hanadb_read_write_lock_collision_count

hanadb_admission_control_admit_count

hanadb_admission_control_reject_count

hanadb_admission_control_queue_size_mb

hanadb_admission_control_wait_time_milliseconds

SAP ASE

CloudWatch Application Insights supports the following metrics:

asedb_database_availability

asedb_trunc_log_on_chkpt_enabled

asedb_last_db_backup_age_in_days

asedb_last_transaction_log_backup_age_in_hours

asedb_suspected_database

asedb_db_space_usage_percent

asedb_db_log_space_usage_percent

asedb_locked_login

asedb_has_mixed_log_and_data

asedb_runtime_for_open_transactions

asedb_data_cache_hit_ratio

asedb_data_cache_usage

asedb_sql_cache_hit_ratio

asedb_cache_usage

asedb_run_queue_length

asedb_number_of_rollbacks

asedb_number_of_commits

asedb_number_of_transactions

asedb_outstanding_disk_io

asedb_percent_io_busy

asedb_percent_system_busy

asedb_percent_locks_active

asedb_scheduled_jobs_failed_percent

asedb_user_connections_percent

asedb_query_logical_reads

asedb_query_physical_reads

asedb_query_cpu_time

asedb_query_memory_usage

SAP ASE High Availability on Amazon EC2

CloudWatch Application Insights supports the following metrics:

asedb_ha_replication_state

asedb_ha_replication_mode

asedb_ha_replication_latency_in_minutes

SAP NetWeaver

CloudWatch Application Insights supports the following metrics:

Metric	Description
sap_alerts_ResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>ResponseTime.
sap_alerts_ResponseTimeDialog	The SAP response time dialog alert from CCMS (RZ20)>R3Services>Dialog> ResponseTimeDialog.
sap_alerts_ResponseTimeDialogRFC	The SAP response time alert from CCMS (RZ20)>R3Services> Dialog>ResponseTimeDialogRFC.
sap_alerts_DBRequestTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>DBRequestTime.
sap_alerts_FrontendResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services > Dialog>FrontEndResponseTime.
sap_alerts_Database	The SAP system has logged database-related errors. Alert from SM21 or CCMS (RZ20)>R3 Syslog>Database.
sap_alerts_QueueTime	The SAP queue time alert from CCMS (RZ20)>R3Services>Dialog>QueueTime.
sap_alerts_AbortedJobs	Failed background jobs in SAP system. Alert from (RZ20)>R3Services > Background>AbortedJobs.

Metric	Description
sap_alerts_BasisSystem	SAP system logged system-level errors. Alert from SM21 or CCMS (RZ20)>R3Syslog>BasisSystem.
sap_alerts_Security	The SAP system logged security-related messages. Alert from SM21 or CCMS (RZ20)>R3Syslog>Security.
sap_alerts_System	The SAP system logged security or audit-related messages. Alert from SM21 or CCMS (RZ20)>Security>System.
sap_alerts_LongRunners	There are long running programs in your SAP system. Alert from CCMS (RZ20)>R3Services > Dialog>LongRunners.
sap_alerts_SqlError	There are SAP database client layer error logs. Alert from CCMS(RZ20)>DatabaseClient>AbapSql>SqlError.
sap_alerts_State	State alert from CCMS (RZ20)>OS Collector >State.
sap_alerts_Shortdumps	Shortdumps alert from ST22 and CCMS (RZ20)>R3Abap>Shortdumps.
sap_alerts_Availability	Availability alert for SAP application server instance from SM21, SM50, SM51, SM66, and CCMS (RZ20)>InstanceAsTask>Availability.
sap_dispatcher_queue_high	The SAPControl Web Service function GetQueueStatistic provides the dispatcher queue high count.
sap_dispatcher_queue_max	The SAPControl Web Service function GetQueueStatistic provides the dispatcher queue max count.

Metric	Description
sap_dispatcher_queue_now	The SAPControl Web Service function <code>GetQueueStatistic</code> provides the dispatcher queue now count.
sap_dispatcher_queue_reads	The SAPControl Web Service function <code>GetQueueStatistic</code> provides the dispatcher queue reads count.
sap_dispatcher_queue_writes	The SAPControl Web Service function <code>GetQueueStatistic</code> provides the dispatcher queue writes count.
sap_enqueue_server_arguments_high	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue arguments high.
sap_enqueue_server_arguments_max	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue arguments max.
sap_enqueue_server_arguments_now	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue arguments now.
sap_enqueue_server_arguments_state	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue arguments state.
sap_enqueue_server_backup_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue backup requests.
sap_enqueue_server_cleanup_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue cleanup requests.

Metric	Description
sap_enqueue_server_dequeue_all_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the dequeue all requests.
sap_enqueue_server_dequeue_errors	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the dequeue errors.
sap_enqueue_server_dequeue_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the dequeue requests.
sap_enqueue_server_enqueue_errors	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue errors.
sap_enqueue_server_enqueue_rejects	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue rejects.
sap_enqueue_server_enqueue_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue requests.
sap_enqueue_server_lock_time	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue lock time.
sap_enqueue_server_lock_wait_time	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue lock wait time.
sap_enqueue_server_locks_high	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue locks high.

Metric	Description
sap_enqueue_server_locks_max	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue locks max.
sap_enqueue_server_locks_now	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue locks now.
sap_enqueue_server_locks_state	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue locks state.
sap_enqueue_server_owner_high	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue owner high.
sap_enqueue_server_owner_max	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue owner max.
sap_enqueue_server_owner_now	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue owner now.
sap_enqueue_server_owner_state	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue owner state.
sap_enqueue_server_replication_state	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue replication state status.
sap_enqueue_server_reporting_requests	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the reporting requests status.

Metric	Description
sap_enqueue_server_server_time	The SAPControl Web Service function <code>EnqGetStatistic</code> provides the enqueue server time.
sap_HA_check_failover_config_state	The SAPControl Web Service function <code>HACheckFailoverConfig</code> provides the SAP High Availability status.
sap_HA_get_failover_config_HAActive	The SAPControl Web Service function <code>HAGetFailoverConfig</code> provides the SAP High Availability Cluster configuration and status.
sap_start_service_processes	The SAPControl Web Service function <code>GetProcessList</code> provides the <code>disp+work</code> , <code>IGS</code> , <code>gwr</code> , <code>icman</code> , message server, and enqueue server processes status.

HA Cluster

CloudWatch Application Insights supports the following metrics:

`ha_cluster_pacemaker_stonith_enabled`

`ha_cluster_corosync_quorate`

`hanadb_webdispatcher_service_started_status`

`ha_cluster_pacemaker_nodes`

`ha_cluster_corosync_ring_errors`

`ha_cluster_pacemaker_fail_count`

Java

CloudWatch Application Insights supports the following metrics:

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_threading_daemonthreadcount

java_lang_classloading_loadedclasscount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights supports the following metrics:

Metrics

- [CloudWatch built-in metrics](#)

- [Container Insights metrics](#)
- [Container Insights Prometheus metrics](#)

CloudWatch built-in metrics

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights metrics

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity

instance_memory_utilization

instance_memory_working_set

instance_network_total_bytes

instance_number_of_running_tasks

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Kubernetes on AWS

CloudWatch Application Insights supports the following metrics:

Metrics

- [Container Insights metrics](#)
- [Container Insights Prometheus metrics](#)

Container Insights metrics

cluster_failed_node_count

cluster_node_count

namespace_number_of_running_pods

node_cpu_limit

node_cpu_reserved_capacity

node_cpu_usage_total

node_cpu_utilization

node_filesystem_utilization

node_memory_limit

node_memory_reserved_capacity

node_memory_utilization

node_memory_working_set

node_network_total_bytes

node_number_of_running_containers

node_number_of_running_pods

pod_cpu_reserved_capacity

pod_cpu_utilization

pod_cpu_utilization_over_pod_limit

pod_memory_reserved_capacity

pod_memory_utilization

pod_memory_utilization_over_pod_limit

pod_network_rx_bytes

pod_network_tx_bytes

service_number_of_running_pods

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights supports the following metrics:

DataReadBytes

DataWriteBytes

DataReadOperations

DataWriteOperations

MetadataOperations

FreeStorageCapacity

FreeDataStorageCapacity

LogicalDiskUsage

PhysicalDiskUsage

Amazon VPC

CloudWatch Application Insights supports the following metrics:

NetworkAddressUsage

NetworkAddressUsagePeered

VPCFirewallQueryVolume

Amazon VPC NAT gateways

CloudWatch Application Insights supports the following metrics:

ErrorPortAllocation

IdleTimeoutCount

Amazon Route 53 health check

CloudWatch Application Insights supports the following metrics:

ChildHealthCheckHealthyCount

ConnectionTime

HealthCheckPercentageHealthy

HealthCheckStatus

SSLHandshakeTime

TimeToFirstByte

Amazon Route 53 hosted zone

CloudWatch Application Insights supports the following metrics:

DNSQueries

DNSSECInternalFailure

DNSSECKeySigningKeysNeedingAction

DNSSECKeySigningKeyMaxNeedingActionAge

DNSSECKeySigningKeyAge

Amazon Route 53 Resolver endpoint

CloudWatch Application Insights supports the following metrics:

EndpointHealthyENICount

EndpointUnHealthyENICount

InboundQueryVolume

OutboundQueryVolume

OutboundQueryAggregateVolume

AWS Network Firewall rule group

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupQueryVolume

AWS Network Firewall rule group association

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupVpcQueryVolume

Metrics with data points requirements

For metrics without an obvious default threshold to alarm on, Application Insights waits until the metric has enough data points to predict a reasonable threshold to alarm on. The metric data points requirement that CloudWatch Application Insights checks before an alarm is created are:

- The metric has at least 100 data points from the past 15 to the past 2 days.
- The metric has at least 100 data points from the last day.

The following metrics follow these data points requirements. Note that CloudWatch agent metrics require up to one hour to create alarms.

Metrics

- [AWS/ApplicationELB](#)
- [AWS/AutoScaling](#)
- [AWS/EC2](#)
- [Elastic Block Store \(EBS\)](#)
- [AWS/ELB](#)
- [AWS/RDS](#)
- [AWS/Lambda](#)
- [AWS/SQS](#)
- [AWS/CWAgent](#)
- [AWS/DynamoDB](#)
- [AWS/S3](#)
- [AWS/States](#)
- [AWS/ApiGateway](#)
- [AWS/SNS](#)

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS/AutoScaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

AWS/RDS

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

CPUCreditBalance

CommitLatency

CommitThroughput

DDLlatency

DDLThroughput

DMLlatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

DiskQueueDepth

EngineUptime

FreeLocalStorage

FreeStorageSpace

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ReadIOPS

ReadThroughput

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPS

VolumeWriteIOPS

WriteIOPS

WriteThroughput

AWS/Lambda

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:Database Replica File Bytes Received/sec

SQLServer:Database Replica Log Bytes Received/sec

SQLServer:Database Replica Log remaining for undo

SQLServer:Database Replica Log Send Queue

SQLServer:Database Replica Mirrored Write Transaction/sec

SQLServer:Database Replica Recovery Queue

SQLServer:Database Replica Redo Bytes Remaining

SQLServer:Database Replica Redone Bytes/sec

SQLServer:Database Replica Total Log requiring undo

SQLServer:Database Replica Transaction Delay

SQLServer:General Statistics Processes blocked

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

AWS/DynamoDB

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

AWS/S3

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectScannedBytes

SelectReturnedBytes

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS/States

ActivitiesScheduled

ActivitiesStarted

ActivitiesSucceeded

ActivityScheduleTime

ActivityRuntime

ActivityTime

LambdaFunctionsScheduled

LambdaFunctionsStarted

LambdaFunctionsSucceeded

LambdaFunctionScheduleTime

LambdaFunctionRuntime

LambdaFunctionTime

ServiceIntegrationsScheduled

ServiceIntegrationsStarted

ServiceIntegrationsSucceeded

ServiceIntegrationScheduleTime

ServiceIntegrationRuntime

ServiceIntegrationTime

ProvisionedRefillRate

ProvisionedBucketSize

ConsumedCapacity

ThrottledEvents

AWS/ApiGateway

4XXError

IntegrationLatency

Latency

DataProcessed

CacheHitCount

CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered

NumberOfMessagesPublished

NumberOfNotificationsFailed

NumberOfNotificationsFilteredOut

NumberOfNotificationsFilteredOut-InvalidAttributes

NumberOfNotificationsFilteredOut-NoMessageAttributes

NumberOfNotificationsRedrivenToDlq

NumberOfNotificationsFailedToRedriveToDlq

SMSSuccessRate

Recommended metrics

The following table lists the recommended metrics for each component type.

Component type	Workload type	Recommended metric
EC2 instance (Windows servers)	Default/Custom	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Active Directory	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Database ==> Instances Database Cache % Hit DirectoryServices DRA Pending Replication Operations DirectoryServices DRA Pending Replication Synchronizations

Component type	Workload type	Recommended metric
		DNS Recursive Query Failure/sec LogicalDisk Avg. Disk Queue Length
	Java Application	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
	Microsoft IIS/.NET Web Front-End	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts

Component type	Workload type	Recommended metric
	Microsoft SQL Server Database Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Paging File % Usage System Processor Queue Length Network Interface Bytes Total/Sec PhysicalDisk % Disk Time SQLServer:Buffer Manager Buffer Cache Hit ratio SQLServer:Buffer Manager Page Life Expectancy SQLServer:General Statistics Processes Blocked SQLServer:General Statistics User Connections SQLServer:Locks Number of Deadlocks/Sec SQLServer:SQL Statistics Batch Requests/Sec

Component type	Workload type	Recommended metric
	MySQL	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	.NET workerpool/Mid-Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC

Component type	Workload type	Recommended metric
	.NET Core Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes
	Oracle	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Postgres	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes

Component type	Workload type	Recommended metric
	SharePoint	<p>CPUUtilization</p> <p>StatusCheckFailed</p> <p>Processor % Processor Time</p> <p>Memory % Committed Bytes In Use</p> <p>Memory Available Mbytes</p> <p>ASP.NET Applications Cache API trims</p> <p>ASP.NET Requests Rejected</p> <p>ASP.NET Worker Process Restarts</p> <p>Memory Pages/sec</p> <p>SharePoint Publishing Cache Publishing cache flushes / second</p> <p>SharePoint Foundation Executing Time/Page Request</p> <p>SharePoint Disk-Based Cache Total number of cache compactions</p> <p>SharePoint Disk-Based Cache Blob cache hit ratio</p> <p>SharePoint Disk-Based Cache Blob Cache fill ratio</p>

Component type	Workload type	Recommended metric
		SharePoint Disk-Based Cache Blob cache flushes / second ASP.NET Requests Queued ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk Avg. Disk sec/ Write LogicalDisk Avg. Disk sec/ Read Processor % Interrupt Time
EC2 instance (Linux servers)	Default/Custom	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freevirtualmemorysize
	.NET Core Tier or SQL Server Database Tier	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Oracle	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
	Postgres	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
EC2 instance group	SAP HANA multi-node or single node	<ul style="list-style-type: none"> • hanadb_server_startup_time_variation_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_ratio_percent • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

Component type	Workload type	Recommended metric
EBS volume	Any	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	Any	HTTPCode_Backend_4XX HTTPCode_Backend_5XX Latency SurgeQueueLength UnHealthyHostCount
Application ELB	Any	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount

Component type	Workload type	Recommended metric
RDS Database instance	Any	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS Database cluster	Any	CPUUtilization CommitLatency DatabaseConnections Deadlocks FreeableMemory NetworkThroughput VolumeBytesUsed
Lambda Function	Any	Duration Errors IteratorAge ProvisionedConcurrencySpilloverInvocations Throttles

Component type	Workload type	Recommended metric
SQS Queue	Any	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB table	Any	SystemErrors UserErrors ConsumedReadCapacityUnits ConsumedWriteCapacityUnits ReadThrottleEvents WriteThrottleEvents ConditionalCheckFailedRequests TransactionConflict

Component type	Workload type	Recommended metric
Amazon S3 bucket	Any	<p>If replication configuration with Replication Time Control (RTC) is enabled:</p> <ul style="list-style-type: none">ReplicationLatencyBytesPendingReplicationOperationsPendingReplication <p>If request metrics are turned on:</p> <ul style="list-style-type: none">5xxErrors4xxErrorsBytesDownloadedBytesUploaded

Component type	Workload type	Recommended metric
AWS Step Functions	Any	<p>General</p> <ul style="list-style-type: none"> • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity <p>If state machine type is EXPRESS or log group level is OFF</p> <ul style="list-style-type: none"> • ExecutionsFailed • ExecutionsTimedOut <p>If state machine has Lambda functions</p> <ul style="list-style-type: none"> • LambdaFunctionsFailed • LambdaFunctionsTimedOut <p>If state machine has activities</p> <ul style="list-style-type: none"> • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbeatTimedOut <p>If state machine has service integrations</p> <ul style="list-style-type: none"> • ServiceIntegrationsFailed

Component type	Workload type	Recommended metric
		<ul style="list-style-type: none">• ServiceIntegration sTimedOut
API Gateway REST API stage	Any	<ul style="list-style-type: none">• 4XXErrors• 5XXErrors• Latency

Component type	Workload type	Recommended metric
ECS Cluster	Any	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utilization (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only)

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only)

Component type	Workload type	Recommended metric
	Java Application	<p>CpuUtilized</p> <p>MemoryUtilized</p> <p>NetworkRxBytes</p> <p>NetworkTxBytes</p> <p>RunningTaskCount</p> <p>PendingTaskCount</p> <p>StorageReadBytes</p> <p>StorageWriteBytes</p> <p>CPUReservation (EC2 Launch Type only)</p> <p>CPUUtilization (EC2 Launch Type only)</p> <p>MemoryReservation (EC2 Launch Type only)</p> <p>MemoryUtilization (EC2 Launch Type only)</p> <p>GPUReservation (EC2 Launch Type only)</p> <p>instance_cpu_utilization (EC2 Launch Type only)</p> <p>instance_filesystem_utilization (EC2 Launch Type only)</p> <p>instance_memory_utilization (EC2 Launch Type only)</p>

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only) java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
ECS Service	Any	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
EKS Cluster	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_h eapmemoryusage_used java_lang_memory_h eapmemoryusage_committed java_lang_operatingsystem_f reephysicalmemorysize java_lang_operatingsystem_f reeswapspacesize

Component type	Workload type	Recommended metric
Kubernetes Cluster on EC2	Any	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

The following table lists the recommended processes and process metrics for each component type. CloudWatch Application Insights does not recommend process monitoring for processes that do not run on an instance.

Component type	Workload type	Recommended process	Recommended metric
EC2 instance (Windows servers)	Microsoft IIS/.NET Web Front-End	w3wp	procstat cpu_usage , procstat memory_iss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	Microsoft SQL Server Database Tier	SQLAgent	procstat cpu_usage ,

Component type	Workload type	Recommended process	Recommended metric
			procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlservr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlwriter	procstat cpu_usage , procstat memory_rss

Component type	Workload type	Recommended process	Recommended metric
		Reporting ServicesService	procstat cpu_usage , procstat memory_rss
		MsDtsServr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		Msmdsrv	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Component type	Workload type	Recommended process	Recommended metric
	.NET workerpool/ Mid-Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	.NET Core Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Performance Counter metrics

Performance Counter metrics are recommended for instances only when the corresponding Performance Counter sets are installed on the Windows instances.

Performance Counter metric name	Performance Counter set name
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions
.NET CLR Exceptions Throw to Catch Depth/Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory

Performance Counter metric name	Performance Counter set name
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequest Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequest Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequest Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET Requests Rejected	ASP.NET
ASP.NET Worker Process Restarts	ASP.NET
ASP.NET Applications Cache API trims	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET

Performance Counter metric name	Performance Counter set name
Database ==> Instances Database Cache % Hit	Database ==> Instances
Database ==> Instances I/O Database Reads Average Latency	Database ==> Instances
Database ==> Instances I/O Database Reads/sec	Database ==> Instances
Database ==> Instances I/O Log Writes Average Latency	Database ==> Instances
DirectoryServices DRA Pending Replication Operations	DirectoryServices
DirectoryServices DRA Pending Replication Synchronizations	DirectoryServices
DirectoryServices LDAP Bind Time	DirectoryServices
DNS Recursive Queries/sec	DNS
DNS Recursive Query Failure/sec	DNS
DNS TCP Query Received/sec	DNS
DNS Total Query Received/sec	DNS
DNS Total Response Sent/sec	DNS
DNS UDP Query Received/sec	DNS
HTTP Service Request Queues CurrentQueueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk Avg. Disk sec/Write	LogicalDisk
LogicalDisk Avg. Disk sec/Read	LogicalDisk

Performance Counter metric name	Performance Counter set name
LogicalDisk Avg. Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	Memory
Memory Available Mbytes	Memory
Memory Pages/Sec	Memory
Memory Long-Term Average Standby Cache Lifetime (s)	Memory
Network Interface Bytes Total/Sec	Network Interface
Network Interface Bytes Received/sec	Network Interface
Network Interface Bytes Sent/sec	Network Interface
Network Interface Current Bandwidth	Network Interface
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg. Disk Queue Length	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	Processor
Processor % Interrupt Time	Processor

Performance Counter metric name	Performance Counter set name
Processor % Processor Time	Processor
Processor % User Time	Processor
SharePoint Disk-Based Cache Blob Cache fill ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache flushes / second	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache hit ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Total number of cache compactions	SharePoint Disk-Based Cache
SharePoint Foundation Executing Time/Page Request	SharePoint Foundation
SharePoint Publishing Cache Publishing cache flushes / second	SharePoint Publishing Cache
Security System-Wide Statistics Kerberos Authentications	Security System-Wide Statistics
Security System-Wide Statistics NTLM Authentications	Security System-Wide Statistics
SQLServer:Access Methods Forwarded Records/Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager

Performance Counter metric name	Performance Counter set name
SQLServer:Buffer Manager Page life Expectancy	SQLServer:Buffer Manager
SQLServer:Database Replica File Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log remaining for undo	SQLServer:Database Replica
SQLServer:Database Replica Log Send Queue	SQLServer:Database Replica
SQLServer:Database Replica Mirrored Write Transaction/sec	SQLServer:Database Replica
SQLServer:Database Replica Recovery Queue	SQLServer:Database Replica
SQLServer:Database Replica Redo Bytes Remaining	SQLServer:Database Replica
SQLServer:Database Replica Redone Bytes/sec	SQLServer:Database Replica
SQLServer:Database Replica Total Log requiring undo	SQLServer:Database Replica
SQLServer:Database Replica Transaction Delay	SQLServer:Database Replica
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks

Performance Counter metric name	Performance Counter set name
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compilations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	System
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	Web Service
Web Service Bytes Sent/Sec	Web Service
Web Service Connection Attempts/Sec	Web Service
Web Service Current Connections	Web Service

Performance Counter metric name	Performance Counter set name
Web Service Get Requests/Sec	Web Service
Web Service Post Requests/Sec	Web Service

Using the resource health view in the CloudWatch console

You can use the resource health view to automatically discover, manage, and visualize the health and performance of hosts across their applications in a single view. You can visualize the health of their hosts by a performance dimension such as CPU or memory, and slice and dice hundreds of hosts in a single view using filters. You can filter by tags or by use cases, such as hosts in the same Auto Scaling group or hosts that use the same load balancer,

Prerequisites

To make sure that you get the full benefit of the resource health view, check that you have the following prerequisites.

- To see the memory utilization of your hosts and use it as a filter, you must install the CloudWatch agent on your hosts and set it up to send a memory metric to CloudWatch in the default CWAgent namespace. On Linux and macOS instances, the CloudWatch agent must send the `mem_used_percent` metric. On Windows instances, the agent must send the `Memory % Committed Bytes In Use` metric. These metrics are included if you use the wizard to create the CloudWatch agent configuration file and select any of the pre-defined sets of metrics. Metrics collected by the CloudWatch agent are billed as custom metrics. For more information, see [Installing the CloudWatch agent](#).

When you use the CloudWatch agent to collect these memory metrics to use with the resource health view, you must include the following section in the CloudWatch agent configuration file. This section contains the default dimension settings and is created by default, so do not change any part of this section to anything different than what is shown in the following example.

```
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
```

```
},
```

- To view all the information available in the resource health view, you must be signed in to an account that has the following permissions. If you are signed on with fewer permissions, you can still use the resource health view but some performance data will not be visible.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
        "logs:Get*",
        "logs:Describe*",
        "sns:Get*",
        "sns:List*",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeRegions"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To view resource health in your account

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Infrastructure Monitoring, Resource Health**.

The resource health page appears, showing a square for each host in your account. Each square is colored based on the current status of that host, based on the setting for **Color by**. Host squares with an alarm symbol have one or more alarms currently in ALARM state.

You can see up to 500 hosts in a single view. If you have more hosts in your account, use the filter settings in step 6 of this procedure.

3. To change what criteria is used to show each host's health, choose a setting for **Color by**. You can choose **CPU Utilization**, **Memory Utilization**, or **Status check**. Memory utilization metrics are available only for hosts that are running the CloudWatch agent and have it configured to collect memory metrics and send them to the default CWAgent namespace. For more information, see [Collect metrics, logs, and traces with the CloudWatch agent](#).
4. To change the thresholds and the colors that are used for the health indicators in the grid, choose the gear icon above the grid.
5. To toggle whether to show alarms in the host grid, choose or clear **Show alarms across all metrics**.
6. To split the hosts in the map into groups, choose a grouping criteria for **Group by**.
7. To narrow the view to fewer hosts, choose a filter criteria for **Filter by**. You can filter by tags and by resource groupings such as Auto Scaling group, instance type, security group, and more.
8. To sort hosts, choose a sorting criteria for **Sort by**. You can sort by status check results, instance state, CPU or memory utilization, and the number of alarms that are in ALARM state.
9. To see more information about a host, choose the square that represents that host. A popup pane appears. To then dive deeper into information about that host, choose **View dashboard** or **View on list**.

Monitor across accounts and Regions

To enable unified monitoring across accounts, CloudWatch offers the following features:

- [CloudWatch cross-account observability](#)– facilitate observability within a single Region with the Observability Access Manager (OAM) service. You can link accounts and easily view metrics, logs, traces, and other telemetry between accounts. This helps you to unify observability in central monitoring accounts that view telemetry shared from source accounts, and operate on this shared telemetry as if it were native to the monitoring account.
- [Cross-account cross-Region CloudWatch console](#)– delivers a console experience that allows you to view dashboards, metrics, and alarms consoles of other accounts across Regions by toggling between accounts. After you set up the necessary permissions, you use an account selector integrated into the alarms, dashboards, and metrics consoles to view metrics, dashboards, and alarms in other accounts without having to log in and out of the accounts. By enabling this feature, you can also set up dashboards that contain cross-account cross-Region metrics for centralized visibility within an account.

These two features are complementary to each other and can be used independently or together. See the following table for a comparison of the two features. We recommend that you use CloudWatch cross-account observability for the richest cross-account observability and discovery experience within a Region for your metrics, logs, and traces.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
What is it?	Unified access to underlying telemetry and other observability resources across multiple accounts. After this is configured, observability resources are seamlessly viewable between accounts, eliminating the need for role assumptions. The central monitoring account gains direct access to the telemetry data and resources from source accounts,	A designated monitoring account assume a CrossAccountSharingRole defined in source accounts from the CloudWatch console. By assuming this role, the monitoring account can invoke operations such as dashboard viewing on behalf of source accounts, directly from its console.

	<u>CloudWatch cross-account observability</u>	<u>Cross-account cross-Region CloudWatch console</u>
	streamlining the monitoring and observability process.	
How does it work?	<p>A monitoring account, using the Observability Access Monitoring service, creates a <i>sink</i> and attaches a sink policy to it. The sink policy defines which resources they would like to view and which source accounts should share them. Then source accounts can create a link to the monitoring account sink, establishing what they actually want to share. After the link is created, the specified resources are visible in the monitoring account.</p>	<p>A source account initiates the configuration by setting up a CrossAccountSharingRole, allowing a monitoring account to run operations in the source account. Then, a monitoring account enables the cross-account cross-Region selector in the console by specifying the source account ID. This enables the monitoring account to be able to switch into the source account. When switching, the CloudWatch console checks for the existence of a service-linked role that allows CloudWatch to assume the CrossAccountSharingRole that was created in the source account.</p>
What telemetry is supported?	<ul style="list-style-type: none"> • Metrics • Traces • Logs • 	<ul style="list-style-type: none"> • Metrics • Traces

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
What functionality is supported?	<ul style="list-style-type: none"> • Dashboarding • Alarms • Metrics Insights • Anomaly Detection • CloudWatch Logs insights • Application Insights • Other functionalities. For more details see CloudWatch cross-account observability. 	<ul style="list-style-type: none"> • Console switching between accounts and Regions in metrics, alarms, and traces consoles. • Custom dashboards with metrics and alarms from other accounts and Regions <p>For more details, see Cross-account cross-Region CloudWatch console.</p>
How many accounts can I use it with?	A monitoring account can see resources from as many as 100,000 source accounts at the same time. A source account can share their resources with as many as five different monitoring accounts.	By using the cross-account cross-Region selector in the console, a monitoring account can switch to one other account at a time but there is no limit on the number of accounts that can be linked. When defining cross-account dashboards and alarms, many source accounts can be referenced.
Does it move telemetry data?	No. Resources are shared between accounts with the exception of copied traces.	No. An IAM policy is configured to allow embedded account switching for cross-account cross-Region resource visibility.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console
How much does it cost?	No extra charges for shared logs and metrics, and the first trace copy is free. For more information about pricing, see Amazon CloudWatch pricing .	No additional charges for cross-account or cross-Region actions.
Does it support observability across Regions?	No	Yes
Does it support programmatic access?	Yes. the AWS CLI, AWS Cloud Development Kit (AWS CDK), and APIs are supported.	No.
Does it support programmatic setup?	Yes	Yes
Does it support AWS Organizations?	Yes	Yes

Topics

- [CloudWatch cross-account observability](#)
- [Cross-account cross-Region CloudWatch console](#)

CloudWatch cross-account observability

With Amazon CloudWatch cross-account observability, you can monitor and troubleshoot applications that span multiple accounts within a Region. Seamlessly search, visualize, and analyze your metrics, logs, traces, Application Insights applications, and Internet Monitor monitors in any of the linked accounts without account boundaries.

Set up one or more AWS accounts as *monitoring accounts* and link them with multiple *source accounts*. A monitoring account is a central AWS account that can view and interact with

observability data generated from source accounts. A source account is an individual AWS account that generates observability data for the resources that reside in it. Source accounts share their observability data with the monitoring account. The shared observability data can include the following types of telemetry:

- Metrics in Amazon CloudWatch. You can choose to share the metrics from all namespaces with the monitoring account, or filter to a subset of namespaces.
- Log groups in Amazon CloudWatch Logs. You can choose to share all log groups with the monitoring account, or filter to a subset of log groups.
- Traces in AWS X-Ray
- Applications in Amazon CloudWatch Application Insights
- Monitors in CloudWatch Internet Monitor

To create links between monitoring accounts and source accounts, you can use the CloudWatch console. Alternatively, use the *Observability Access Manager* commands in the AWS CLI and API. For more information, see [Observability Access Manager API Reference](#).

A *sink* is a resource that represents an attachment point in a monitoring account. Source accounts can link to the sink to share observability data. Each account can have one sink per Region. Each sink is managed by the monitoring account where it is located. An *observability link* is a resource that represents the link established between a source account and a monitoring account. Links are managed by the source account.

For a video demonstration of setting up CloudWatch cross-account observability, see the following video.

The next topic explains how to set up CloudWatch cross-account observability in both monitoring accounts and source accounts. For information about the cross-account cross-Region CloudWatch dashboard, see [Cross-account cross-Region CloudWatch console](#).

Use Organizations for source accounts

There are two options for linking source accounts to your monitoring account. You can use one or both options.

- Use AWS Organizations to link accounts in an organization or organizational unit to the monitoring account.

- Connect individual AWS accounts to the monitoring account.

We recommend that you use Organizations so that new AWS accounts created later in the organization are automatically onboarded to cross-account observability as source accounts.

Details about linking monitoring accounts and source accounts

- Each monitoring account can be linked to as many as 100,000 source accounts.
- Each source account can share data with as many as five monitoring accounts.
- You can set up a single account as both a monitoring account and a source account. If you do, this account sends only the observability data from itself to the its linked monitoring account. It does not relay the data from its source accounts.
- A monitoring account specifies which telemetry types can be shared with it. A source account specifies which telemetry types it wants to share.
 - If there are more telemetry types selected in the *monitoring account* than in the source account, the accounts are linked. Only the data types that are selected in both accounts are shared.
 - If there are more telemetry types selected in the *source account* than in the monitoring account, the link creation fails and nothing is shared.
 - A metric name doesn't appear in the monitoring account console until that metric emits new data points after the link is created.
- To remove a link between accounts, do so from the source account.
- To delete a sink in a monitoring account, you must first remove all links to that sink the monitoring account.

Pricing

Cross-account observability in CloudWatch comes with no extra cost for logs and metrics, and the first trace copy is free. For more information about pricing, see [Amazon CloudWatch Pricing](#).

Contents

- [Link monitoring accounts with source accounts](#)
 - [Necessary permissions](#)
 - [Setup overview](#)
 - [Step 1: Set up a monitoring account](#)

- [Step 2: \(Optional\) Download an AWS CloudFormation template or URL](#)
- [Step 3: Link the source accounts](#)
 - [Use an AWS CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts](#)
 - [Use an AWS CloudFormation template to set up individual source accounts](#)
 - [Use a URL to set up individual source accounts](#)
- [Manage monitoring accounts and source accounts](#)
 - [Link more source accounts to an existing monitoring account](#)
 - [Remove the link between a monitoring account and source account](#)
 - [View information about a monitoring account](#)

Link monitoring accounts with source accounts

The topics in this section explain how to set up links between monitoring accounts and source accounts.

We recommend that you create a new AWS account to serve as the monitoring account for your organization.

Contents

- [Necessary permissions](#)
- [Setup overview](#)
- [Step 1: Set up a monitoring account](#)
- [Step 2: \(Optional\) Download an AWS CloudFormation template or URL](#)
- [Step 3: Link the source accounts](#)
 - [Use an AWS CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts](#)
 - [Use an AWS CloudFormation template to set up individual source accounts](#)
 - [Use a URL to set up individual source accounts](#)

Necessary permissions

To create links between a monitoring account and a source account, you must be signed in with certain permissions.

- **To set up a monitoring account** – You must have either full administrator access in the monitoring account, or you must sign in to that account with the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSinkModification",
      "Effect": "Allow",
      "Action": [
        "oam:CreateSink",
        "oam>DeleteSink",
        "oam:PutSinkPolicy",
        "oam:TagResource"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnly",
      "Effect": "Allow",
      "Action": ["oam:Get*", "oam:List*"],
      "Resource": "*"
    }
  ]
}
```

- **Source account, scoped to a specific monitoring account** – To create, update, and manage links for just one specified monitoring account, you must sign in to account with at least the following permissions. In this example, the monitoring account is 999999999999.

If the link isn't going to share all five resource types (metrics, logs, traces, Application Insights applications, and Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, or `internetmonitor:Link` as needed.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "oam:CreateLink",
        "oam:UpdateLink",
        "oam>DeleteLink",
```

```

        "oam:GetLink",
        "oam:TagResource"
    ],
    "Effect": "Allow",
    "Resource": "arn:*:oam:*:*:link/*"
},
{
    "Action": [
        "oam:CreateLink",
        "oam:UpdateLink"
    ],
    "Effect": "Allow",
    "Resource": "arn:*:oam:*:*:sink/*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": [
                "999999999999"
            ]
        }
    }
},
{
    "Action": "oam:ListLinks",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "cloudwatch:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "logs:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "xray:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "applicationinsights:Link",
    "Effect": "Allow",

```

```

        "Resource": "*"
      },
      {
        "Action": "internetmonitor:Link",
        "Effect": "Allow",
        "Resource": "*"
      }
    ]
  }
}

```

- **Source account, with permissions to link to any monitoring account** – To create a link to any existing monitoring account sink and share metrics, log groups, traces, Application Insights applications, and Internet Monitor monitors, you must sign in to the source account with full administrator permissions or sign in there with the following permissions

If the link isn't going to share all five resource types (metrics, logs, traces, Application Insights applications, and Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, or `internetmonitor:Link` as needed.

```

{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "oam:CreateLink",
      "oam:UpdateLink"
    ],
    "Resource": [
      "arn:aws:oam:*:*:link/*",
      "arn:aws:oam:*:*:sink/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "oam:List*",
      "oam:Get*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [

```

```
        "oam:DeleteLink",
        "oam:GetLink",
        "oam:TagResource"
    ],
    "Resource": "arn:aws:oam:*:*:link/*"
},
{
    "Action": "cloudwatch:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "xray:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "logs:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "applicationinsights:Link",
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": "internetmonitor:Link",
    "Effect": "Allow",
    "Resource": "*"
}
]
}
```

Setup overview

The following high-level steps show you how to set up CloudWatch cross-account observability.

Note

We recommend creating a new AWS account to use as your organization's monitoring account.

1. Set up a dedicated monitoring account.
2. (Optional) Download an AWS CloudFormation template or copy a URL to link source accounts.
3. Link source accounts to the monitoring account.

After completing these steps, you can use the monitoring account to view the observability data of the source accounts.

Step 1: Set up a monitoring account

Follow the steps in this section to set up an AWS account as a monitoring account for CloudWatch cross-account observability.

Prerequisites

- **If you're setting up accounts in an AWS Organizations organization as the source accounts** – Get the organization path or organization ID.
- **If you're not using Organizations for the source accounts** – Get the account IDs of the source accounts.

To set up an account as a monitoring account, you must have certain permissions. For more information, see [Necessary permissions](#).

To set up a monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Configure**.
5. For **Select data**, choose whether this monitoring account will be able to view **Logs, Metrics, Traces, Application Insights - Applications**, and **Internet Monitor - Monitors** data from the source accounts it is linked to.

6. For **List source accounts**, enter the source accounts that this monitoring account will view. To identify the source accounts, enter individual account IDs, organization paths, or organization IDs. If you enter an organization path or organization ID, this monitoring account is allowed to view observability data from all linked accounts in that organization.

Separate the entries in this list with commas.

 **Important**

When you enter an organization path, follow the exact format. The ou-id must end with a / (a slash character). For example: o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbb/

7. For **Define a label to identify your source account**, specify whether to use account names or email addresses to identify the source accounts when you use the monitoring account to view them.
8. Choose **Configure**.

 **Important**

The link between the monitoring and source accounts is not complete until you configure the source accounts. For more information, see the following sections.

Step 2: (Optional) Download an AWS CloudFormation template or URL

To link source accounts to a monitoring account, we recommend using an AWS CloudFormation template or a URL.

- **If you are linking an entire organization** – CloudWatch provides an AWS CloudFormation template.
- **If you are linking individual accounts** – Use either an AWS CloudFormation template or a URL that CloudWatch provides.

To use an AWS CloudFormation template, you must download it during these steps. After you link the monitoring account with at least one source account, the AWS CloudFormation template is no longer available to download.

To download an AWS CloudFormation template or copy a URL for linking source accounts to the monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Resources to link accounts**.
5. Do one of the following:
 - Choose **AWS organization** to get a template to use to link accounts in an organization to this monitoring account.
 - Choose **Any account** to get a template or URL for setting up individual accounts as source accounts.
6. Do one of the following:
 - If you chose **AWS organization**, choose **Download CloudFormation template**.
 - If you chose **Any account**, choose either **Download CloudFormation template** or **Copy URL**.
7. (Optional) Repeat steps 5-6 to download both the AWS CloudFormation template and the URL.

Step 3: Link the source accounts

Use the steps in these sections to link source accounts to a monitoring account.

To link monitoring accounts with source accounts, you must have certain permissions. For more information, see [Necessary permissions](#).

Use an AWS CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts

These steps assume that you already downloaded the necessary AWS CloudFormation template by performing the steps in [Step 2: \(Optional\) Download an AWS CloudFormation template or URL](#).

To use an AWS CloudFormation template to link accounts in an organization or organizational unit to the monitoring account

1. Sign in to the organization's management account.

2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. In the left navigation bar, choose **StackSets**.
4. Check that you are signed in to the Region that you want, then choose **Create StackSet**.
5. Choose **Next**.
6. Choose **Template is ready** and choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify StackSet details**, enter a name for the StackSet and choose **Next**.
10. For **Add stacks to stack set**, choose **Deploy new stacks**.
11. For **Deployment targets**, choose whether to deploy to the entire organization or to specified organizational units.
12. For **Specify regions**, choose which Regions to deploy CloudWatch cross-account observability to.
13. Choose **Next**.
14. On the **Review** page, confirm your selected options and choose **Submit**.
15. In the **Stack instances** tab, refresh the screen until you see that your stack instances have the status **CREATE_COMPLETE**.

Use an AWS CloudFormation template to set up individual source accounts

These steps assume that you already downloaded the necessary AWS CloudFormation template by performing the steps in [Step 2: \(Optional\) Download an AWS CloudFormation template or URL](#).

To use an AWS CloudFormation template to set up individual source accounts for CloudWatch cross-account observability

1. Sign in to the source account.
2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. In the left navigation bar, choose **Stacks**.
4. Check that you are signed in to the Region that you want, then choose **Create stack, With new resources (standard)**.
5. Choose **Next**.

6. Choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify stack details**, enter a name for the stack and choose **Next**.
10. On the **Configure stack options** page, choose **Next**.
11. On the **Review** page, choose **Submit**.
12. On the status page for your stack, refresh the screen until you see that your stack has the status **CREATE_COMPLETE**.
13. To use this same template to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-12.

Use a URL to set up individual source accounts

These steps assume that you already copied the necessary URL by performing the steps in [Step 2: \(Optional\) Download an AWS CloudFormation template or URL](#).

To use a URL to link individual source accounts to the monitoring account

1. Sign in to the account that you want to use as a source account.
2. Enter the URL that you copied from the monitoring account.

You see the CloudWatch settings page, with some information filled in.

3. For **Select data**, choose whether this source account will share **Logs**, **Metrics**, **Traces**, **Application Insights - Applications**, and **Internet Monitor - Monitors** data to this monitoring account.

For both **Logs** and **Metrics**, you can choose whether to share all resources or a subset with the monitoring account.

- a. (Optional) To share a subset of this account's log groups with the monitoring account, select **Logs** and choose **Filter Logs**. Then use the **Filter Logs** box to construct a query to find the log groups that you want to share. The query will use the term `LogGroupName` and one or more of the following operands.

- = and !=

- AND
- OR
- ^ indicates LIKE and !^ indicates NOT LIKE. These can be used only as prefix searches. Include a % at the end of the string that you want to search for and include.
- IN and NOT IN, using parentheses (())

The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are AND and OR. There isn't a limit on the number of other operands.

 **Tip**

Choose **View sample queries** to see the correct syntax for common query formats.

- b. (Optional) To share a subset of this account's metric namespaces with the monitoring account, select **Metrics** and choose **Filter Metrics**. Then use the **Filter Metrics** box to construct a query to find the metric namespaces that you want to share. Use the term Namespace and one or more of the following operands.
- = and !=
 - AND
 - OR
 - LIKE and NOT LIKE. These can be used only as prefix searches. Include a % at the end of the string that you want to search for and include.
 - IN and NOT IN, using parentheses (())

The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are AND and OR. There isn't a limit on the number of other operands.

 **Tip**

Choose **View sample queries** to see the correct syntax for common query formats.

4. Do not change the ARN in **Enter monitoring account configuration ARN**.

5. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account. Optionally, choose **Edit** to change it.
6. Choose **Link**.
7. Enter **Confirm** in the box and choose **Confirm**.
8. To use this same URL to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-7.

Manage monitoring accounts and source accounts

After you set up your monitoring accounts and source accounts, you can use the steps in these sections to manage them.

Contents

- [Link more source accounts to an existing monitoring account](#)
- [Remove the link between a monitoring account and source account](#)
- [View information about a monitoring account](#)

Link more source accounts to an existing monitoring account

Follow the steps in this section to add links from additional source accounts to an existing monitoring account.

Each source account can be linked to as many as five monitoring accounts. Each monitoring account can be linked to as many as 100,000 source accounts.

To manage a source account, you must have certain permissions. For more information, see [Necessary permissions](#).

To add more source accounts to a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage source accounts**.
5. Choose the **Configuration policy** tab.

6. In the **Configuration policy** box, add the new source account ID in the **Principal** line.

For example, suppose the **Principal** line is currently the following:

```
"Principal": {"AWS": ["111111111111", "222222222222"]}
```

To add 999999999999 as a third source account, edit the line to the following:

```
"Principal": {"AWS": ["111111111111", "222222222222", "999999999999"]}
```

7. Choose **Update**.
8. Choose the **Configuration details** tab.
9. Choose the copy icon that is next to the monitoring account's sink ARN.
10. Sign in to the account that you want to use as a new source account.
11. Paste the monitoring account's sink ARN that you copied in Step 9.

You see the CloudWatch settings page, with some information filled in.

12. For **Select data**, choose whether this source account will send **Logs**, **Metrics**, **Traces**, and **Application Insights - Applications** data to the monitoring accounts it is linked to.
13. Do not change the ARN in **Enter monitoring account configuration ARN**.
14. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account. Optionally, choose **Edit** to change it.
15. Choose **Link**.
16. Enter **Confirm** in the box and choose **Confirm**.

Remove the link between a monitoring account and source account

Follow the steps in this section to stop sending data from one source account to a monitoring account.

You must have the permissions required to manage a source account to complete this task. For more information, see [Necessary permissions](#).

To remove the link between a source account and a monitoring account

1. Sign in to the source account.

2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the left navigation pane, choose **Settings**.
4. By **Source account information**, choose **View monitoring accounts**.
5. Select the check box next to the monitoring account that you want to stop sharing data with.
6. Choose **Stop sharing data, Confirm**.
7. Sign in to the monitoring account.
8. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
9. Choose **Settings**.
10. By **Monitoring account information**, choose **View configuration**.
11. In the **Policy** box, delete the source account ID from the **Principal** line and choose **Update**.

View information about a monitoring account

Follow the steps in this section to view the cross-account settings for a monitoring account.

To manage a monitoring account, you must have certain permissions. For more information, see [Necessary permissions](#).

To manage a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage source accounts**.
5. To view the Observability Access Manager policy that enables this account to be a monitoring account, choose the **Configuration policy** tab.
6. To view the source accounts that are linked to this monitoring account, choose the **Linked source accounts** tab.
7. To view the monitoring account sink ARN, and the types of data that this monitoring account can view in linked source accounts, choose the **Linked source accounts** tab.

Cross-account cross-Region CloudWatch console

Note

We recommend that you use CloudWatch cross-account observability to get the richest cross-account observability and discovery experience for your metrics, logs, and traces within a Region. For more information, see [CloudWatch cross-account observability](#).

The cross-account, cross-Region CloudWatch console allows you to easily switch between different accounts and Region by using selectors in the console to view the dashboards, alarms, and metrics in other accounts and Regions. This feature also allows you to create cross-account, cross-Region dashboards which summarize your CloudWatch metrics from multiple AWS accounts and multiple Regions into a single dashboard, making them accessible without having to switch accounts or Regions.

Many organizations have their AWS resources deployed in multiple accounts, to provide billing and security boundaries. In this case, we recommend that you designate one or more of your accounts as your *monitoring accounts*, and build your cross-account cross-Region dashboards in these accounts. Cross-account cross-Region console functionality is integrated with AWS Organizations, to help you efficiently build your cross-account cross-Region dashboards.

The cross-account, cross-region CloudWatch console experience does not provide cross-account cross-Region visibility for logs. Additionally, it does not support the creation of alarms on metrics in other accounts or Regions from within a monitoring account.

Topics

- [Enabling cross-account cross-Region functionality in CloudWatch](#)
- [\(Optional\) Integrate with AWS Organizations](#)
- [Troubleshooting your CloudWatch cross-account setup](#)
- [Disabling and cleaning up after using cross-account](#)

Enabling cross-account cross-Region functionality in CloudWatch

To set up cross-account cross-Region functionality in your CloudWatch console, use the CloudWatch console to set up your sharing accounts and monitoring accounts.

Set up a sharing account

You must enable sharing in each account that will make data available to the monitoring account.

This will grant the read-only permissions that you choose in step 5 to all users that view a cross account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To enable your account to share CloudWatch data with other accounts

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. For **Share your CloudWatch data**, choose **Configure**.
4. For **Sharing**, choose **Specific accounts** and enter the IDs of the accounts that you want to share data with.

Any accounts that you specify here can view your account's CloudWatch data. Specify the IDs only of accounts that you know and trust.

5. For **Permissions**, specify how to share your data with one of the following options:
 - **Provide read-only access to your CloudWatch metrics, dashboards, and alarms.** This option enables the monitoring accounts to create cross-account dashboards that include widgets that contain CloudWatch data from your account.
 - **Include CloudWatch automatic dashboards.** If you select this option, users in the monitoring account can also view the information in this account's automatic dashboards. For more information, see [Getting started with Amazon CloudWatch](#).
 - **Include X-Ray read-only access for the X-Ray Trace Map.** If you select this option, users in the monitoring account can also view the X-Ray trace map and X-Ray trace information in this account. For more information, see [Using the X-Ray Trace Map](#).
 - **Full read-only access to everything in your account.** This option enables the accounts that you use for sharing to create cross-account dashboards that include widgets that contain CloudWatch data from your account. It also enables those accounts to look deeper into your account and view your account's data in the consoles of other AWS services.
6. Choose **Launch CloudFormation template**.

In the confirmation screen, type **Confirm**, and choose **Launch template**.

7. Select the **I acknowledge...** check box, and choose **Create stack**.

Sharing with an entire organization

Completing the preceding procedure creates an IAM role which enables your account to share data with one account. You can create or edit an IAM role that shares your data with all accounts in an organization. Do this only if you know and trust all accounts in the organization.

This will grant the read-only permissions listed in the policies shown in step 5 of the previous procedure to all users that view a cross-account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To share your CloudWatch account data with all accounts in an organization

1. If you haven't already, complete the preceding procedure to share your data with one AWS account.
2. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
3. In the navigation pane, choose **Roles**.
4. In the list of roles, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.

You see a policy like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. Change the policy to the following, replacing *org-id* with the ID of your organization.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Effect": "Allow",
    "Principal": {
      "AWS": "*"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:PrincipalOrgID": "org-id"
      }
    }
  }
]
```

7. Choose **Update Trust Policy**.

Set up a monitoring account

Enable each monitoring account if you want to view cross-account CloudWatch data.

When you complete the following procedure, CloudWatch creates a service-linked role that CloudWatch uses in the monitoring account to access data shared from your other accounts. This service-linked role is called **AWSServiceRoleForCloudWatchCrossAccount**. For more information, see [Using service-linked roles for CloudWatch](#).

To enable your account to view cross-account CloudWatch data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**, and then, in the **Cross-account cross-region** section, choose **Configure**.
3. Under the **View cross-account cross-region** section, choose **Enable**, and then select the **Show selector in the console** checkbox to enable an account selector to appear in the CloudWatch console when you're graphing a metric or creating an alarm.
4. Under **View cross-account cross-region**, choose one of the following options:
 - **Account Id Input**. This option prompts you to manually input an account ID each time that you want to switch accounts when you view cross-account data.
 - **AWS Organization account selector**. This option causes the accounts that you specified when you completed your cross-account integration with Organizations to appear. When

you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

To do this, you must have first used your organization management account to allow CloudWatch to see a list of accounts in your organization. For more information, see [\(Optional\) Integrate with AWS Organizations](#).

- **Custom account selector.** This option prompts you to enter a list of account IDs. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

You can also enter a label for each of these accounts to help you identify them when choosing accounts to view.

The account selector settings that a user makes here are retained only for that user, not for all other users in the monitoring account.

5. Choose **Enable**.

After you complete this setup, you can create cross-account dashboards. For more information, see [Cross-account cross-Region dashboards](#).

Cross-Region functionality

Cross-Region functionality is built in to this feature automatically. You do not need to take any extra steps to be able to display metrics from different Regions in a single account on the same graph or the same dashboard. Cross-Region functionality is not supported for alarms, so you can't create an alarm in one Region that watches a metric in a different Region.

(Optional) Integrate with AWS Organizations

If you want to integrate cross-account functionality with AWS Organizations, you must make a list of all accounts in the organization available to the monitoring accounts.

To enable cross-account CloudWatch functionality to access a list of all accounts in your organization

1. Sign in to your organization's management account.
2. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
3. In the navigation pane, choose **Settings**, then choose **Configure**.

4. For **Grant permission to view the list of accounts in the organization**, choose **Specific accounts** to be prompted to enter a list of account IDs. The list of accounts in your organization are shared with only the accounts that you specify here.
5. Choose **Share organization account list**.
6. Choose **Launch CloudFormation template**.

In the confirmation screen, type **Confirm**, and choose **Launch template**.

Troubleshooting your CloudWatch cross-account setup

This section contains troubleshooting tips for cross-account, console deployment in CloudWatch.

I am getting access denied errors displaying cross-account data

Check the following:

- Your monitoring account should have a role named **AWSServiceRoleForCloudWatchCrossAccount**. If it does not, you need to create this role. For more information, see [Set Up a Monitoring Account](#).
- Each sharing account should have a role named **CloudWatch-CrossAccountSharingRole**. If it does not, you need to create this role. For more information, see [Set Up A Sharing Account](#).
- The sharing role must trust the monitoring account.

To confirm that your roles are set up properly for the CloudWatch cross-account console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, make sure the needed role exists. In a sharing account, look for **CloudWatch-CrossAccountSharingRole**. In a monitoring account, look for **AWSServiceRoleForCloudWatchCrossAccount**.
4. If you are in a sharing account and **CloudWatch-CrossAccountSharingRole** already exists, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.
6. Confirm that the policy lists either the account ID of the monitoring account, or the organization ID of an organization that contains the monitoring account.

I don't see an account dropdown in the console

First, check that you have created the correct IAM roles, as discussed in the preceding troubleshooting section. If those are set up correctly, make sure that you have enabled this account to view cross-account data, as described in [Enable Your Account to View Cross-Account Data](#).

Disabling and cleaning up after using cross-account

To disable cross-account functionality for CloudWatch, follow these steps.

Step 1: Remove the cross-account stacks or roles

The best method is to remove the AWS CloudFormation stacks that were used to enable cross-account functionality.

- In each of the sharing accounts, remove the **CloudWatch-CrossAccountSharingRole** stack.
- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, remove the **CloudWatch-CrossAccountListAccountsRole** stack in the organization's management account.

If you didn't use the AWS CloudFormation stacks to enable cross-account functionality, do the following:

- In each of the sharing accounts, delete the **CloudWatch-CrossAccountSharingRole** IAM role.
- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, delete the **CloudWatch-CrossAccountSharing-ListAccountsRole** IAM role in the organization's management account.

Step 2: Remove the service-linked role

In the monitoring account, delete the **AWSServiceRoleForCloudWatchCrossAccount** service-linked IAM role.

Query metrics from other data sources

You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources. To do so, you connect CloudWatch to the other data sources. This gives you a single, consolidated monitoring experience within the CloudWatch console. You can have a unified view of your infrastructure and application metrics regardless of where the data is stored, helping you identify and resolve issues faster.

After you connect to a data source using a CloudWatch wizard, CloudWatch creates an AWS CloudFormation stack that deploys and configures an AWS Lambda function. This Lambda function runs on demand every time you query the data source. The CloudWatch query builder shows you in real time a list of elements that can be queried, such as metrics, tables, fields, or labels. As you make choices, the query builder pre-populates a query in the native language of the selected source.

CloudWatch provides guided wizards for you to connect to the following data sources. For these data sources, you provide basic information to identify the data source and credentials. You can also manually create connectors to other data sources by creating your own Lambda functions.

- Amazon OpenSearch Service– Derive metrics from your OpenSearch Service logs and traces.
- Amazon Managed Service for Prometheus– Query these metrics using PromQL.
- Amazon RDS for MySQL – Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon RDS for PostgreSQL– Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon S3 CSV files– Display metrics data from a CSV file stored in an Amazon S3 bucket.
- Microsoft Azure Monitor– Query metrics from your Microsoft Azure Monitor account.
- Prometheus– Query these metrics using PromQL.

After you create connectors to data sources, see [Creating a graph of metrics from another data source](#) for information about graphing a metric from a data source. For information about setting an alarm on a metric from a data source, see [Create an alarm based on a connected data source](#).

Topics

- [Managing access to data sources](#)

- [Connect to a prebuilt data source with a wizard](#)
- [Create a custom connector to a data source](#)
- [Use your custom data source](#)
- [Delete a connector to a data source](#)

Managing access to data sources

CloudWatch uses AWS CloudFormation to create the required resources in your account. We recommend that you use the `cloudformation:TemplateUrl` condition to control access to AWS CloudFormation templates when you grant `CreateStack` permissions to IAM users.

Warning

Any user that you grant data source invoke permission to can query metrics from that data source even if that user does not have direct IAM permissions to the data source. For example, if you grant `lambda:InvokeFunction` permissions on a Amazon Managed Service for Prometheus data source Lambda function to a user, that user will be able to query metrics from the corresponding Amazon Managed Service for Prometheus workspace even if you didn't grant them direct IAM access to that workspace.

You can find template URLs for data sources on the **Create stack** page in the CloudWatch Settings Console.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "cloudformation:CreateStack" ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudformation:TemplateUrl": [ data-source-template-url ]
        }
      }
    }
  ]
}
```

```
}
```

For more information about controlling AWS CloudFormation access, see [Controlling access with AWS Identity and Access Management](#)

Connect to a prebuilt data source with a wizard

This topic provides instructions for using the wizard to connect CloudWatch to the following data sources.

- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon RDS for MySQL
- Amazon RDS for PostgreSQL
- Amazon S3 CSV files
- Microsoft Azure Monitor
- Prometheus

Later in this section are subsections with notes about managing and querying with each of these data sources.

To create a connector to a data source

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Select the source that you want, then choose **Next**.
6. Enter a name for the data source.
7. Enter the other required information, depending on the data source that you chose. This can include credentials to access the data source and data source identifying information such as Prometheus workspace name, database name, or Amazon S3 bucket name. For AWS services, the wizard discovers the resources and populates them into the selection dropdown.

For more notes about the data source you are using, see the sections after this procedure.

8. To have CloudWatch connect to the data source in a VPC, choose **Use a VPC** and select the VPC to use. Then select the subnet and security group.
9. Choose **I acknowledge that AWS CloudFormation will create IAM resources**. This resource is the Lambda function execution role.
10. Choose **Create data source**.

The new source that you just added doesn't appear until the AWS CloudFormation stack is done creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

When your new data source appears in this list, it is ready to use. You can choose **Query from CloudWatch metrics** to begin querying with it. For more information, see [Creating a graph of metrics from another data source](#).

Amazon Managed Service for Prometheus

Updating the data source configuration

- You can update your data source manually by doing the following:
 - To update the Amazon Managed Service for Prometheus workspace ID, update the `AMAZON_PROMETHEUS_WORKSPACE_ID` environment variable for the data source connector Lambda function.
 - To update the VPC configuration, see [Configuring VPC access \(console\)](#) for more information.

Querying the data source

- When querying Amazon Managed Service for Prometheus, after you select the data source in the **Multi source query** tab and select an Amazon Managed Service for Prometheus connector, you can use the **Query helper** to discover metrics and labels and provide simple PromQL queries. You can also use the PromQL query editor to build a PromQL query.
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Amazon OpenSearch Service

Creating the data source

If the OpenSearch domain is enabled for FGAC, you must map the execution role of the connector Lambda function to a user in OpenSearch Service. For more information, see the **Mapping users to roles** section in [Managing permissions](#) in the OpenSearch Service documentation.

If your OpenSearch domain is only accessible within a Virtual Private Cloud (VPC), you need to manually include a new environment variable in the Lambda function called `AMAZON_OPENSEARCH_ENDPOINT`. The value for this variable should be the root domain of the OpenSearch endpoint. You can obtain this root domain by removing `https://` and `<region>.es.amazonaws.com` from the domain endpoint listed in the OpenSearch Service console. For example, if your domain endpoint is `https://sample-domain.us-east-1.es.amazonaws.com`, the root domain would be `sample-domain`.

Updating the data source

- You can update your data source manually by doing the following:
 - To update the OpenSearch Service domain, update the `AMAZON_OPENSEARCH_DOMAIN_NAME` environment variable for the data source connector Lambda function.
 - To update the VPC configuration, see [Configuring VPC access \(console\)](#) for more information.

Querying the data source

- When querying OpenSearch Service, after you select the data source in the **Multi source query** tab, do the following:
 - Select the Index to query.
 - Select the Metric name (Any numeric field in the document) and Stat.
 - Select the Time axis (Any date field in the document).
 - Select Filters to apply (Any String field in the document).
 - Choose **Graph query**.

Amazon RDS for PostgreSQL and Amazon RDS for MySQL

Creating the data source

- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in [Connect to a prebuilt data source with a wizard](#). If the data source is to connect to the VPC for credentials, the endpoint must be configured in the VPC. For more information, see [Using an AWS Secrets Manager VPC endpoint](#).

Additionally, you must create a VPC endpoint for the Amazon RDS service. For more information, see [Amazon RDS API and interface VPC endpoints \(AWS PrivateLink\)](#).

Updating the data source

- You can update your data source manually by doing the following:
 - To update the database instance, update the RDS_INSTANCE environment variable for the data source connector Lambda function.
 - To update the username and password used to connect to Amazon RDS, use AWS Secrets Manager. You can find the ARN of the secret used for the data source in the environment variable RDS_SECRET on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see [Modify an AWS Secrets Manager secret](#).
 - To update the VPC configuration, see [Configuring VPC access \(console\)](#) for more information.

Querying the data source

- When querying Amazon RDS, after you select the data source in the **Multi source query** tab and select an Amazon RDS connector, you can use the database discoverer to view available databases, tables, and columns. You can also use the SQL editor to create an SQL query.

You can use the following variables in the query:

- `$start.iso` – The start time in ISO date format
- `$end.iso` – The end time in ISO date format
- `$period` – The selected period in seconds

For example, you can query `SELECT value, timestamp FROM table WHERE timestamp BETWEEN $start.iso and $end.iso`

- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard

or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Note

If no date field is found in the results, the values for each numeric field are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using SUM and aligned with the period in CloudWatch.

Amazon S3 CSV files

Querying the data source

- When querying Amazon S3 CSV files, after you select the data source in the **Multi source query** tab and select an Amazon S3 connector, you select the Amazon S3 bucket and key.

The CSV file must be formatted in the following ways:

- The time stamp must be the first column.
- The table must have a header row. The headers are used to name your metrics. The title of the time stamp column will be ignored, only the titles of the metrics columns are used.
- The time stamps must be in ISO date format.
- The metrics must be numeric fields.

```
Timestamp, Metric-1, Metric-2, ...
```

The following is an example:

timestamp	CPU (%)	Memory (%)	Storage (%)
2023-11-23T17:09:41+00:00	1	2	3
2023-11-23T17:04:41+00:00	4	5	6

timestamp	CPU (%)	Memory (%)	Storage (%)
2023-11-23T16:59:41+00:00	7	8	9
2023-11-23T16:54:41+00:00	10	11	12

Note

If no timestamp is provided, the values for each metric are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using SUM and aligned with the period in CloudWatch.

Microsoft Azure Monitor

Creating the data source

- You must provide your tenant ID, client ID, and client secret to connect to Microsoft Azure Monitor. The credentials will be stored in AWS Secrets Manager. For more information, see [Create a Microsoft Entra application and service principal that can access resources](#) in the Microsoft documentation.

Updating the data source

- You can update your data source manually by doing the following:
 - To update the tenant ID, client ID, and client secret used to connect to Azure Monitor, you can find the ARN of the secret used for the data source as the AZURE_CLIENT_SECRET environment variable on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see [Modify an AWS Secrets Manager secret](#).

Querying the data source

- When querying Azure Monitor, after you select the data source in the **Multi source query** tab and select an Azure Monitor connector, you specify the Azure subscription, and the resource group and resource. You can then select the metric namespace, metric, and aggregation, and filter by dimensions.

Prometheus

Creating the data source

- You must provide the Prometheus endpoint and the user and password required to query Prometheus. The credentials will be stored in AWS Secrets Manager.
- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in [Connect to a prebuilt data source with a wizard](#). If the data source is to connect to for credentials, the endpoint must be configured in the VPC. For more information, see [Using an AWS Secrets Manager VPC endpoint](#).

Updating data source configuration

- You can update your data source manually by doing the following:
 - To update the Prometheus endpoint, specify the new endpoint as the `PROMETHEUS_API_ENDPOINT` environment variable on the data source Lambda function.
 - To update the username and password used to connect to Prometheus, you can find the ARN of the secret used for the data source as the `PROMETHEUS_API_SECRET` environment variable on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see [Modify an AWS Secrets Manager secret](#).
 - To update the VPC configuration, see [Configuring VPC access \(console\)](#) for more information.

Querying the data source

Important

Prometheus metric types are different than CloudWatch metrics and many metrics available through Prometheus are cumulative by design. When you query Prometheus metrics, CloudWatch doesn't apply any additional transformation to the data: if you specify only the metric name or label, the displayed value will be cumulative. For more information, see [Metric types](#) in the Prometheus documentation.

To see Prometheus metrics data as discrete values, like CloudWatch metrics, you need to edit the query before you run it. For example, you might need to add a call to the `rate` function over your Prometheus metric name. For documentation about the `rate` function and other Prometheus functions, see [rate\(\)](#) in the Prometheus documentation.

Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Notification of Available Updates

From time to time, Amazon might notify you that we recommend that you update your connectors with a newer available version and will provide instructions for how to do so.

Create a custom connector to a data source

To connect a custom data source to CloudWatch, you have two options:

- Get started by using a sample template that CloudWatch provides. You can use either JavaScript or Python with this template. These templates include sample Lambda code that will be useful to you as you create your Lambda function. You can then modify the Lambda function from the template to connect to your custom data source.
- Create an AWS Lambda function from scratch that implements the data source connector, the data query, and the preparation of the time series for use by CloudWatch. This function must pre-aggregate or merge datapoints if needed, and also align the period and timestamps to be compatible with CloudWatch.

Contents

- [Use a template](#)
- [Create a custom data source from scratch](#)
 - [Step 1: Create the function](#)
 - [GetMetricData event](#)

- [DescribeGetMetricData event](#)
- [Important considerations for CloudWatch alarms](#)
- [\(Optional\) Use AWS Secrets Manager to store credentials](#)
- [\(Optional\) Connect to a data source in a VPC](#)
- [Step 2: Create a Lambda permissions policy](#)
- [Step 3: Attach a resource tag to the Lambda function](#)

Use a template

Using a template creates a sample Lambda function, and can help you get your custom connector built faster. These sample functions provide sample code for many common scenarios involved with building a custom connector. You can examine the Lambda code after you create a connector with a template, then modify it to use to connect to your data source.

Additionally, if you use the template, CloudWatch takes care of creating the Lambda permissions policy and attaching resource tags to the Lambda function.

To use the template to create a connector to a custom data source


1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Choose the radio button for **Custom - getting started template** and then choose **Next**.
6. Enter a name for the data source.
7. Select the one of the listed templates.
8. Select either Node.js or Python.
9. Choose **Create data source**.

The new custom source that you just added doesn't appear until the AWS CloudFormation stack finishes creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

When your new data source appears in this list, it is ready for you to test in the console and modify.

10. (Optional) To query the test data from this source in the console, follow the instructions in [Creating a graph of metrics from another data source](#).
11. Modify the Lambda function for your needs.
 - a. In the navigation pane, choose **Settings**.
 - b. Choose the **Metrics data sources** tab.
 - c. Choose **View in Lambda console** for the source that you want to modify.

You can now modify the function to access your data source. For more information, see [Step 1: Create the function](#).

 **Note**

By using the template, when you write your Lambda function you don't need to follow the instructions in [Step 2: Create a Lambda permissions policy](#) or [Step 3: Attach a resource tag to the Lambda function](#). These steps were performed by CloudWatch because you used the template.

Create a custom data source from scratch

Follow the steps in this section to create a Lambda function that connects CloudWatch to a data source.

Step 1: Create the function

A custom data source connector must support `GetMetricData` events from CloudWatch. Optionally, you can also implement a `DescribeGetMetricData` event to provide documentation to users in the CloudWatch console for how to use the connector. The `DescribeGetMetricData` response can also be used to set defaults that are used in the CloudWatch custom query builder.

CloudWatch provides code snippets as samples to help you get started. For more information, see the samples repository at <https://github.com/aws-samples/cloudwatch-data-source-samples>.

Constraints

- The response from Lambda must be smaller than 6 Mb. If the response exceeds 6 Mb, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must complete its execution within 10 seconds for visualization and dashboarding purposes, or within 4.5 seconds for alarms usage. If the execution time exceeds that time, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must send its output using epoch timestamps in seconds.
- If the Lambda function doesn't resample the data and instead returns data that doesn't correspond to the start time and period length that was requested by the CloudWatch user, that data is ignored by CloudWatch. The extra data is discarded from any visualization or alarming. Any data that is not between the start time and end time is also discarded.

For example, if a user asks for data from 10:00 to 11:00 with a period of 5 min, then "10:00:00 to 10:04:59" and "10:05:00 to 10:09:59" are the valid time ranges for data to be returned. You must return a time series that includes `10:00 value1`, `10:05 value2`, and so on. If the function returns `10:03 valueX`, for example, it gets dropped because 10:03 does not correspond to the requested start time and period.

- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

GetMetricData event

Request payload

The following is an example of a `GetMetricData` request payload sent as input to the Lambda function.

```
{
  "EventType": "GetMetricData",
  "GetMetricDataRequest": {
    "StartTime": 1697060700,
    "EndTime": 1697061600,
    "Period": 300,
    "Arguments": ["serviceregistry_external_http_requests{host_cluster!=\"prod\"}"]
  }
}
```

- **StartTime**– The timestamp specifying the earliest data to return. The **Type** is timestamp epoch seconds.
- **EndTime**– The timestamp specifying the latest data to return. The **Type** is timestamp epoch seconds.
- **Period**– The number of seconds that each aggregation of the metrics data represents. The minimum is 60 seconds. The **Type** is Seconds.
- **Arguments**– An array of arguments to pass to the Lambda metric math expression. For more information about passing arguments, see [How to pass arguments to your Lambda function](#).

Response payload

The following is an example of a `GetMetricData` response payload returned by the Lambda function.

```
{
  "MetricDataResults": [
    {
      "StatusCode": "Complete",
      "Label": "CPUUtilization",
      "Timestamps": [ 1697060700, 1697061000, 1697061300 ],
      "Values": [ 15000, 14000, 16000 ]
    }
  ]
}
```

The response payload will contain either a `MetricDataResults` field or an `Error` field, but not both.

A `MetricDataResults` field is a list of time-series fields of type `MetricDataResult`. Each of those time-series fields can include the following fields.

- **StatusCode**– (Optional) `Complete` indicates that all data points in the requested time range were returned. `PartialData` means that an incomplete set of data points were returned. If this is omitted, the default is `Complete`.

Valid Values: `Complete` | `InternalError` | `PartialData` | `Forbidden`

- **Messages**– Optional list of messages with additional information about the data returned.

Type: Array of [MessageData](#) objects with `Code` and `Value` strings.

- **Label**– The human-readable label associated with the data.

Type: String

- **Timestamps**– The timestamps for the data points, formatted in epoch time. The number of timestamps always matches the number of values and the value for `Timestamps[x]` is `Values[x]`.

Type: Array of timestamps

- **Values**– The data point values for the metric, corresponding to `Timestamps`. The number of values always matches the number of timestamps and the value for `Timestamps[x]` is `Values[x]`.

Type: Array of doubles

For more information about `Error` objects, see the following sections.

Error response formats

You can optionally use the error response to provide more information about errors. We recommend that you return an error with `Code Validation` when a validation error occurs, such as when a parameter is missing or is the wrong type.

The following is an example of the response when the Lambda function wants to raise a `GetMetricData` validation exception.

```
{
  "Error": {
    "Code": "Validation",
    "Value": "Invalid Prometheus cluster"
  }
}
```

The following is an example of the response when the Lambda function indicates that it's unable to return data because of an access issue. The response is translated into a single time series with a status code of `Forbidden`.

```
{
  "Error": {
    "Code": "Forbidden",
```

```
    "Value": "Unable to access ..."  
  }  
}
```

The following is an example of when the Lambda function raises an overall `InternalError` exception, which is translated into a single time series with a status code of `InternalError` and a message. Whenever an error code has a value other than `Validation` or `Forbidden`, CloudWatch assumes that it's a generic internal error.

```
{  
  "Error": {  
    "Code": "PrometheusClusterUnreachable",  
    "Value": "Unable to communicate with the cluster"  
  }  
}
```

DescribeGetMetricData event

Request payload

The following is an example of a `DescribeGetMetricData` request payload.

```
{  
  "EventType": "DescribeGetMetricData"  
}
```

Response payload

The following is an example of a `DescribeGetMetricData` response payload.

```
{  
  "Description": "Data source connector",  
  "ArgumentDefaults": [{  
    "Value": "default value"  
  }]  
}
```

- **Description**– A description of how to use the data source connector. This description will appear in the CloudWatch console. Markdown is supported.

Type: String

- **ArgumentDefaults**– Optional array of argument default values used pre-populate the custom data source builder.

If `[{ Value: "default value 1"}, { Value: 10}]`, is returned, the query builder in the CloudWatch console displays two inputs, the first with “default value 1” and the second with 10.

If `ArgumentDefaults` is not provided, a single input is displayed with type default set to `String`.

Type: Array of objects containing `Value` and `Type`.

- **Error**– (Optional) An error field can be included in any response. You can see examples in [GetMetricData event](#).

Important considerations for CloudWatch alarms

If you are going to use the data source to set CloudWatch alarms, you should set it up to report data with timestamps every minute to CloudWatch. For more information and other considerations for creating alarms on metrics from connected data sources, see [Create an alarm based on a connected data source](#).

(Optional) Use AWS Secrets Manager to store credentials

If your Lambda function needs to use credentials to access the data source, we recommend using AWS Secrets Manager to store these credentials instead of hardcoding them into your Lambda function. For more information about using AWS Secrets Manager with Lambda, see [Use AWS Secrets Manager secrets in AWS Lambda functions](#).

(Optional) Connect to a data source in a VPC

If your data source is in a VPC managed by Amazon Virtual Private Cloud, you must configure your Lambda function to access it. For more information, see [Connecting outbound networking to resources in a VPC](#).

You might also need to configure VPC service endpoints to access services such as AWS Secrets Manager. For more information, see [Access an AWS service using an interface VPC endpoint](#).

Step 2: Create a Lambda permissions policy

You must use create a policy statement that grants CloudWatch permission to use the Lambda function that you created. You can use the AWS CLI or the Lambda console to create the policy statement.

To use the AWS CLI to create the policy statement

- Enter the following command. Replace *123456789012* with your account ID, replace *my-data-source-function* with the name of your Lambda function, and replace *MyDataSource-DataSourcePermission1234* with an arbitrary unique value.

```
aws lambda add-permission --function-name my-data-source-function --statement-id MyDataSource-DataSourcePermission1234 --action lambda:InvokeFunction --principal lambda.datasources.cloudwatch.amazonaws.com --source-account 123456789012
```

Step 3: Attach a resource tag to the Lambda function

The CloudWatch console determines which of your Lambda functions are data source connectors by using a tag. When you create a data source using one of the wizards, the tag is automatically applied by the AWS CloudFormation stack that configures it. When you create a data source yourself, you can use the following tag for your Lambda function. This makes your connector appear in the **Data source** dropdown in the CloudWatch console when you query metrics.

- A tag with `cloudwatch:datasource` as the key and `custom` as the value.

Use your custom data source

After you've created a data source, you can use it to query data from that source to visualize it and to set alarms. If you used the template to create your custom data source connector or you added the tag listed in [Step 3: Attach a resource tag to the Lambda function](#), you can follow the steps in [Creating a graph of metrics from another data source](#) to query it.

You can also use the metric math function LAMBDA to query it, as explained in the following section.

For information about creating alarms on metrics from your data source, see [Create an alarm based on a connected data source](#).

How to pass arguments to your Lambda function

The recommended way for you to pass arguments to your custom data source is to use the query builder in the CloudWatch console when you query the data source.

You can also use your Lambda function to retrieve data from your data source by using the new LAMBDA expression in CloudWatch metric math.

```
LAMBDA("LambdaFunctionName" [, optional-arg]*)
```

`optional-arg` is up to 20 strings, numbers, or Booleans. For example, `param`, `3.14`, or `true`.

Note

Multi-line strings are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

When you use the LAMBDA metric math function, you can provide the function name ("MyFunction"). If your resource policy allows, you can also use a specific version of the function ("MyFunction:22"), or a Lambda function alias ("MyFunction:MyAlias"). You can't use *

The following are some examples of calling the LAMBDA function.

```
LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query")
```

```
LAMBDA("MyCustomDataSource", true, "fuzzy", 99.9)
```

The LAMBDA metric math function returns a list of time series that can be returned to the requester or combined with other metric math functions. The following is an example of combining LAMBDA with other metric math functions.

```
FILL(LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query"), 0)
```

Delete a connector to a data source

To delete a connector to a data source, follow the instructions in this section.

To delete a connector to a data source

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Manage in CloudFormation** in the row of the data source that you want to delete.

You are taken to the AWS CloudFormation console.

5. In the section with the name of your data source, choose **Delete**.
6. In the confirmation pop-up, choose **Delete**.

Collect metrics, logs, and traces with the CloudWatch agent

The unified CloudWatch agent enables you to do the following:

- Collect internal system-level metrics from Amazon EC2 instances across operating systems. The metrics can include in-guest metrics, in addition to the metrics for EC2 instances. The additional metrics that can be collected are listed in [Metrics collected by the CloudWatch agent](#).
- Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by AWS.
- Retrieve custom metrics from your applications or services using the StatsD and collectd protocols. StatsD is supported on both Linux servers and servers running Windows Server. collectd is supported only on Linux servers.
- Collect logs from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server.

Note

The CloudWatch agent does not support collecting logs from FIFO pipes.

- Version 1.300031.0 and later can be used to enable CloudWatch Application Signals. For more information, see [Application Signals](#).
- Version 1.300025.0 and later can collect traces from [OpenTelemetry](#) or [X-Ray](#) client SDKs, and send them to X-Ray.

Using the CloudWatch agent allows you to collect traces without needing to run a separate trace collection daemon, helping to reduce the number of agents that you run and manage.

You can store and view the metrics that you collect with the CloudWatch agent in CloudWatch just as you can with any other CloudWatch metrics. The default namespace for metrics collected by the CloudWatch agent is CWAgent, although you can specify a different namespace when you configure the agent.

The logs collected by the unified CloudWatch agent are processed and stored in Amazon CloudWatch Logs, just like logs collected by the older CloudWatch Logs agent. For information about CloudWatch Logs pricing, see [Amazon CloudWatch Pricing](#).

Metrics collected by the CloudWatch agent are billed as custom metrics. For more information about CloudWatch metrics pricing, see [Amazon CloudWatch Pricing](#).

The CloudWatch agent is open-source under the MIT license, and is [hosted on GitHub](#). If you would like to build, customize or contribute to the CloudWatch agent, see the GitHub repository for the latest instructions. If you think you've found a potential security issue, do not post it on GitHub or any public forum. Instead, please follow the instructions at [Vulnerability Reporting](#) or [email AWS security directly](#).

The steps in this section explain how to install the unified CloudWatch agent on Amazon EC2 instances and on-premises servers. For more information about the metrics that the CloudWatch agent can collect, see [Metrics collected by the CloudWatch agent](#).

Supported operating systems

The CloudWatch agent is supported on x86-64 architecture on the following operating systems. It is also supported on all minor version updates for each of the major versions listed here.

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server versions 23.10, 22.04, 20.04, 18.04, 16.04, and 14.04
- CentOS versions 9, 8, and 7
- Red Hat Enterprise Linux (RHEL) versions 9, 8, and 7
- Debian versions 12, 11 and 10
- SUSE Linux Enterprise Server (SLES) versions 15 and 12
- Oracle Linux versions 9, 8 and 7
- AlmaLinux versions 9 and 8
- Rocky Linux versions 9 and 8
- The following macOS computers: EC2 M1 Mac1 instances, and computers running macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)
- 64-bit versions of Windows Server 2022, Windows Server 2019, and Windows Server 2016
- 64-bit Windows 10

The agent is supported on ARM64 architecture on the following operating systems. It is also supported on all minor version updates for each of the major versions listed here.

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server versions 23.10, 22.04, 20.04, 18.04, and 16.04
- CentOS versions 9 and 8
- Red Hat Enterprise Linux (RHEL) versions 9, 8, and 7
- Debian versions 12, 11 and 10
- SUSE Linux Enterprise Server 15
- The following macOS computers: macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)

Installation process overview

You can download and install the CloudWatch agent manually using the command line, or you can integrate it with SSM. The general flow of installing the CloudWatch agent using either method is as follows:

1. Create IAM roles or users that enable the agent to collect metrics from the server and optionally to integrate with AWS Systems Manager.
2. Download the agent package.
3. Modify the CloudWatch agent configuration file and specify the metrics that you want to collect.
4. Install and start the agent on your servers. As you install the agent on an EC2 instance, you attach the IAM role that you created in step 1. As you install the agent on an on-premises server, you specify a named profile that contains the credentials of the IAM user that you created in step 1.

Contents

- [Installing the CloudWatch agent](#)
- [Create the CloudWatch agent configuration file](#)
- [Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart](#)
- [Metrics collected by the CloudWatch agent](#)
- [Common scenarios with the CloudWatch agent](#)
- [Troubleshooting the CloudWatch agent](#)

Installing the CloudWatch agent

The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using one of these operating systems, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems including Linux and Windows Server, you can download and install the CloudWatch agent using either the command line with an Amazon S3 download link, using Amazon EC2 Systems Manager, or using an AWS CloudFormation template. See the following sections for details.

Contents

- [Installing the CloudWatch agent using the command line](#)
- [Installing the CloudWatch agent using AWS Systems Manager](#)
- [Installing the CloudWatch agent on new instances using AWS CloudFormation](#)
- [CloudWatch agent credentials preference](#)
- [Verifying the signature of the CloudWatch agent package](#)

Installing the CloudWatch agent using the command line

Use the following topics to download, configure, and install the CloudWatch agent package.

Topics

- [Download and configure the CloudWatch agent using the command line](#)
- [Create IAM roles and users for use with CloudWatch agent](#)
- [Installing and running the CloudWatch agent on your servers](#)

Download and configure the CloudWatch agent using the command line

Use the following steps to download the CloudWatch agent package, create IAM roles or users, and optionally modify the common configuration file.

Download the CloudWatch agent package

Note

To download the CloudWatch agent, your connection must use TLS 1.2 or later.

The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download and install the CloudWatch agent using the command line.

For each download link, there is a general link as well as links for each Region. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid download links are:

- https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is available for download. These files are in the following locations:

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES or [https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/RELEASE_NOTES](https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/info/latest/RELEASE_NOTES)

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION or [https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/CWAGENT_VERSION](https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/info/latest/CWAGENT_VERSION)

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
x86-64	Ubuntu	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p>
x86-64	Oracle	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p>
x86-64	macOS	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p>

Architecture	Platform	Download link	Signature file link
x86-64	Windows	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p>
ARM64	Amazon Linux 2023 and Amazon Linux 2	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>
ARM64	Redhat	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>

Architecture	Platform	Download link	Signature file link
ARM64	Ubuntu	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig</p>
ARM64	SUSE	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>
ARM64	MacOS	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig</p>

To use the command line to download and install the CloudWatch agent package

1. Download the CloudWatch agent.

On a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

```
wget download-link
```

On a server running Windows Server, download the following file:

```
https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see [Verifying the signature of the CloudWatch agent package](#).
3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

```
msiexec /i amazon-cloudwatch-agent.msi
```

This command also works from within PowerShell. For more information about MSI command options, see [Command-Line Options](#) in the Microsoft Windows documentation.

If you downloaded a PKG package on a macOS server, change to the directory containing the package and enter the following:

```
sudo installer -pkg ./amazon-cloudwatch-agent.pkg -target /
```

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers. For more information, see [Create the CloudWatch agent configuration file](#).

Create IAM roles and users for use with CloudWatch agent

Access to AWS resources requires permissions. You create an IAM role, an IAM user, or both to grant permissions that the CloudWatch agent needs to write metrics to CloudWatch. If you're going to use the agent on Amazon EC2 instances, you must create an IAM role. If you're going to use the agent on on-premises servers, you must create an IAM user.

Note

We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. For writing files to and downloading files from the Parameter Store, the policies created by Amazon support only files with names that start with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or download it from Parameter Store.

If you're going to run the CloudWatch agent on Amazon EC2 instances, use the following steps to create the necessary IAM role. This role provides permissions for reading information from the instance and writing it to CloudWatch.

To create the IAM role necessary to run the CloudWatch agent on EC2 instances

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose **Roles** and then **Create role**.
3. Make sure that **AWS service** is selected under **Trusted entity type**.
4. For **Use case**, choose **EC2** under **Common use cases**,
5. Choose **Next**.

6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. (Optional) If the agent is sending traces to X-Ray you need to also give the role the **AWSXRayDaemonWriteAccess** policy. To do so, find that policy in the list and select the check box next to it.
8. Choose **Next**.
9. In **Role name**, enter a name for the role, such as *CloudWatchAgentServerRole*. Optionally give it a description. Then choose **Create role**.

The role is now created.

10. (Optional) If the agent is going to send logs to CloudWatch Logs and you want the agent to be able to set retention policies for these log groups, you need to add the `logs:PutRetentionPolicy` permission to the role. For more information, see [Allowing the CloudWatch agent to set log retention policy](#).

If you're going to run the CloudWatch agent on on-premises servers, use the following steps to create the necessary IAM user.

Warning

This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed. Access keys can be updated if necessary. For more information, see [Updating access keys](#) in the *IAM User Guide*.

To create the IAM user necessary for the CloudWatch agent to run on on-premises servers

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose **Users** and then **Add users**.
3. Enter the user name for the new user.
4. Select **Access key - Programmatic access** and choose **Next: Permissions**.
5. Choose **Attach existing policies directly**.

6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. (Optional) If the agent is going to traces to X-Ray you need to also give the role the **AWSXRayDaemonWriteAccess** policy. To do so, find that policy in the list and select the check box next to it.
8. Choose **Next: Tags**.
9. Optionally create tags for the new IAM user, and then choose **Next:Review**.
10. Confirm that the correct policy is listed, and choose **Create user**.
11. Next to the name of the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Allowing the CloudWatch agent to set log retention policy

You can configure the CloudWatch agent to set the retention policy for log groups that it sends log events to. If you do this, you must grant the `logs:PutRetentionPolicy` to the IAM role or user that the agent uses. The agent uses an IAM role to run on Amazon EC2 instances, and uses an IAM user for on-premises servers.

To grant the CloudWatch agent's IAM role permission to set log retention policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM role. You chose this name when you created the role. It might be named `CloudWatchAgentServerRole`.

When you see the role, choose the name of the role.

4. In the **Permissions** tab, choose **Add permissions, Create inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Effect": "Allow",
    "Action": "logs:PutRetentionPolicy",
    "Resource": "*"
  }
]
```

6. Choose **Review policy**.
7. For **Name**, enter **CloudWatchAgentPutLogsRetention** or something similar, and choose **Create policy**.

To grant the CloudWatch agent's IAM user permission to set log retention policies

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Users**.
3. In the search box, Type the beginning of the name of the CloudWatch agent's IAM user. You chose this name when you created the user.

When you see the user, choose the name of the user.

4. In the **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. Choose **Review policy**.
7. For **Name**, enter **CloudWatchAgentPutLogsRetention** or something similar, and choose **Create policy**.

Installing and running the CloudWatch agent on your servers

After you have created the agent configuration file that you want and created an IAM role or IAM user, use the following steps to install and run the agent on your servers, using that configuration. First, attach an IAM role or IAM user to the server that will run the agent. Then, on that server, download the agent package and start it using the agent configuration you created.

Download the CloudWatch agent package using an S3 download link

Note

To download the CloudWatch agent, your connection must use TLS 1.2 or later.

You need to install the agent on each server where you will run the agent.

Amazon Linux AMIs

The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).

```
sudo yum install amazon-cloudwatch-agent
```

All operating systems

On all supported operating systems, you can download and install the CloudWatch agent using the command line with an Amazon S3 download link as described in the following steps.

For each download link, there is a general link as well as links for each Region. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid download links are:

- https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

- https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig <a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
x86-64	Ubuntu	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p>
x86-64	Oracle	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p>
x86-64	macOS	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p>

Architecture	Platform	Download link	Signature file link
x86-64	Windows	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p>
ARM64	Amazon Linux 2023 and Amazon Linux 2	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>
ARM64	Redhat	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>

Architecture	Platform	Download link	Signature file link
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To use the command line to install the CloudWatch agent on an Amazon EC2 instance

1. Download the CloudWatch agent. For a Linux server, enter the following. For *download-link*, use the appropriate download link from the previous table.

```
wget download-link
```

For a server running Windows Server, download the following file:

```
https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see [Verifying the signature of the CloudWatch agent package](#).
3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

If you downloaded an MSI package on a server running Windows Server, change to the directory containing the package and enter the following:

```
msiexec /i amazon-cloudwatch-agent.msi
```

This command also works from within PowerShell. For more information about MSI command options, see [Command-Line Options](#) in the Microsoft Windows documentation.

(Installing on an EC2 instance) Attaching an IAM role

To enable the CloudWatch agent to send data from the instance, you must attach an IAM role to the instance. The role to attach is **CloudWatchAgentServerRole**. You should have created this role previously. For more information see [Create IAM roles and users for use with CloudWatch agent](#).

For more information on attaching an IAM role to an instance, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide*.

(Installing on an on-premises server) Specify IAM credentials and AWS Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see [Create IAM roles and users for use with CloudWatch agent](#).

You also must specify the AWS Region to send the metrics to, using the `region` field in the `[AmazonCloudWatchAgent]` section of the AWS config file, as in the following example.

```
[profile AmazonCloudWatchAgent]
```

```
region = us-west-1
```

The following is an example of using the `aws configure` command to create a named profile for the CloudWatch agent. This example assumes that you are using the default profile name of `AmazonCloudWatchAgent`.

To create the `AmazonCloudWatchAgent` profile for the CloudWatch agent

1. If you haven't already done so, install the AWS Command Line Interface on the server. For more information, see [Installing the AWS CLI](#).
2. On Linux servers, enter the following command and follow the prompts:

```
sudo aws configure --profile AmazonCloudWatchAgent
```

On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

Verify internet access

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must add the CloudWatch endpoints for the appropriate Regions to the allow list. These endpoints are listed in [Amazon CloudWatch endpoints and quotas](#).
- If you're using the agent to collect logs, you must add the CloudWatch Logs endpoints for the appropriate Regions to the allow list. These endpoints are listed in [Amazon CloudWatch Logs endpoints and quotas](#).
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must add the Systems Manager endpoints for the appropriate Regions to the allow list. These endpoints are listed in [AWS Systems Manager endpoints and quotas](#).

(Optional) Modify the common configuration for proxy or Region information

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

Note

We recommend that you use the `common-config.toml` file to provide shared configuration and credentials when you run the CloudWatch agent in an on-premise mode, and it can also be useful when you are running on Amazon EC2 and you want to reuse existing shared credential profiles and files. Enabling it via the `common-config.toml` has the added advantage that if your shared credentials file gets rotated with renewed credentials after they expire, the new credentials are automatically picked up by the agent without requiring a restart.

The default `common-config.toml` is as follows.

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##           Instance role is used for EC2 case by default.
##           AmazonCloudWatchAgent profile is used for the on-premises case by
default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#   http_proxy = "{http_url}"
```

```
# https_proxy = "{https_url}"  
# no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [\(Installing on an on-premises server\) Specify IAM credentials and AWS Region](#).

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\Users\Administrator\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the `\` characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Start the CloudWatch agent using the command line

Follow these steps to use the command line to start the CloudWatch agent on a server.

To use the command line to start the CloudWatch agent on a server

1. Copy the agent configuration file that you want to use to the server where you're going to run the agent. Note the pathname where you copy it to.
2. In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Enter one of the following commands. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

On an EC2 instance running Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Linux, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
```

On an EC2 instance running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c file:configuration-file-path
```

On an on-premises server running Windows Server, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Installing the CloudWatch agent using AWS Systems Manager

Use the following topics to install and run the CloudWatch agent using AWS Systems Manager.

Topics

- [Create IAM roles and users for use with the CloudWatch agent](#)
- [Download and configure the CloudWatch agent](#)
- [Installing the CloudWatch agent on EC2 instances using your agent configuration](#)
- [Installing the CloudWatch agent on on-premises servers](#)

Create IAM roles and users for use with the CloudWatch agent

Access to AWS resources requires permissions. You can create IAM roles and users that include the permissions that you need for the CloudWatch agent to write metrics to CloudWatch and for the CloudWatch agent to communicate with Amazon EC2 and AWS Systems Manager. You use IAM roles on Amazon EC2 instances, and you use IAM users with on-premises servers.

One role or user enables CloudWatch agent to be installed on a server and send metrics to CloudWatch. The other role or user is needed to store your CloudWatch agent configuration in Systems Manager Parameter Store. Parameter Store enables multiple servers to use one CloudWatch agent configuration.

The ability to write to Parameter Store is a broad and powerful permission. You should use it only when you need it, and it shouldn't be attached to multiple instances in your deployment. If you store your CloudWatch agent configuration in Parameter Store, we recommend the following:

- Set up one instance where you perform this configuration.
- Use the IAM role with permissions to write to Parameter Store only on this instance.
- Use the IAM role with permissions to write to Parameter Store only while you are working with and saving the CloudWatch agent configuration file.

Note

We recently modified the following procedures by using new `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies created by Amazon, instead of requiring customers to create these policies themselves. To use these policies to write the agent configuration file to Parameter Store and then download

it from Parameter Store, your agent configuration file must have a name that starts with `AmazonCloudWatch-`. If you have a CloudWatch agent configuration file with a file name that doesn't start with `AmazonCloudWatch-`, these policies can't be used to write the file to Parameter Store or to download the file from Parameter Store.

Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances

The first procedure creates the IAM role that you must attach to each Amazon EC2 instance that runs the CloudWatch agent. This role provides permissions for reading information from the instance and writing it to CloudWatch.

The second procedure creates the IAM role that you must attach to the Amazon EC2 instance being used to create the CloudWatch agent configuration file. This step is necessary if you're going to store this file in Systems Manager Parameter Store so that other servers can use it. This role provides permissions for writing to Parameter Store, in addition to the permissions for reading information from the instance and writing it to CloudWatch. This role includes permissions sufficient to run the CloudWatch agent as well as to write to Parameter Store.

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM role necessary for each server to run the CloudWatch agent

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Immediately under **Common use cases**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use

Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.

7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as **CloudWatchAgentServerRole** or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentServerPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

The following procedure creates the IAM role that can also write to Parameter Store. You can use this role to store the agent configuration file in Parameter Store so that other servers can retrieve it.

The permissions for writing to Parameter Store provide broad access. This role shouldn't be attached to all your servers, and only administrators should use it. After you create the agent configuration file and copy it to Parameter Store, you should detach this role from the instance and use `CloudWatchAgentServerRole` instead.

To create the IAM role for an administrator to write to Parameter Store

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Immediately under **Choose the service that will use this role**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
6. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use

Systems Manager service core functionality. If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.

7. Choose **Next: Tags**.
8. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for your new role, such as **CloudWatchAgentAdminRole** or another name that you prefer.
10. (Optional) For **Role description**, enter a description.
11. Confirm that **CloudWatchAgentAdminPolicy** and optionally **AmazonSSMManagedInstanceCore** appear next to **Policies**.
12. Choose **Create role**.

The role is now created.

Create IAM users to use with the CloudWatch agent on on-premises servers

The first procedure creates the IAM user that you need to run the CloudWatch agent. This user provides permissions to send data to CloudWatch.

The second procedure creates the IAM user that you can use when creating the CloudWatch agent configuration file. Use this procedure to store this file in Systems Manager Parameter Store so that other servers can use it. This user provides permissions to write to Parameter Store, in addition to the permissions to write data to CloudWatch.

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of details available with the CloudWatch Agent predefined metric sets.

To create the IAM user necessary for the CloudWatch agent to write data to CloudWatch

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.

4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.
6. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.
9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.
11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

The following procedure creates the IAM user that can also write to Parameter Store. If you're going to store the agent configuration file in Parameter Store so that other servers can use it, you need to use this IAM user. This IAM user provides permissions for writing to Parameter Store. This user also provides the permissions for reading information from the instance and writing it to CloudWatch. The permissions for writing to Systems Manager Parameter Store provide broad access. This IAM user shouldn't be attached to all your servers, and only administrators should use it. You should use this IAM user only when you are storing the agent configuration file in Parameter Store.

To create the IAM user necessary to store the configuration file in Parameter Store and send information to CloudWatch

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. Enter the user name for the new user.
4. For **Access type**, select **Programmatic access**, and then choose **Next: Permissions**.
5. For **Set permissions**, choose **Attach existing policies directly**.

6. In the list of policies, select the check box next to **CloudWatchAgentAdminPolicy**. If necessary, use the search box to find the policy.
7. To use Systems Manager to install or configure the CloudWatch agent, select the check box next to **AmazonSSMManagedInstanceCore**. This AWS managed policy enables an instance to use Systems Manager service core functionality. (If necessary, use the search box to find the policy. This policy isn't necessary if you start and configure the agent only through the command line.)
8. Choose **Next: Tags**.
9. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
10. Confirm that the correct policies are listed, and then choose **Create user**.
11. In the row for the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Download and configure the CloudWatch agent

This section explains how to use Systems Manager to download the agent and then how to create your agent configuration file. Before you can use Systems Manager to download the agent, you must make sure that the instance is configured correctly for Systems Manager.

Installing or updating SSM Agent

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see [Installing and Configuring SSM Agent on Linux Instances](#) in the *AWS Systems Manager User Guide*.

For information about installing or updating the SSM Agent, see [Working with SSM Agent](#) in the *AWS Systems Manager User Guide*.

(Optional) Verify Systems Manager prerequisites

Verify internet access

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must allow list the CloudWatch endpoints for the appropriate Regions. These endpoints are listed in [Amazon CloudWatch](#) in the *Amazon Web Services General Reference*.
- If you're using the agent to collect logs, you must allow list the CloudWatch Logs endpoints for the appropriate Regions. These endpoints are listed in [Amazon CloudWatch Logs](#) in the *Amazon Web Services General Reference*.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must allow list the Systems Manager endpoints for the appropriate Regions. These endpoints are listed in [AWS Systems Manager](#) in the *Amazon Web Services General Reference*.

Use the following steps to download the CloudWatch agent package using Systems Manager.

To download the CloudWatch agent using Systems Manager

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance to install the CloudWatch agent on. If you don't see a specific instance, it might not be configured as a managed instance for use with Systems Manager. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** field, enter *AmazonCloudWatchAgent*.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers.

If you're going to save your agent configuration file in the Systems Manager Parameter Store, you must use an EC2 instance to save to the Parameter Store. Additionally, you must first attach to that instance the `CloudWatchAgentAdminRole` IAM role. For more information about attaching roles, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide*.

For more information about creating the CloudWatch agent configuration file, see [Create the CloudWatch agent configuration file](#).

Installing the CloudWatch agent on EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

Topics

- [Attach an IAM role to the instance](#)
- [Download the CloudWatch agent package on an Amazon EC2 instance](#)
- [\(Optional\) Modify the common configuration and named profile for CloudWatch agent](#)
- [Start the CloudWatch agent](#)

Attach an IAM role to the instance

You must attach the `CloudWatchAgentServerRole` IAM role to the EC2 instance to be able to run the CloudWatch agent on the instance. This role enables the CloudWatch agent to perform actions on the instance. You should have created this role previously. For more information see [Create IAM roles and users for use with CloudWatch agent](#).

For more information, see [Attaching an IAM Role to an Instance](#) in the *Amazon EC2 User Guide*.

Download the CloudWatch agent package on an Amazon EC2 instance

You need to install the agent on each server where you will run the agent. The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make

sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see [Download the CloudWatch agent package](#).

Note

When you install or update the CloudWatch agent, only the **Uninstall and reinstall** option is supported. You can't use the **In-place update** option.

Download the CloudWatch agent on an Amazon EC2 instance Using Systems Manager

Before you can use Systems Manager to install the CloudWatch agent, you must make sure that the instance is configured correctly for Systems Manager.

Installing or updating SSM Agent

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see [Installing and Configuring the SSM Agent on Linux Instances](#) in the *AWS Systems Manager User Guide*.

For information about installing or updating SSM Agent on an instance running Windows Server, see [Installing and Configuring SSM Agent on Windows Instances](#) in the *AWS Systems Manager User Guide*.

(Optional) Verify Systems Manager prerequisites

Before you use Systems Manager Run Command to install and configure the CloudWatch agent, verify that your instances meet the minimum Systems Manager requirements. For more information, see [Setting Up AWS Systems Manager](#) in the *AWS Systems Manager User Guide*.

Verify internet access

Your Amazon EC2 instances must have outbound internet access in order to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see [Internet Gateways](#) in the *Amazon VPC User Guide*.

Download the CloudWatch agent package

Systems Manager Run Command enables you to manage the configuration of your instances. You specify a Systems Manager document, specify parameters, and execute the command on one or more instances. SSM Agent on the instance processes the command and configures the instance as specified.

To download the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance on which to install the CloudWatch agent. If you do not see a specific instance, it might not be configured for Run Command. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter *AmazonCloudWatchAgent*.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

(Optional) Modify the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can use this file to optionally specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##      Instance role is used for EC2 case by default.
##      AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#   http_proxy = "{http_url}"
#   https_proxy = "{https_url}"
#   no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the `#` from that line and specify a value. You can edit this file manually, or by using the RunShellScript Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used.

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the `#` from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the `\` characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\.credentials"
```

If you specify a `shared_credential_file`, you must also remove the `#` from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Start the CloudWatch agent

You can start the agent using Systems Manager Run Command or the command line.

Start the CloudWatch agent using Systems Manager Run Command

Follow these steps to start the agent using Systems Manager Run Command.

To start the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

- If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
 4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
 5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
 6. In the **Action** list, choose **configure**.
 7. In the **Optional Configuration Source** list, choose **ssm**.
 8. In the **Optional Configuration Location** box, enter the name of the Systems Manager parameter name of the agent configuration file that you created and saved to Systems Manager Parameter Store, as explained in [Create the CloudWatch agent configuration file](#).
 9. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
 10. Choose **Run**.
 11. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

Start the CloudWatch agent on an Amazon EC2 instance using the command line

Follow these steps to use the command line to install the CloudWatch agent on an Amazon EC2 instance.

To use the command line to start the CloudWatch agent on an Amazon EC2 instance

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Linux and macOS: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
```

Linux and macOS: If you saved the configuration file on the local computer, enter the following command. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-  
config -m ec2 -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -  
a fetch-config -m ec2 -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"  
-a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent  
\config.json"
```

Installing the CloudWatch agent on on-premises servers

If you have downloaded the CloudWatch agent on one computer and created the agent configuration file you want, you can use that configuration file to install the agent on other on-premises servers.

Download the CloudWatch agent on an on-premises server

You can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link. For information about using an Amazon S3 download link, see [Download the CloudWatch agent package](#).

Download using Systems Manager

To use Systems Manager Run Command, you must register your on-premises server with Amazon EC2 Systems Manager. For more information, see [Setting Up Systems Manager in Hybrid Environments](#) in the *AWS Systems Manager User Guide*.

If you have already registered your server, update SSM Agent to the latest version.

For information about updating SSM Agent on a server running Linux, see [Install SSM Agent for a Hybrid Environment \(Linux\)](#) in the *AWS Systems Manager User Guide*.

For information about updating the SSM Agent on a server running Windows Server, see [Install SSM Agent for a Hybrid Environment \(Windows\)](#) in the *AWS Systems Manager User Guide*.

To use the SSM Agent to download the CloudWatch agent package on an on-premises server

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, select the server to install the CloudWatch agent on. If you don't see a specific server, it might not be configured for Run Command. For more information, see [Setting Up AWS Systems Manager for Hybrid Environments](#) in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter *AmazonCloudWatchAgent*.
8. Keep **Version** blank to install the latest version of the agent.
9. Choose **Run**.

The agent package is downloaded, and the next steps are to configure and start it.

(Installing on an on-premises server) Specify IAM credentials and AWS Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier. For more information about creating this user, see [Create IAM roles and users for use with the CloudWatch agent](#).

You also must specify the AWS Region to send the metrics to, using the `region` field.

Following is an example of this file.

```
[AmazonCloudWatchAgent]
aws_access_key_id=my_access_key
```

```
aws_secret_access_key=my_secret_key
region = us-west-1
```

For *my_access_key* and *my_secret_key*, use the keys from the IAM user that doesn't have the permissions to write to Systems Manager Parameter Store. For more information about the IAM users needed for CloudWatch agent, see [Create IAM users to use with the CloudWatch agent on on-premises servers](#).

If you name this profile `AmazonCloudWatchAgent`, you don't need to do anything more. Optionally, you can give it a different name and specify that name as the value for `shared_credential_profile` in the `common-config.toml` file, which is explained in the following section.

Following is an example of using the **aws configure** command to create a named profile for the CloudWatch agent. This example assumes that you're using the default profile name of `AmazonCloudWatchAgent`.

To create the `AmazonCloudWatchAgent` profile for the CloudWatch agent

1. If you haven't already done so, install the AWS Command Line Interface on the server. For more information, see [Installing the AWS CLI](#).
2. On Linux servers, enter the following command and follow the prompts:

```
sudo aws configure --profile AmazonCloudWatchAgent
```

On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

```
aws configure --profile AmazonCloudWatchAgent
```

(Optional) Modifying the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access

## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##           Instance role is used for EC2 case by default.
##           AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#   shared_credential_profile = "{profile_name}"
#   shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#   http_proxy = "{http_url}"
#   https_proxy = "{https_url}"
#   no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the `#` from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see [\(Installing on an on-premises server\) Specify IAM credentials and AWS Region](#).

On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

If you specify a `shared_credential_profile`, you must also remove the `#` from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\Users\Administrator\.aws` on Windows Server.

The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the `\` characters.

```
shared_credential_file= "/usr/username/credentials"
```

```
shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
```

If you specify a `shared_credential_file`, you must also remove the `#` from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Starting the CloudWatch agent

You can start the CloudWatch agent using either Systems Manager Run Command or the command line.

To use SSM Agent to start the CloudWatch agent on an on-premises server

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, select the instance where you installed the agent.
6. In the **Action** list, choose **configure**.
7. In the **Mode** list, choose **onPremise**.

8. In the **Optional Configuration Location** box, enter the name of the agent configuration file that you created with the wizard and stored in the Parameter Store.
9. Choose **Run**.

The agent starts with the configuration you specified in the configuration file.

To use the command line to start the CloudWatch agent on an on-premises server

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

Linux: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Linux: If you saved the configuration file on the local computer, enter the following command. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m onPremise -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -  
a fetch-config -m onPremise -s -c file:configuration-file-path
```

Installing the CloudWatch agent on new instances using AWS CloudFormation

Amazon has uploaded several AWS CloudFormation templates to GitHub to help you install and update the CloudWatch agent on new Amazon EC2 instances. For more information about using AWS CloudFormation, see [What is AWS CloudFormation?](#)

The template location is [Deploy the Amazon CloudWatch agent to EC2 instances using AWS CloudFormation](#). This location includes both `inline` and `ssm` directories. Each of these directories contains templates for both Linux and Windows instances.

- The templates in the `inline` directory have the CloudWatch agent configuration embedded into the AWS CloudFormation template. By default, the Linux templates collect the metrics `mem_used_percent` and `swap_used_percent`, and the Windows templates collect `Memory Committed Bytes In Use` and `Paging File % Usage`.

To modify these templates to collect different metrics, modify the following section of the template. The following example is from the template for Linux servers. Follow the format and syntax of the agent configuration file to make these changes. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

```
{  
  "metrics":{  
    "append_dimensions":{  
      "AutoScalingGroupName":"${!aws:AutoScalingGroupName}",  
      "ImageId":"${!aws:ImageId}",  
      "InstanceId":"${!aws:InstanceId}",  
      "InstanceType":"${!aws:InstanceType}"  
    },  
    "metrics_collected":{  
      "mem":{  
        "measurement":[  
          "mem_used_percent"  
        ]  
      },  
    },  
  },  
}
```

```
    "swap":{
      "measurement":[
        "swap_used_percent"
      ]
    }
  }
}
```

Note

In the inline templates, all placeholder variables must have an exclamation mark (!) before them as an escape character. You can see this in the example template. If you add other placeholder variables, be sure to add an exclamation mark before the name.

- The templates in the `ssm` directory load an agent configuration file from Parameter Store. To use these templates, you must first create a configuration file and upload it to Parameter Store. You then provide the Parameter Store name of the file in the template. You can create the configuration file manually or by using the wizard. For more information, see [Create the CloudWatch agent configuration file](#).

You can use both types of templates for installing the CloudWatch agent and for updating the agent configuration.

Tutorial: Install and configure the CloudWatch agent using an AWS CloudFormation inline template

This tutorial walks you through using AWS CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using the inline templates, which don't require the use of the JSON configuration file or Parameter Store. The inline template includes the agent configuration in the template. In this tutorial, you use the default agent configuration contained in the template.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use AWS CloudFormation to install the CloudWatch agent on a new instance

1. Download the template from GitHub. In this tutorial, download the inline template for Amazon Linux 2 as follows:

```
curl -O https://raw.githubusercontent.com/aws-cloudformation/aws-cloudformation-templates/main/Solutions/AmazonCloudWatchAgent/inline/amazon_linux.yaml
```

2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. Choose **Create stack**.
4. For **Choose a template**, select **Upload a template to Amazon S3**, choose the downloaded template, and choose **Next**.
5. On the **Specify Details** page, fill out the following parameters and choose **Next**:
 - **Stack name**: Choose a stack name for your AWS CloudFormation stack.
 - **IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics, logs, and traces. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).
 - **InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
 - **InstanceType**: Choose a valid instance type.
 - **KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide*.
 - **SSHLocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.
6. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

If you refresh the console, you see that the new stack has the CREATE_IN_PROGRESS status.

8. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

Use the following command to confirm that the agent is installed:

```
rpm -qa amazon-cloudwatch-agent
```

Use the following command to confirm that the agent is running:

```
ps aux | grep amazon-cloudwatch-agent
```

The next procedure demonstrates using AWS CloudFormation to update the CloudWatch agent using an inline template. The default inline template collects the `mem_used_percent` metric. In this tutorial, you change the agent configuration to stop collecting that metric.

To use AWS CloudFormation to update the CloudWatch agent

1. In the template that you downloaded in the previous procedure, remove the following lines and then save the template:

```
"mem": {  
    "measurement": [  
        "mem_used_percent"  
    ]  
},
```

2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
3. On the AWS CloudFormation dashboard, select the stack that you created and choose **Update Stack**.
4. For **Select Template**, select **Upload a template to Amazon S3**, choose the template that you modified, and choose **Next**.
5. On the **Options** page, choose **Next** and then **Next**.
6. On the **Review** page, review your information and choose **Update**.

After some time, you see `UPDATE_COMPLETE`.

Tutorial: Install the CloudWatch agent using AWS CloudFormation and Parameter Store

This tutorial walks you through using AWS CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using an agent configuration file that you create and save in Parameter Store.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use AWS CloudFormation to install the CloudWatch agent on a new instance using a configuration from Parameter Store

1. If you haven't done so already, download the CloudWatch agent package to one of your computers so that you can create the agent configuration file. For more information and downloading the agent using Parameter Store, see [Download and configure the CloudWatch agent](#). For more information on downloading the package using the command line, see [Download and configure the CloudWatch agent using the command line](#).
2. Create the agent configuration file and save it in Parameter Store. For more information, see [Create the CloudWatch agent configuration file](#).
3. Download the template from GitHub as follows:

```
curl -O https://raw.githubusercontent.com/aws-labs/aws-cloudformation-templates/master/aws/solutions/AmazonCloudWatchAgent/ssm/amazon_linux.template
```

4. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
5. Choose **Create stack**.
6. For **Choose a template**, select **Upload a template to Amazon S3**, choose the template that you downloaded, and choose **Next**.
7. On the **Specify Details** page, fill out the following parameters accordingly and choose **Next**:
 - **Stack name**: Choose a stack name for your AWS CloudFormation stack.
 - **IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics, logs, and traces. For more information, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).
 - **InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
 - **InstanceType**: Choose a valid instance type.
 - **KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide*.
 - **SSHLocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.
 - **SSMKey**: Specifies the agent configuration file that you created and saved in Parameter Store.

8. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
9. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

If you refresh the console, you see that the new stack has the CREATE_IN_PROGRESS status.

10. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

Use the following command to confirm that the agent is installed:

```
rpm -qa amazon-cloudwatch-agent
```

Use the following command to confirm that the agent is running:

```
ps aux | grep amazon-cloudwatch-agent
```

The next procedure demonstrates using AWS CloudFormation to update the CloudWatch agent, using an agent configuration that you saved in Parameter Store.

To use AWS CloudFormation to update the CloudWatch agent using a configuration in Parameter Store

1. Change the agent configuration file stored in Parameter Store to the new configuration that you want.
2. In the AWS CloudFormation template that you downloaded in the [the section called "Tutorial: Install the CloudWatch agent using AWS CloudFormation and Parameter Store"](#) topic, change the version number. For example, you might change VERSION=1.0 to VERSION=2.0.
3. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
4. On the AWS CloudFormation dashboard, select the stack that you created and choose **Update Stack**.
5. For **Select Template**, select **Upload a template to Amazon S3**, select the template that you just modified, and choose **Next**.
6. On the **Options** page, choose **Next** and then **Next**.
7. On the **Review** page, review your information and choose **Update**.

After some time, you see UPDATE_COMPLETE.

Troubleshooting installation of the CloudWatch agent with AWS CloudFormation

This section helps you troubleshoot issues with installing and updating the CloudWatch agent using AWS CloudFormation.

Detecting when an update fails

If you use AWS CloudFormation to update your CloudWatch agent configuration, and use an invalid configuration, the agent stops sending any metrics to CloudWatch. A quick way to check whether an agent configuration update succeeded is to look at the `cfn-init-cmd.log` file. On a Linux server, the file is located at `/var/log/cfn-init-cmd.log`. On a Windows instance, the file is located at `C:\cfn\log\cfn-init-cmd.log`.

Metrics are missing

If you don't see metrics that you expect to see after installing or updating the agent, confirm that the agent is configured to collect that metric. To do this, check the `amazon-cloudwatch-agent.json` file to make sure that the metric is listed, and check that you are looking in the correct metric namespace. For more information, see [CloudWatch agent files and locations](#).

CloudWatch agent credentials preference

This section outlines the credentials provider chain that the CloudWatch agent uses to obtain credentials when communicating with other AWS services and APIs. The ordering is as follows. The preferences listed in numbers two through five of the following list are the same preference order as defined in the AWS SDK. For more information, see [Specifying Credentials](#) in the SDK documentation.

1. Shared config and credentials files as defined in the CloudWatch agent's `common-config.toml` file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information](#).
2. AWS SDK environment variables

Important

On Linux, if you run the CloudWatch agent using the `amazon-cloudwatch-agent-ctl` script, the script starts the agent as a `systemd` service. In this case, environment variables such as `HOME`, `AWS_ACCESS_KEY_ID`, and `AWS_SECRET_ACCESS_KEY` are not accessible by the agent.

3. Shared configuration and credentials files found in \$HOME/%USERPROFILE%

Note

The CloudWatch agent looks for `.aws/credentials` in `$HOME` for Linux and MacOS and looks in `%USERPROFILE%` for Windows. Unlike the AWS SDK, the CloudWatch agent does not have fallback methods to determine the home directory if the environment variables are inaccessible. This difference in behavior is to maintain backwards compatibility with earlier implementations of the AWS SDK.

Furthermore, unlike with the shared credentials found in `common-config.toml`, if the AWS SDK-derived shared credentials expire and are rotated, the renewed credentials are not automatically picked up by the CloudWatch agent and require a restart of the agent to do so.

4. An AWS Identity and Access Management role for tasks if an application is present that uses an Amazon Elastic Container Service task definition or a RunTask API operation.
5. An instance profile attached to an Amazon EC2 instance.

As a best practice, we recommend that you specify credentials in the following order when you use the CloudWatch agent.

1. Use IAM roles for tasks if your application uses an Amazon Elastic Container Service task definition or a RunTask API operation.
2. Use IAM roles if your application runs on an Amazon EC2 instance.
3. Use the CloudWatch agent `common-config.toml` file to specify the credentials file. This credentials file is the same one used by other AWS SDKs and the AWS CLI. If you're already using a shared credentials file, you can also use it for this purpose. If you provide it by using the CloudWatch agent's `common-config.toml` file, you ensure that the agent will consume rotated credentials when they expire and get replaced without requiring you to restart the agent.
4. Use environment variables. Setting environment variables is useful if you're doing development work on a computer other than an Amazon EC2 instance.

Note

If you send telemetry to a different account as explained in [Sending metrics, logs, and traces to a different account](#), the CloudWatch agent uses the credentials provider

chain described in this section to obtain the initial set of credentials. It then uses those credentials when assuming the IAM role specified by `role_arn` in the CloudWatch agent configuration file.

Verifying the signature of the CloudWatch agent package

GPG signature files are included for CloudWatch agent packages on Linux servers. You can use a public key to verify that the agent download file is original and unmodified.

For Windows Server, you can use the MSI to verify the signature.

For macOS computers, the signature is included in the agent download package.

To find the correct signature file, see the following table. For each architecture and operating system there is a general link as well as links for each Region. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid links are:

- https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
- https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
- https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm

Note

To download the CloudWatch agent, your connection must use TLS 1.2 or later.

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/

Architecture	Platform	Download link	Signature file link
		amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Centos	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Redhat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent- <i>region</i> .s3. <i>region</i> .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	SUSE	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p>
x86-64	Debian	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig</p>
x86-64	Ubuntu	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig</p>

Architecture	Platform	Download link	Signature file link
x86-64	Oracle	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig</p>
x86-64	macOS	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig</p>
x86-64	Windows	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig</p>

Architecture	Platform	Download link	Signature file link
ARM64	Amazon Linux 2023 and Amazon Linux 2	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>
ARM64	Redhat	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig</p>
ARM64	Ubuntu	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb</p>	<p>https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig</p> <p><a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig</p>

Architecture	Platform	Download link	Signature file link
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig
		<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	<a href="https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig">https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To verify the CloudWatch agent package on a Linux server

1. Download the public key.

```
shell$ wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-cloudwatch-agent.gpg
```

2. Import the public key into your keyring.

```
shell$ gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value, as you need it in the next step. In the preceding example, the key value is 3B789C72.

3. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
shell$ gpg --fingerprint key-value
pub 2048R/3B789C72 2017-11-14
    Key fingerprint = 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
uid                               Amazon CloudWatch Agent
```

The fingerprint string should be equal to the following:

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

4. Download the package signature file using **wget**. To determine the correct signature file, see the preceding table.

```
wget Signature File Link
```

5. To verify the signature, run **gpg --verify**.

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a server running Windows Server

1. Download and install GnuPG for Windows from <https://gnupg.org/download/>. When installing, include the **Shell Extension (GpgEx)** option.

You can perform the remaining steps in Windows PowerShell.

2. Download the public key.


```
PS> wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-cloudwatch-agent.gpg -OutFile amazon-cloudwatch-agent.gpg
```

3. Import the public key into your keyring.

```
PS> gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value because you need it in the next step. In the preceding example, the key value is 3B789C72.

4. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
PS> gpg --fingerprint key-value
pub   rsa2048 2017-11-14 [SC]
       9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid           [ unknown] Amazon CloudWatch Agent
```

The fingerprint string should be equal to the following:

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

5. Download the package signature file using `wget`. To determine the correct signature file, see [CloudWatch Agent Download Links](#).
6. To verify the signature, run `gpg --verify`.

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:          using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

```
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a macOS computer

- There are two methods for signature verification on macOS.
- Verify the fingerprint by running the following command.

```
pkgutil --check-signature amazon-cloudwatch-agent.pkg
```

You should see a result similar to the following.

```
Package "amazon-cloudwatch-agent.pkg":
  Status: signed by a developer certificate issued by Apple for
distribution
  Signed with a trusted timestamp on: 2020-10-02 18:13:24 +0000
  Certificate Chain:
  1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
  Expires: 2024-10-18 22:31:30 +0000
  SHA256 Fingerprint:
  81 B4 6F AF 1C CA E1 E8 3C 6F FB 9E 52 5E 84 02 6E 7F 17 21 8E FB
  0C 40 79 13 66 8D 9F 1F 10 1C

-----

  2. Developer ID Certification Authority
  Expires: 2027-02-01 22:12:15 +0000
  SHA256 Fingerprint:
  7A FC 9D 01 A6 2F 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 8D 03
  F2 9C 88 CF B0 B1 BA 63 58 7F

-----

  3. Apple Root CA
  Expires: 2035-02-09 21:40:36 +0000
```

```
SHA256 Fingerprint:  
B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C  
68 C5 BE 91 B5 A1 10 01 F0 24
```

- Or, download and use the .sig file To use this method, follow these steps.
- Install the GPG application to your macOS host by entering the following command.

```
brew install GnuPG
```

- Download the package signature file using curl. To determine the correct signature file, see [CloudWatch Agent Download Links](#).
- To verify the signature, run **gpg --verify**.

```
PS> gpg --verify sig-filename agent-download-filename  
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time  
gpg: using RSA key D58167303B789C72  
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]  
gpg: WARNING: This key is not certified with a trusted signature!  
gpg: There is no indication that the signature belongs to the owner.  
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Create the CloudWatch agent configuration file

Before running the CloudWatch agent on any servers, you must create one or more CloudWatch agent configuration files.

The agent configuration file is a JSON file that specifies the metrics, logs, and traces that the agent is to collect, including custom metrics. You can create it by using the wizard or by creating it yourself from scratch. You could also use the wizard to initially create the configuration file and then modify it manually. If you create or modify the file manually, the process is more complex, but

you have more control over the metrics collected and can specify metrics not available through the wizard.

Any time you change the agent configuration file, you must then restart the agent to have the changes take effect. To restart the agent, follow the instructions in [Start the CloudWatch agent](#).

After you have created a configuration file, you can save it manually as a JSON file and then use this file when installing the agent on your servers. Alternatively, you can store it in Systems Manager Parameter Store if you're going to use Systems Manager when you install the agent on servers.

The CloudWatch agent supports using multiple configuration files. For more information, see [Multiple CloudWatch agent configuration files](#).

Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see [Amazon CloudWatch Pricing](#).

Contents

- [Create the CloudWatch agent configuration file with the wizard](#)
- [Manually create or edit the CloudWatch agent configuration file](#)

Create the CloudWatch agent configuration file with the wizard

The agent configuration file wizard, `amazon-cloudwatch-agent-config-wizard`, asks a series of questions to help you configure the CloudWatch agent for your needs.

Required credentials

The wizard can autodetect the credentials and AWS Region to use if you have the AWS credentials and configuration files in place before you start the wizard. For more information about these files, see [Configuration and Credential Files](#) in the *AWS Systems Manager User Guide*.

In the AWS credentials file, the wizard checks for default credentials and also looks for an `AmazonCloudWatchAgent` section such as the following:

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

The wizard displays the default credentials, the credentials from the AmazonCloudWatchAgent, and an Others option. You can select which credentials to use. If you choose Others, you can input credentials.

For *my_access_key* and *my_secret_key*, use the keys from the IAM user that has the permissions to write to Systems Manager Parameter Store. For more information about the IAM users needed for the CloudWatch agent, see [Create IAM users to use with the CloudWatch agent on on-premises servers](#).

In the AWS configuration file, you can specify the Region that the agent sends metrics to if it's different than the [default] section. The default is to publish the metrics to the Region where the Amazon EC2 instance is located. If the metrics should be published to a different Region, specify the Region here. In the following example, the metrics are published to the us-west-1 Region.

```
[AmazonCloudWatchAgent]
region = us-west-1
```

Run the CloudWatch agent configuration wizard

To create the CloudWatch agent configuration file

1. Start the CloudWatch agent configuration wizard by entering the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

On a server running Windows Server, run the following commands to launch the wizard:

```
cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
```

```
.\amazon-cloudwatch-agent-config-wizard.exe
```

2. Answer the questions to customize the configuration file for your server.
3. If you're storing the configuration file locally, the configuration file `config.json` is stored in `/opt/aws/amazon-cloudwatch-agent/bin/` on Linux servers, and is stored in `C:\Program Files\Amazon\AmazonCloudWatchAgent` on Windows Server. You can then copy this file to other servers where you want to install the agent.

If you're going to use Systems Manager to install and configure the agent, be sure to answer **Yes** when prompted whether to store the file in Systems Manager Parameter Store. You can also choose to store the file in Parameter Store even if you aren't using the SSM Agent to install the CloudWatch agent. To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent](#).

CloudWatch agent predefined metric sets

The wizard is configured with predefined sets of metrics, with different detail levels. These sets of metrics are shown in the following tables. For more information about these metrics, see [Metrics collected by the CloudWatch agent](#).

Note

Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of metric details that are described in these tables.

Amazon EC2 instances running Linux

Detail level	Metrics included
Basic	<p>Mem: mem_used_percent</p> <p>Disk: disk_used_percent</p> <p>The disk metrics such as <code>disk_used_percent</code> have a dimension for <code>Partition</code>, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.</p>
Standard	<p>CPU: <code>cpu_usage_idle</code>, <code>cpu_usage_iowait</code>, <code>cpu_usage_user</code>, <code>cpu_usage_system</code></p> <p>Disk: <code>disk_used_percent</code>, <code>disk_inodes_free</code></p>

Detail level	Metrics included
	<p>Diskio: diskio_io_time</p> <p>Mem: mem_used_percent</p> <p>Swap: swap_used_percent</p>
Advanced	<p>CPU: cpu_usage_idle , cpu_usage_iowait , cpu_usage_user , cpu_usage_system</p> <p>Disk: disk_used_percent , disk_inodes_free</p> <p>Diskio: diskio_io_time , diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads</p> <p>Mem: mem_used_percent</p> <p>Netstat: netstat_tcp_established , netstat_tcp_time_wait</p> <p>Swap: swap_used_percent</p>

On-premises servers running Linux

Detail level	Metrics included
Basic	<p>Disk: disk_used_percent</p> <p>Diskio: diskio_write_bytes , diskio_read_bytes , diskio_writes , diskio_reads</p> <p>Mem: mem_used_percent</p> <p>Net: net_bytes_sent , net_bytes_recv , net_packets_sent , net_packets_recv</p> <p>Swap: swap_used_percent</p>
Standard	<p>CPU: cpu_usage_idle , cpu_usage_iowait</p> <p>Disk: disk_used_percent , disk_inodes_free</p>

Detail level	Metrics included
	<p>Diskio: <code>diskio_io_time</code> , <code>diskio_write_bytes</code> , <code>diskio_read_bytes</code> , <code>diskio_writes</code> , <code>diskio_reads</code></p> <p>Mem: <code>mem_used_percent</code></p> <p>Net: <code>net_bytes_sent</code> , <code>net_bytes_recv</code> , <code>net_packets_sent</code> , <code>net_packets_recv</code></p> <p>Swap: <code>swap_used_percent</code></p>
Advanced	<p>CPU: <code>cpu_usage_guest</code> , <code>cpu_usage_idle</code> , <code>cpu_usage_iowait</code> , <code>cpu_usage_steal</code> , <code>cpu_usage_user</code> , <code>cpu_usage_system</code></p> <p>Disk: <code>disk_used_percent</code> , <code>disk_inodes_free</code></p> <p>Diskio: <code>diskio_io_time</code> , <code>diskio_write_bytes</code> , <code>diskio_read_bytes</code> , <code>diskio_writes</code> , <code>diskio_reads</code></p> <p>Mem: <code>mem_used_percent</code></p> <p>Net: <code>net_bytes_sent</code> , <code>net_bytes_recv</code> , <code>net_packets_sent</code> , <code>net_packets_recv</code></p> <p>Netstat: <code>netstat_tcp_established</code> , <code>netstat_tcp_time_wait</code></p> <p>Swap: <code>swap_used_percent</code></p>

Amazon EC2 instances running Windows Server

Note

The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for LogicalDisk % Free Space is just % Free Space.

Detail level	Metrics included
Basic	<p>Memory: Memory % Committed Bytes In Use</p> <p>LogicalDisk: LogicalDisk % Free Space</p>
Standard	<p>Memory: Memory % Committed Bytes In Use</p> <p>Paging: Paging File % Usage</p> <p>Processor: Processor % Idle Time, Processor % Interrupt Time, Processor % User Time</p> <p>PhysicalDisk: PhysicalDisk % Disk Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p>
Advanced	<p>Memory: Memory % Committed Bytes In Use</p> <p>Paging: Paging File % Usage</p> <p>Processor: Processor % Idle Time, Processor % Interrupt Time, Processor % User Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk % Disk Time , PhysicalDisk Disk Write Bytes/sec , PhysicalDisk Disk Read Bytes/sec , PhysicalDisk Disk Writes/sec , PhysicalDisk Disk Reads/sec</p> <p>TCP: TCPv4 Connections Established , TCPv6 Connections Established</p>

On-premises server running Windows Server

Note

The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for LogicalDisk % Free Space is just % Free Space.

Detail level	Metrics included
Basic	<p>Paging: Paging File % Usage</p> <p>Processor: Processor % Processor Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk Disk Write Bytes/sec , PhysicalDisk Disk Read Bytes/sec , PhysicalDisk Disk Writes/sec , PhysicalDisk Disk Reads/sec</p> <p>Memory: Memory % Committed Bytes In Use</p> <p>Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec , Network Interface Packets Sent/sec, Network Interface Packets Received/sec</p>
Standard	<p>Paging: Paging File % Usage</p> <p>Processor: Processor % Processor Time, Processor % Idle Time, Processor % Interrupt Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk % Disk Time , PhysicalDisk Disk Write Bytes/sec , PhysicalDisk Disk Read Bytes/sec , PhysicalDisk Disk Writes/sec , PhysicalDisk Disk Reads/sec</p> <p>Memory: Memory % Committed Bytes In Use</p>

Detail level	Metrics included
	Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec , Network Interface Packets Sent/sec, Network Interface Packets Received/sec
Advanced	<p>Paging: Paging File % Usage</p> <p>Processor: Processor % Processor Time, Processor % Idle Time, Processor % Interrupt Time, Processor % User Time</p> <p>LogicalDisk: LogicalDisk % Free Space</p> <p>PhysicalDisk: PhysicalDisk % Disk Time , PhysicalDisk Disk Write Bytes/sec , PhysicalDisk Disk Read Bytes/sec , PhysicalDisk Disk Writes/sec , PhysicalDisk Disk Reads/sec</p> <p>Memory: Memory % Committed Bytes In Use</p> <p>Network Interface: Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec , Network Interface Packets Sent/sec, Network Interface Packets Received/sec</p> <p>TCP: TCPv4 Connections Established , TCPv6 Connections Established</p>

Manually create or edit the CloudWatch agent configuration file

The CloudWatch agent configuration file is a JSON file with four sections, `agent`, `metrics`, `logs`, and `traces`, described as follows:

- The `agent` section includes fields for the overall configuration of the agent.
- The `metrics` section specifies the custom metrics for collection and publishing to CloudWatch. If you're using the agent only to collect logs, you can omit the `metrics` section from the file.
- The `logs` section specifies what log files are published to CloudWatch Logs. This can include events from the Windows Event Log if the server runs Windows Server.
- The `traces` section specifies the sources for traces that are collected and sent to AWS X-Ray.

The following sections explain the structure and fields of this JSON file. You can also view the schema definition for this configuration file. The schema definition is located at *installation-directory*/doc/amazon-cloudwatch-agent-schema.json on Linux servers, and at *installation-directory*/amazon-cloudwatch-agent-schema.json on servers running Windows Server.

If you create or edit the agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json on a Linux server and \$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to install the agent.

Note

Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see [Amazon CloudWatch Pricing](#).

CloudWatch agent configuration file: Agent section

The agent section can include the following fields. The wizard doesn't create an agent section. Instead, the wizard omits it and uses the default values for all fields in this section.

- `metrics_collection_interval` – Optional. Specifies how often all metrics specified in this configuration file are to be collected. You can override this value for specific types of metrics.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

The default value is 60.

- `region` – Specifies the Region to use for the CloudWatch endpoint when an Amazon EC2 instance is being monitored. The metrics collected are sent to this Region, such as `us-west-1`. If you omit this field, the agent sends metrics to the Region where the Amazon EC2 instance is located.

If you are monitoring an on-premises server, this field isn't used, and the agent reads the Region from the AmazonCloudWatchAgent profile of the AWS configuration file.

- `credentials` – Specifies an IAM role to use when sending metrics, logs, and traces to a different AWS account. If specified, this field contains one parameter, `role_arn`.
 - `role_arn` – Specifies the Amazon Resource Name (ARN) of an IAM role to use for authentication when sending metrics, logs, and traces to a different AWS account. For more information, see [Sending metrics, logs, and traces to a different account](#).
- `debug` – Optional. Specifies running the CloudWatch agent with debug log messages. The default value is `false`.
- `aws_sdk_log_level` – Optional. Supported only in versions 1.247350.0 and later of the CloudWatch agent.

You can specify this field to have the agent perform logging for AWS SDK endpoints. The value for this field can include one or more of the following options. Separate multiple options with the `|` character.

- `LogDebug`
- `LogDebugWithSigning`
- `LogDebugWithHTTPBody`
- `LogDebugRequestRetries`
- `LogDebugWithEventStreamBody`

For more information about these options, see [LogLevelType](#).

- `logfile` – Specifies the location where the CloudWatch agent writes log messages. If you specify an empty string, the log goes to `stderr`. If you don't specify this option, the default locations are the following:
 - Linux: `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log`
 - Windows Server: `c:\ProgramData\Amazon\CloudWatchAgent\Logs\amazon-cloudwatch-agent.log`

The CloudWatch agent automatically rotates the log file that it creates. A log file is rotated out when it reaches 100 MB in size. The agent keeps the rotated log files for up to seven days, and it keeps as many as five backup log files that have been rotated out. Backup log files have a timestamp appended to their filename. The timestamp shows

the date and time that the file was rotated out: for example, `amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz`.

- `omit_hostname` – Optional. By default, the hostname is published as a dimension of metrics that are collected by the agent, unless you are using the `append_dimensions` field in the `metrics` section. Set `omit_hostname` to `true` to prevent the hostname from being published as a dimension even if you are not using `append_dimensions`. The default value is `false`.
- `run_as_user` – Optional. Specifies a user to use to run the CloudWatch agent. If you don't specify this parameter, the root user is used. This option is valid only on Linux servers.

If you specify this option, the user must exist before you start the CloudWatch agent. For more information, see [Running the CloudWatch agent as a different user](#).

- `user_agent` – Optional. Specifies the user-agent string that is used by the CloudWatch agent when it makes API calls to the CloudWatch backend. The default value is a string consisting of the agent version, the version of the Go programming language that was used to compile the agent, the runtime operating system and architecture, the build time, and the plugins enabled.
- `usage_data` – Optional. By default, the CloudWatch agent sends health and performance data about itself to CloudWatch whenever it publishes metrics or logs to CloudWatch. This data incurs no costs to you. You can prevent the agent from sending this data by specifying `false` for `usage_data`. If you omit this parameter, the default of `true` is used and the agent sends the health and performance data.

If you set this value to `false`, you must stop and restart the agent for it to take effect.

The following is an example of an agent section.

```
"agent": {
  "metrics_collection_interval": 60,
  "region": "us-west-1",
  "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
  "debug": false,
  "run_as_user": "cwagent"
}
```

CloudWatch agent configuration file: Metrics section

Fields common to Linux and Windows

On servers running either Linux or Windows Server, the `metrics` section includes the following fields:

- `namespace` – Optional. The namespace to use for the metrics collected by the agent. The default value is `CWAgent`. The maximum length is 255 characters. The following is an example:

```
{
  "metrics": {
    "namespace": "Development/Product1Metrics",
    .....
  },
}
```

- `append_dimensions` – Optional. Adds Amazon EC2 metric dimensions to all metrics collected by the agent. This also causes the agent to not publish the hostname as a dimension.

The only supported key-value pairs for `append_dimensions` are shown in the following list. Any other key-value pairs are ignored. The agent supports these key-value pairs exactly as they are shown in the following list. You can't change the key values to publish different dimension names for them.

- `"ImageId": "${aws:ImageId}"` sets the instance's AMI ID as the value of the `ImageId` dimension.
- `"InstanceId": "${aws:InstanceId}"` sets the instance's instance ID as the value of the `InstanceId` dimension.
- `"InstanceType": "${aws:InstanceType}"` sets the instance's instance type as the value of the `InstanceType` dimension.
- `"AutoScalingGroupName": "${aws:AutoScalingGroupName}"` sets the instance's Auto Scaling group name as the value of the `AutoScalingGroupName` dimension.

If you want to append dimensions to metrics with arbitrary key-value pairs, use the `append_dimensions` parameter in the field for that particular type of metric.

If you specify a value that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about

these endpoints, see [Amazon Elastic Compute Cloud \(Amazon EC2\)](#) in the *Amazon Web Services General Reference*.

- `aggregation_dimensions` – Optional. Specifies the dimensions that collected metrics are to be aggregated on. For example, if you roll up metrics on the `AutoScalingGroupName` dimension, the metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

You can roll up metrics along single or multiple dimensions. For example, specifying `[["InstanceId"], ["InstanceType"], ["InstanceId", "InstanceType"]]` aggregates metrics for instance ID singly, instance type singly, and for the combination of the two dimensions.

You can also specify `[]` to roll up all metrics into one collection, disregarding all dimensions.

- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends metrics. Specifying this and setting a private link enables you to send the metrics to an Amazon VPC endpoint. For more information, see [What Is Amazon VPC?](#)

The value of `endpoint_override` must be a string that is a URL.

For example, the following part of the metrics section of the configuration file sets the agent to use a VPC Endpoint when sending metrics.

```
{
  "metrics": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXXXXXX.monitoring.us-
east-1.vpce.amazonaws.com",
    .....
  },
}
```

- `metrics_collected` – Required. Specifies which metrics are to be collected, including custom metrics collected through StatsD or collectd. This section includes several subsections.

The contents of the `metrics_collected` section depend on whether this configuration file is for a server running Linux or Windows Server.

- `force_flush_interval` – Specifies in seconds the maximum amount of time that metrics remain in the memory buffer before being sent to the server. No matter the setting for this, if the size of the metrics in the buffer reaches 1 MB or 1000 different metrics, the metrics are immediately sent to the server.

The default value is 60.

- `credentials` – Specifies an IAM role to use when sending metrics to a different account. If specified, this field contains one parameter, `role_arn`.
- `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending metrics to a different account. For more information, see [Sending metrics, logs, and traces to a different account](#). If specified here, this value overrides the `role_arn` specified in the `agent` section of the configuration file, if any.

Linux section

On servers running Linux, the `metrics_collected` section of the configuration file can also contain the following fields.

Many of these fields can include a `measurement` section that lists the metrics you want to collect for that resource. These measurement sections can either specify the complete metric name such as `swap_used`, or just the part of the metric name that will be appended to the type of resource. For example, specifying `reads` in the measurement section of the `diskio` section causes the `diskio_reads` metric to be collected.

- `collectd` – Optional. Specifies that you want to retrieve custom metrics using the `collectd` protocol. You use `collectd` software to send the metrics to the CloudWatch agent. For more information about the configuration options available for `collectd`, see [Retrieve custom metrics with collectd](#).
- `cpu` – Optional. Specifies that CPU metrics are to be collected. This section is valid only for Linux instances. You must include at least one of the `resources` and `totalcpu` fields for any CPU metrics to be collected. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `resources` – Optional. Specify this field with a value of `*` to cause per-cpu metrics to be collected. The only allowed value is `*`.

- `totalcpu` – Optional. Specifies whether to report cpu metrics aggregated across all cpu cores. The default is true.
- `measurement` – Specifies the array of cpu metrics to be collected. Possible values are `time_active`, `time_guest`, `time_guest_nice`, `time_idle`, `time_iowait`, `time_irq`, `time_nice`, `time_softirq`, `time_steal`, `time_system`, `time_user`, `usage_active`, `usage_guest`, `usage_guest_nice`, `usage_idle`, `usage_iowait`, `usage_irq`, `usage_nice`, `usage_softirq`, `usage_steal`, `usage_system`, and `usage_user`. This field is required if you include `cpu`.

By default, the unit for `cpu_usage_*` metrics is Percent, and `cpu_time_*` metrics don't have a unit.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the cpu metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the cpu metrics. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics that the agent collects.
- `disk` – Optional. Specifies that disk metrics are to be collected. Collects metrics only for mounted volumes. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value

of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `resources` – Optional. Specifies an array of disk mount points. This field limits CloudWatch to collect metrics from only the listed mount points. You can specify `*` as the value to collect metrics from all mount points. The default value is to collect metrics from all mount points.
- `measurement` – Specifies the array of disk metrics to be collected. Possible values are `free`, `total`, `used`, `used_percent`, `inodes_free`, `inodes_used`, and `inodes_total`. This field is required if you include `disk`.

 **Note**

The disk metrics have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.

To see the default units for each disk metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `ignore_file_system_types` – Specifies file system types to exclude when collecting disk metrics. Valid values include `sysfs`, `devtmpfs`, and so on.
- `drop_device` – Setting this to `true` causes `Device` to not be included as a dimension for disk metrics.

Preventing `Device` from being used as a dimension can be useful on instances that use the Nitro system because on those instances the device names change for each disk mount when

the instance is rebooted. This can cause inconsistent data in your metrics and cause alarms based on these metrics to go to `INSUFFICIENT DATA` state.

The default is `false`.

- `metrics_collection_interval` – Optional. Specifies how often to collect the disk metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Specify key-value pairs to use as additional dimensions for only the disk metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

One key-value pair that you can use is the following. You can also specify other custom key-value pairs.

- `"VolumeId": "${aws:VolumeId}"` adds a `VolumeId` dimension to your block device disk metrics. For Amazon EBS volumes, this will be the Amazon EBS Volume ID. For EC2 instance store, this will be the device serial. Using this requires the `drop_device` parameter to be set to `false`.
- `diskio` – Optional. Specifies that disk i/o metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `resources` – Optional. If you specify an array of devices, CloudWatch collects metrics from only those devices. Otherwise, metrics for all devices are collected. You can also specify `*` as the value to collect metrics from all devices.

- `measurement` – Specifies the array of diskio metrics to be collected. Possible values are `reads`, `writes`, `read_bytes`, `write_bytes`, `read_time`, `write_time`, `io_time`, and `iops_in_progress`. This field is required if you include `diskio`.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the diskio metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the diskio metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `swap` – Optional. Specifies that swap memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `measurement` – Specifies the array of swap metrics to be collected. Possible values are `free`, `used`, and `used_percent`. This field is required if you include `swap`.

To see the default units for each swap metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the swap metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the swap metrics. If you specify this field, it is used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent. It's collected as a high-resolution metric.
- `mem` – Optional. Specifies that memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `measurement` – Specifies the array of memory metrics to be collected. Possible values are `active`, `available`, `available_percent`, `buffered`, `cached`, `free`, `inactive`, `total`, `used`, and `used_percent`. This field is required if you include `mem`.

To see the default units for each `mem` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the mem metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the mem metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics that the agent collects.
- `net` – Optional. Specifies that networking metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `resources` – Optional. If you specify an array of network interfaces, CloudWatch collects metrics from only those interfaces. Otherwise, metrics for all devices are collected. You can also specify `*` as the value to collect metrics from all interfaces.
 - `measurement` – Specifies the array of networking metrics to be collected. Possible values are `bytes_sent`, `bytes_recv`, `drop_in`, `drop_out`, `err_in`, `err_out`, `packets_sent`, and `packets_recv`. This field is required if you include `net`.

To see the default units for each net metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the net metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the net metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `netstat` – Optional. Specifies that TCP connection state and UDP connection metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `measurement` – Specifies the array of netstat metrics to be collected. Possible values are `tcp_close`, `tcp_close_wait`, `tcp_closing`, `tcp_established`, `tcp_fin_wait1`,

`tcp_fin_wait2`, `tcp_last_ack`, `tcp_listen`, `tcp_none`, `tcp_syn_sent`, `tcp_syn_recv`, `tcp_time_wait`, and `udp_socket`. This field is required if you include `netstat`.

To see the default units for each `netstat` metric, see [Metrics collected by the CloudWatch agent on Linux and macOS instances](#).

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the `netstat` metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the `netstat` metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `processes` – Optional. Specifies that process metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
 - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
 - `measurement` – Specifies the array of processes metrics to be collected. Possible values are `blocked`, `dead`, `idle`, `paging`, `running`, `sleeping`, `stopped`, `total`, `total_threads`, `wait`, and `zombies`. This field is required if you include `processes`.

For all processes metrics, the default unit is None.

Within the entry for each individual metric, you might optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.
- `nvidia_gpu` – Optional. Specifies that NVIDIA GPU metrics are to be collected. This section is valid only for Linux instances on hosts that are configured with a NVIDIA GPU accelerator and have the NVIDIA System Management Interface (`nvidia-smi`) installed.

The NVIDIA GPU metrics that are collected are prefixed with the string `nvidia_smi_` to distinguish them from the metrics collected for other accelerator types. This section can include the following fields:

- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of NVIDIA GPU metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in [Collect NVIDIA GPU metrics](#).

Within the entry for each individual metric, you can optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).
- `metrics_collection_interval` – Optional. Specifies how often to collect the NVIDIA GPU metrics, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.
- `procstat` – Optional. Specifies that you want to retrieve metrics from individual processes. For more information about the configuration options available for `procstat`, see [Collect process metrics with the procstat plugin](#).
- `statsd` – Optional. Specifies that you want to retrieve custom metrics using the StatsD protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard StatsD client to send the metrics to the CloudWatch agent. For more information about the configuration options available for StatsD, see [Retrieve custom metrics with StatsD](#).
- `ethtool` – Optional. Specifies that you want to retrieve network metrics using the `ethtool` plugin. This plugin can import both the metrics collected by the standard `ethtool` utility, as well as network performance metrics from Amazon EC2 instances. For more information about the configuration options available for `ethtool`, see [Collect network performance metrics](#).

The following is an example of a `metrics` section for a Linux server. In this example, three CPU metrics, three `netstat` metrics, three process metrics, and one disk metric are collected, and the agent is set up to receive additional metrics from a `collectd` client.

```
"metrics": {
  "aggregation_dimensions" : [{"AutoScalingGroupName"}, {"InstanceId",
"InstanceType"}],
  "metrics_collected": {
    "collectd": {},
    "cpu": {
      "resources": [
        "*"
      ],
    },
  },
}
```

```
    "measurement": [
      {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
      {"name": "cpu_usage_nice", "unit": "Percent"},
      "cpu_usage_guest"
    ],
    "totalcpu": false,
    "drop_original_metrics": [ "cpu_usage_guest" ],
    "metrics_collection_interval": 10,
    "append_dimensions": {
      "test": "test1",
      "date": "2017-10-01"
    }
  },
  "netstat": {
    "measurement": [
      "tcp_established",
      "tcp_syn_sent",
      "tcp_close"
    ],
    "metrics_collection_interval": 60
  },
  "disk": {
    "measurement": [
      "used_percent"
    ],
    "resources": [
      "*"
    ],
    "drop_device": true
  },
  "processes": {
    "measurement": [
      "running",
      "sleeping",
      "dead"
    ]
  }
},
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
}
```

```
}
```

Windows Server

In the `metrics_collected` section for Windows Server, you can have subsections for each Windows performance object, such as `Memory`, `Processor`, and `LogicalDisk`. For information about what objects and counters are available, see [Performance Counters](#) in the Microsoft Windows documentation.

Within the subsection for each object, you specify a measurement array of the counters to collect. The measurement array is required for each object that you specify in the configuration file. You can also specify a `resources` field to name the instances to collect metrics from. You can also specify `*` for `resources` to collect separate metrics for every instance. If you omit `resources` for counters that have instances, the data for all instances is aggregated into one set. If you omit `resources` for counters that don't have instances, the counters are not collected by the CloudWatch agent. To determine whether counters have instances, you can use one of the following commands.

Powershell:

```
Get-Counter -ListSet *
```

Command line (not Powershell):

```
TypePerf.exe -q
```

Within each object section, you can also specify the following optional fields:

- `metrics_collection_interval` – Optional. Specifies how often to collect the metrics for this object, overriding the global `metrics_collection_interval` specified in the agent section of the configuration file.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see [High-resolution metrics](#).

- `append_dimensions` – Optional. Specifies additional dimensions to use for only the metrics for this object. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent.
- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

Within each counter section, you can also specify the following optional fields:

- `rename` – Specifies a different name to be used in CloudWatch for this metric.
- `unit` – Specifies the unit to use for this metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the Unit description in [MetricDatum](#).

There are two other optional sections that you can include in `metrics_collected`:

- `statsd` – Enables you to retrieve custom metrics using the StatsD protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard StatsD client to send the metrics to the CloudWatch agent. For more information, see [Retrieve custom metrics with StatsD](#).
- `procstat` – Enables you to retrieve metrics from individual processes. For more information, see [Collect process metrics with the procstat plugin](#).

The following is an example `metrics` section for use on Windows Server. In this example, many Windows metrics are collected, and the computer is also set to receive additional metrics from a StatsD client.

```
"metrics": {
  "metrics_collected": {
    "statsd": {},
    "Processor": {
      "measurement": [
        {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
        "% Interrupt Time",
        "% User Time",
```

```
    "% Processor Time"
  ],
  "resources": [
    "*"
  ],
  "append_dimensions": {
    "d1": "win_foo",
    "d2": "win_bar"
  }
},
"LogicalDisk": {
  "measurement": [
    {"name": "% Idle Time", "unit": "Percent"},
    {"name": "% Disk Read Time", "rename": "DISK_READ"},
    "% Disk Write Time"
  ],
  "resources": [
    "*"
  ]
},
"Memory": {
  "metrics_collection_interval": 5,
  "measurement": [
    "Available Bytes",
    "Cache Faults/sec",
    "Page Faults/sec",
    "Pages/sec"
  ],
  "append_dimensions": {
    "d3": "win_bo"
  }
},
"Network Interface": {
  "metrics_collection_interval": 5,
  "measurement": [
    "Bytes Received/sec",
    "Bytes Sent/sec",
    "Packets Received/sec",
    "Packets Sent/sec"
  ],
  "resources": [
    "*"
  ],
  "append_dimensions": {
```

```

        "d3": "win_bo"
    }
},
"System": {
    "measurement": [
        "Context Switches/sec",
        "System Calls/sec",
        "Processor Queue Length"
    ],
    "append_dimensions": {
        "d1": "win_foo",
        "d2": "win_bar"
    }
}
},
"append_dimensions": {
    "ImageId": "${aws:ImageId}",
    "InstanceId": "${aws:InstanceId}",
    "InstanceType": "${aws:InstanceType}",
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
"aggregation_dimensions" : [{"ImageId"}, {"InstanceId", "InstanceType"}, {"d1"},[]]
}
}

```

CloudWatch agent configuration file: Logs section

The logs section includes the following fields:

- `logs_collected` – Required if the logs section is included. Specifies which log files and Windows event logs are to be collected from the server. It can include two fields, `files` and `windows_events`.
- `files` – Specifies which regular log files the CloudWatch agent is to collect. It contains one field, `collect_list`, which further defines these files.
 - `collect_list` – Required if `files` is included. Contains an array of entries, each of which specifies one log file to collect. Each of these entries can include the following fields:
 - `file_path` – Specifies the path of the log file to upload to CloudWatch Logs. Standard Unix glob matching rules are accepted, with the addition of `**` as a *super asterisk*. For example, specifying `/var/log/**/*.log` causes all `.log` files in the `/var/log` directory tree to be collected. For more examples, see [Glob Library](#).

You can also use the standard asterisk as a standard wildcard. For example, `/var/log/system.log*` matches files such as `system.log_1111`, `system.log_2222`, and so on in `/var/log`.

Only the latest file is pushed to CloudWatch Logs based on file modification time. We recommend that you use wildcards to specify a series of files of the same type, such as `access_log.2018-06-01-01` and `access_log.2018-06-01-02`, but not multiple kinds of files, such as `access_log_80` and `access_log_443`. To specify multiple kinds of files, add another log stream entry to the agent configuration file so that each kind of log file goes to a different log stream.

- `auto_removal` – Optional. If this is `true`, the CloudWatch agent automatically deletes this log file after reading it and it has been rotated. Usually the log files are deleted after their entire contents are uploaded to CloudWatch Logs, but if the agent reaches the EOF (end of file) and also detects another newer log file that matches the same `file_path`, the agent deletes the OLD file, so you must make sure that you are done writing to the OLD file before creating the NEW file. The [RUST tracing library](#) has a known incompatibility because it will potentially create a NEW log file and then still attempt to write to the OLD log file.

The agent only removes complete files from logs that create multiple files, such as logs that create separate files for each date. If a log continuously writes to a single file, it is not removed.

If you already have a log file rotation or removal method in place, we recommend that you omit this field or set it to `false`.

If you omit this field, the default value of `false` is used.

- `log_group_name` – Optional. Specifies what to use as the log group name in CloudWatch Logs.

We recommend that you use this field to specify a log group name to prevent confusion. If you omit `log_group_name`, the value of `file_path` up to the final dot is used as the log group name. For example, if the file path is `/tmp/TestLogFile.log.2017-07-11-14`, the log group name is `/tmp/TestLogFile.log`.

If you specify a log group name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}`

retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you use these variables to create many different log groups, keep in mind the limit of 1,000,000 log groups per Region per account.

Allowed characters include a–z, A–Z, 0–9, '_' (underscore), '-' (hyphen), '/' (forward slash), and '.' (period).

- `log_group_class` – Optional. Specifies which log group class to use for the new log group. For more information about log group classes, see [Log classes](#).

Valid values are `STANDARD` and `INFREQUENT_ACCESS`. If you omit this field, the default of `STANDARD` is used.

 **Important**

After a log group is created, its class can't be changed.

- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you omit this field, the value of the `log_stream_name` parameter in the global logs section is used. If that is also omitted, the default value of `{instance_id}` is used.

If a log stream doesn't already exist, it's created automatically.

- `retention_in_days` – Optional. Specifies the number of days to retain the log events in the specified log group.
 - If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.
 - If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

The CloudWatch agent wizard uses `-1` as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention.

This `-1` value set by the wizard specifies that the events in the log group will never expire. However, manually editing this value to `-1` has no effect.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

 **Note**

The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies. For more information, see [Allowing the CloudWatch agent to set log retention policy](#).

 **Warning**

If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `filters` – Optional. Can contain an array of entries, each of which specifies a regular expression and a filter type to specify whether to publish or drop log entries that match the filter. If you omit this field, all logs in the log file are published to CloudWatch Logs. If you include this field, the agent processes each log message with all of the filters that you specify, and only the log events that pass all of the filters are published to CloudWatch Logs. The log entries that don't pass all of the filters will still remain in the host's log file, but will not be sent to CloudWatch Logs.

Each entry in the `filters` array can include the following fields:

- `type`– Denotes the type of filter. Valid values are `include` and `exclude`. With `include`, the log entry must match the expression to be published to CloudWatch Logs. With `exclude`, each log entry that matches the filter is not sent to CloudWatch Logs.
- `expression`– A regular expression string that follows the [RE2 Syntax](#).

 **Note**

The CloudWatch agent doesn't check the performance of any regular expression that you supply, or restrict the run time of the evaluation of the regular expressions. We recommend that you are careful not to write an expression that is expensive to evaluate. For more information about possible issues, see [Regular expression Denial of Service - ReDoS](#)

For example, the following excerpt of the CloudWatch agent configuration file publishes logs that are PUT and POST requests to CloudWatch Logs, but excluding logs that come from Firefox.

```
"collect_list": [  
  {  
    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",  
    "log_group_name": "test.log",  
    "log_stream_name": "test.log",  
    "filters": [  
      {  
        "type": "exclude",  
        "expression": "Firefox"  
      },  
      {  
        "type": "include",  
        "expression": "P(UT|OST)"  
      }  
    ]  
  },  
  .....  
]
```

Note

The order of the filters in the configuration file matters for performance. In the preceding example, the agent drops all the logs that match Firefox before it starts evaluating the second filter. To cause fewer log entries to be evaluated by more than one filter, put the filter that you expect to rule out more logs first in the configuration file.

- `timezone` – Optional. Specifies the time zone to use when putting timestamps on log events. The valid values are `UTC` and `Local`. The default value is `Local`.

This parameter is ignored if you don't specify a value for `timestamp_format`.

- `timestamp_format` – Optional. Specifies the timestamp format, using plaintext and special symbols that start with `%`. If you omit this field, the current time is used. If you use this field, you can use the symbols in the following list as part of the format.

If a single log entry contains two time stamps that match the format, the first time stamp is used.

This list of symbols is different than the list used by the older CloudWatch Logs agent. For a summary of these differences, see [Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent](#)

`%y`

Year without century as a zero-padded decimal number. For example, 19 to represent 2019.

`%Y`

Year with century as a decimal number. For example, 2019.

`%b`

Month as the locale's abbreviated name

`%B`

Month as the locale's full name

`%m`

Month as a zero-padded decimal number

`%-m`

Month as a decimal number (not zero-padded)

`%d`

Day of the month as a zero-padded decimal number

`%-d`

Day of the month as a decimal number (not zero-padded)

`%A`

Full name of weekday, such as Monday

`%a`

Abbreviation of weekday, such as Mon

`%H`

Hour (in a 24-hour clock) as a zero-padded decimal number

`%I`

Hour (in a 12-hour clock) as a zero-padded decimal number

`%-I`

Hour (in a 12-hour clock) as a decimal number (not zero-padded)

`%p`

AM or PM

`%M`

Minutes as a zero-padded decimal number

`%-M`

Minutes as a decimal number (not zero-padded)

`%S`

Seconds as a zero-padded decimal number

`%-S`

Seconds as a decimal number (not zero padded)

`%f`

Fractional seconds as a decimal number (1-9 digits), zero-padded on the left.

`%Z`

Time zone, for example PST

`%z`

Time zone, expressed as the offset between the local time zone and UTC. For example, `-0700`. Only this format is supported. For example, `-07:00` isn't a valid format.

- `multi_line_start_pattern` – Specifies the pattern for identifying the start of a log message. A log message is made of a line that matches the pattern and any subsequent lines that don't match the pattern.

If you omit this field, multi-line mode is disabled, and any line that begins with a non-whitespace character closes the previous log message and starts a new log message.

If you include this field, you can specify `{timestamp_format}` to use the same regular expression as your timestamp format. Otherwise, you can specify a different regular expression for CloudWatch Logs to use to determine the start lines of multi-line entries.


- `encoding` – Specified the encoding of the log file so that it can be read correctly. If you specify an incorrect coding, there might be data loss because characters that can't be decoded are replaced with other characters.

The default value is `utf-8`. The following are all possible values:

`ascii, big5, euc-jp, euc-kr, gbk, gb18030, ibm866, iso2022-jp, iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8, utf-16, utf-16le, UTF-16, UTF-16LE, windows-874, windows-1250,`

windows-1251, windows-1252, windows-1253, windows-1254,
windows-1255, windows-1256, windows-1257, windows-1258, x-mac-
cyrillic

- The `windows_events` section specifies the type of Windows events to collect from servers running Windows Server. It includes the following fields:
 - `collect_list` – Required if `windows_events` is included. Specifies the types and levels of Windows events to be collected. Each log to be collected has an entry in this section, which can include the following fields:
 - `event_name` – Specifies the type of Windows events to log. This is equivalent to the Windows event log channel name: for example, System, Security, Application, and so on. This field is required for each type of Windows event to log.

 **Note**

When CloudWatch retrieves messages from a Windows log channel, it looks up the log channel based on its `Full Name` property. Meanwhile, the Windows Event Viewer navigation pane displays the `Log Name` property of log channels. The `Full Name` and `Log Name` do not always match. To confirm the `Full Name` of a channel, right-click on it in the Windows Event viewer and open **Properties**.

- `event_levels` – Specifies the levels of event to log. You must specify each level to log. Possible values include INFORMATION, WARNING, ERROR, CRITICAL, and VERBOSE. This field is required for each type of Windows event to log.
- `log_group_name` – Required. Specifies what to use as the log group name in CloudWatch Logs.
- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

If you omit this field, the value of the `log_stream_name` parameter in the global logs section is used. If that is also omitted, the default value of `{instance_id}` is used.

If a log stream doesn't already exist, it's created automatically.

- `event_format` – Optional. Specifies the format to use when storing Windows events in CloudWatch Logs. `xml` uses the XML format as in Windows Event Viewer. `text` uses the legacy CloudWatch Logs agent format.
- `retention_in_days` – Optional. Specifies the number of days to retain the Windows events in the specified log group.
 - If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.
 - If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

The CloudWatch agent wizard uses `-1` as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention. This `-1` value specifies set by the wizard specifies that the events in the log group don't expire. However, manually editing this value to `-1` has no effect.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

Note

The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies. For more information, see [Allowing the CloudWatch agent to set log retention policy](#).

Warning

If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify

are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `log_stream_name` – Required. Specifies the default log stream name to be used for any logs or Windows events that don't have individual log stream names defined in the `log_stream_name` parameter within their entry in `collect_list`.
- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends logs. Specifying this field and setting a private link enables you to send the logs to an Amazon VPC endpoint. For more information, see [What Is Amazon VPC?](#)

The value of `endpoint_override` must be a string that is a URL.

For example, the following part of the logs section of the configuration file sets the agent to use a VPC Endpoint when sending logs.

```
{
  "logs": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXXX.logs.us-east-1.vpce.amazonaws.com",
    .....
  },
}
```

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5.

If you are using the agent to report high-resolution metrics in embedded metric format, and you are setting alarms on those metrics, keep this parameter set to the default value of 5. Otherwise, the metrics are reported with a delay that can cause alarming on partial or incomplete data.

- `credentials` – Specifies an IAM role to use when sending logs to a different AWS account. If specified, this field contains one parameter, `role_arn`.
 - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending logs to a different AWS account. For more information, see [Sending metrics, logs, and traces to a different account](#). If specified here, this overrides the `role_arn` specified in the agent section of the configuration file, if any.

- `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.
- `application_signals` (Optional) Specifies that you want to enable [CloudWatch Application Signals](#). For more information about this configuration, see [Enable CloudWatch Application Signals](#).
- `kubernetes` – This field can contain an `enhanced_container_insights` parameter, which you can use to enable Container Insights with enhanced observability for Amazon EKS.
 - `enhanced_container_insights` – Set this to `true` to enable Container Insights with enhanced observability for Amazon EKS. For more information, see [Container Insights with enhanced observability for Amazon EKS](#).
 - `accelerated_compute_metrics` – Set this to `false` to opt out of collecting Nvidia GPU metrics on Amazon EKS clusters. For more information, see [NVIDIA GPU metrics](#).
- `emf` – To collect metrics embedded in logs, it is no longer necessary to add this `emf` field. This is a legacy field that specified that the agent is to collect logs that are in embedded metric format. You can generate metric data from these logs. For more information, see [Embedding metrics within logs](#).

The following is an example of a `logs` section.

```
"logs":
  {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\
\\Logs\\amazon-cloudwatch-agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log",
            "log_stream_name": "my_log_stream_name_1",
            "timestamp_format": "%H: %M: %S%y%b%-d"
          },
          {
            "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\
\\Logs\\test.log",
            "log_group_name": "test.log",
            "log_stream_name": "my_log_stream_name_2"
          }
        ]
      }
    }
  }
```

```
    ]
  },
  "windows_events": {
    "collect_list": [
      {
        "event_name": "System",
        "event_levels": [
          "INFORMATION",
          "ERROR"
        ],
        "log_group_name": "System",
        "log_stream_name": "System"
      },
      {
        "event_name": "CustomizedName",
        "event_levels": [
          "INFORMATION",
          "ERROR"
        ],
        "log_group_name": "CustomizedLogGroup",
        "log_stream_name": "CustomizedLogStream"
      }
    ]
  }
},
"log_stream_name": "my_log_stream_name",
"metrics_collected": {
  "kubernetes": {
    "enhanced_container_insights": true
  }
}
}
```

CloudWatch agent configuration file: Traces section

By adding a `traces` section to the CloudWatch agent configuration file, you can enable CloudWatch Application Signals or collect traces from X-Ray and from the OpenTelemetry instrumentation SDK and send them to X-Ray.

⚠ Important

The agent's IAM role or IAM user must have the **AWSXrayWriteOnlyAccess** policy to send trace data to X-Ray. For more information, see [Create IAM roles and users for use with CloudWatch agent](#).

For a quick start for collecting traces, you can add just the following to the CloudWatch agent configuration file.

```
"traces_collected": {
  "xray": {
  },
  "otlp": {
  }
}
```

If you add the previous section to the CloudWatch agent configuration file and restart the agent, this causes the agent to start collecting traces using the following default options and values. For more information about these parameters, see the parameter definitions later in this section.

```
"traces_collected": {
  "xray": {
    "bind_address": "127.0.0.1:2000",
    "tcp_proxy": {
      "bind_address": "127.0.0.1:2000"
    }
  },
  "otlp": {
    "grpc_endpoint": "127.0.0.1:4317",
    "http_endpoint": "127.0.0.1:4318"
  }
}
```

The traces section can include the following fields:

- **traces_collected** – Required if the traces section is included. Specifies which SDKs to collect traces from. It can include the following fields:

- `application_signals` – Optional. Specifies that you want to enable [CloudWatch Application Signals](#). For more information about this configuration, see [Enable CloudWatch Application Signals](#).
- `xray` – Optional. Specifies that you want to collect traces from the X-Ray SDK. This section can include the following fields:
 - `bind_address` – Optional. Specifies the UDP address for the CloudWatch agent to use to listen for X-Ray traces. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

If you omit this field, the default of `127.0.0.1:2000` is used.

- `tcp_proxy` – Optional. Configures the address for a proxy used to support X-Ray remote sampling. For more information, see [Configuring sampling rules](#) in the X-Ray documentation.

This section can contain the following field.

- `bind_address` – Optional. Specifies the TCP address to which the CloudWatch agent should set up the proxy. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

If you omit this field, the default of `127.0.0.1:2000` is used.

- `otlp` – Optional. Specifies that you want to collect traces from the OpenTelemetry SDK. For more information about the AWS Distro for OpenTelemetry, see [AWS Distro for OpenTelemetry](#). For more information about the AWS Distro for OpenTelemetry SDKs, see [Introduction](#).

This section can include the following fields:

- `grpc_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry traces sent using gRPC Remote Procedure Calls. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK.

If you omit this field, the default of `127.0.0.1:4317` is used.

- `http_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OTLP traces sent over HTTP. The format is `ip:port`. This address must match the address set for the HTTP exporter in the OpenTelemetry SDK.

If you omit this field, the default of `127.0.0.1:4318` is used.

- `concurrency` – Optional. Specifies the maximum number of concurrent calls to X-Ray that can be used to upload traces. The default value is 8
- `local_mode` – Optional. If `true`, the agent doesn't collect Amazon EC2 instance metadata. The default is `false`
- `endpoint_override` – Optional. Specifies a FIPS endpoint or private link to use as the endpoint where the CloudWatch agent sends traces. Specifying this field and setting a private link enables you to send the traces to an Amazon VPC endpoint. For more information, see [What is Amazon VPC](#)

The value of `endpoint_override` must be a string that is a URL.

- `region_override` – Optional. Specifies the Region to use for the X-Ray endpoint. The CloudWatch agent sends the traces to X-Ray in the specified Region. If you omit this field, the agent sends the traces to the Region where the Amazon EC2 instance is located.

If you specify a Region here, it takes precedence over the setting of the `region` parameter in the agent section of the configuration file.

- `proxy_override` – Optional. Specifies the proxy server address for the CloudWatch agent to use when sending requests to X-Ray. The proxy server's protocol must be specified as part of this address.
- `credentials` – Specifies an IAM role to use when sending traces to a different AWS account. If specified, this field contains one parameter, `role_arn`.
 - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending traces to a different AWS account. For more information, see [Sending metrics, logs, and traces to a different account](#). If specified here, this overrides the `role_arn` specified in the agent section of the configuration file, if any.

CloudWatch agent configuration file: Complete examples

The following is an example of a complete CloudWatch agent configuration file for a Linux server.

The items listed in the measurement sections for the metrics you want to collect can either specify the complete metric name such or just the part of the metric name that will be appended to the type of resource. For example, specifying either `reads` or `diskio_reads` in the measurement section of the `diskio` section will cause the `diskio_reads` metric to be collected.

This example includes both ways of specifying metrics in the measurement section.

```

{
  "agent": {
    "metrics_collection_interval": 10,
    "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
  },
  "metrics": {
    "namespace": "MyCustomNamespace",
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit":
"Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "metrics_collection_interval": 10,
        "append_dimensions": {
          "customized_dimension_key_1": "customized_dimension_value_1",
          "customized_dimension_key_2": "customized_dimension_value_2"
        }
      },
      "disk": {
        "resources": [
          "/",
          "/tmp"
        ],
        "measurement": [
          {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
          "total",
          "used"
        ],
        "ignore_file_system_types": [
          "sysfs", "devtmpfs"
        ],
        "metrics_collection_interval": 60,
        "append_dimensions": {
          "customized_dimension_key_3": "customized_dimension_value_3",
          "customized_dimension_key_4": "customized_dimension_value_4"
        }
      }
    }
  }
}

```



```
    },
    "diskio": {
      "resources": [
        "*"
      ],
      "measurement": [
        "reads",
        "writes",
        "read_time",
        "write_time",
        "io_time"
      ],
      "metrics_collection_interval": 60
    },
    "swap": {
      "measurement": [
        "swap_used",
        "swap_free",
        "swap_used_percent"
      ]
    },
    "mem": {
      "measurement": [
        "mem_used",
        "mem_cached",
        "mem_total"
      ],
      "metrics_collection_interval": 1
    },
    "net": {
      "resources": [
        "eth0"
      ],
      "measurement": [
        "bytes_sent",
        "bytes_recv",
        "drop_in",
        "drop_out"
      ]
    },
    "netstat": {
      "measurement": [
        "tcp_established",
        "tcp_syn_sent",
```

```

        "tcp_close"
    ],
    "metrics_collection_interval": 60
},
"processes": {
    "measurement": [
        "running",
        "sleeping",
        "dead"
    ]
}
},
"append_dimensions": {
    "ImageId": "${aws:ImageId}",
    "InstanceId": "${aws:InstanceId}",
    "InstanceType": "${aws:InstanceType}",
    "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
},
"aggregation_dimensions" : [ ["ImageId"], ["InstanceId", "InstanceType"],
["d1"],[]],
    "force_flush_interval" : 30
},
"logs": {
    "logs_collected": {
        "files": {
            "collect_list": [
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
                    "log_group_name": "amazon-cloudwatch-agent.log",
                    "log_stream_name": "amazon-cloudwatch-agent.log",
                    "timezone": "UTC"
                },
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                    "log_group_name": "test.log",
                    "log_stream_name": "test.log",
                    "timezone": "Local"
                }
            ]
        }
    }
},
"log_stream_name": "my_log_stream_name",
"force_flush_interval" : 15,

```

```

    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true
      }
    }
  }
}

```

The following is an example of a complete CloudWatch agent configuration file for a server running Windows Server.

```

{
  "agent": {
    "metrics_collection_interval": 60,
    "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
  },
  "metrics": {
    "namespace": "MyCustomNamespace",
    "metrics_collected": {
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "customized_dimension_key_1": "customized_dimension_value_1",
          "customized_dimension_key_2": "customized_dimension_value_2"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}

```

```
    ]
  },
  "customizedObjectName": {
    "metrics_collection_interval": 60,
    "customizedCounterName": [
      "metric1",
      "metric2"
    ],
    "resources": [
      "customizedInstances"
    ]
  },
  "Memory": {
    "metrics_collection_interval": 5,
    "measurement": [
      "Available Bytes",
      "Cache Faults/sec",
      "Page Faults/sec",
      "Pages/sec"
    ]
  },
  "Network Interface": {
    "metrics_collection_interval": 5,
    "measurement": [
      "Bytes Received/sec",
      "Bytes Sent/sec",
      "Packets Received/sec",
      "Packets Sent/sec"
    ],
    "resources": [
      "*"
    ],
    "append_dimensions": {
      "customized_dimension_key_3": "customized_dimension_value_3"
    }
  },
  "System": {
    "measurement": [
      "Context Switches/sec",
      "System Calls/sec",
      "Processor Queue Length"
    ]
  }
},
```

```
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [{"ImageId"}, {"InstanceId", "InstanceType"},
["d1"],[]]
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\
\\amazon-cloudwatch-agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log",
            "timezone": "UTC"
          },
          {
            "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\
\\test.log",
            "log_group_name": "test.log",
            "timezone": "Local"
          }
        ]
      }
    },
    "windows_events": {
      "collect_list": [
        {
          "event_name": "System",
          "event_levels": [
            "INFORMATION",
            "ERROR"
          ],
          "log_group_name": "System",
          "log_stream_name": "System",
          "event_format": "xml"
        },
        {
          "event_name": "CustomizedName",
          "event_levels": [
            "WARNING",
            "ERROR"
          ]
        }
      ]
    }
  }
}
```

```
    ],
    "log_group_name": "CustomizedLogGroup",
    "log_stream_name": "CustomizedLogStream",
    "event_format": "xml"
  }
]
}
},
"log_stream_name": "example_log_stream_name"
}
}
```

Save the CloudWatch agent configuration file manually

If you create or edit the CloudWatch agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on a Linux server and `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to run the agent.

Uploading the CloudWatch agent configuration file to Systems Manager Parameter Store

If you plan to use the SSM Agent to install the CloudWatch agent on servers, after you manually edit the CloudWatch agent configuration file, you can upload it to Systems Manager Parameter Store. To do so, use the Systems Manager `put-parameter` command.

To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions. For more information, see [Create IAM roles and users for use with the CloudWatch agent](#).

Use the following command, where *parameter name* is the name to be used for this file in Parameter Store and *configuration_file_pathname* is the path and file name of the configuration file that you edited.

```
aws ssm put-parameter --name "parameter name" --type "String" --value
file://configuration_file_pathname
```

Enable CloudWatch Application Signals

Use CloudWatch Application Signals to automatically instrument your applications on AWS so that you can track application performance against your business objectives. Application Signals provides you a unified, application-centric view of your Java applications, their dependencies, and their edges. For more information, see [Application Signals](#).

CloudWatch Application Signals leverages the CloudWatch agent to receive metrics and traces from your auto-instrumented applications, optionally apply rules to reduce high cardinality, and then publish the processed telemetry to CloudWatch. You can provide custom configuration to the CloudWatch agent specifically for Application Signals using the agent configuration file. To start with, the presence of an `application_signals` section under the `metrics_collected` section within the `logs` section of the agent configuration file specifies that the CloudWatch agent will receive metrics from your auto-instrumented applications. Similarly, the presence of an `application_signals` section under the `traces_collected` section within the `traces` section of the agent configuration file specifies that the CloudWatch agent is enabled to receive traces from your auto-instrumented applications. In addition, you can optionally pass in custom configuration rules to reduce publishing high-cardinality telemetry as outlined in this section.

- For Amazon EKS clusters, when you install the [Amazon CloudWatch Observability](#) EKS add-on, the CloudWatch agent is by default enabled to receive both metrics and traces from your auto-instrumented applications. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration to the Amazon EKS add-on when you create or update it by using additional configuration, as outlined in [\(Optional\) Additional configuration](#).
- For other supported platforms including Amazon EC2, you must start the CloudWatch agent with an agent configuration that enables Application Signals by specifying the `application_signals` sections and optionally any custom configuration rules as outlined later in this section.

The following is an overview of the fields in the CloudWatch agent configuration file that are related to CloudWatch Application Signals.

- `logs`
 - `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.

Note


Previously this section was also used to specify that the agent is to collect logs that are in embedded metric format. Those settings are no longer needed.

- `application_signals` (Optional) Specifies that you want to enable CloudWatch Application Signals to receive metrics from your auto-instrumented applications to facilitate CloudWatch Application Signals.
- `rules` (Optional) An array of rules to conditionally select metrics and traces and apply actions to handle high-cardinality scenarios. Each rule can contain the following fields:
 - `rule_name` (Optional) The name of the rule.
 - `selectors` (Optional) An array of metrics and traces dimension matchers. Each selector must provide the following fields:
 - `dimension` Required if `selectors` is not empty. This specifies the dimension of metrics and traces to use as a filter.
 - `match` Required if `selectors` is not empty. A wildcard pattern used for matching values of the specified dimension.
 - `action` (Optional) The action to be applied to metrics and traces that match the specified selectors. The value of `action` must be one of the following keywords:
 - `keep` Specifies to send only the metrics and traces to CloudWatch if matched by the selectors.
 - `drop` Specifies to drop the metric and traces that match the selectors.
 - `replace` Specifies to replace the dimensions of the metrics and traces that match selectors. They are replaced according to the `replacements` section.
 - `replacements` Required if `action` is `replace`. An array of dimension and value pairs that will be applied to metrics and traces that match the specified selectors when the action is `replace`. Each replacement must provide the following fields:
 - `target_dimension` Required if `replacements` is not empty. Specifies the dimension that needs to be replace.
 - `value` Required if `replacements` is not empty. The value to replace the original value of `target_dimension` with.

- **limiter** (Optional) Use this section to limit how many metrics and dimensions Application Signals sends to CloudWatch, to optimize your costs.
 - **disabled** (Optional) If `true`, the metric limiting feature is disabled. The default is `false`
 - **drop_threshold** (Optional) The maximum number of distinct metrics per service in one rotation interval that can be exported by one CloudWatch agent. The default is 500.
 - **rotation_interval** (Optional) The interval at which the limiter resets the metric records for distinction counting. This is expressed as a string with a sequence of numbers and a unit suffix. Fractions are supported. The supported unit suffixes are `s`, `m`, `h`, `ms`, `us`, and `ns`


The default is 1h for one hour.

- **log_dropped_metrics** (Optional) Specifies whether the agent should write logs to the CloudWatch agent logs when Application Signals metrics are dropped. The default is `false`.

 **Note**

To activate this logging, the `debug` parameter in the agent section must also be set to `true`.

- **traces**
 - **traces_collected**
 - **application_signals** Optional. Specify this to enable the CloudWatch agent to receive traces from your auto-instrumented applications for facilitating CloudWatch Application Signals.

 **Note**

Even though the custom `application_signals` rules are specified under the `metrics_collected` section that is contained in the `logs` section, they also implicitly apply to the `traces_collected` section as well. The same set of rules will apply to both metrics and traces.

When there are multiple rules with different actions, they apply in the following sequence: keep, then drop, then replace.

The following is an example of a full CloudWatch agent configuration file that applies custom rules.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "rules": [
          {
            "rule_name": "keep01",
            "selectors": [
              {
                "dimension": "Service",
                "match": "pet-clinic-frontend"
              },
              {
                "dimension": "RemoteService",
                "match": "customers-service"
              }
            ],
            "action": "keep"
          },
          {
            "rule_name": "drop01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "GET /api/customer/owners/*"
              }
            ],
            "action": "drop"
          },
          {
            "rule_name": "replace01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "PUT /api/customer/owners/*/pets/*"
              }
            ],
            "action": "replace"
          }
        ]
      }
    }
  }
}
```

```

        "dimension": "RemoteOperation",
        "match": "PUT /owners"
    }
  ],
  "replacements": [
    {
      "target_dimension": "Operation",
      "value": "PUT /api/customer/owners/{ownerId}/pets{petId}"
    }
  ],
  "action": "replace"
}
]
}
}
},
"traces": {
  "traces_collected": {
    "application_signals": {}
  }
}
}
}

```

For the previous example configuration file, the rules are processed as follows:

1. Rule `keep01` ensures that any metrics and traces with the dimension `Service` as `pet-clinic-frontend` and the dimension `RemoteService` as `customers-service` are kept.
2. For the processed metrics and traces after applying `keep01`, the `drop01` rule ensures that metrics and traces with the dimension `Operation` as `GET /api/customer/owners/*` are dropped.
3. For the processed metrics and traces after applying `drop01`, the `replace01` rule updates metrics and traces that have the dimension `Operation` as `PUT /api/customer/owners/*/pets/*` and the dimension `RemoteOperation` as `PUT /owners` such that their `Operation` dimension is now replaced to be `PUT /api/customer/owners/{ownerId}/pets{petId}`.

The following is a complete example of a CloudWatch configuration file that manages cardinality in Application Signals by changing the metric limit to 100, enabling the logging of dropped metrics, and setting the rotation interval to two hours.

```
{
```

```
"logs": {
  "metrics_collected": {
    "application_signals": {
      "limiter": {
        "disabled": false,
        "drop_threshold": 100,
        "rotation_interval": "2h",
        "log_dropped_metrics": true
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}
```

Collect network performance metrics


EC2 instances running on Linux that use the Elastic Network Adapter (ENA) publish network performance metrics. Version 1.246396.0 and later of the CloudWatch agent enable you to import these network performance metrics into CloudWatch. When you import these network performance metrics into CloudWatch, they are charged as CloudWatch custom metrics.

For more information about the ENA driver, see [Enabling enhanced networking with the Elastic Network Adapter \(ENA\) on Linux instances](#) and [Enabling enhanced networking with the Elastic Network Adapter \(ENA\) on Windows instances](#).

How you set up the collection of network performance metrics differs on Linux servers and Windows servers.

The following table lists these network performance metrics enabled by the ENA adapter. When the CloudWatch agent imports these metrics into CloudWatch from Linux instances, it prepends `ethtool_` at the beginning of each of these metric names.

Metric	Description
<p>Name on Linux servers: bw_in_allowance_exceeded</p> <p>Name on Windows servers: Aggregate inbound BW allowance exceeded</p>	<p>The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
<p>Name on Linux servers: bw_out_allowance_exceeded</p> <p>Name on Windows servers: Aggregate outbound BW allowance exceeded</p>	<p>The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
<p>Name on Linux servers: conntrack_allowance_available</p> <p>Name on Windows servers: Available connection tracking allowance</p>	<p>Reports the number of tracked connections that can be established by the instance before hitting the Connections Tracked allowance of that instance type. This metric is available only on Nitro-based EC2 instances using the Linux driver for Elastic Network Adapter (ENA) starting from version 2.8.1, and on computers using the Windows driver for Elastic Network Adapter (ENA) starting from version 2.6.0.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code></p>

Metric	Description
	<p>collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
<p>Name on Linux servers: ena_srd_mode</p> <p>Name on Windows servers: ena_srd_mode</p>	<p>Describes which ENA Express features are enabled. For more information about ENA Express, see Improve network performance with ENA Express on Linux instances Values are as follows:</p> <ul style="list-style-type: none"> • 0 = ENA Express off, UDP off • 1 = ENA Express on, UDP off • 2 = ENA Express off, UDP on <div data-bbox="781 884 1507 1199" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; background-color: #E6F2FF;"> <p> Note</p> <p>This happens only when ENA Express was originally enabled, and UDP was configured to use it. The prior value is retained for UDP traffic.</p> </div> <ul style="list-style-type: none"> • 3 = ENA Express on, UDP on

Metric	Description
<p>Name on Linux servers:ena_srd_eligible_tx_pkts</p> <p>Name on Windows servers: ena_srd_eligible_tx_pkts</p>	<p>The number of network packets sent within a given time period that meet AWS Scalable Reliable Datagram (SRD) requirements for eligibility, as follows:</p> <ul style="list-style-type: none"> • Both sending and receiving instance types are supported. • Both sending and receiving instances must have ENA Express configured. • The sending and receiving instances must be on the same subnet. • The network path between the instances must not include middleware boxes. ENA Express doesn't currently support middleware boxes.
<p>Name on Linux servers:ena_srd_tx_pkts</p> <p>Name on Windows servers: ena_srd_tx_pkts</p>	<p>The number of SRD packets transmitted within a given time period.</p>
<p>Name on Linux servers:ena_srd_rx_pkts</p> <p>Name on Windows servers: ena_srd_rx_pkts</p>	<p>The number of SRD packets received within a given time period.</p>
<p>Name on Linux servers:ena_srd_resource_utilization</p> <p>Name on Windows servers: ena_srd_resource_utilization</p>	<p>The percentage of the maximum allowed memory utilization for concurrent SRD connections that the instance has consumed.</p>

Metric	Description
<p>Name on Linux servers: linklocal_allowance_exceeded</p> <p>Name on Windows servers: Link local packet rate allowance exceeded</p>	<p>The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
<p>Name on Linux servers: pps_allowance_exceeded</p> <p>Name on Windows servers: PPS allowance exceeded</p>	<p>The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>

Linux setup

On Linux servers, the *ethtool plugin* enables you to import the network performance metrics into CloudWatch.

`ethtool` is a standard Linux utility that can collect statistics about Ethernet devices on Linux servers. The statistics it collects depend on the network device and driver. Examples of these statistics include `tx_cnt`, `rx_bytes`, `tx_errors`, and `align_errors`. When you use the `ethtool` plugin with the CloudWatch agent, you can also import these statistics into CloudWatch, along with the EC2 network performance metrics listed earlier in this section.

Tip

To find the statistics available on our operating system and network device, use the `ethtool -S` command.

When the CloudWatch agent imports metrics into CloudWatch, it adds an `ethtool_` prefix to the names of all imported metrics. So the standard `ethtool` statistic `rx_bytes` is called `ethtool_rx_bytes` in CloudWatch, and the EC2 network performance metric `bw_in_allowance_exceeded` is called `ethtool_bw_in_allowance_exceeded` in CloudWatch.

On Linux servers, to import `ethtool` metrics, add an `ethtool` section to the `metrics_collected` section of the CloudWatch agent configuration file. The `ethtool` section can include the following subsections:

- **interface_include**— Including this section causes the agent to collect metrics from only the interfaces that have names listed in this section. If you omit this section, metrics are collected from all Ethernet interfaces that aren't listed in `interface_exclude`.

The default ethernet interface is `eth0`.

- **interface_exclude**— If you include this section, list the Ethernet interfaces that you don't want to collect metrics from.

The `ethtool` plugin always ignores loopback interfaces.

- **metrics_include**— This section lists the metrics to import into CloudWatch. It can include both standard statistics collected by `ethtool` and Amazon EC2 high-resolution network metrics.

The following example displays part of the CloudWatch agent configuration file. This configuration collects the standard `ethtool` metrics `rx_packets` and `tx_packets`, and the Amazon EC2 network performance metrics from only the `eth1` interface.

For more information about the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](#).

```
"metrics": {
  "append_dimensions": {
    "InstanceId": "${aws:InstanceId}"
  },
```

```

"metrics_collected": {
  "ethtool": {
    "interface_include": [
      "eth1"
    ],
    "metrics_include": [
      "rx_packets",
      "tx_packets",
      "bw_in_allowance_exceeded",
      "bw_out_allowance_exceeded",
      "contrack_allowance_exceeded",
      "linklocal_allowance_exceeded",
      "pps_allowance_exceeded"
    ]
  }
}

```

Windows setup

On Windows servers, the network performance metrics are available through Windows Performance Counters, which the CloudWatch agent already collects metrics from. So you do not need a plugin to collect these metrics from Windows servers.

The following is a sample configuration file to collect network performance metrics from Windows. For more information about editing the CloudWatch agent configuration file, see [Manually create or edit the CloudWatch agent configuration file](#).

```

{
  "metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "ENA Packets Shaping": {
        "measurement": [
          "Aggregate inbound BW allowance exceeded",
          "Aggregate outbound BW allowance exceeded",
          "Connection tracking allowance exceeded",
          "Link local packet rate allowance exceeded",
          "PPS allowance exceeded"
        ],
        "metrics_collection_interval": 60,

```

```
        "resources": [
            "*"
        ]
    }
}
```

Viewing network performance metrics

After importing network performance metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view `ethtool` metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

Because all of these metrics are aggregate counters, you can use CloudWatch metric math functions such as `RATE(METRICS())` to calculate the rate for these metrics in graphs or use them to set alarms. For more information about metric math functions, see [Use metric math](#).

To view network performance metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID, and then choose **Add to search**.
 - d. To filter by metric, choose the metric name, and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, and then choose **Add to dashboard**.

Collect NVIDIA GPU metrics

You can use the CloudWatch agent to collect NVIDIA GPU metrics from Linux servers. To set this up, add a `nvidia_gpu` section inside the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see [Linux section](#).

Additionally, the instance must have an NVIDIA driver installed. NVIDIA drivers are pre-installed on some Amazon Machine Images (AMIs). Otherwise, you can manually install the driver. For more information, see [Install NVIDIA drivers on Linux instances](#).

The following metrics can be collected. All of these metrics are collected with no CloudWatch Unit, but you can specify a unit for each metric by adding a parameter to the CloudWatch agent configuration file. For more information, see [Linux section](#).

Metric	Metric name in CloudWatch	Description
<code>utilization_gpu</code>	<code>nvidia_smi_utilization_gpu</code>	The percentage of time over the past sample period during which one or more kernels on the GPU was running.
<code>temperature_gpu</code>	<code>nvidia_smi_temperature_gpu</code>	The core GPU temperature in degrees Celsius.
<code>power_draw</code>	<code>nvidia_smi_power_draw</code>	The last measured power draw for the entire board, in watts.
<code>utilization_memory</code>	<code>nvidia_smi_utilization_memory</code>	The percentage of time over the past sample period during which global (device) memory was being read or written.
<code>fan_speed</code>	<code>nvidia_smi_fan_speed</code>	The percentage of maximum fan speed that the device's fan is currently intended to run at.
<code>memory_total</code>	<code>nvidia_smi_memory_total</code>	Reported total memory, in MB.
<code>memory_used</code>	<code>nvidia_smi_memory_used</code>	Memory used, in MB.

Metric	Metric name in CloudWatch	Description
memory_free	nvidia_smi_memory_free	Memory free, in MB.
pcie_link_gen_current	nvidia_smi_pcie_link_gen_current	The current link generation.
pcie_link_width_current	nvidia_smi_pcie_link_width_current	The current link width.
encoder_stats_session_count	nvidia_smi_encoder_stats_session_count	Current number of encoder sessions.
encoder_stats_average_fps	nvidia_smi_encoder_stats_average_fps	The moving average of the encode frames per second.
encoder_stats_average_latency	nvidia_smi_encoder_stats_average_latency	The moving average of the encode latency in microseconds.
clocks_current_graphics	nvidia_smi_clocks_current_graphics	The current frequency of the graphics (shader) clock.
clocks_current_sm	nvidia_smi_clocks_current_sm	The current frequency of the Streaming Multiprocessor (SM) clock.
clocks_current_memory	nvidia_smi_clocks_current_memory	The current frequency of the memory clock.

Metric	Metric name in CloudWatch	Description
clocks_current_video	nvidia_smi_clocks_current_video	The current frequency of the video (encoder plus decoder) clocks.

All of these metrics are collected with the following dimensions:

Dimension	Description
index	A unique identifier for the GPU on this server. Represents the NVIDIA Management Library (NVML) index of the device.
name	The type of GPU. For example, NVIDIA Tesla A100
host	The server host name.

Collect process metrics with the procstat plugin

The *procstat* plugin enables you to collect metrics from individual processes. It is supported on Linux servers and on servers running supported version of Windows Server.

Topics

- [Configuring the CloudWatch agent for procstat](#)
- [Metrics collected by procstat](#)
- [Viewing process metrics imported by the CloudWatch agent](#)

Configuring the CloudWatch agent for procstat

To use the procstat plugin, add a `procstat` section in the `metrics_collected` section of the CloudWatch agent configuration file. There are three ways to specify the processes to monitor. You can use only one of these methods, but you can use that method to specify one or more processes to monitor.

- `pid_file`: Selects processes by the names of the process identification number (PID) files they create.
- `exe`: Selects the processes that have process names that match the string that you specify, using regular expression matching rules. The match is a "contains" match, meaning that if you specify `agent` as the term to match, processes with names like `cloudwatchagent` match the term. For more information, see [Syntax](#).
- `pattern`: Selects processes by the command lines used to start the processes. All processes are selected that have command lines matching the specified string using regular expression matching rules. The entire command line is checked, including parameters and options used with the command.

The match is a "contains" match, meaning that if you specify `-c` as the term to match, processes with parameters like `-config` match the term.

- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

The CloudWatch agent uses only one of these methods, even if you include more than one of the above sections. If you specify more than one section, the CloudWatch agent uses the `pid_file` section if it is present. If not, it uses the `exe` section.

On Linux servers, the strings that you specify in an `exe` or `pattern` section are evaluated as regular expressions. On servers running Windows Server, these strings are evaluated as WMI queries. An example would be `pattern: "%apache%"`. For more information, see [LIKE Operator](#).

Whichever method you use, you can include an optional `metrics_collection_interval` parameter, which specifies how often in seconds to collect those metrics. If you omit this parameter, the default value of 60 seconds is used.

In the examples in the following sections, the `procstat` section is the only section included in the `metrics_collected` section of the agent configuration file. Actual configuration files can also include other sections in `metrics_collected`. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

Configuring with `pid_file`

The following example `procstat` section monitors the processes that create the PID files `example1.pid` and `example2.pid`. Different metrics are collected from each process. Metrics collected from the process that creates `example2.pid` are collected every 10 seconds, and the metrics collected from the `example1.pid` process are collected every 60 seconds, the default value.

```
{
  "metrics": {
    "metrics_collected": {
      "procstat": [
        {
          "pid_file": "/var/run/example1.pid",
          "measurement": [
            "cpu_usage",
            "memory_rss"
          ]
        },
        {
          "pid_file": "/var/run/example2.pid",
          "measurement": [
            "read_bytes",
            "read_count",
            "write_bytes"
          ],
          "metrics_collection_interval": 10
        }
      ]
    }
  }
}
```



```

    }
  }
}

```

Configuring with exe

The following example `procstat` section monitors all processes with names that match the strings `agent` or `plugin`. The same metrics are collected from each process.

```

{
  "metrics": {
    "metrics_collected": {
      "procstat": [
        {
          "exe": "agent",
          "measurement": [
            "cpu_time",
            "cpu_time_system",
            "cpu_time_user"
          ]
        },
        {
          "exe": "plugin",
          "measurement": [
            "cpu_time",
            "cpu_time_system",
            "cpu_time_user"
          ]
        }
      ]
    }
  }
}

```

Configuring with pattern

The following example `procstat` section monitors all processes with command lines that match the strings `config` or `-c`. The same metrics are collected from each process.

```

{
  "metrics": {
    "metrics_collected": {

```

```

    "procstat": [
      {
        "pattern": "config",
        "measurement": [
          "rlimit_memory_data_hard",
          "rlimit_memory_data_soft",
          "rlimit_memory_stack_hard",
          "rlimit_memory_stack_soft"
        ]
      },
      {
        "pattern": "-c",
        "measurement": [
          "rlimit_memory_data_hard",
          "rlimit_memory_data_soft",
          "rlimit_memory_stack_hard",
          "rlimit_memory_stack_soft"
        ]
      }
    ]
  }
}

```

Metrics collected by procstat

The following table lists the metrics that you can collect with the `procstat` plugin.

The CloudWatch agent adds `procstat` to the beginning of the following metric names. There is a different syntax depending on whether it was collected from a Linux server or a server running Windows Server. For example, the `cpu_time` metric appears as `procstat_cpu_time` when collected from Linux and as `procstat cpu_time` when collected from Windows Server.

Metric name	Available on	Description
<code>cpu_time</code>	Linux	The amount of time that the process uses the CPU. This metric is measured in

Metric name	Available on	Description
		hundredths of a second. Unit: Count
cpu_time_guest	Linux	The amount of time that the process is in guest mode. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_guest_nice	Linux	The amount of time that the process is running in a niced guest. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
<code>cpu_time_idle</code>	Linux	<p>The amount of time that the process is in idle mode. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>
<code>cpu_time_iowait</code>	Linux	<p>The amount of time that the process is waiting for I/O operations to complete. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>

Metric name	Available on	Description
cpu_time_irq	Linux	<p>The amount of time that the process is servicing interrupts. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>
cpu_time_nice	Linux	<p>The amount of time that the process is in nice mode. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>

Metric name	Available on	Description
cpu_time_soft_irq	Linux	<p>The amount of time that the process is servicing software interrupts. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>
cpu_time_steal	Linux	<p>The amount of time spent running in other operating systems when running in a virtualized environment. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>

Metric name	Available on	Description
cpu_time_stolen	Linux, Windows Server	<p>The amount of time that the process is in <i>stolen time</i>, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: None</p>
cpu_time_system	Linux, Windows Server, macOS	<p>The amount of time that the process is in system mode. This metric is measured in hundredths of a second.</p> <p>Type: Float</p> <p>Unit: Count</p>

Metric name	Available on	Description
cpu_time_user	Linux, Windows Server, macOS	The amount of time that the process is in user mode. This metric is measured in hundredths of a second. Unit: Count
cpu_usage	Linux, Windows Server, macOS	The percentage of time that the process is active in any capacity. Unit: Percent
memory_data	Linux, macOS	The amount of memory that the process uses for data. Unit: Bytes
memory_locked	Linux, macOS	The amount of memory that the process has locked. Unit: Bytes

Metric name	Available on	Description
memory_rss	Linux, Windows Server, macOS	The amount of real memory (resident set) that the process is using. Unit: Bytes
memory_stack	Linux, macOS	The amount of stack memory that the process is using. Unit: Bytes
memory_swap	Linux, macOS	The amount of swap memory that the process is using. Unit: Bytes
memory_vms	Linux, Windows Server, macOS	The amount of virtual memory that the process is using. Unit: Bytes

Metric name	Available on	Description
num_fds	Linux	The number of file descriptors that this process has open. Unit: None
num_threads	Linux, Windows, macOS	The number of threads in this process. Unit: None
pid	Linux, Windows Server, macOS	Process identifier (ID). Unit: None

Metric name	Available on	Description
pid_count	Linux, Windows Server, macOS	<p>The number of process IDs associated with the process.</p> <p>On Linux servers and macOS computers the full name of this metric is <code>procstat_lookup_pid_count</code> and on Windows Server it is <code>procstat_lookup_pid_count</code>.</p> <p>Unit: None</p>
read_bytes	Linux, Windows Server	<p>The number of bytes that the process has read from disks.</p> <p>Unit: Bytes</p>

Metric name	Available on	Description
write_bytes	Linux, Windows Server	The number of bytes that the process has written to disks. Unit: Bytes
read_count	Linux, Windows Server	The number of disk read operations that the process has executed. Unit: None
rlimit_realttime_priority_hard	Linux	The hard limit on the real-time priority that can be set for this process. Unit: None
rlimit_realttime_priority_soft	Linux	The soft limit on the real-time priority that can be set for this process. Unit: None

Metric name	Available on	Description
<code>rlimit_signals_pending_hard</code>	Linux	The hard limit on maximum number of signals that can be queued by this process. Unit: None
<code>rlimit_signals_pending_soft</code>	Linux	The soft limit on maximum number of signals that can be queued by this process. Unit: None
<code>rlimit_nice_priority_hard</code>	Linux	The hard limit on the maximum nice priority that can be set by this process. Unit: None

Metric name	Available on	Description
<code>rlimit_nice_priority_soft</code>	Linux	The soft limit on the maximum nice priority that can be set by this process. Unit: None
<code>rlimit_num_fds_hard</code>	Linux	The hard limit on the maximum number of file descriptors that this process can have open. Unit: None
<code>rlimit_num_fds_soft</code>	Linux	The soft limit on the maximum number of file descriptors that this process can have open. Unit: None

Metric name	Available on	Description
<code>write_count</code>	Linux, Windows Server	The number of disk write operations that the process has executed. Unit: None
<code>involuntary_context_switches</code>	Linux	The number of times that the process was involuntarily context-switched. Unit: None
<code>voluntary_context_switches</code>	Linux	The number of times that the process was context-switched voluntarily. Unit: None
<code>realtime_priority</code>	Linux	The current usage of real-time priority for the process. Unit: None

Metric name	Available on	Description
nice_priority	Linux	The current usage of nice priority for the process. Unit: None
signals_pending	Linux	The number of signals pending to be handled by the process. Unit: None
rlimit_cpu_time_hard	Linux	The hard CPU time resource limit for the process. Unit: None
rlimit_cpu_time_soft	Linux	The soft CPU time resource limit for the process. Unit: None
rlimit_file_locks_hard	Linux	The hard file locks resource limit for the process. Unit: None

Metric name	Available on	Description
<code>rlimit_file_locks_soft</code>	Linux	The soft file locks resource limit for the process. Unit: None
<code>rlimit_memory_data_hard</code>	Linux	The hard resource limit on the process for memory used for data. Unit: Bytes
<code>rlimit_memory_data_soft</code>	Linux	The soft resource limit on the process for memory used for data. Unit: Bytes
<code>rlimit_memory_locked_hard</code>	Linux	The hard resource limit on the process for locked memory. Unit: Bytes

Metric name	Available on	Description
<code>rlimit_memory_locked_soft</code>	Linux	The soft resource limit on the process for locked memory. Unit: Bytes
<code>rlimit_memory_rss_hard</code>	Linux	The hard resource limit on the process for physical memory. Unit: Bytes
<code>rlimit_memory_rss_soft</code>	Linux	The soft resource limit on the process for physical memory. Unit: Bytes
<code>rlimit_memory_stack_hard</code>	Linux	The hard resource limit on the process stack. Unit: Bytes
<code>rlimit_memory_stack_soft</code>	Linux	The soft resource limit on the process stack. Unit: Bytes

Metric name	Available on	Description
<code>rlimit_memory_vms_hard</code>	Linux	The hard resource limit on the process for virtual memory. Unit: Bytes
<code>rlimit_memory_vms_soft</code>	Linux	The soft resource limit on the process for virtual memory. Unit: Bytes

Viewing process metrics imported by the CloudWatch agent

After importing process metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view process metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

To view process metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

- b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Retrieve custom metrics with StatsD

You can retrieve additional custom metrics from your applications or services using the CloudWatch agent with the StatsD protocol. StatsD is a popular open-source solution that can gather metrics from a wide variety of applications. StatsD is especially useful for instrumenting your own metrics. For an example of using the CloudWatch agent and StatsD together, see [How to better monitor your custom application metrics using Amazon CloudWatch Agent](#).

StatsD is supported on both Linux servers and servers running Windows Server. CloudWatch supports the following StatsD format:

```
MetricName:value | type | @sample_rate | #tag1:  
value, tag1...
```

- *MetricName* – A string with no colons, bars, # characters, or @ characters.
- *value* – This can be either integer or float.
- *type* – Specify c for counter, g for gauge, ms for timer, h for histogram, or s for set.
- *sample_rate* – (Optional) A float between 0 and 1, inclusive. Use only for counter, histogram, and timer metrics. The default value is 1 (sampling 100% of the time).
- *tags* – (Optional) A comma-separated list of tags. StatsD tags are similar to dimensions in CloudWatch. Use colons for key/value tags, such as `env:prod`.

You can use any StatsD client that follows this format to send the metrics to the CloudWatch agent. For more information about some of the available StatsD clients, see the [StatsD client page on GitHub](#).

To collect these custom metrics, add a `"statsd": {}` line to the `metrics_collected` section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it's done for you. For more information, see [Create the CloudWatch agent configuration file](#).

The StatsD default configuration works for most users. There are optional fields that you can add to the **statsd** section of the agent configuration file as needed:

- `service_address` – The service address to which the CloudWatch agent should listen. The format is *ip:port*. If you omit the IP address, the agent listens on all available interfaces. Only the UDP format is supported, so you don't need to specify a UDP prefix.

The default value is `:8125`.

- `metrics_collection_interval` – How often in seconds that the StatsD plugin runs and collects metrics. The default value is 10 seconds. The range is 1–172,000.
- `metrics_aggregation_interval` – How often in seconds CloudWatch aggregates metrics into single data points. The default value is 60 seconds.

For example, if `metrics_collection_interval` is 10 and `metrics_aggregation_interval` is 60, CloudWatch collects data every 10 seconds. After each minute, the six data readings from that minute are aggregated into a single data point, which is sent to CloudWatch.

The range is 0–172,000. Setting `metrics_aggregation_interval` to 0 disables the aggregation of StatsD metrics.

- `allowed_pending_messages` – The number of UDP messages that are allowed to queue up. When the queue is full, the StatsD server starts dropping packets. The default value is 10000.
- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

The following is an example of the **statsd** section of the agent configuration file, using the default port and custom collection and aggregation intervals.

```
{
  "metrics":{
    "metrics_collected":{
      "statsd":{
```

```
        "service_address":":8125",
        "metrics_collection_interval":60,
        "metrics_aggregation_interval":300
    }
}
}
```

Viewing StatsD metrics imported by the CloudWatch agent

After importing StatsD metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view StatsD metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

To view StatsD metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Retrieve custom metrics with collectd

You can retrieve additional metrics from your applications or services using the CloudWatch agent with the collectd protocol, which is supported only on Linux servers. collectd is a popular open-source solution with plugins that can gather system statistics for a wide variety of applications. By

combining the system metrics that the CloudWatch agent can already collect with the additional metrics from `collectd`, you can better monitor, analyze, and troubleshoot your systems and applications. For more information about `collectd`, see [collectd - The system statistics collection daemon](#).

You use the `collectd` software to send the metrics to the CloudWatch agent. For the `collectd` metrics, the CloudWatch agent acts as the server while the `collectd` plugin acts as the client.

The `collectd` software is not installed automatically on every server. On a server running Amazon Linux 2, follow these steps to install `collectd`

```
sudo amazon-linux-extras install collectd
```

For information about installing `collectd` on other systems, see the [Download page for collectd](#).

To collect these custom metrics, add a `"collectd": {}` line to the `metrics_collected` section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it is done for you. For more information, see [Create the CloudWatch agent configuration file](#).

Optional parameters are also available. If you are using `collectd` and you do not use `/etc/collectd/auth_file` as your `collectd_auth_file`, you must set some of these options.

- **service_address:** The service address to which the CloudWatch agent should listen. The format is `"udp://ip:port"`. The default is `udp://127.0.0.1:25826`.
- **name_prefix:** A prefix to attach to the beginning of the name of each `collectd` metric. The default is `collectd_`. The maximum length is 255 characters.
- **collectd_security_level:** Sets the security level for network communication. The default is **encrypt**.

encrypt specifies that only encrypted data is accepted. **sign** specifies that only signed and encrypted data is accepted. **none** specifies that all data is accepted. If you specify a value for `collectd_auth_file`, encrypted data is decrypted if possible.

For more information, see [Client setup](#) and [Possible interactions](#) in the `collectd` Wiki.

- **collectd_auth_file** Sets a file in which user names are mapped to passwords. These passwords are used to verify signatures and to decrypt encrypted network packets. If given, signed data is verified and encrypted packets are decrypted. Otherwise, signed data is accepted without checking the signature and encrypted data cannot be decrypted.

The default is `/etc/collectd/auth_file`.

If `collectd_security_level` is set to `none`, this is optional. If you set `collectd_security_level` to `encrypt` or `sign`, you must specify `collectd_auth_file`.

For the format of the auth file, each line is a user name followed by a colon and any number of spaces followed by the password. For example:

```
user1: user1_password
```

```
user2: user2_password
```

- **collectd_typesdb:** A list of one or more files that contain the dataset descriptions. The list must be surrounded by brackets, even if there is just one entry in the list. Each entry in the list must be surrounded by double quotes. If there are multiple entries, separate them with commas. The default on Linux servers is `["/usr/share/collectd/types.db"]`. The default on macOS computers depends on the version of collectd. For example, `["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]`.

For more information, see <https://www.collectd.org/documentation/manpages/types.db.html>.

- **metrics_aggregation_interval:** How often in seconds CloudWatch aggregates metrics into single data points. The default is 60 seconds. The range is 0 to 172,000. Setting it to 0 disables the aggregation of collectd metrics.

The following is an example of the collectd section of the agent configuration file.

```
{
  "metrics":{
    "metrics_collected":{
      "collectd":{
        "name_prefix":"My_collectd_metrics_",
        "metrics_aggregation_interval":120
      }
    }
  }
}
```


Viewing collectd metrics imported by the CloudWatch agent

After importing collectd metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view collectd metrics as a time series graph. For more information about setting alarms, see [Using Amazon CloudWatch alarms](#).

To view collectd metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - b. To sort the table, use the column heading.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions, Add to dashboard**.

Set up and configure Prometheus metrics collection on Amazon EC2 instances

The following sections explain how to install the CloudWatch agent with Prometheus monitoring on EC2 instances, and how to configure the agent to scrape additional targets. It also provides tutorials for setting up sample workloads to use testing with Prometheus monitoring.

For information about the operating systems supported by the CloudWatch agent, see [Collect metrics, logs, and traces with the CloudWatch agent](#)

VPC security group requirements

If you are using a VPC, the following requirements apply.

- The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.
- The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- [Step 1: Install the CloudWatch agent](#)
- [Step 2: Scrape Prometheus sources and import metrics](#)
- [Example: Set up Java/JMX sample workloads for Prometheus metric testing](#)

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see [Installing the CloudWatch agent](#).

Step 2: Scrape Prometheus sources and import metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in [<scrape_config>](#) in the Prometheus documentation. The other is for the CloudWatch agent configuration.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in [<scrape_config>](#) in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. A sample configuration file contains the following global configuration lines:

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
```

```
- files: ["C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_1.yaml",  
"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_2.yaml"]
```

The `global` section specifies parameters that are valid in all configuration contexts. They also serve as defaults for other configuration sections. It contains the following parameters:

- `scrape_interval`— Defines how frequently to scrape targets.
- `scrape_timeout`— Defines how long to wait before a scrape request times out.

The `scrape_configs` section specifies a set of targets and parameters that define how to scrape them. It contains the following parameters:

- `job_name`— The job name assigned to scraped metrics by default.
- `sample_limit`— Per-scrape limit on the number of scraped samples that will be accepted.
- `file_sd_configs`— List of file service discovery configurations. It reads a set of files containing a list of zero or more static configs. The `file_sd_configs` section contains a `files` parameter which defines patterns for files from which target groups are extracted.

The CloudWatch agent supports the following service discovery configuration types.

static_config Allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

The following is a sample static config to scrape Prometheus metrics from a local host. Metrics can also be scraped from other servers if the Prometheus port is open to the server where the agent runs.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml  
- targets:  
  - 127.0.0.1:9404  
labels:  
  key1: value1  
  key2: value2
```

This example contains the following parameters:

- `targets`— The targets scraped by the static config.
- `labels`— Labels assigned to all metrics that are scraped from the targets.

ec2_sd_config Allows retrieving scrape targets from Amazon EC2 instances. The following is a sample `ec2_sd_config` to scrape Prometheus metrics from a list of EC2 instances. The Prometheus ports of these instances have to open to the server where the CloudWatch agent runs. The IAM role for the EC2 instance where the CloudWatch agent runs must include the `ec2:DescribeInstance` permission. For example, you could attach the managed policy **AmazonEC2ReadOnlyAccess** to the instance running the CloudWatch agent.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

This example contains the following parameters:

- **region**— The AWS Region where the target EC2 instance is. If you leave this blank, the Region from the instance metadata is used.
- **port**— The port to scrape metrics from.
- **filters**— Optional filters to use to filter the instance list. This example filters based on EC2 instance IDs. For more criteria that you can filter on, see [DescribeInstances](#).

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file includes `prometheus` sections under both `logs` and `metrics_collected`. It includes the following parameters.

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics.

- **prometheus_config_path**— specifies the Prometheus scrape configuration file path.
- **emf_processor**— specifies the embedded metric format processor configuration. For more information about embedded metric format, see [Embedding metrics within logs](#).

The `emf_processor` section can contain the following parameters:

- **metric_declaration_dedup**— If set to true, the de-duplication function for the embedded metric format metrics is enabled.
- **metric_namespace**— Specifies the metric namespace for the emitted CloudWatch metrics.
- **metric_unit**— Specifies the metric name:metric unit map. For information about supported metric units, see [MetricDatum](#).
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
 - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
 - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.
 - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
 - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

The following is an example CloudWatch agent configuration for Prometheus.

```
{
  "logs":{
    "metrics_collected":{
      "prometheus":{
        "cluster_name":"prometheus-cluster",
        "log_group_name":"Prometheus",
        "prometheus_config_path":"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\
\\prometheus.yaml",
        "emf_processor":{
          "metric_declaration_dedup":true,
          "metric_namespace":"CWAgent-Prometheus",
```



```

"CloudWatchMetrics": [
  {
    "Metrics": [
      {
        "Unit": "Count",
        "Name": "jvm_threads_current"
      },
      {
        "Unit": "Milliseconds",
        "Name": "jvm_gc_collection_seconds_sum"
      }
    ],
    "Dimensions": [
      [
        "key1",
        "key2"
      ],
      [
        "key2"
      ]
    ],
    "Namespace": "CWAgent-Prometheus"
  }
],
"ClusterName": "prometheus-cluster",
"InstanceId": "i-0e45bd06f196096c8",
"Timestamp": "1607966368109",
"Version": "0",
"host": "EC2AMAZ-PDD0IUM",
"instance": "127.0.0.1:9404",
"jvm_threads_current": 2,
"jvm_gc_collection_seconds_sum": 0.0060000000000000002,
"prom_metric_type": "gauge",
...
}

```

Example: Set up Java/JMX sample workloads for Prometheus metric testing

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see [prometheus/jmx_exporter](#).

The CloudWatch agent can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Hjava, and Tomcat (Catalina), from a JMX exporter on EC2 instances.

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see [Installing the CloudWatch agent](#).

Step 2: Start the Java/JMX workload

The next step is to start the Java/JMX workload.

First, download the latest JMX exporter jar file from the following location: [prometheus/jmx_exporter](#).

Use the jar for your sample application

The example commands in the following sections use `SampleJavaApplication-1.0-SNAPSHOT.jar` as the jar file. Replace these parts of the commands with the jar for your application.

Prepare the JMX exporter configuration

The `config.yaml` file is the JMX exporter configuration file. For more information, see [Configuration](#) in the JMX exporter documentation.

Here is a sample configuration for Java and Tomcat.

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\\w+-\\w+)-(\\d+)\"><>(\\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
```



```

    port: "$2"
    protocol: "$1"
    help: Catalina global $3
    type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//[(-a-zA-Z0-9+&@#/%=?~_!|:.,;]*[-a-zA-Z0-9+&@#/%=?~_!|:.,;]*), name=(-a-zA-Z0-9+/$%~_!|.)*, J2EEApplication=none, J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\\w+-\\w+)-(\\d+)"><>(currentThreadCount|currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=(-a-zA-Z0-9+&@#/%=?~_!|:.,;)*[-a-zA-Z0-9+&@#/%=?~_!|:.,;]*), context=(-a-zA-Z0-9+/$%~_!|.)*><>(processingTime|sessionCounter|rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"

```

Start the Java application with the Prometheus exporter

Start the sample application. This will emit Prometheus metrics to port 9404. Be sure to replace the entry point `com.gubupt.sample.app.App` with the correct information for your sample java application.

On Linux, enter the following command.

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp
./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

On Windows, enter the following command.

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp .
.\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

Verify the Prometheus metrics emission

Verify that Prometheus metrics are being emitted.

On Linux, enter the following command.

```
$ curl localhost:9404
```

On Windows, enter the following command.

```
PS C:\> curl http://localhost:9404
```

Sample output on Linux:

```
StatusCode      : 200
StatusDescription : OK
Content         : # HELP jvm_classes_loaded The number of classes that are currently
loaded in the JVM
                # TYPE jvm_classes_loaded gauge
                jvm_classes_loaded 2526.0
                # HELP jvm_classes_loaded_total The total number of class...
RawContent      : HTTP/1.1 200 OK
                Content-Length: 71908
                Content-Type: text/plain; version=0.0.4; charset=utf-8
                Date: Fri, 18 Dec 2020 16:38:10 GMT

                # HELP jvm_classes_loaded The number of classes that are
currentl...
Forms           : {}
Headers         : [[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
charset=utf-8], [Date, Fri, 18
                Dec 2020 16:38:10 GMT]]
Images          : {}
InputFields     : {}
```

```
Links          : {}
ParsedHtml     : System.__ComObject
RawContentLength : 71908
```

Step 3: Configure the CloudWatch agent to scrape Prometheus metrics

Next, set up the Prometheus scrape configuration in the CloudWatch agent configuration file.

To set up the Prometheus scrape configuration for the Java/JMX example

1. Set up the configuration for `file_sd_config` and `static_config`.

On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: [ "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\
prometheus_file_sd.yaml" ]
```

2. Set up the scrape targets configuration.

On Linux, enter the following command.

```
$ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
```

```
- targets:
  - 127.0.0.1:9404
labels:
  application: sample_java_app
  os: linux
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
labels:
  application: sample_java_app
  os: windows
```

3. Set up the Prometheus scrape configuration by `ec2_sc_config`. Replace *your-ec2-instance-id* with the correct EC2 instance ID.

On Linux, enter the following command.

```
$ cat .\prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
labels:
  application: sample_java_app
```

```
os: windows
```

4. Set up the CloudWatch agent configuration. First, navigate to the correct directory. On Linux, it is `/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json`. On Windows, it is `C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json`.

The following is a sample configuration with Java/JMX Prometheus metrics defined. Be sure to replace *path-to-Prometheus-Scrape-Configuration-file* with the correct path.

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "prometheus": {
        "cluster_name": "my-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",
        "emf_processor": {
          "metric_declaration_dedup": true,
          "metric_namespace": "PrometheusTest",
          "metric_unit": {
            "jvm_threads_current": "Count",
            "jvm_classes_loaded": "Count",
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",
            "catalina_manager_activesessions": "Count",
            "jvm_gc_collection_seconds_sum": "Seconds",
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",
            "jvm_memory_bytes_used": "Bytes",
            "jvm_memory_pool_bytes_used": "Bytes"
          }
        },
        "metric_declaration": [
          {
            "source_labels": ["job"],
            "label_matcher": "^jmx$",
            "dimensions": [["instance"]],
            "metric_selectors": [
              "^jvm_threads_current$",
              "^jvm_classes_loaded$",
              "^java_lang_operatingsystem_freephysicalmemorysize$",
              "^catalina_manager_activesessions$",
              "^jvm_gc_collection_seconds_sum$",
            ]
          }
        ]
      }
    }
  }
}
```

```
        "^catalina_globalrequestprocessor_bytesreceived$"
    ]
  },
  {
    "source_labels": ["job"],
    "label_matcher": "^jmx$",
    "dimensions": [["area"]],
    "metric_selectors": [
      "^jvm_memory_bytes_used$"
    ]
  },
  {
    "source_labels": ["job"],
    "label_matcher": "^jmx$",
    "dimensions": [["pool"]],
    "metric_selectors": [
      "^jvm_memory_pool_bytes_used$"
    ]
  }
]
}
},
"force_flush_interval": 5
}
}
```

5. Restart the CloudWatch agent by entering one of the following commands.

On Linux, enter the following command.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
```

On Windows, enter the following command.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json
```

Viewing the Prometheus metrics and logs

You can now view the Java/JMX metrics being collected.

To view the metrics for your sample Java/JMX workload

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **PrometheusTest** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **prometheus-test**.

Install the CloudWatch agent by using the Amazon CloudWatch Observability EKS add-on or the Helm chart

You can use either the Amazon CloudWatch Observability EKS add-on or the Amazon CloudWatch Observability Helm chart to install the CloudWatch Agent and the Fluent-bit agent on an Amazon EKS cluster. You can also use the Helm chart to install the CloudWatch Agent and the Fluent-bit agent on a Kubernetes cluster that is not hosted on Amazon EKS.

Using either method on an Amazon EKS cluster enables both [Container Insights](#) with enhanced observability for Amazon EKS and [CloudWatch Application Signals](#) by default. Both features help you to collect infrastructure metrics, application performance telemetry, and container logs from the cluster.

With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics are charged per observation instead of being charged per metric stored or log ingested. For Application Signals, billing is based on inbound requests to your applications, outbound requests from your applications, and each configured service level objective (SLO). Each inbound request received generates one application signal, and each outbound request made generates one application signal. Every SLO creates two application signals per measurement period. For more information about CloudWatch pricing, see [Amazon CloudWatch Pricing](#).

Both methods enable Container Insights on both Linux and Windows worker nodes in the Amazon EKS cluster. To enable Container Insights on Windows, you must use version 1.5.0 or later of the Amazon EKS add-on or the Helm chart. Currently, Application Signals is not supported on Windows in Amazon EKS clusters.

The Amazon CloudWatch Observability EKS add-on is supported on Amazon EKS clusters running with Kubernetes version 1.23 or later.

When you install the add-on or the Helm chart, you must also grant IAM permissions to enable the CloudWatch agent to send metrics, logs, and traces to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This option grants permissions to worker nodes to send telemetry to CloudWatch.
- Use an IAM role for service accounts for the agent pods, and attach the policy to this role. This works only for Amazon EKS clusters. This option gives CloudWatch access only to the appropriate agent pods.

Option 1: Install with IAM permissions on worker nodes

To use this method, first attach the **CloudWatchAgentServerPolicy** IAM policy to your worker nodes by entering the following command. In this command, replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \  
--role-name my-worker-node-role \  
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Then install the Amazon CloudWatch Observability EKS add-on. To install the add-on, you can use the AWS CLI, the console, AWS CloudFormation, or Terraform.

AWS CLI

To use the AWS CLI to install the Amazon CloudWatch Observability EKS add-on

Enter the following command. Replace *my-cluster-name* with the name of your cluster.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-name my-  
cluster-name
```

Amazon EKS console

To use the Amazon EKS console to add the Amazon CloudWatch Observability EKS add-on

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose **Clusters**.

3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
 - a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.
 - b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:
 - a. Select the **Version** you'd like to use.
 - b. For **Select IAM role**, select **Inherit from node**
 - c. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.
 - d. Choose **Next**.
8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

AWS CloudFormation

To use AWS CloudFormation to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see [AWS::EKS::Addon](#).

```
{
  "Resources": {
    "EKSAAddOn": {
      "Type": "AWS::EKS::Addon",
      "Properties": {
        "AddonName": "amazon-cloudwatch-observability",
```

```
        "ClusterName": "my-cluster-name"
    }
}
}
```

Helm chart

To use the amazon-cloudwatch-observability Helm chart

1. You must have Helm installed to use this chart. For more information about installing Helm, see the [Helm documentation](#).
2. After you have installed Helm, enter the following commands. Replace *my-cluster-name* with the name of your cluster, and replace *my-cluster-region* with the Region that the cluster runs in.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update aws-observability
helm install --wait --create-namespace --namespace amazon-cloudwatch amazon-
cloudwatch-observability aws-observability/amazon-cloudwatch-observability --set
clusterName=my-cluster-name --set region=my-cluster-region
```

Terraform

To use Terraform to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see [Resource: aws_eks_addon](#).

```
resource "aws_eks_addon" "example" {
  addon_name = "amazon-cloudwatch-observability"
  cluster_name = "my-cluster-name"
}
```

Option 2: Install using IAM service account role (applies only to using the add-on)

This method is valid only if you are using the Amazon CloudWatch Observability EKS add-on. Before using this method, verify the following prerequisites:

- You have a functional Amazon EKS cluster with nodes attached in one of the AWS Regions that supports Container Insights. For the list of supported Regions, see [Container Insights](#).
- You have `kubectl` installed and configured for the cluster. For more information, see [Installing kubectl](#) in the *Amazon EKS User Guide*.
- You have `eksctl` installed. For more information, see [Installing or updating eksctl](#) in the *Amazon EKS User Guide*.

To install the Amazon CloudWatch Observability EKS add-on using the IAM service account role

1. Enter the following command to create an OpenID Connect (OIDC) provider, if the cluster doesn't have one already. For more information, see [Configuring a Kubernetes service account to assume an IAM role](#) in the *Amazon EKS User Guide*.

```
eksctl utils associate-iam-oidc-provider --cluster my-cluster-name --approve
```

2. Enter the following command to create the IAM role with the **CloudWatchAgentServerPolicy** policy attached, and configure the agent service account to assume that role using OIDC. Replace *my-cluster-name* with the name of your cluster, and replace *my-service-account-role* with the name of the role that you want to associate the service account with. If the role doesn't already exist, `eksctl` creates it for you.

```
eksctl create iamserviceaccount \  
  --name cloudwatch-agent \  
  --namespace amazon-cloudwatch --cluster my-cluster-name \  
  --role-name my-service-account-role \  
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \  
  --role-only \  
  --approve
```

3. Install the add-on by entering the following command. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-service-account-role* with the IAM role created in the previous step.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-  
name my-cluster-name --service-account-role-arn arn:aws:iam::111122223333:role/my-  
service-account-role
```

(Optional) Additional configuration

Topics

- [Opt out of collecting container logs](#)
- [Use a custom Fluent Bit configuration](#)
- [Manage Kubernetes tolerations for the installed pod workloads](#)
- [Opt out of accelerated compute metrics collection](#)
- [Use a custom CloudWatch agent configuration](#)
- [Manage admission webhook TLS certificates](#)
- [Collect Amazon EBS volume IDs](#)

Opt out of collecting container logs

By default, the add-on uses Fluent Bit to collect container logs from all pods and then sends the logs to CloudWatch Logs. For information about which logs are collected, see [Setting up Fluent Bit](#).

Note

Neither the add-on or the Helm chart manage existing Fluentd or Fluent Bit resources in a cluster. You can delete the existing Fluentd or Fluent Bit resources before installing the add-on or Helm chart. Alternatively, to keep your existing setup and avoid having the add-on or the Helm chart from also installing Fluent Bit, you can disable it by following the instructions in this section.

To opt out of the collection of container logs if you are using the Amazon CloudWatch Observability EKS add-on, pass the following option when you create or update the add-on:

```
--configuration-values '{ "containerLogs": { "enabled": false } }'
```

To opt out of the collection of container logs if you are using the Helm chart, pass the following option when you create or update the add-on:

```
--set containerLogs.enabled=false
```

Use a custom Fluent Bit configuration

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, you can modify the Fluent Bit configuration when you create or update the add-on or Helm chart. You supply the custom Fluent Bit configuration in the `containerLogs` root level section of the advanced configuration of the add-on or the value overrides in the Helm chart. Within this section, you supply the custom Fluent Bit configuration in the `config` section (for Linux) or `configWindows` section (for Windows). The `config` is further broken down into the following sub-sections:

- `service`– This section represents the SERVICE config to define the global behavior of the Fluent Bit engine.
- `customParsers`– This section represents any global PARSERS that you want to include that are capable of taking unstructured log entries and giving them a structure to make it easier for processing and further filtering.
- `extraFiles`– This section can be used to provide additional Fluent Bit conf files to be included. By default, the following 3 conf files are included:
 - `application-log.conf`– A conf file for sending application logs from your cluster to the log group `/aws/containerinsights/my-cluster-name/application` in CloudWatch Logs.
 - `dataplane-log.conf`– A conf file for sending logs corresponding to your cluster's data plane components including the CRI logs, kubelet logs, kube-proxy logs and Amazon VPC CNI logs to the log group `/aws/containerinsights/my-cluster-name/dataplane` in CloudWatch Logs.
 - `host-log.conf`– A conf for sending logs from `/var/log/dmesg`, `/var/log/messages`, and `/var/log/secure` on Linux, and `System winlogs` on Windows, to the log group `/aws/containerinsights/my-cluster-name/host` in CloudWatch.

Note

Provide the full configuration for each of these individual sections even if you are modifying only one field within a sub-section. We recommend that you use the default configuration provided below as a baseline and then modify it accordingly so that you don't disable functionality that is enabled by default. You can use the following YAML configuration when modifying the advanced config for the Amazon EKS add-on or when you supply value overrides for the Helm chart.

To find the config section for your cluster, see [aws-observability / helm-charts](#) on GitHub and find the release corresponding to the version of the add-on or Helm chart that you are installing. Then navigate to `/charts/amazon-cloudwatch-observability/values.yaml` to find the config section (for Linux) and `configWindows` section (for Windows) within the `fluentBit` section under `containerLogs`.

As an example, the default Fluent Bit configuration for version 1.7.0 can be found [here](#).

We recommend that you provide the config as YAML when you supply it using the Amazon EKS add-on's advanced config or when you supply it as value overrides for your Helm installation. Be sure that the YAML conforms to the following structure.

```
containerLogs:
  fluentBit:
    config:
      service: |
        ...
      customParsers: |
        ...
      extraFiles:
        application-log.conf: |
          ...
        dataplane-log.conf: |
          ...
        host-log.conf: |
          ...
```

The following example config changes the global setting for the flush interval to be 45 seconds. Even though the only modification is to the `Flush` field, you must still provide the full `SERVICE` definition for the service sub-section. Because this example didn't specify overrides for the other sub-sections, the defaults are used for them.

```
containerLogs:
  fluentBit:
    config:
      service: |
        [SERVICE]
        Flush           45
        Grace           30
        Log_Level       error
        Daemon          off
```

```

Parsers_File          parsers.conf
storage.path          /var/fluent-bit/state/flb-storage/
storage.sync          normal
storage.checksum      off
storage.backlog.mem_limit 5M

```

The following example configuration includes an extra Fluent bit conf file. In this example, we are adding a custom `my-service.conf` under `extraFiles` and it will be included in addition to the three default `extraFiles`.

```

containerLogs:
  fluentBit:
    config:
      extraFiles:
        my-service.conf: |
          [INPUT]
            Name          tail
            Tag           myservice.*
            Path          /var/log/containers/*myservice*.log
            DB            /var/fluent-bit/state/flb_myservice.db
            Mem_Buf_Limit 5MB
            Skip_Long_Lines On
            Ignore_Older  1d
            Refresh_Interval 10

          [OUTPUT]
            Name          cloudwatch_logs
            Match         myservice.*
            region        ${AWS_REGION}
            log_group_name /aws/containerinsights/${CLUSTER_NAME}/myservice
            log_stream_prefix ${HOST_NAME}-
            auto_create_group true

```

The next example removes an existing conf file entirely from `extraFiles`. This excludes the `application-log.conf` entirely by overriding it with an empty string. Simply omitting `application-log.conf` from `extraFiles` would instead imply to use the default, which is not what we are trying to achieve in this example. The same applies to removing any custom conf file that you might have previously added to `extraFiles`.

```

containerLogs:
  fluentBit:
    config:

```

```
extraFiles:
  application-log.conf: ""
```

Manage Kubernetes tolerations for the installed pod workloads

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, the add-on and the Helm chart by default set Kubernetes *tolerations* to tolerate all taints on the pod workloads that are installed by the add-on or the Helm chart. This ensures that daemonsets such as the CloudWatch agent and Fluent Bit can schedule pods on all nodes in your cluster by default. For more information about tolerations and taints, see [Taints and Tolerations](#) in the Kubernetes documentation.

The default tolerations set by the add-on or the Helm chart are as follows:

```
tolerations:
- operator: Exists
```

You can override the default tolerations by setting the `tolerations` field at the root level when using the add-on advanced config or when you install or upgrade the Helm chart with value overrides. An example would look like the following:

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

To omit tolerations completely, you can use a config that looks like the following:

```
tolerations: []
```

Any changes to tolerations apply to all pod workloads that are installed by the add-on or the Helm chart.

Opt out of accelerated compute metrics collection

By default, Container Insights with enhanced observability collects metrics for Accelerated Compute monitoring, including NVIDIA GPU metrics, AWS Neuron metrics for AWS Trainium and AWS Inferentia, and AWS Elastic Fabric Adapter (EFA) metrics.

NVIDIA GPU metrics from Amazon EKS workloads are collected by default beginning with version `v1.3.0-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300034.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see [NVIDIA GPU metrics](#).

AWS Neuron metrics for AWS Trainium and AWS Inferentia accelerators are collected by default beginning with version `v1.5.0-eksbuild.1` of the EKS add-on or the Helm chart, and version `1.300036.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see [AWS Neuron metrics for AWS Trainium and AWS Inferentia](#).

AWS Elastic Fabric Adapter (EFA) metrics from Linux nodes on Amazon EKS clusters are collected by default beginning with version `v1.5.2-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300037.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see [AWS Elastic Fabric Adapter \(EFA\) metrics](#).

You can opt out of collecting these metrics by setting the `accelerated_compute_metrics` field in the CloudWatch agent configuration file to `false`. This field is in the `kubernetes` section of the `metrics_collected` section in the CloudWatch configuration file. The following is an example of an opt-out configuration. For more information about how to use custom CloudWatch agent configurations, see the following section, [Use a custom CloudWatch agent configuration](#).

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true,
        "accelerated_compute_metrics": false
      }
    }
  }
}
```

Use a custom CloudWatch agent configuration

To collect other metrics, logs or traces using the CloudWatch agent, you can specify a custom configuration while also keeping Container Insights and CloudWatch Application Signals enabled. To do so, embed the CloudWatch agent configuration file within the `config` key under the `agent` key of the advanced configuration that you can use when creating or updating the EKS add-on or the Helm chart. The following represents the default agent configuration when you do not provide any additional configuration.

⚠ Important

Any custom configuration that you provide using additional configuration settings overrides the default configuration used by the agent. Be cautious not to unintentionally disable functionality that is enabled by default, such as Container Insights with enhanced observability and CloudWatch Application Signals. In the scenario that you are required to provide a custom agent configuration, we recommend using the following default configuration as a baseline and then modifying it accordingly.

- For using the Amazon CloudWatch observability EKS add-on

```
--configuration-values '{
  "agent": {
    "config": {
      "logs": {
        "metrics_collected": {
          "application_signals": {},
          "kubernetes": {
            "enhanced_container_insights": true
          }
        }
      },
      "traces": {
        "traces_collected": {
          "application_signals": {}
        }
      }
    }
  }
}'
```

- For using the Helm chart

```
--set agent.config='{
  "logs": {
    "metrics_collected": {
      "application_signals": {},
      "kubernetes": {
        "enhanced_container_insights": true
      }
    }
  }
}'
```

```
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}'
```

The following example shows the default agent configuration for the CloudWatch agent on Windows. The CloudWatch agent on Windows does not support custom configuration.

```
{
  "logs": {
    "metrics_collected": {
      "kubernetes": {
        "enhanced_container_insights": true
      },
    }
  }
}
```

Manage admission webhook TLS certificates

The Amazon CloudWatch Observability EKS add-on and the Helm chart leverage Kubernetes [admission webhooks](#) to validate and mutate AmazonCloudWatchAgent and Instrumentation custom resource (CR) requests, and optionally Kubernetes pod requests on the cluster if CloudWatch Application Signals is enabled. In Kubernetes, webhooks require a TLS certificate that the API server is configured to trust in order to ensure secure communication.

By default, the Amazon CloudWatch Observability EKS add-on and the Helm chart auto-generate a self-signed CA and a TLS certificate signed by this CA for securing the communication between the API server and the webhook server. This auto-generated certificate has a default expiry of 10 years and is not auto-renewed upon expiry. In addition, the CA bundle and the certificate are re-generated every time the add-on or Helm chart is upgraded or re-installed, thus resetting the expiry. If you want to change the default expiry of the auto-generated certificate, you can use the following additional configurations when creating or updating the add-on. Replace *expiry-in-days* with your desired expiry duration in days.

- Use this for the Amazon CloudWatch Observability EKS add-on

```
--configuration-values '{ "admissionWebhooks": { "autoGenerateCert":  
  { "expiryDays": expiry-in-days } } }'
```

- Use this for the Helm chart

```
--set admissionWebhooks.autoGenerateCert.expiryDays=expiry-in-days
```

For a more secure and feature-rich certificate authority solution, the add-on has opt-in support for [cert-manager](#), a widely-adopted solution for TLS certificate management in Kubernetes that simplifies the process of obtaining, renewing, managing and using those certificates. It ensures that certificates are valid and up to date, and attempts to renew certificates at a configured time before expiry. cert-manager also facilitates issuing certificates from a variety of supported sources, including [AWS Certificate Manager Private Certificate Authority](#).

We recommend that you review best practices for management of TLS certificates on your clusters and advise you to opt in to cert-manager for production environments. Note that if you opt-in to enabling cert-manager for managing the admission webhook TLS certificates, you are required to pre-install cert-manager on your Amazon EKS cluster before you install the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about available installation options, see [cert-manager documentation](#). After you install it, you can opt in to using cert-manager for managing the admission webhook TLS certificates using the following additional configuration.

- If you are using the Amazon CloudWatch Observability EKS add-on

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":  
  true } } }'
```

- If you are using the Helm chart

```
--set admissionWebhooks.certManager.enabled=true
```

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":  
  true } } }'
```

The advanced configuration discussed in this section will by default use a [SelfSigned](#) issuer.

Collect Amazon EBS volume IDs

If you want to collect Amazon EBS volume IDs in the performance logs, you must add another policy to the IAM role that is attached to the worker nodes or to the service account. Add the following as an inline policy. For more information, see [Adding and Removing IAM Identity Permissions](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:DescribeVolumes"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Troubleshooting the Amazon CloudWatch Observability EKS add-on or the Helm chart

Use the following information to help troubleshoot problems with the Amazon CloudWatch Observability EKS add-on or the Helm chart

Topics

- [Updating and deleting the Amazon CloudWatch Observability EKS add-on or the Helm chart](#)
- [Verify the version of the CloudWatch agent used by the Amazon CloudWatch Observability EKS add-on or the Helm chart](#)
- [Handling a ConfigurationConflict when managing the add-on or the Helm chart](#)

Updating and deleting the Amazon CloudWatch Observability EKS add-on or the Helm chart

For instructions about updating or deleting the Amazon CloudWatch Observability EKS add-on, see [Managing Amazon EKS add-ons](#). Use `amazon-cloudwatch-observability` as the name of the add-on.

To delete the Helm chart in a cluster, enter the following command.

```
helm delete amazon-cloudwatch -n amazon-cloudwatch --wait
```

Verify the version of the CloudWatch agent used by the Amazon CloudWatch Observability EKS add-on or the Helm chart

The Amazon CloudWatch Observability EKS add-on and the Helm chart installs a custom resource of kind `AmazonCloudWatchAgent` that controls the behavior of the CloudWatch agent daemonset on the cluster, including the version of the CloudWatch agent being used. You can get a list of all the `AmazonCloudWatchAgent` custom resources installed on your cluster `u` by entering the following command:

```
kubectl get amazoncloudwatchagent -A
```

In the output of this command, you should be able to check the version of the CloudWatch agent. Alternatively, you can also describe the `amazoncloudwatchagent` resource or one of the `cloudwatch-agent-*` pods running on your cluster to inspect the image being used.

Handling a ConfigurationConflict when managing the add-on or the Helm chart

When you install or update the Amazon CloudWatch Observability EKS add-on or the Helm chart, if you notice a failure caused by existing resources, it is likely because you already have the CloudWatch agent and its associated components such as the `ServiceAccount`, the `ClusterRole` and the `ClusterRoleBinding` installed on the cluster.

The error displayed by the add-on will include `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode,`

The error displayed by the Helm chart will be similar to `Error: INSTALLATION FAILED: Unable to continue with install and invalid ownership metadata..`

When the add-on or the Helm chart tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on or Helm chart. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the add-on or Helm chart when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see [Deleting the CloudWatch agent and Fluent Bit for Container Insights](#) for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the AWS CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

Metrics collected by the CloudWatch agent

You can collect metrics from servers by installing the CloudWatch agent on the server. You can install the agent on both Amazon EC2 instances and on-premises servers, and on computers running either Linux, Windows Server, or macOS. If you install the agent on an Amazon EC2 instance, the metrics it collects are in addition to the metrics enabled by default on Amazon EC2 instances.

For information about installing the CloudWatch agent on an instance, see [Collect metrics, logs, and traces with the CloudWatch agent](#).

All metrics discussed in this section are collected directly by the CloudWatch agent.

Metrics collected by the CloudWatch agent on Windows Server instances

On a server running Windows Server, installing the CloudWatch agent enables you to collect the metrics associated with the counters in Windows Performance Monitor. The CloudWatch metric names for these counters are created by putting a space between the object name and the counter

name. For example, the `% Interrupt Time` counter of the `Processor` object is given the metric name `Processor % Interrupt Time` in CloudWatch. For more information about Windows Performance Monitor counters, see the [Microsoft Windows Server documentation](#).

The default namespace for metrics collected by the CloudWatch agent is `CWAgent`, although you can specify a different namespace when you configure the agent.

Metrics collected by the CloudWatch agent on Linux and macOS instances

The following table lists the metrics that you can collect with the CloudWatch agent on Linux servers and macOS computers.

Metric	Description
<code>cpu_time_active</code>	<p>The amount of time that the CPU is active in any capacity. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
<code>cpu_time_guest</code>	<p>The amount of time that the CPU is running a virtual CPU for a guest operating system. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
<code>cpu_time_guest_nice</code>	<p>The amount of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
<code>cpu_time_idle</code>	<p>The amount of time that the CPU is idle. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>

Metric	Description
cpu_time_iowait	<p>The amount of time that the CPU is waiting for I/O operations to complete. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_time_irq	<p>The amount of time that the CPU is servicing interrupts. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_time_nice	<p>The amount of time that the CPU is in user mode with low-priority processes, which can easily be interrupted by higher-priority processes. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_time_softirq	<p>The amount of time that the CPU is servicing software interrupts. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_time_steal	<p>The amount of time that the CPU is in <i>stolen time</i>, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_time_system	<p>The amount of time that the CPU is in system mode. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>

Metric	Description
cpu_time_user	<p>The amount of time that the CPU is in user mode. This metric is measured in hundredths of a second.</p> <p>Unit: None</p>
cpu_usage_active	<p>The percentage of time that the CPU is active in any capacity.</p> <p>Unit: Percent</p>
cpu_usage_guest	<p>The percentage of time that the CPU is running a virtual CPU for a guest operating system.</p> <p>Unit: Percent</p>
cpu_usage_guest_nice	<p>The percentage of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes.</p> <p>Unit: Percent</p>
cpu_usage_idle	<p>The percentage of time that the CPU is idle.</p> <p>Unit: Percent</p>
cpu_usage_iowait	<p>The percentage of time that the CPU is waiting for I/O operations to complete.</p> <p>Unit: Percent</p>
cpu_usage_irq	<p>The percentage of time that the CPU is servicing interrupts.</p> <p>Unit: Percent</p>

Metric	Description
cpu_usage_nice	The percentage of time that the CPU is in user mode with low-priority processes, which higher-priority processes can easily interrupt. Unit: Percent
cpu_usage_softirq	The percentage of time that the CPU is servicing software interrupts. Unit: Percent
cpu_usage_steal	The percentage of time that the CPU is in <i>stolen time</i> , or time spent in other operating systems in a virtualized environment. Unit: Percent
cpu_usage_system	The percentage of time that the CPU is in system mode. Unit: Percent
cpu_usage_user	The percentage of time that the CPU is in user mode. Unit: Percent
disk_free	Free space on the disks. Unit: Bytes
disk_inodes_free	The number of available index nodes on the disk. Unit: Count
disk_inodes_total	The total number of index nodes reserved on the disk. Unit: Count

Metric	Description
disk_inodes_used	The number of used index nodes on the disk. Unit: Count
disk_total	Total space on the disks, including used and free. Unit: Bytes
disk_used	Used space on the disks. Unit: Bytes
disk_used_percent	The percentage of total disk space that is used. Unit: Percent
diskio_iops_in_progress	The number of I/O requests that have been issued to the device driver but have not yet completed. Unit: Count
diskio_io_time	The amount of time that the disk has had I/O requests queued. Unit: Milliseconds The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_reads	The number of disk read operations. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
diskio_read_bytes	<p>The number of bytes read from the disks.</p> <p>Unit: Bytes</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
diskio_read_time	<p>The amount of time that read requests have waited on the disks. Multiple read requests waiting at the same time increase the number. For example, if 5 requests all wait for an average of 100 milliseconds, 500 is reported.</p> <p>Unit: Milliseconds</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
diskio_writes	<p>The number disk write operations.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
diskio_write_bytes	<p>The number of bytes written to the disks.</p> <p>Unit: Bytes</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>

Metric	Description
diskio_write_time	<p>The amount of time that write requests have waited on the disks. Multiple write requests waiting at the same time increase the number. For example, if 8 requests all wait for an average of 1000 milliseconds, 8000 is reported.</p> <p>Unit: Milliseconds</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
ethtool_bw_in_allowance_exceeded	<p>The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
ethtool_bw_out_allowance_exceeded	<p>The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance..</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>

Metric	Description
ethtool_contrack_allowance_exceeded	<p>The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>
ethtool_linklocal_allowance_exceeded	<p>The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics</p> <p>Unit: None</p>

Metric	Description
ethtool_pps_allowance_exceeded	<p>The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance.</p> <p>This metric is collected only if you have listed it in the <code>ethtool</code> subsection of the <code>metrics_collected</code> section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics.</p> <p>Unit: None</p>
mem_active	<p>The amount of memory that has been used in some way during the last sample period.</p> <p>Unit: Bytes</p>
mem_available	<p>The amount of memory that is available and can be given instantly to processes.</p> <p>Unit: Bytes</p>
mem_available_percent	<p>The percentage of memory that is available and can be given instantly to processes.</p> <p>Unit: Percent</p>
mem_buffered	<p>The amount of memory that is being used for buffers.</p> <p>Unit: Bytes</p>
mem_cached	<p>The amount of memory that is being used for file caches.</p> <p>Unit: Bytes</p>

Metric	Description
mem_free	The amount of memory that isn't being used. Unit: Bytes
mem_inactive	The amount of memory that hasn't been used in some way during the last sample period Unit: Bytes
mem_total	The total amount of memory. Unit: Bytes
mem_used	The amount of memory currently in use. Unit: Bytes
mem_used_percent	The percentage of memory currently in use. Unit: Percent
net_bytes_recv	The number of bytes received by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
net_bytes_sent	The number of bytes sent by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
net_drop_in	<p>The number of packets received by this network interface that were dropped.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
net_drop_out	<p>The number of packets transmitted by this network interface that were dropped.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
net_err_in	<p>The number of receive errors detected by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
net_err_out	<p>The number of transmit errors detected by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
net_packets_sent	<p>The number of packets sent by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>

Metric	Description
net_packets_recv	<p>The number of packets received by this network interface.</p> <p>Unit: Count</p> <p>The only statistic that should be used for this metric is Sum. Do not use Average.</p>
netstat_tcp_close	<p>The number of TCP connections with no state.</p> <p>Unit: Count</p>
netstat_tcp_close_wait	<p>The number of TCP connections waiting for a termination request from the client.</p> <p>Unit: Count</p>
netstat_tcp_closing	<p>The number of TCP connections that are waiting for a termination request with acknowledgement from the client.</p> <p>Unit: Count</p>
netstat_tcp_established	<p>The number of TCP connections established.</p> <p>Unit: Count</p>
netstat_tcp_fin_wait1	<p>The number of TCP connections in the FIN_WAIT1 state during the process of closing a connection.</p> <p>Unit: Count</p>
netstat_tcp_fin_wait2	<p>The number of TCP connections in the FIN_WAIT2 state during the process of closing a connection.</p> <p>Unit: Count</p>

Metric	Description
netstat_tcp_last_ack	<p>The number of TCP connections waiting for the client to send acknowledgement of the connection termination message. This is the last state right before the connection is closed down.</p> <p>Unit: Count</p>
netstat_tcp_listen	<p>The number of TCP ports currently listening for a connection request.</p> <p>Unit: Count</p>
netstat_tcp_none	<p>The number of TCP connections with inactive clients.</p> <p>Unit: Count</p>
netstat_tcp_syn_sent	<p>The number of TCP connections waiting for a matching connection request after having sent a connection request.</p> <p>Unit: Count</p>
netstat_tcp_syn_recv	<p>The number of TCP connections waiting for connection request acknowledgement after having sent and received a connection request.</p> <p>Unit: Count</p>
netstat_tcp_time_wait	<p>The number of TCP connections currently waiting to ensure that the client received the acknowledgement of its connection termination request.</p> <p>Unit: Count</p>
netstat_udp_socket	<p>The number of current UDP connections.</p> <p>Unit: Count</p>

Metric	Description
processes_blocked	The number of processes that are blocked. Unit: Count
processes_dead	The number of processes that are dead, indicated by the X state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_idle	The number of processes that are idle (sleeping for more than 20 seconds). Available only on FreeBSD instances. Unit: Count
processes_paging	The number of processes that are paging, indicated by the W state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_running	The number of processes that are running, indicated by the R state code. Unit: Count
processes_sleeping	The number of processes that are sleeping, indicated by the S state code. Unit: Count
processes_stopped	The number of processes that are stopped, indicated by the T state code. Unit: Count

Metric	Description
processes_total	The total number of processes on the instance. Unit: Count
processes_total_threads	The total number of threads making up the processes. This metric is available only on Linux instances. This metric is not collected on macOS computers. Unit: Count
processes_wait	The number of processes that are paging, indicated by the W state code on FreeBSD instances. This metric is available only on FreeBSD instances, and is not available on Linux, Windows Server, or macOS instances. Unit: Count
processes_zombies	The number of zombie processes, indicated by the Z state code. Unit: Count
swap_free	The amount of swap space that isn't being used. Unit: Bytes
swap_used	The amount of swap space currently in use. Unit: Bytes
swap_used_percent	The percentage of swap space currently in use. Unit: Percent

Definitions of memory metrics collected by the CloudWatch agent

When the CloudWatch agent collects memory metrics, the source is the host's memory management subsystem. For example, the Linux kernel exposes OS-maintained data in `/proc`. For memory, the data is in `/proc/meminfo`.

Each different operating system and architecture has different calculations of the resources that are used by processes. For more information, see the following sections.

During each collection interval, the CloudWatch agent on each instance collects the instance resources and calculates the resources being used by all processes which are running in that instance. This information is reported back to CloudWatch metrics. You can configure the length of the collection interval in the CloudWatch agent configuration file. For more information, see [CloudWatch agent configuration file: Agent section](#).

The following list explains how the memory metrics that the CloudWatch agent collects are defined.

- **Active Memory**– Memory that is being used by a process. In other words, the memory used by current running apps.
- **Available Memory**– The memory that can be instantly given to the processes without the system going into swap (also known as virtual memory).
- **Buffer Memory**– The data area shared by hardware devices or program processes that operate at different speeds and priorities.
- **Cached Memory**– Stores program instructions and data that are used repeatedly in the operation of programs that the CPU is likely to need next.
- **Free Memory**– Memory that is not being used at all and is readily available. It is completely free for the system to be used when needed.
- **Inactive Memory**– Pages that have not been accessed "recently".
- **Total Memory**– The size of the actual physical memory RAM.
- **Used Memory**– Memory that is currently in use by programs and processes.

Topics

- [Linux: Metrics collected and calculations used](#)
- [macOS: Metrics collected and calculations used](#)

- [Windows: Metrics collected](#)
- [Example: Calculating memory metrics on Linux](#)

Linux: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Buffered (Bytes)
- Cached (Bytes)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Used memory = Total Memory - Free Memory - Cached memory - Buffer memory

Total memory = Used Memory + Free Memory + Cached memory + Buffer memory

macOS: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Available memory = Free Memory + Inactive memory

Used memory = Total Memory - Available memory

Total memory = Available Memory + Used Memory

Windows: Metrics collected

The metrics collected on Windows hosts are listed below. All of these metrics have None for Unit.

- Available bytes
- Cache Faults/sec
- Page Faults/sec
- Pages/sec

There are no calculations used for Windows metrics because the CloudWatch agent parses events from performance counters.

Example: Calculating memory metrics on Linux

As an example, suppose that entering the `cat /proc/meminfo` command on a Linux host shows the following results:

```
MemTotal:      3824388 kB
MemFree:       462704 kB
MemAvailable:  2157328 kB
Buffers:       126268 kB
Cached:        1560520 kB
SReclaimable: 289080 kB>
```

In this example, the CloudWatch agent will collect the following values. All the values that the CloudWatch agent collects and reports are in bytes.

- `mem_total`: 3916173312 bytes
- `mem_available`: 2209103872 bytes (MemFree + Cached)
- `mem_free`: 473808896 bytes
- `mem_cached`: 1893990400 bytes (cached + SReclaimable)
- `mem_used`: 1419075584 bytes (MemTotal - (MemFree + Buffers + (Cached + SReclaimable)))

- `mem_buffered`: 129667072 bytes
- `mem_available_percent`: 56.41%
- `mem_used_percent`: 36.24% ($\text{mem_used} / \text{mem_total} * 100$)

Common scenarios with the CloudWatch agent

The following sections outline how to complete common configuration and customization tasks for the CloudWatch agent.

Topics

- [Running the CloudWatch agent as a different user](#)
- [How the CloudWatch agent handles sparse log files](#)
- [Adding custom dimensions to metrics collected by the CloudWatch agent](#)
- [Multiple CloudWatch agent configuration files](#)
- [Aggregating or rolling up metrics collected by the CloudWatch agent](#)
- [Collecting high-resolution metrics with the CloudWatch agent](#)
- [Sending metrics, logs, and traces to a different account](#)
- [Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent](#)

Running the CloudWatch agent as a different user

On Linux servers, the CloudWatch agent runs as the root user by default. To have the agent run as a different user, use the `run_as_user` parameter in the agent section in the CloudWatch agent configuration file. This option is available only on Linux servers.

If you're already running the agent with the root user and want to change to using a different user, use one of the following procedures.

To run the CloudWatch agent as a different user on an EC2 instance running Linux

1. Download and install a new CloudWatch agent package. For more information, see [Download the CloudWatch agent package](#).
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.

3. Provide credentials for this user in one of these ways:
 - If the file `.aws/credentials` exists in the home directory of the root user, you must create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/username/.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information](#).
 - If the file `.aws/credentials` does not exist in the home directory of the root user, you can do one of the following:
 - Create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be `/home/username/.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information](#).
 - Instead of creating a credentials file, attach an IAM role to the instance. The agent uses this role as the credential provider.
4. In the CloudWatch agent configuration file, add the following line in the agent section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see [Create the CloudWatch agent configuration file](#)

5. Give the user the required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
6. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

To run the CloudWatch agent as a different user on an on-premises server running Linux

1. Download and install a new CloudWatch agent package. For more information, see [Download the CloudWatch agent package](#).

2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Store the credentials of this user to a path that the user can access, such as `/home/username/.aws/credentials`.
4. Set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see [\(Optional\) Modify the common configuration for proxy or Region information](#).
5. In the CloudWatch agent configuration file, add the following line in the agent section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see [Create the CloudWatch agent configuration file](#)

6. Give the user required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
7. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:configuration-file-path
```

How the CloudWatch agent handles sparse log files

Sparse files are files with both empty blocks and real contents. A sparse file uses disk space more efficiently by writing brief information representing the empty blocks to disk instead of the actual null bytes which makes up the block. This makes the actual size of a sparse file usually much smaller than its apparent size.

However, the CloudWatch agent doesn't treat sparse files differently than it treats normal files. When the agent reads a sparse file, the empty blocks are treated as "real" blocks filled with null bytes. Because of this, the CloudWatch agent publishes as many bytes as the apparent size of a sparse file to CloudWatch.

Configuring the CloudWatch agent to publish a sparse file can cause higher than expected CloudWatch costs, so we recommend not to do so. For example, `/var/logs/lastlog` in Linux is usually a very sparse file, and we recommend that you don't publish it to CloudWatch.

Adding custom dimensions to metrics collected by the CloudWatch agent

To add custom dimensions such as tags to metrics collected by the agent, add the `append_dimensions` field to the section of the agent configuration file that lists those metrics.

For example, the following example section of the configuration file adds a custom dimension named `stackName` with a value of `Prod` to the `cpu` and `disk` metrics collected by the agent.

```
"cpu":{
  "resources":[
    "*"
  ],
  "measurement":[
    "cpu_usage_guest",
    "cpu_usage_nice",
    "cpu_usage_idle"
  ],
  "totalcpu":false,
  "append_dimensions":{
    "stackName":"Prod"
  }
},
"disk":{
  "resources":[
    "/",
    "/tmp"
  ],
  "measurement":[
    "total",
    "used"
  ],
  "append_dimensions":{
    "stackName":"Prod"
  }
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Multiple CloudWatch agent configuration files

On both Linux servers and Windows servers, you can set up the CloudWatch agent to use multiple configuration files. For example, you can use a common configuration file that collects a set of metrics, logs, and traces that you always want to collect from all servers in your infrastructure. You can then use additional configuration files that collect metrics from certain applications or in certain situations.

To set this up, first create the configuration files that you want to use. Any configuration files that will be used together on the same server must have different file names. You can store the configuration files on servers or in Parameter Store.

Start the CloudWatch agent using the `fetch-config` option and specify the first configuration file. To append the second configuration file to the running agent, use the same command but with the `append-config` option. All metrics, logs, and traces listed in either configuration file are collected. The following example commands illustrate this scenario using configurations stores as files. The first line starts the agent using the `infrastructure.json` configuration file, and the second line appends the `app.json` configuration file.

The following example commands are for Linux.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2  
-s -c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m  
ec2 -s -c file:/tmp/app.json
```

The following example commands are for Windows Server.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"  
-a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent  
\infrastructure.json"
```

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"  
-a append-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent  
\app.json"
```

The following example configuration files illustrate a use for this feature. The first configuration file is used for all servers in the infrastructure, and the second collects only logs from a certain application and is appended to servers running that application.

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      }
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log"
          },
          {
            "file_path": "/var/log/messages",
            "log_group_name": "/var/log/messages"
          }
        ]
      }
    }
  }
}
```

app.json

```
{
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/app/app.log*",
            "log_group_name": "/app/app.log"
          }
        ]
      }
    }
  }
}
```

Any configuration files appended to the configuration must have different file names from each other and from the initial configuration file. If you use `append-config` with a configuration file with the same file name as a configuration file that the agent is already using, the `append` command overwrites the information from the first configuration file instead of appending to it. This is true even if the two configuration files with the same file name are on different file paths.

The preceding example shows the use of two configuration files, but there is no limit to the number of configuration files that you can append to the agent configuration. You can also mix the use of configuration files located on servers and configurations located in Parameter Store.

Aggregating or rolling up metrics collected by the CloudWatch agent

To aggregate or roll up metrics collected by the agent, add an `aggregation_dimensions` field to the section for that metric in the agent configuration file.

For example, the following configuration file snippet rolls up metrics on the `AutoScalingGroupName` dimension. The metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

```
"metrics": {
  "cpu": {...}
  "disk": {...}
  "aggregation_dimensions" : [ ["AutoScalingGroupName"] ]
}
```


To roll up along the combination of each InstanceId and InstanceType dimensions in addition to rolling up on the Auto Scaling group name, add the following.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [ ["AutoScalingGroupName"], ["InstanceId", "InstanceType"] ]
}
```

To roll up metrics into one collection instead, use [].

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [[]]
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Collecting high-resolution metrics with the CloudWatch agent

The `metrics_collection_interval` field specifies the time interval for the metrics collected, in seconds. By specifying a value of less than 60 for this field, the metrics are collected as high-resolution metrics.

For example, if your metrics should all be high-resolution and collected every 10 seconds, specify 10 as the value for `metrics_collection_interval` under the agent section as a global metrics collection interval.

```
"agent": {
  "metrics_collection_interval": 10
}
```

Alternatively, the following example sets the cpu metrics to be collected every second, and all other metrics are collected every minute.

```
"agent":{
  "metrics_collection_interval": 60
```

```
},
"metrics":{
  "metrics_collected":{
    "cpu":{
      "resources":[
        "*"
      ],
      "measurement":[
        "cpu_usage_guest"
      ],
      "totalcpu":false,
      "metrics_collection_interval": 1
    },
    "disk":{
      "resources":[
        "/",
        "/tmp"
      ],
      "measurement":[
        "total",
        "used"
      ]
    }
  }
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Sending metrics, logs, and traces to a different account

To have the CloudWatch agent send the metrics, logs, or traces to a different account, specify a `role_arn` parameter in the agent configuration file on the sending server. The `role_arn` value specifies an IAM role in the target account that the agent uses when sending data to the target account. This role enables the sending account to assume a corresponding role in the target account when delivering the metrics or logs to the target account.

You can also specify separate `role_arn` strings in the agent configuration file: one to use when sending metrics, another for sending logs, and another for sending traces.

The following example of part of the agent section of the configuration file sets the agent to use `CrossAccountAgentRole` when sending data to a different account.

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  .....
}
```

Alternatively, the following example sets different roles for the sending account to use for sending metrics, logs, and traces:

```
"metrics": {
  "credentials": {
    "role_arn": "RoleToSendMetrics"
  },
  "metrics_collected": {....
```

```
"logs": {
  "credentials": {
    "role_arn": "RoleToSendLogs"
  },
  ....
```

Policies needed

When you specify a `role_arn` in the agent configuration file, you must also make sure the IAM roles of the sending and target accounts have certain policies. The roles in both the sending and target accounts should have `CloudWatchAgentServerPolicy`. For more information about assigning this policy to a role, see [Create IAM roles to use with the CloudWatch agent on Amazon EC2 instances](#).

The role in the sending account also must include the following policy. You add this policy on the **Permissions** tab in the IAM console when you edit the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::target-account-ID:role/agent-role-in-target-account"
      ]
    }
  ]
}

```

The role in the target account must include the following policy so that it recognizes the IAM role used by the sending account. You add this policy on the **Trust relationships** tab in the IAM console when you edit the role. The role in the target account where you add this policy is the role you created in [Create IAM roles and users for use with CloudWatch agent](#). This role is the role specified in *agent-role-in-target-account* in the policy used by the sending account.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::sending-account-ID:role/role-in-sender-account"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

Timestamp differences between the unified CloudWatch agent and the earlier CloudWatch Logs agent

The CloudWatch agent supports a different set of symbols for timestamp formats, compared to the earlier CloudWatch Logs agent. These differences are shown in the following table.

Symbols supported by both agents	Symbols supported only by unified CloudWatch agent	Symbols supported only by earlier CloudWatch Logs agent
%A, %a, %b, %B, %d, %f, %H, %l, %m, %M, %p, %S, %y, %Y, %Z, %z	%d, %l, %m, %M, %S	%c, %j, %U, %W, %w

For more information about the meanings of the symbols supported by the new CloudWatch agent, see [CloudWatch Agent Configuration File: Logs Section](#) in the *Amazon CloudWatch User Guide*. For information about symbols supported by the CloudWatch Logs agent, see [Agent Configuration File](#) in the *Amazon CloudWatch Logs User Guide*.

Troubleshooting the CloudWatch agent

Use the following information to help troubleshoot problems with the CloudWatch agent.

Topics

- [CloudWatch agent command line parameters](#)
- [Installing the CloudWatch agent using Run Command fails](#)
- [The CloudWatch agent won't start](#)
- [Verify that the CloudWatch agent is running](#)
- [The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region](#)
- [The CloudWatch agent won't start on Windows Server](#)
- [Where are the metrics?](#)
- [The CloudWatch agent takes a long time to run in a container or logs a hop limit error](#)
- [I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console](#)
- [CloudWatch agent files and locations](#)
- [Finding information about CloudWatch agent versions](#)
- [Logs generated by the CloudWatch agent](#)
- [Stopping and restarting the CloudWatch agent](#)

CloudWatch agent command line parameters

To see the full list of parameters supported by the CloudWatch agent, enter the following at the command line at a computer where you have it installed:

```
amazon-cloudwatch-agent-ctl -help
```

Installing the CloudWatch agent using Run Command fails

To install the CloudWatch agent using Systems Manager Run Command, the SSM Agent on the target server must be version 2.2.93.0 or later. If your SSM Agent isn't the correct version, you might see errors that include the following messages:

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

For information about updating your SSM Agent version, see [Installing and Configuring SSM Agent](#) in the *AWS Systems Manager User Guide*.

The CloudWatch agent won't start

If the CloudWatch agent fails to start, there might be an issue in your configuration. Configuration information is logged in the `configuration-validation.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` on servers running Windows Server.

Verify that the CloudWatch agent is running

You can query the CloudWatch agent to find whether it's running or stopped. You can use AWS Systems Manager to do this remotely. You can also use the command line, but only to check the local server.

To query the status of the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.

2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose the button next to **AmazonCloudWatch-ManagedAgent**.
5. In the **Action** list, choose **status**.
6. For **Optional Configuration Source** choose **default** and keep **Optional Configuration Location** blank.
7. In the **Target** area, choose the instance to check.
8. Choose **Run**.

If the agent is running, the output resembles the following.

```
{
  "status": "running",
  "starttime": "2017-12-12T18:41:18",
  "version": "1.73.4"
}
```

If the agent is stopped, the "status" field displays "stopped".

To query the status of the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
```

On a server running Windows Server, enter the following in PowerShell as an administrator:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2 -a status
```

The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region

If the agent doesn't start and the error message mentions an Amazon EC2 Region endpoint, you might have configured the agent to need access to the Amazon EC2 endpoint without granting that access.

For example, if you specify a value for the `append_dimensions` parameter in the agent configuration file that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see [Amazon Elastic Compute Cloud \(Amazon EC2\)](#) in the *Amazon Web Services General Reference*.

The CloudWatch agent won't start on Windows Server

On Windows Server, you might see the following error:

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be
started due to the following
error: Cannot start service AmazonCloudWatchAgent on computer '.'.
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113
char:12
+     $svc | Start-Service
+     ~~~~~
+ CategoryInfo          : OpenError:
(System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
ServiceCommandException
+ FullyQualifiedErrorId :
CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

To fix this, first make sure that the server service is running. This error can be seen if the agent tries to start when the server service isn't running.

If the server service is already running, the following may be the issue. On some Windows Server installations, the CloudWatch agent takes more than 30 seconds to start. Because Windows Server, by default, allows only 30 seconds for services to start, this causes the agent to fail with an error similar to the following:

To fix this issue, increase the service timeout value. For more information, see [A service does not start, and events 7000 and 7011 are logged in the Windows event log](#).

Where are the metrics?

If the CloudWatch agent has been running but you can't find metrics collected by it in the AWS Management Console or the AWS CLI, confirm that you're using the correct namespace. By default, the namespace for metrics collected by the agent is `CWAgent`. You can customize this namespace using the `namespace` field in the `metrics` section of the agent configuration file. If you don't see the metrics that you expect, check the configuration file to confirm the namespace being used.

When you first download the CloudWatch agent package, the agent configuration file is `amazon-cloudwatch-agent.json`. This file is in the directory where you ran the configuration wizard, or you might have moved it to a different directory. If you use the configuration wizard, the agent configuration file output from the wizard is named `config.json`. For more information about the configuration file, including the `namespace` field, see [CloudWatch agent configuration file: Metrics section](#).

The CloudWatch agent takes a long time to run in a container or logs a hop limit error

When you run the CloudWatch agent as a container service and want to add Amazon EC2 metric dimensions to all metrics collected by the agent, you might see the following errors in version `v1.247354.0` of the agent:

```
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Unable to retrieve Instance Metadata Tags. This plugin must only be used on an EC2 instance.
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Please increase hop limit to 2 by following this document https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-options.html#configuring-IMDS-existing-instances.
2022-06-07T03:36:11Z E! [telegraf] Error running agent: could not initialize processor ec2tagger: EC2MetadataRequestError: failed to get EC2 instance identity document caused by: EC2MetadataError: failed to make EC2Metadata request
    status code: 401, request id:
caused by:
```

You might see this error if the agent tries to get metadata from IMDSv2 inside a container without an appropriate hop limit. In versions of the agent earlier than `v1.247354.0`, you can experience this issue without seeing the log message.

To solve this, increase the hop limit to 2 by following the instructions in [Configure the instance metadata options](#).

I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console

If you update your CloudWatch agent configuration file, the next time that you start the agent, you need to use the **fetch-config** option. For example, if you stored the updated file on the local computer, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -s -m ec2 -c file:configuration-file-path
```

CloudWatch agent files and locations

The following table lists the files installed by and used with the CloudWatch agent, along with their locations on servers running Linux or Windows Server.

File	Linux location	Windows Server location
The control script that controls starting, stopping, and restarting the agent.	/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl or /usr/bin/amazon-cloudwatch-agent-ctl	\$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1
The log file the agent writes to. You might need to attach this when contacting AWS Support.	/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log or /var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log
Agent configuration validation file.	/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log or /	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\con

File	Linux location	Windows Server location
	var/log/amazon/amazon-cloudwatch-agent/configuration-validation.log	figuration-validation.log
The JSON file used to configure the agent immediately after the wizard creates it. For more information, see Create the CloudWatch agent configuration file .	/opt/aws/amazon-cloudwatch-agent/bin/config.json	\$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\config.json
The JSON file used to configure the agent if this configuration file has been downloaded from Parameter Store.	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json
The TOML file used to specify Region and credential information to be used by the agent, overriding system defaults.	/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml or /etc/amazon/amazon-cloudwatch-agent/common-config.toml	\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml

File	Linux location	Windows Server location
The TOML file that contains the converted contents of the JSON configuration file. The <code>amazon-cloudwatch-agent-ctl</code> script generates this file. Users should not directly modify this file. It can be useful for verifying that JSON to TOML translation was successful.	<code>/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml</code> or <code>/etc/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml</code>	<code>\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.toml</code>
The YAML file that contains the converted contents of the JSON configuration file. The <code>amazon-cloudwatch-agent-ctl</code> script generates this file. You should not directly modify this file. This file can be useful for verifying that the JSON to YAML translation was successful.	<code>/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.yaml</code> or <code>/etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.yaml</code>	<code>\$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.yaml</code>

Finding information about CloudWatch agent versions

To find the version number of the CloudWatch agent on a Linux server, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

To find the version number of the CloudWatch agent on Windows Server, enter the following command:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2  
-a status
```

Note

Using this command is the correct way to find the version of the CloudWatch agent. If you use **Programs and Features** in the Control Panel, you will see an incorrect version number.

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is currently available for download. These files are in the following locations:

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES or [https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/RELEASE_NOTES](https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/info/latest/RELEASE_NOTES)
- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION or [https://amazoncloudwatch-agent-*region*.s3.*region*.amazonaws.com/info/latest/CWAGENT_VERSION](https://amazoncloudwatch-agent-<i>region</i>.s3.<i>region</i>.amazonaws.com/info/latest/CWAGENT_VERSION)

Logs generated by the CloudWatch agent

The agent generates a log while it runs. This log includes troubleshooting information. This log is the `amazon-cloudwatch-agent.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log` on servers running Windows Server.

You can configure the agent to log additional details in the `amazon-cloudwatch-agent.log` file. In the agent configuration file, in the agent section, set the `debug` field to `true`, then reconfigure and restart the CloudWatch agent. To disable the logging of this extra information, set the `debug` field to `false`. Then, reconfigure and restart the agent. For more information, see [Manually create or edit the CloudWatch agent configuration file](#).

In versions 1.247350.0 and later of the CloudWatch agent, you can optionally set the `aws_sdk_log_level` field in the agent section of the agent configuration file to one or more of the following options. Separate multiple options with the `|` character.

- LogDebug
- LogDebugWithSigning
- LogDebugWithHTTPBody
- LogDebugRequestRetries
- LogDebugWithEventStreamBody

For more information about these options, see [LogLevelType](#).

Stopping and restarting the CloudWatch agent

You can manually stop the CloudWatch agent using either AWS Systems Manager or the command line.

To stop the CloudWatch agent using Run Command

1. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager/>.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **stop**.
7. Keep **Optional Configuration Source** and **Optional Configuration Location** blank.
8. Choose **Run**.

To stop the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a stop
```

On a server running Windows Server, enter the following in PowerShell as an administrator:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m  
ec2 -a stop
```

To restart the agent, follow the instructions in [Start the CloudWatch agent](#).

Embedding metrics within logs

The CloudWatch embedded metric format allows you to generate custom metrics asynchronously in the form of logs written to CloudWatch Logs. You can embed custom metrics alongside detailed log event data, and CloudWatch automatically extracts the custom metrics so that you can visualize and alarm on them, for real-time incident detection. Additionally, the detailed log events associated with the extracted metrics can be queried using CloudWatch Logs Insights to provide deep insights into the root causes of operational events.

Embedded metric format helps you generate actionable custom metrics from ephemeral resources such as Lambda functions and containers. By using the embedded metric format to send logs from these ephemeral resources, you can now easily create custom metrics without having to instrument or maintain separate code, while gaining powerful analytical capabilities on your log data.

No setup is required to use the embedded metric format. Either structure your logs by following the [Embedded metric format specification](#), or generate them using our client libraries and send them to CloudWatch Logs using the [PutLogEvents API](#) or the [CloudWatch agent](#).

Charges are incurred for logs ingestion and archival, and custom metrics that are generated. For more information, see [Amazon CloudWatch Pricing](#).

Note

Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see [Dimensions](#).

Topics

- [Publishing logs with the embedded metric format](#)
- [Viewing your metrics and logs in the console](#)
- [Setting alarms on metrics created with the embedded metric format](#)

Publishing logs with the embedded metric format

You can generate embedded metric format logs with the following methods:

- Generate and send the logs by using the [open-sourced client libraries](#).
- Manually generate the logs using the [embedded metric format specification](#), and then use the [CloudWatch agent](#) or the [PutLogEvents API](#) to send the logs.

Topics

- [Creating logs in embedded metric format using the client libraries](#)
- [Specification: Embedded metric format](#)
- [Using the PutLogEvents API to send manually-created embedded metric format logs](#)
- [Using the CloudWatch agent to send embedded metric format logs](#)
- [Using the embedded metric format with AWS Distro for OpenTelemetry](#)

Creating logs in embedded metric format using the client libraries

Amazon provides open-sourced client libraries, which you can use to create embedded metric format logs. Currently those libraries are available for the languages in the following list. Full examples for different setups can be found in our client libraries under **/examples**.

The libraries and the instructions for how to use them are located on Github. Use the following links.

- [Node.js](#)

Note

For Node.js, versions 4.1.1+, 3.0.2+, 2.0.7+ are required for use with the Lambda JSON log format. Using previous versions in such Lambda environments will lead to metric loss. For more information, see [Accessing Amazon CloudWatch logs for AWS Lambda](#).

- [Python](#)
- [Java](#)
- [C#](#)

Client libraries are meant to work out of the box with the CloudWatch agent. Generated embedded metric format logs are sent to the CloudWatch agent, which then aggregates and publishes them to CloudWatch Logs for you.

Note

When using Lambda, no agent is required to send the logs to CloudWatch. Anything logged to STDOUT is sent to CloudWatch Logs via the Lambda Logging Agent.

Specification: Embedded metric format

The CloudWatch embedded metric format is a JSON specification used to instruct CloudWatch Logs to automatically extract metric values embedded in structured log events. You can use CloudWatch to graph and create alarms on the extracted metric values.

Embedded metric format specification conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this format specification are to be interpreted as described in [Key Words RFC2119](#).

The terms "JSON", "JSON text", "JSON value", "member", "element", "object", "array", "number", "string", "boolean", "true", "false", and "null" in this format specification are to be interpreted as defined in [JavaScript Object Notation RFC8259](#).

Note

If you plan to create alarms on metrics created using embedded metric format, see [Setting alarms on metrics created with the embedded metric format](#) for recommendations.

Embedded metric format document structure

This section describes the structure of an embedded metric format document. Embedded metric format documents are defined in [JavaScript Object Notation RFC8259](#).

Unless otherwise noted, objects defined by this specification **MUST NOT** contain any additional members. Members not recognized by this specification **MUST** be ignored. Members defined in this specification are case-sensitive.

The embedded metric format is subject to the same limits as standard CloudWatch Logs events and are limited to a maximum size of 256 KB.

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the AWS/Logs namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see [Monitoring with CloudWatch metrics](#).

Root node

The LogEvent message MUST be a valid JSON object with no additional data at the beginning or end of the LogEvent message string. For more information about the LogEvent structure, see [InputLogEvent](#).

Embedded metric format documents MUST contain the following top-level member on the root node. This is a [Metadata object](#) object.

```
{
  "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

The root node MUST contain all [Target members](#) members defined by the references in the [MetricDirective object](#).

The root node MAY contain any other members that are not included in the above requirements. The values of these members MUST be valid JSON types.

Metadata object

The `_aws` member can be used to represent metadata about the payload that informs downstream services how they should process the LogEvent. The value MUST be an object and MUST contain the following members:

- **CloudWatchMetrics**— An array of [MetricDirective object](#) used to instruct CloudWatch to extract metrics from the root node of the LogEvent.

```
{
  "_aws": {
```

```

    "CloudWatchMetrics": [ ... ]
  }
}

```

- **Timestamp**— A number representing the time stamp used for metrics extracted from the event. Values MUST be expressed as the number of milliseconds after Jan 1, 1970 00:00:00 UTC.

```

{
  "_aws": {
    "Timestamp": 1559748430481
  }
}

```

MetricDirective object

The MetricDirective object instructs downstream services that the LogEvent contains metrics that will be extracted and published to CloudWatch. MetricDirectives MUST contain the following members:

- **Namespace**— A string representing the CloudWatch namespace for the metric.
- **Dimensions**— A [DimensionSet](#) array.
- **Metrics**— An array of [MetricDefinition](#) objects. This array MUST NOT contain more than 100 MetricDefinition objects.

DimensionSet array

A DimensionSet is an array of strings containing the dimension keys that will be applied to all metrics in the document. The values within this array MUST also be members on the root-node—referred to as the [Target members](#)

A DimensionSet MUST NOT contain more than 30 dimension keys. A DimensionSet MAY be empty.

The target member MUST have a string value. This value MUST NOT contain more than 1024 characters. The target member defines a dimension that will be published as part of the metric identity. Every DimensionSet used creates a new metric in CloudWatch. For more information about dimensions, see [Dimension](#) and [Dimensions](#).

```

{
  "_aws": {

```

```
"CloudWatchMetrics": [
  {
    "Dimensions": [ [ "functionVersion" ] ],
    ...
  }
],
"functionVersion": "$LATEST"
}
```

Note

Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see [Dimensions](#).

MetricDefinition object

A `MetricDefinition` is an object that **MUST** contain the following member:

- **Name**— A string [Reference values](#) to a metric [Target members](#). Metric targets **MUST** be either a numeric value or an array of numeric values.

A `MetricDefinition` object **MAY** contain the following members:

- **Unit**— An **OPTIONAL** string value representing the unit of measure for the corresponding metric. Values **SHOULD** be valid CloudWatch metric units. For information about valid units, see [MetricDatum](#). If a value is not provided, then a default value of `NONE` is assumed.
- **StorageResolution**— An **OPTIONAL** integer value representing the storage resolution for the corresponding metric. Setting this to 1 specifies this metric as a high-resolution metric, so that CloudWatch stores the metric with sub-minute resolution down to one second. Setting this to 60 specifies this metric as standard-resolution, which CloudWatch stores at 1-minute resolution. Values **SHOULD** be valid CloudWatch supported resolutions, 1 or 60. If a value is not provided, then a default value of 60 is assumed.

For more information about high-resolution metrics, see [High-resolution metrics](#).

Note

If you plan to create alarms on metrics created using embedded metric format, see [Setting alarms on metrics created with the embedded metric format](#) for recommendations.

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ],
        ...
      }
    ]
  },
  "Time": 1
}
```

Reference values

Reference values are string values that reference [Target members](#) members on the root node. These references should NOT be confused with the JSON Pointers described in [RFC6901](#). Target values cannot be nested.

Target members

Valid targets MUST be members on the root node and cannot be nested objects. For example, a `_reference_` value of `"A.a"` MUST match the following member:

```
{ "A.a" }
```

It MUST NOT match the nested member:

```
{ "A": { "a" } }
```

Valid values of target members depend on what is referencing them. A metric target **MUST** be a numeric value or an array of numeric values. Numeric array metric targets **MUST NOT** have more than 100 members. A dimension target **MUST** have a string value.

Embedded metric format example and JSON schema

The following is a valid example of embedded metric format.

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ]
      }
    ]
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

You can use the following schema to validate embedded metric format documents.

```
{
  "type": "object",
  "title": "Root Node",
  "required": [
    "_aws"
  ],
  "properties": {
    "_aws": {
      "$id": "#/properties/_aws",
      "type": "object",
      "title": "Metadata",

```

```

    "required": [
      "Timestamp",
      "CloudWatchMetrics"
    ],
    "properties": {
      "Timestamp": {
        "$id": "#/properties/_aws/properties/Timestamp",
        "type": "integer",
        "title": "The Timestamp Schema",
        "examples": [
          1565375354953
        ]
      },
      "CloudWatchMetrics": {
        "$id": "#/properties/_aws/properties/CloudWatchMetrics",
        "type": "array",
        "title": "MetricDirectives",
        "items": {
          "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
          "type": "object",
          "title": "MetricDirective",
          "required": [
            "Namespace",
            "Dimensions",
            "Metrics"
          ],
          "properties": {
            "Namespace": {
              "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/namespace",
              "type": "string",
              "title": "CloudWatch Metrics Namespace",
              "examples": [
                "MyApp"
              ],
              "pattern": "^(.*)$",
              "minLength": 1,
              "maxLength": 1024
            },
            "Dimensions": {
              "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
              "type": "array",
              "title": "The Dimensions Schema",

```



```

        "minItems": 1,
        "items": {
            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items",
            "type": "array",
            "title": "DimensionSet",
            "minItems": 0,
            "maxItems": 30,
            "items": {
                "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                "type": "string",
                "title": "DimensionReference",
                "examples": [
                    "Operation"
                ],
                "pattern": "^(.*)$",
                "minLength": 1,
                "maxLength": 250
            }
        }
    },
    "Metrics": {
        "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
        "type": "array",
        "title": "MetricDefinitions",
        "items": {
            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items",
            "type": "object",
            "title": "MetricDefinition",
            "required": [
                "Name"
            ],
            "properties": {
                "Name": {
                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Name",
                    "type": "string",
                    "title": "MetricName",
                    "examples": [
                        "ProcessingLatency"
                    ],

```


Using the PutLogEvents API to send manually-created embedded metric format logs

You can send embedded metric format logs to CloudWatch Logs using the CloudWatch Logs PutLogEvents API. When calling PutLogEvents, you may optionally include the following HTTP header to instruct CloudWatch Logs that the metrics should be extracted, but it is no longer necessary.

```
x-amzn-logs-format: json/emf
```

The following is a full example using the AWS SDK for Java 2.x:

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
    public static void main(String[] args) {

        final String usage = "To run this example, supply a Region code (eg.
us-east-1), log group, and stream name as command line arguments"
            + "Ex: PutLogEvents <region-id> <log-group-name>
<stream-name>";

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String regionId = args[0];
        String logGroupName = args[1];
        String logStreamName = args[2];

        CloudWatchLogsClient logsClient =
            CloudWatchLogsClient.builder().region(Region.of(regionId)).build();
```

```

// Build a JSON log using the EmbeddedMetricFormat.
long timestamp = System.currentTimeMillis();
String message = "{" +
    "  \"_aws\": {" +
    "    \"Timestamp\": " + timestamp + "," +
    "    \"CloudWatchMetrics\": [" +
    "      {" +
    "        \"Namespace\": \"MyApp\", " +
    "        \"Dimensions\": [[\"Operation\"], [\"Operation
\", \"Cell\"]], " +
    "        \"Metrics\": [{ \"Name\": \"ProcessingLatency
\", \"Unit\": \"Milliseconds\", \"StorageResolution\": 60 }]" +
    "      }" +
    "    ]" +
    "  }, " +
    "  \"Operation\": \"Aggregator\", " +
    "  \"Cell\": \"001\", " +
    "  \"ProcessingLatency\": 100" +
    "}";

InputLogEvent inputLogEvent = InputLogEvent.builder()
    .message(message)
    .timestamp(timestamp)
    .build();

// Specify the request parameters.
PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
    .logEvents(Collections.singletonList(inputLogEvent))
    .logGroupName(logGroupName)
    .logStreamName(logStreamName)
    .build();

logsClient.putLogEvents(putLogEventsRequest);

System.out.println("Successfully put CloudWatch log event");
}
}

```

Note

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the AWS/Logs namespace of your account. These can be

used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see [Monitoring with CloudWatch metrics](#).

Using the CloudWatch agent to send embedded metric format logs

To use this method, first install the CloudWatch agent for the services you want to send embedded metric format logs from, and then you can begin sending the events.

The CloudWatch agent must be version 1.230621.0 or later.

Note

You do not need to install the CloudWatch agent to send logs from Lambda functions. Lambda function timeouts are not handled automatically. This means that if your function times out before the metrics get flushed, then the metrics for that invocation will not be captured.

Installing the CloudWatch agent

Install the CloudWatch agent for each service which is to send embedded metric format logs.

Installing the CloudWatch agent on EC2

First, install the CloudWatch agent on the instance. For more information, see [Installing the CloudWatch agent](#).

Once you have installed the agent, configure the agent to listen on a UDP or TCP port for the embedded metric format logs. The following is an example of this configuration that listens on the default socket `tcp:25888`. For more information about agent configuration, see [Manually create or edit the CloudWatch agent configuration file](#).

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Installing the CloudWatch agent on Amazon ECS

The easiest way to deploy the CloudWatch agent on Amazon ECS is to run it as a sidecar, defining it in the same task definition as your application.

Create agent configuration file

Create your CloudWatch agent configuration file locally. In this example, the relative file path will be `amazon-cloudwatch-agent.json`.

For more information about agent configuration, see [Manually create or edit the CloudWatch agent configuration file](#).

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Push configuration to SSM Parameter Store

Enter the following command to push the CloudWatch agent configuration file to the AWS Systems Manager (SSM) Parameter Store.

```
aws ssm put-parameter \
  --name "cwagentconfig" \
  --type "String" \
  --value "`cat amazon-cloudwatch-agent.json`" \
  --region "{{region}}"
```

Configure the task definition

Configure your task definition to use the CloudWatch Agent and expose the TCP or UDP port. The sample task definition that you should use depends on your networking mode.

Notice that the `webapp` specifies the `AWS_EMF_AGENT_ENDPOINT` environment variable. This is used by the library and should point to the endpoint that the agent is listening on. Additionally, the `cwagent` specifies the `CW_CONFIG_CONTENT` as a “valueFrom” parameter that points to the SSM configuration that you created in the previous step.

This section contains one example for bridge mode and one example for host or awsvpc mode. For more examples of how you can configure the CloudWatch agent on Amazon ECS, see the [Github samples repository](#)

The following is an example for bridge mode. When bridge mode networking is enabled, the agent needs to be linked to your application using the `links` parameter and must be addressed using the container name.

```
{
  "containerDefinitions": [
    {
      "name": "webapp",
      "links": [ "cwagent" ],
      "image": "my-org/web-app:latest",
      "memory": 256,
      "cpu": 256,
      "environment": [{
        "name": "AWS_EMF_AGENT_ENDPOINT",
        "value": "tcp://cwagent:25888"
      }],
    },
    {
      "name": "cwagent",
      "mountPoints": [],
      "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
      "memory": 256,
      "cpu": 256,
      "portMappings": [{
        "protocol": "tcp",
        "containerPort": 25888
      }],
      "environment": [{
        "name": "CW_CONFIG_CONTENT",
        "valueFrom": "cwagentconfig"
      }],
    }
  ],
}
```

The following is an example for host mode or awsvpc mode. When running on these network modes, the agent can be addressed over `localhost`.

```
{
  "containerDefinitions": [
    {
      "name": "webapp",
      "image": "my-org/web-app:latest",
      "memory": 256,
      "cpu": 256,
      "environment": [{
        "name": "AWS_EMF_AGENT_ENDPOINT",
        "value": "tcp://127.0.0.1:25888"
      }],
    },
    {
      "name": "cwagent",
      "mountPoints": [],
      "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
      "memory": 256,
      "cpu": 256,
      "portMappings": [{
        "protocol": "tcp",
        "containerPort": 25888
      }],
      "environment": [{
        "name": "CW_CONFIG_CONTENT",
        "valueFrom": "cwagentconfig"
      }],
    }
  ],
}
```

Note

In `awsvpc` mode, you must either give a public IP address to the VPC (Fargate only), set up a NAT gateway, or set up a CloudWatch Logs VPC endpoint. For more information about setting up a NAT, see [NAT Gateways](#). For more information about setting up a CloudWatch Logs VPC endpoint, see [Using CloudWatch Logs with Interface VPC Endpoints](#).

The following is an example of how to assign a public IP address to a task that uses the Fargate launch type.

```
aws ecs run-task \
--cluster {{cluster-name}} \
```



```
--task-definition cwagent-fargate \  
--region {{region}} \  
--launch-type FARGATE \  
--network-configuration  
"awsVpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENA
```

Ensure permissions

Ensure the IAM role executing your tasks has permission to read from the SSM Parameter Store. You can add this permission by attaching the **AmazonSSMReadOnlyAccess** policy. To do so, enter the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess  
\  
--role-name CWAgentECSExecutionRole
```

Installing the CloudWatch agent on Amazon EKS

Parts of this process can be skipped if you have already installed CloudWatch Container Insights on this cluster.

Permissions

If you have not already installed Container Insights, then first ensure that your Amazon EKS nodes have the appropriate IAM permissions. They should have the **CloudWatchAgentServerPolicy** attached. For more information, see [Verify prerequisites](#).

Create ConfigMap

Create a ConfigMap for the agent. The ConfigMap also tells the agent to listen on a TCP or UDP port. Use the following ConfigMap.

```
# cwagent-emf-configmap.yaml  
apiVersion: v1  
data:  
  # Any changes here must not break the JSON format  
  cwagentconfig.json: |  
    {  
      "agent": {
```

```
    "omit_hostname": true
  },
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
kind: ConfigMap
metadata:
  name: cwagentemfconfig
  namespace: default
```

If you have already installed Container Insights, add the following "emf": { } line to your existing ConfigMap.

Apply the ConfigMap

Enter the following command to apply the ConfigMap.

```
kubectl apply -f cwagent-emf-configmap.yaml
```

Deploy the agent

To deploy the CloudWatch agent as a sidecar, add the agent to your pod definition, as in the following example.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
    - name: web-app
      image: my-org/web-app:latest
    # CloudWatch Agent configuration
    - name: cloudwatch-agent
      image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
      imagePullPolicy: Always
  resources:
```

```

limits:
  cpu: 200m
  memory: 100Mi
requests:
  cpu: 200m
  memory: 100Mi
volumeMounts:
  - name: cwagentconfig
    mountPath: /etc/cwagentconfig
ports:
# this should match the port configured in the ConfigMap
  - protocol: TCP
    hostPort: 25888
    containerPort: 25888
volumes:
  - name: cwagentconfig
    configMap:
      name: cwagentemfconfig

```

Using the CloudWatch agent to send embedded metric format logs

When you have the CloudWatch agent installed and running, you can send the embedded metric format logs over TCP or UDP. There are two requirements when sending the logs over the agent:

- The logs must contain a `LogGroupName` key that tells the agent which log group to use.
- Each log event must be on a single line. In other words, a log event cannot contain the newline (`\n`) character.

The log events must also follow the embedded metric format specification. For more information, see [Specification: Embedded metric format](#).

If you plan to create alarms on metrics created using embedded metric format, see [Setting alarms on metrics created with the embedded metric format](#) for recommendations.

The following is an example of sending log events manually from a Linux bash shell. You can instead use the UDP socket interfaces provided by your programming language of choice.

```

echo '{"_aws":{"Timestamp":1574109732004,"LogGroupName":"Foo","CloudWatchMetrics":
[{"Namespace":"MyApp","Dimensions":[["Operation"]],"Metrics":
[{"Name":"ProcessingLatency","Unit":"Milliseconds","StorageResolution":60}]}}',"Operation":"Agg
\n

```

```
> /dev/udp/0.0.0.0/25888
```

Note

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the AWS/Logs namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see [Monitoring with CloudWatch metrics](#).

Using the embedded metric format with AWS Distro for OpenTelemetry

You can use the embedded metric format as a part of the OpenTelemetry project. OpenTelemetry is an open-source initiative that removes boundaries and restrictions between vendor-specific formats for tracing, logs, and metrics by offering a single set of specifications and APIs. For more information, see [OpenTelemetry](#).

Using embedded metric format with OpenTelemetry requires two components: an OpenTelemetry-compliant data source, and the AWS Distro for OpenTelemetry Collector enabled for use with CloudWatch embedded metric format logs.

We have preconfigured redistributions of the OpenTelemetry components, maintained by AWS, to make onboarding as easy as possible. For more information about using OpenTelemetry with embedded metric format, in addition to other AWS services, see [AWS Distro for OpenTelemetry](#).

For additional information regarding language support and usage, see [AWS Observability on Github](#).

Viewing your metrics and logs in the console

After you generate embedded metric format logs that extract metrics, you can use the CloudWatch console to view the metrics. Embedded metrics have the dimensions that you specified when you generated the logs. Also, embedded metrics that you generated using the client libraries have the following default dimensions:

- ServiceType
- ServiceName
- LogGroup

To view metrics that were generated from embedded metric format logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select a namespace that you specified for your embedded metrics when you generated them. If you used the client libraries to generate the metrics and did not specify a namespace, then select **aws-embedded-metrics**. This is the default namespace for embedded metrics generated using the client libraries.
4. Select a metric dimension (for example, **ServiceName**).
5. The **All metrics** tab displays all metrics for that dimension in the namespace. You can do the following:
 - a. To sort the table, use the column heading.
 - b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
 - c. To filter by resource, choose the resource ID and then choose **Add to search**.
 - d. To filter by metric, choose the metric name and then choose **Add to search**.

Querying logs using CloudWatch Logs Insights

You can query the detailed log events associated with the extracted metrics by using CloudWatch Logs Insights to provide deep insights into the root causes of operational events. One of the benefits of extracting metrics from your logs is that you can filter your logs later by the unique metric (metric name plus unique dimension set) and metric values, to get context on the events that contributed to the aggregated metric value

For example, to get an impacted request id or x-ray trace id, you could run the following query in CloudWatch Logs Insights.

```
filter Latency > 1000 and Operation = "Aggregator"  
| fields RequestId, TraceId
```

You can also perform query-time aggregation on high-cardinality keys, such as finding the customers impacted by an event. The following example illustrates this.

```
filter Latency > 1000 and Operation = "Aggregator"  
| stats count() by CustomerId
```

For more information, see [Analyzing Log Data with CloudWatch Logs Insights](#)

Setting alarms on metrics created with the embedded metric format

In general, creating alarms on metrics generated by embedded metric format follows the same pattern as creating alarms on any other metrics. For more information, see [Using Amazon CloudWatch alarms](#).

Embedded metric format metric generation depends on your log publishing flow, because the logs have to be processed by CloudWatch Logs to be transformed into metrics. So it's important for you to publish logs in a timely manner so that your metric datapoints are created within the period of time in which alarms are evaluated.

If you plan use embedded metric format to send high-resolution metrics and create alarms on these metrics, we recommended that you flush logs to CloudWatch Logs at an interval of 5 seconds or less to avoid introducing additional delay that can cause alarming on partial or missing data. If you are using the CloudWatch agent, you can adjust the flush interval by setting the `force_flush_interval` parameter in the CloudWatch agent configuration file. This value defaults to 5 seconds.

If you are using Lambda on other platforms where you can't control the log flush interval, consider using "M out of N" alarms to control the number of datapoints used to alarm. For more information, see [Evaluating an alarm](#).

AWS services that publish CloudWatch metrics

The following AWS services publish metrics to CloudWatch. For information about the metrics and dimensions, see the specified documentation.

Service	Namespace	Documentation
AWS Amplify	AWS/AmplifyHosting	Monitoring
Amazon API Gateway	AWS/ApiGateway	Monitor API Execution with Amazon CloudWatch
Amazon AppFlow	AWS/AppFlow	Monitoring Amazon AppFlow with Amazon CloudWatch
AWS Application Migration Service	AWS/MGN	Monitoring Application Migration Service with Amazon CloudWatch
AWS App Runner	AWS/AppRunner	Viewing App Runner service metrics reported to CloudWatch
AppStream 2.0	AWS/AppStream	Monitoring Amazon AppStream 2.0 Resources
AWS AppSync	AWS/AppSync	CloudWatch Metrics
Amazon Athena	AWS/Athena	Monitoring Athena Queries with CloudWatch Metrics
Amazon Aurora	AWS/RDS	Amazon Aurora metrics
AWS Backup	AWS/Backup	Monitoring AWS Backup Metrics with CloudWatch
Amazon Bedrock	AWS/Bedrock	Monitoring Amazon Bedrock with Amazon CloudWatch
AWS Billing and Cost Management	AWS/Billing	Monitoring Charges with Alerts and Notifications

Service	Namespace	Documentation
Amazon Braket	AWS/Braket/ By Device	Monitoring Amazon Braket with Amazon CloudWatch
AWS Certificate Manager	AWS/CertificateManager	Supported CloudWatch Metrics
AWS Private CA	AWS/ACMPPrivateCA	Supported CloudWatch Metrics
AWS Chatbot	AWS/Chatbot	Monitoring AWS Chatbot with Amazon CloudWatch
Amazon Chime	AWS/ChimeVoiceConnector	Monitoring Amazon Chime with Amazon CloudWatch
Amazon Chime SDK	AWS/ChimeSDK	Service metrics
AWS Client VPN	AWS/ClientVPN	Monitoring with Amazon CloudWatch
Amazon CloudFront	AWS/CloudFront	Monitoring CloudFront Activity Using CloudWatch
AWS CloudHSM	AWS/CloudHSM	Getting CloudWatch Metrics
Amazon CloudSearch	AWS/CloudSearch	Monitoring an Amazon CloudSearch Domain with Amazon CloudWatch
AWS CloudTrail	AWS/CloudTrail	Supported CloudWatch metrics
CloudWatch agent	CWAgent or a custom namespace	Metrics collected by the CloudWatch agent

Service	Namespace	Documentation
CloudWatch Application Signals	ApplicationSignals	Standard application metrics collected
CloudWatch metric streams	AWS/CloudWatch/MetricStreams	Monitoring your metric streams with CloudWatch metrics
CloudWatch RUM	AWS/RUM	CloudWatch metrics that you can collect with CloudWatch RUM
CloudWatch Synthetics	CloudWatchSynthetics	CloudWatch metrics published by canaries
Amazon CloudWatch Logs	AWS/Logs	Monitoring Usage with CloudWatch Metrics
AWS CodeBuild	AWS/CodeBuild	Monitoring AWS CodeBuild
Amazon CodeGuru Reviewer		Monitoring CodeGuru Reviewer with Amazon CloudWatch
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch
Amazon Q Developer	AWS/CodeWhisperer	Monitoring Amazon Q Developer with Amazon CloudWatch
Amazon Cognito	AWS/Cognito	Monitoring Amazon Cognito
Amazon Comprehend	AWS/Comprehend	Monitoring Amazon Comprehend endpoints
AWS Config	AWS/Config	AWS Config Usage and Success Metrics

Service	Namespace	Documentation
Amazon Connect	AWS/Connect	Monitoring Amazon Connect in Amazon CloudWatch Metrics
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	Monitor your policies using Amazon CloudWatch
AWS DataSync	AWS/DataSync	Monitoring Your Task
Amazon DataZone		Monitoring Amazon DataZone with Amazon CloudWatch
Amazon DevOps Guru	AWS/DevOps-Guru	Monitoring Amazon DevOps Guru with Amazon CloudWatch
AWS Database Migration Service	AWS/DMS	Monitoring AWS DMS Tasks
AWS Direct Connect	AWS/DX	Monitoring with Amazon CloudWatch
AWS Directory Service	AWS/DirectoryService	Use Amazon CloudWatch metrics to determine when to add domain controllers
Amazon DocumentDB	AWS/DocDB	Amazon DocumentDB Metrics
Amazon DynamoDB	AWS/DynamoDB	DynamoDB Metrics and Dimensions
DynamoDB Accelerator (DAX)	AWS/DAX	Viewing DAX Metrics and Dimensions

Service	Namespace	Documentation
Amazon EC2	AWS/EC2	Monitoring Your Instances Using CloudWatch
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	Using CloudWatch metrics to monitor Elastic Graphics
Amazon EC2 Spot Fleet	AWS/EC2Spot	CloudWatch Metrics for Spot Fleet
Amazon EC2 Auto Scaling	AWS/AutoScaling	Monitoring Your Auto Scaling Groups and Instances Using CloudWatch
AWS Elastic Beanstalk	AWS/ElasticBeanstalk	Publishing Amazon CloudWatch Custom Metrics for an Environment
Amazon Elastic Block Store	AWS/EBS	Amazon CloudWatch Metrics for Amazon EBS
Amazon Elastic Container Registry	AWS/ECR	Amazon ECR repository metrics
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch Metrics
Amazon ECS through CloudWatch Container Insights	ECS/ContainerInsights	Amazon ECS Container Insights metrics
Amazon ECS Cluster auto scaling	AWS/ECS/ManagedScaling	Amazon ECS cluster auto scaling

Service	Namespace	Documentation
AWS Elastic Disaster Recovery		CloudWatch Metrics for DRS
Amazon Elastic File System	AWS/EFS	Monitoring with CloudWatch
Amazon Elastic Inference	AWS/ElasticInference	Using CloudWatch Metrics to Monitor Amazon Elastic Inference
Amazon EKS through CloudWatch Container Insights	Container Insights	Amazon EKS and Kubernetes Container Insights metrics
Elastic Load Balancing	AWS/ApplicationELB	CloudWatch Metrics for your Application Load Balancer
Elastic Load Balancing	AWS/NetworkELB	CloudWatch Metrics for your Network Load Balancer
Elastic Load Balancing	AWS/GatewayELB	CloudWatch Metrics for your Gateway Load Balancer
Elastic Load Balancing	AWS/ELB	CloudWatch Metrics for your Classic Load Balancer
Amazon Elastic Transcoder	AWS/ElasticTranscoder	Monitoring with Amazon CloudWatch
Amazon ElastiCache for Memcached	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics

Service	Namespace	Documentation
Amazon ElastiCache for Redis	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon OpenSearch Service	AWS/ES	Monitoring OpenSearch cluster metrics with Amazon CloudWatch
Amazon EMR	AWS/ElasticMapReduce	Monitor Metrics with CloudWatch
AWS Elemental MediaConnect	AWS/MediaConnect	Monitoring MediaConnect with Amazon CloudWatch
AWS Elemental MediaConvert	AWS/MediaConvert	Using CloudWatch Metrics to View Metrics for AWS Elemental MediaConvert Resources
AWS Elemental MediaLive	AWS/MediaLive	Monitoring activity using Amazon CloudWatch metrics
AWS Elemental MediaPackage	AWS/MediaPackage	Monitoring AWS Elemental MediaPackage with Amazon CloudWatch Metrics
AWS Elemental MediaStore	AWS/MediaStore	Monitoring AWS Elemental MediaStore with Amazon CloudWatch Metrics
AWS Elemental MediaTailor	AWS/MediaTailor	Monitoring AWS Elemental MediaTailor with Amazon CloudWatch
Amazon EventBridge	AWS/Events	Monitoring Amazon EventBridge
Amazon FinSpace		Logging and monitoring

Service	Namespace	Documentation
Amazon Forecast		CloudWatch Metrics for Amazon Forecast
Amazon Fraud Detector		Monitoring Amazon Fraud Detector with Amazon CloudWatch
Amazon FSx for Lustre	AWS/FSx	Monitoring Amazon FSx for Lustre
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon FSx for Windows File Server	AWS/FSx	Monitoring Amazon FSx for Windows File Server
Amazon FSx for NetApp ONTAP	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon GameLift	AWS/GameLift	Monitor Amazon GameLift with CloudWatch
AWS Global Accelerator	AWS/GlobalAccelerator	Using Amazon CloudWatch with AWS Global Accelerator
AWS Glue	Glue	Monitoring AWS Glue Using CloudWatch Metrics
AWS Ground Station	AWS/GroundStation	Metrics Using Amazon CloudWatch
AWS HealthLake	AWS/HealthLake	Monitoring HealthLake with CloudWatch

Service	Namespace	Documentation
Amazon Inspector	AWS/Inspector	Monitoring Amazon Inspector Using CloudWatch
Amazon Interactive Video Service	AWS/IVS	Monitoring Amazon IVS with Amazon CloudWatch
Amazon Interactive Video Service Chat	AWS/IVSChat	Monitoring Amazon IVS with Amazon CloudWatch
AWS IoT	AWS/IoT	AWS IoT Metrics and Dimensions
AWS IoT Analytics	AWS/IoTAnalytics	Namespace, Metrics, and Dimensions
AWS IoT FleetWise	AWS/IoTFleetWise	Monitoring AWS IoT FleetWise with Amazon CloudWatch
AWS IoT SiteWise	AWS/IoTSiteWise	Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics
AWS IoT TwinMaker	AWS/IoTTwinMaker	Monitoring AWS IoT TwinMaker with Amazon CloudWatch metrics
AWS IoT 1-Click		Monitoring AWS IoT 1-Click with Amazon CloudWatch
AWS Key Management Service	AWS/KMS	Monitoring with CloudWatch
Amazon Keyspaces (for Apache Cassandra)	AWS/Cassandra	Amazon Keyspaces Metrics and Dimensions
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch

Service	Namespace	Documentation
Amazon Managed Service for Apache Flink	AWS/KinesisAnalytics	<p><i>Managed Service for Apache Flink for SQL Applications:</i> Monitoring with CloudWatch</p> <p><i>Managed Service for Apache Flink for Apache Flink:</i> Viewing Amazon Managed Service for Apache Flink Metrics and Dimensions</p>
Amazon Data Firehose	AWS/Firehose	Monitoring Firehose Using CloudWatch Metrics
Amazon Kinesis Data Streams	AWS/Kinesis	Monitoring Amazon Kinesis Data Streams with Amazon CloudWatch
Amazon Kinesis Video Streams	AWS/KinesisVideo	Monitoring Kinesis Video Streams Metrics with CloudWatch
AWS Lambda	AWS/Lambda	AWS Lambda Metrics
Amazon Lex	AWS/Lex	Monitoring Amazon Lex with Amazon CloudWatch
AWS License Manager	AWS/LicenseManager/licenseUsage AWS/LicenseManager/LinuxSubscriptions	<p>Monitoring license usage with Amazon CloudWatch</p> <p>Usage metrics and Amazon CloudWatch alarms for Linux subscriptions</p>
Amazon Location Service	AWS/Location	Amazon Location Service metrics exported to Amazon CloudWatch
Amazon Lookout for Equipment	AWS/lookoutequipment	Monitoring Lookout for Equipment with Amazon CloudWatch

Service	Namespace	Documentation
Amazon Lookout for Metrics	AWS/LookoutMetrics	Monitoring Lookout for Metrics with Amazon CloudWatch
Amazon Lookout for Vision	AWS/LookoutVision	Monitoring Lookout for Vision with Amazon CloudWatch
AWS Mainframe Modernization		Monitoring AWS Mainframe Modernization with Amazon CloudWatch
Amazon Machine Learning	AWS/ML	Monitoring Amazon ML with CloudWatch Metrics
Amazon Managed Blockchain	AWS/managedblockchain	Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain
Amazon Managed Service for Prometheus	AWS/Prometheus	Amazon CloudWatch metrics
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	Monitoring Amazon MSK with Amazon CloudWatch
Amazon Managed Streaming for Apache Kafka	AWS/Kafka Connect	Monitoring MSK Connect
Amazon Managed Workflows for Apache Airflow	AWS/MWAA	Container, queue, and database metrics for Amazon MWAA

Service	Namespace	Documentation
Amazon MemoryDB for Redis	AWS/MemoryDB	Monitoring CloudWatch metrics
Amazon MQ	AWS/AmazonMQ	Monitoring Amazon MQ Brokers Using Amazon CloudWatch
Amazon Neptune	AWS/Neptune	Monitoring Neptune with CloudWatch
AWS Network Firewall	AWS/NetworkFirewall	AWS Network Firewall metrics in Amazon CloudWatch
AWS Network Manager	AWS/NetworkManager	CloudWatch metrics for on-premises resources
Amazon Nimble Studio	AWS/NimbleStudio	Monitoring Nimble Studio with Amazon CloudWatch
AWS HealthOmics	AWS/Omics	Monitoring AWS HealthOmics with Amazon CloudWatch
AWS OpsWorks	AWS/OpsWorks	Monitoring Stacks using Amazon CloudWatch
AWS Outposts	AWS/Outposts	CloudWatch metrics for AWS Outposts
AWS Panorama	AWS/PanoramaDeviceMetrics	Monitoring appliances and applications with Amazon CloudWatch
Amazon Personalize	AWS/Personalize	CloudWatch metrics for Amazon Personalize
Amazon Pinpoint	AWS/Pinpoint	View Amazon Pinpoint metrics in CloudWatch

Service	Namespace	Documentation
Amazon Polly	AWS/Polly	Integrating CloudWatch with Amazon Polly
AWS PrivateLink	AWS/PrivateLinkEndpoints	CloudWatch metrics for AWS PrivateLink
AWS PrivateLink	AWS/PrivateLinkServices	CloudWatch metrics for AWS PrivateLink
AWS Private 5G	AWS/Private5G	Amazon CloudWatch metrics
Amazon QLDB	AWS/QLDB	Monitoring data in Amazon QuickSight
Amazon QuickSight	AWS/QuickSight	Monitoring with Amazon CloudWatch
Amazon Redshift	AWS/Redshift	Amazon Redshift Performance Data
Amazon Relational Database Service	AWS/RDS	Monitoring Amazon RDS metrics with Amazon CloudWatch
Amazon Rekognition	AWS/Rekognition	Monitoring Rekognition with Amazon CloudWatch
AWS re:Post Private	AWS/rePostPrivate	Monitoring AWS re:Post Private with Amazon CloudWatch
AWS RoboMaker	AWS/RoboMaker	Monitoring AWS RoboMaker with Amazon CloudWatch
Amazon Route 53	AWS/Route53	Monitoring Amazon Route 53

Service	Namespace	Documentation
Route 53 Application Recovery Controller	AWS/Route53RecoveryReadiness	Using Amazon CloudWatch with Application Recovery Controller
Amazon SageMaker	AWS/SageMaker	Monitoring SageMaker with CloudWatch
Amazon SageMaker Model Building Pipelines	AWS/SageMaker/ModelBuildingPipeline	SageMaker Pipelines Metrics
AWS Secrets Manager	AWS/SecretsManager	Monitoring Secrets Manager with Amazon CloudWatch
Amazon Security Lake	AWS/SecurityLake	CloudWatch metrics for Amazon Security Lake
Service Catalog	AWS/ServiceCatalog	Service Catalog CloudWatch Metrics
AWS Shield Advanced	AWS/DDoSProtection	Monitoring with CloudWatch
Amazon Simple Email Service	AWS/SES	Retrieving Amazon SES Event Data from CloudWatch
AWS SimSpace Weaver	AWS/simspaceweaver	Monitoring AWS SimSpace Weaver with Amazon CloudWatch
Amazon Simple Notification Service	AWS/SNS	Monitoring Amazon SNS with CloudWatch
Amazon Simple Queue Service	AWS/SQS	Monitoring Amazon SQS Queues Using CloudWatch

Service	Namespace	Documentation
Amazon S3	AWS/S3	Monitoring Metrics with Amazon CloudWatch
S3 Storage Lens	AWS/S3/Storage-Lens	Monitor S3 Storage Lens metrics in CloudWatch
Amazon Simple Workflow Service	AWS/SWF	Amazon SWF Metrics for CloudWatch
AWS Step Functions	AWS/States	Monitoring Step Functions Using CloudWatch
AWS Storage Gateway	AWS/StorageGateway	Using Amazon CloudWatch metrics
AWS Systems Manager Run Command	AWS/SSM-RunCommand	Monitoring Run Command Metrics Using CloudWatch
Amazon Textract	AWS/Textract	CloudWatch Metrics for Amazon Textract
Amazon Timestream	AWS/Timestream	Timestream Metrics and Dimensions
AWS Transfer for SFTP	AWS/Transfer	AWS SFTP CloudWatch Metrics
Amazon Transcribe	AWS/Transcribe	Monitoring Amazon Transcribe with Amazon CloudWatch
Amazon Translate	AWS/Translate	CloudWatch Metrics and Dimensions for Amazon Translate
AWS Trusted Advisor	AWS/TrustedAdvisor	Creating Trusted Advisor Alarms Using CloudWatch

Service	Namespace	Documentation
Amazon VPC	AWS/NATGateway	Monitoring Your NAT Gateway with CloudWatch
Amazon VPC	AWS/TransitGateway	CloudWatch Metrics for Your Transit Gateways
Amazon VPC	AWS/VPN	Monitoring with CloudWatch
Amazon VPC IP Address Manager	AWS/IPAM	Create alarms with Amazon CloudWatch
AWS WAF	AWS/WAFV2 for AWS WAF resources WAF for AWS WAF classic resources	Monitoring with CloudWatch
Amazon WorkMail	AWS/WorkMail	Monitoring Amazon WorkMail with Amazon CloudWatch
Amazon WorkSpaces	AWS/WorkSpaces	Monitor Your WorkSpaces Using CloudWatch Metrics
Amazon WorkSpaces Web	AWS/WorkSpacesWeb	Monitoring Amazon WorkSpaces Web with Amazon CloudWatch

AWS usage metrics

CloudWatch collects metrics that track the usage of some AWS resources and APIs. These metrics are published in the `AWS/Usage` namespace. Usage metrics in CloudWatch allow you to proactively manage usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch anomaly detection, and configuring alarms that alert you when usage approaches a threshold.

Some AWS services integrate these usage metrics with Service Quotas. For these services, you can use CloudWatch to manage your account's use of your service quotas. For more information, see [Visualizing your service quotas and setting alarms](#).

Topics

- [Visualizing your service quotas and setting alarms](#)
- [AWS API usage metrics](#)
- [CloudWatch usage metrics](#)

Visualizing your service quotas and setting alarms

For some AWS services, you can use the usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information about service quotas, see [What Is Service Quotas](#) in the *Service Quotas User Guide*.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use that monitoring account to visualize service quotas and set alarms for metrics in the source accounts that are linked to that monitoring account. For more information, see [CloudWatch cross-account observability](#).

Currently, the following services integrate their usage metrics with Service Quotas:

- AWS CloudHSM
- [Amazon Chime SDK](#)
- [Amazon CloudWatch](#)

- [Amazon CloudWatch Logs](#)
- [Amazon DynamoDB](#)
- [Amazon EC2](#)
- [Amazon Elastic Container Registry](#)
- Elastic Load Balancing
- AWS Fargate
- [AWS Fault Injection Service](#)
- [AWS Interactive Video Service](#)
- AWS Key Management Service
- [Amazon Data Firehose](#)
- [Amazon Location Service](#)
- [Amazon Managed Blockchain \(AMB\) Query](#)
- [AWS RoboMaker](#)
- Amazon SageMaker

To visualize a service quota and optionally set an alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **Usage**, and then choose **By AWS Resource**.

The list of service quota usage metrics appears.

4. Select the check box next to one of the metrics.

The graph displays your current usage of that AWS resource.

5. To add your service quota to the graph, do the following:
 - a. Choose the **Graphed metrics** tab.
 - b. Choose **Math expression, Start with an empty expression**. In the new row, under **Details**, enter **SERVICE_QUOTA(m1)**.

A new line is added to the graph, displaying the service quota for the resource represented in the metric.

6. To see your current usage as a percentage of the quota, add a new expression or change the current **SERVICE_QUOTA** expression. The new expression to use is **m1/SERVICE_QUOTA(m1)*100**.
7. (Optional) To set an alarm that notifies you if you approach the service quota, do the following:
 - a. On the row with **m1/SERVICE_QUOTA(m1)*100**, under **Actions**, choose the alarm icon. It looks like a bell.

The alarm creation page appears.

- b. Under **Conditions**, ensure that **Threshold type** is **Static** and **Whenever Expression1 is** set to **Greater**. Under **than**, enter **80**. This creates an alarm that goes into ALARM state when your usage exceeds 80 percent of the quota.
- c. Choose **Next**.
- d. On the next page, select an Amazon SNS topic or create a new one, and then choose **Next**. The topic you select is notified when the alarm goes to ALARM state.
- e. On the next page, enter a name and description for the alarm, and then choose **Next**.
- f. Choose **Create alarm**.

AWS API usage metrics

Most APIs that support AWS CloudTrail logging also report usage metrics to CloudWatch. API usage metrics in CloudWatch allow you to proactively manage API usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch Anomaly Detection, and configuring alarms that alert when usage approaches a threshold.

The following table lists the services that report API usage metrics to CloudWatch, and the value to use for the Service dimension to see the usage metrics from that service.

Service	Value for the Service dimension
AWS Identity and Access Management Access Analyzer	Access Analyzer
AWS Account Management	Account Management

Service	Value for the Service dimension
Alexa for Business	A4B
Amazon API Gateway	API Gateway
AWS App Mesh	App Mesh
AWS AppConfig	AWS AppConfig
Amazon AppFlow	AppFlow
Application Auto Scaling	Application Auto Scaling
Application Discovery Service	Application Discovery Service
Amazon AppStream	AppStream
AppStream 2.0 Image Builder	Image Builder
Amazon Athena	Athena
AWS Audit Manager	Audit Manager
AWS Backup	Backup
AWS Batch	Batch
Amazon Braket	Braket
AWS Budgets	Budgets
AWS Certificate Manager	Certificate Manager
Amazon Chime SDK	ChimeSDK
Amazon Cloud Directory	Cloud Directory
AWS Cloud Map	Cloud Map
AWS CloudFormation	CloudFormation

Service	Value for the Service dimension
AWS CloudHSM	CloudHSM
Amazon CloudSearch	CloudSearch
AWS CloudShell	CloudShell
AWS CloudTrail	CloudTrail
Amazon CloudWatch	CloudWatch
Amazon CloudWatch Application Signals	CloudWatch Application Signals
Amazon CloudWatch Logs	Logs
Amazon CloudWatch Application Insights	CloudWatch Application Insights
AWS CodeBuild	CodeBuild
AWS CodeCommit	CodeCommit
Amazon CodeGuru Profiler	CodeGuru Profiler
AWS CodePipeline	CodePipeline
AWS CodeStar	CodeStar
AWS CodeStar Notifications	CodeStar Notifications
AWS CodeStar Connections	CodeStar Connections
Amazon Cognito Identity pools	Cognito Identity Pools
Amazon Cognito Sync	Cognito Sync
Amazon Comprehend	Comprehend
Amazon Comprehend Medical	Comprehend Medical
AWS Compute Optimizer	ComputeOptimizier

Service	Value for the Service dimension
Amazon Connect	Connect
Amazon Connect Customer Profiles	Customer Profiles
AWS Cost and Usage Reports	Cost and Usage Report
AWS Cost Explorer	Cost Explorer
AWS Data Exchange	Data Exchange
AWS Data Lifecycle Manager	Data Lifecycle Manager
AWS Database Migration Service	Database Migration Service
AWS DataSync	DataSync
AWS DeepLens	AWS DeepLens
Amazon Detective	Detective
Device Advisor	Device Advisor
AWS Direct Connect	Direct Connect
AWS Directory Service	Directory Service
DynamoDB Accelerator	DynamoDBAccelerator
Amazon EC2	EC2
EC2 Auto Scaling	EC2 Auto Scaling
Amazon Elastic Container Registry	ECR Public
Amazon Elastic Container Service	ECS
Amazon Elastic File System	EFS
Amazon Elastic Kubernetes Service	EKS

Service	Value for the Service dimension
AWS Elastic Beanstalk	Elastic Beanstalk
Amazon Elastic Inference	Elastic Inference
Elastic Load Balancing	Elastic Load Balancing
Amazon EMR	EMR Containers
AWS Firewall Manager	Firewall Manager
Amazon FSx	FSx
Amazon GameLift	GameLift
AWS Glue DataBrew	DataBrew
Amazon Managed Grafana	Grafana
AWS IoT Greengrass	Greengrass
AWS Ground Station	Ground Station
AWS Health APIs And Notifications	AWS Health APIs And Notifications
Amazon Interactive Video Service	IVS
AWS IoT Core	IoT
AWS IoT 1-Click	IoT 1-Click
AWS IoT Events	IoT Events
AWS IoT RoboRunner	IoT RoboRunner
AWS IoT SiteWise	IoT Sitewise
AWS IoT Wireless	IoT Wireless
Amazon Kendra	Kendra

Service	Value for the Service dimension
Amazon Keyspaces (for Apache Cassandra)	Keyspaces
Amazon Managed Service for Apache Flink	Kinesis Analytics
Amazon Data Firehose	Firehose
Kinesis Video Streams	Kinesis Video Streams
AWS Key Management Service	KMS
AWS Lambda	Lambda
AWS Launch Wizard	Launch Wizard
Amazon Lex	Amazon Lex
Amazon Lightsail	Lightsail
Amazon Location Service	Location
Amazon Lookout for Vision	Lookout for Vision
Amazon Machine Learning	Amazon Machine Learning
Amazon Macie	Macie
Amazon Managed Blockchain (AMB) Query	Amazon Managed Blockchain Query
AWS Managed Services	AWS Managed Services
AWS Marketplace Commerce Analytics	Marketplace Analytics Service
AWS Elemental MediaConnect	MediaConnect
AWS Elemental MediaConvert	MediaConvert
AWS Elemental MediaLive	MediaLive
AWS Elemental MediaStore	Mediastore

Service	Value for the Service dimension
AWS Elemental MediaTailor	MediaTailor
AWS Mobile Hub	Mobile Hub
AWS Network Firewall	Network Firewall
AWS OpsWorks	OpsWorks
AWS OpsWorks for Configuration Management	OpsWorks CM
AWS Outposts	Outposts
AWS Organizations	Organizations
Amazon RDS Performance Insights	Performance Insights
Amazon Pinpoint	Pinpoint
AWS Private Certificate Authority	Private Certificate Authority
Amazon Managed Service for Prometheus	Prometheus
AWS Proton	Proton
Amazon Quantum Ledger Database (Amazon QLDB)	QLDB
Amazon RDS	RDS
Amazon Redshift	Redshift Data API
Amazon Rekognition	Rekognition
AWS Resource Access Manager	Resource Access Manager
AWS Resource Groups	Resource Groups
AWS Resource Groups Tagging API	Resource Groups Tagging API

Service	Value for the Service dimension
AWS RoboMaker	RoboMaker
Amazon Route 53 Domains	Route 53 Domains
Amazon Route 53 Resolver	Route 53 Resolver
Amazon S3	S3
Amazon S3 Glacier	Amazon S3 Glacier
Amazon SageMaker Runtime	Sagemaker
Savings Plans	Savings Plans
AWS Secrets Manager	Secrets Manager
AWS Security Hub	Security Hub
AWS Server Migration Service	AWS Server Migration Service
AWS Service Catalog AppRegistry	Service Catalog AppRegistry
Service Quotas	Service Quotas
AWS Shield	Shield
AWS Signer	Signer
Amazon Simple Notification Service	SNS
Amazon Simple Email Service	SES
Amazon Simple Queue Service	SQS
Identity Store	Identity Store
Storage Gateway	Storage Gateway
AWS Support	Support

Service	Value for the Service dimension
Amazon Simple Workflow Service	SWF
Amazon Textract	Textract
AWS IoT Things Graph	ThingsGraph
Amazon Timestream	Timestream
Amazon Transcribe	Transcribe
Amazon Translate	Translate
Amazon Transcribe streaming transcription	Transcribe Streaming
AWS Transfer Family	Transfer
AWS WAF	WAF
Amazon WorkDocs	Amazon WorkDocs
Amazon WorkLink	WorkLink
Amazon WorkMail	Amazon WorkMail
Amazon WorkSpaces	Workspaces
AWS X-Ray	X-Ray

Some services report usage metrics for additional APIs as well. To see whether an API reports usage metrics to CloudWatch, use the CloudWatch console to see the metrics reported by that service in the `AWS/Usage` namespace.

To see the list of a service's APIs that report usage metrics to CloudWatch

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **Usage**, and then choose **By AWS Resource**.

4. In the search box near the list of metrics, enter the name of the service. The metrics are filtered by the service you entered.

CloudWatch usage metrics

CloudWatch collects metrics that track the usage of some AWS resources. These metrics correspond to AWS service quotas. Tracking these metrics can help you proactively manage your quotas. For more information, see [Visualizing your service quotas and setting alarms](#).

Service quota usage metrics are in the AWS/Usage namespace and are collected every minute.

Currently, the only metric name in this namespace that CloudWatch publishes is `CallCount`. This metric is published with the dimensions `Resource`, `Service`, and `Type`. The `Resource` dimension specifies the name of the API operation being tracked. For example, the `CallCount` metric with the dimensions `"Service": "CloudWatch"`, `"Type": "API"` and `"Resource": "PutMetricData"` indicates the number of times the CloudWatch `PutMetricData` API operation has been called in your account.

The `CallCount` metric does not have a specified unit. The most useful statistic for the metric is `SUM`, which represents the total operation count for the 1-minute period.

Metrics

Metric	Description
<code>CallCount</code>	The number of specified operations performed in your account.

Dimensions

Dimension	Description
<code>Service</code>	The name of the AWS service containing the resource. For CloudWatch usage metrics, the value for this dimension is <code>CloudWatch</code> .
<code>Class</code>	The class of resource being tracked. CloudWatch API usage metrics use this dimension with a value of <code>None</code> .

Dimension	Description
Type	The type of resource being tracked. Currently, when the Service dimension is CloudWatch , the only valid value for Type is API.
Resource	The name of the API operation. Valid values include the following: DeleteAlarms , DeleteDashboards , DescribeAlarmHistory , DescribeAlarms , GetDashboard , GetMetricData , GetMetricStatistics , ListMetrics , PutDashboard , and PutMetricData

CloudWatch tutorials

The following scenarios illustrate uses of Amazon CloudWatch. In the first scenario, you use the CloudWatch console to create a billing alarm that tracks your AWS usage and lets you know when you have exceeded a certain spending threshold. In the second, more advanced scenario, you use the AWS Command Line Interface (AWS CLI) to publish a single metric for a hypothetical application named *GetStarted*.

Scenarios

- [Monitor your estimated charges](#)
- [Publish metrics](#)

Scenario: Monitor your estimated charges using CloudWatch

In this scenario, you create an Amazon CloudWatch alarm to monitor your estimated charges. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and reflects worldwide charges. This data includes the estimated charges for every service in AWS that you use, as well as the estimated overall total of your AWS charges.

You can choose to receive alerts by email when charges have exceeded a certain threshold. These alerts are triggered by CloudWatch and messages are sent using Amazon Simple Notification Service (Amazon SNS).

Note

For information about analyzing CloudWatch charges that you have already been billed for, see [CloudWatch billing and cost](#).

Tasks

- [Step 1: Enable billing alerts](#)
- [Step 2: Create a billing alarm](#)

- [Step 3: Check the alarm status](#)
- [Step 4: Edit a billing alarm](#)
- [Step 5: Delete a billing alarm](#)

Step 1: Enable billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated AWS charges and create an alarm using billing metric data. After you enable billing alerts, you cannot disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using root user credentials or as a user who has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see [AWS Partner Network](#).

To enable the monitoring of estimated charges

1. Open the AWS Billing console at <https://console.aws.amazon.com/billing/>.
2. In the navigation pane, choose **Billing Preferences**.
3. By **Alert preferences** choose **Edit**.
4. Choose **Receive CloudWatch Billing Alerts**.
5. Choose **Save preferences**.

Step 2: Create a billing alarm

Important

Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing). For more information, see [Step 1: Enable billing alerts](#).

In this procedure, you create an alarm that sends a notification when your estimated charges for AWS exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In **Browse**, choose **Billing**, and then choose **Total Estimated Charge**.

Note

If you don't see the **Billing/Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see [Enabling billing alerts](#).

5. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.
6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
10. For **than . . .**, define the value that you want to cause your alarm to trigger. For example, **200 USD**.

The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see [What is AWS Billing?](#).

Note

After you define a threshold value, the preview graph displays your estimated charges for the current month.

11. Choose **Additional Configuration** and do the following:
 - For **Datapoints to alarm**, specify **1 out of 1**.
 - For **Missing data treatment**, choose **Treat missing data as missing**.
12. Choose **Next**.
13. Under **Notification**, ensure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the ALARM state. The Amazon SNS topic can include your email address so that you receive email when the billing amount crosses the threshold that you specified.

You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.
14. Choose **Next**.
15. Under **Name and description**, enter a name for your alarm.
 - (Optional) Enter a description of your alarm.
16. Choose **Next**.
17. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

Step 3: Check the alarm status

Now, check the status of the billing alarm that you just created.

To check the alarm status

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.

3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm. Until the subscription is confirmed, it is shown as "Pending confirmation". After the subscription is confirmed, refresh the console to show the updated status.

Step 4: Edit a billing alarm

For example, you may want to increase the amount of money you spend with AWS each month from \$200 to \$400. You can edit your existing billing alarm and increase the monetary amount that must be exceeded before the alarm is triggered.

To edit a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm and choose **Actions, Modify**.
5. For **Whenever my total AWS charges for the month exceed**, specify the new amount that must be exceeded to trigger the alarm and send an email notification.
6. Choose **Save Changes**.

Step 5: Delete a billing alarm

If you no longer need your billing alarm, you can delete it.

To delete a billing alarm

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**.
4. Select the check box next to the alarm and choose **Actions, Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Scenario: Publish metrics to CloudWatch

In this scenario, you use the AWS Command Line Interface (AWS CLI) to publish a single metric for a hypothetical application named *GetStarted*. If you haven't already installed and configured the AWS CLI, see [Getting Set Up with the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

Tasks

- [Step 1: Define the data configuration](#)
- [Step 2: Add metrics to CloudWatch](#)
- [Step 3: Get statistics from CloudWatch](#)
- [Step 4: View graphs with the console](#)

Step 1: Define the data configuration

In this scenario, you publish data points that track the request latency for the application. Choose names for your metric and namespace that make sense to you. For this example, name the metric *RequestLatency* and place all of the data points into the *GetStarted* namespace.

You publish several data points that collectively represent three hours of latency data. The raw data comprises 15 request latency readings distributed over three hours. Each reading is in milliseconds:

- Hour one: 87, 51, 125, 235
- Hour two: 121, 113, 189, 65, 89
- Hour three: 100, 47, 133, 98, 100, 328

You can publish data to CloudWatch as single data points or as an aggregated set of data points called a *statistic set*. You can aggregate metrics to a granularity as low as one minute. You can publish the aggregated data points to CloudWatch as a set of statistics with four predefined keys: `Sum`, `Minimum`, `Maximum`, and `SampleCount`.

You publish the data points from hour one as single data points. For the data from hours two and three, you aggregate the data points and publish a statistic set for each hour. The key values are shown in the following table.

Hour	Raw data	Sum	Minimum	Maximum	Sample Count
1	87				
1	51				
1	125				
1	235				
2	121, 113, 189, 65, 89	577	65	189	5
3	100, 47, 133, 98, 100, 328	806	47	328	6

Step 2: Add metrics to CloudWatch

After you have defined your data configuration, you are ready to add data.

To publish data points to CloudWatch

1. At a command prompt, run the following [put-metric-data](#) commands to add data for the first hour. Replace the example timestamp with a timestamp that is two hours in the past, in Universal Coordinated Time (UTC).

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 87 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 51 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 125 --unit Milliseconds
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T20:30:00Z --value 235 --unit Milliseconds
```

2. Add data for the second hour, using a timestamp that is one hour later than the first hour.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--timestamp 2016-10-14T21:30:00Z --statistic-values
Sum=577,Minimum=65,Maximum=189,SampleCount=5 --unit Milliseconds
```

3. Add data for the third hour, omitting the timestamp to default to the current time.

```
aws cloudwatch put-metric-data --metric-name RequestLatency --namespace GetStarted \
--statistic-values Sum=806,Minimum=47,Maximum=328,SampleCount=6 --unit Milliseconds
```

Step 3: Get statistics from CloudWatch

Now that you have published metrics to CloudWatch, you can retrieve statistics based on those metrics using the [get-metric-statistics](#) command as follows. Be sure to specify `--start-time` and `--end-time` far enough in the past to cover the earliest timestamp that you published.

```
aws cloudwatch get-metric-statistics --namespace GetStarted --metric-name
RequestLatency --statistics Average \
--start-time 2016-10-14T00:00:00Z --end-time 2016-10-15T00:00:00Z --period 60
```

The following is example output:

```
{
  "Datapoints": [],
  "Label": "Request:Latency"
}
```

Step 4: View graphs with the console

After you have published metrics to CloudWatch, you can use the CloudWatch console to view statistical graphs.

To view graphs of your statistics on the console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the **Navigation** pane, choose **Metrics**.
3. On the **All metrics** tab, in the search box, type **RequestLatency** and press Enter.

4. Select the check box for the **RequestLatency** metric. A graph of the metric data is displayed in the upper pane.


For more information, see [Graphing metrics](#).

Using CloudWatch with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to CloudWatch, see [Code examples for CloudWatch using AWS SDKs](#).

 **Example availability**

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Code examples for CloudWatch using AWS SDKs

The following code examples show how to use CloudWatch with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples are sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello CloudWatch

The following code examples show how to get started using CloudWatch.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using Amazon.CloudWatch;  
using Amazon.CloudWatch.Model;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Hosting;
```

```
namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
        // the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
        // profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
            host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
            ListMetricsRequest
            {
                Namespace = metricNamespace
            });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
            available in the {metricNamespace} namespace:");
        Console.WriteLine();
        foreach (var metric in response.Metrics.Take(5))
        {
            Console.WriteLine($"Metric: {metric.MetricName}");
            Console.WriteLine($"Namespace: {metric.Namespace}");
            Console.WriteLine($"Dimensions: {string.Join(", ",
                metric.Dimensions.Select(m => $"{m.Name}:{m.Value}"))}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class HelloService {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <namespace>\s

                Where:
                namespace - The namespace to filter against (for example, AWS/
EC2).\s

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String namespace = args[0];
Region region = Region.US_EAST_1;
CloudWatchClient cw = CloudWatchClient.builder()
    .region(region)
    .build();

listMets(cw, namespace);
cw.close();
}

public static void listMets(CloudWatchClient cw, String namespace) {
    try {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsIterable listRes = cw.listMetricsPaginator(request);
        listRes.stream()
            .flatMap(r -> r.metrics().stream())
            .forEach(metrics -> System.out.println(" Retrieved metric is:
" + metrics.metricName()));

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
*/
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <namespace>
        Where:
            namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for Kotlin API reference*.

Code examples

- [Actions for CloudWatch using AWS SDKs](#)
 - [Use DeleteAlarms with an AWS SDK or CLI](#)
 - [Use DeleteAnomalyDetector with an AWS SDK or CLI](#)
 - [Use DeleteDashboards with an AWS SDK or CLI](#)
 - [Use DescribeAlarmHistory with an AWS SDK or CLI](#)
 - [Use DescribeAlarms with an AWS SDK or CLI](#)
 - [Use DescribeAlarmsForMetric with an AWS SDK or CLI](#)
 - [Use DescribeAnomalyDetectors with an AWS SDK or CLI](#)
 - [Use DisableAlarmActions with an AWS SDK or CLI](#)
 - [Use EnableAlarmActions with an AWS SDK or CLI](#)
 - [Use GetDashboard with an AWS SDK or CLI](#)
 - [Use GetMetricData with an AWS SDK or CLI](#)
 - [Use GetMetricStatistics with an AWS SDK or CLI](#)
 - [Use GetMetricWidgetImage with an AWS SDK or CLI](#)
 - [Use ListDashboards with an AWS SDK or CLI](#)
 - [Use ListMetrics with an AWS SDK or CLI](#)
 - [Use PutAnomalyDetector with an AWS SDK or CLI](#)
 - [Use PutDashboard with an AWS SDK or CLI](#)
 - [Use PutMetricAlarm with an AWS SDK or CLI](#)
 - [Use PutMetricData with an AWS SDK or CLI](#)
- [Scenarios for CloudWatch using AWS SDKs](#)
 - [Get started with CloudWatch alarms using an AWS SDK](#)
 - [Get started with CloudWatch metrics, dashboards, and alarms using an AWS SDK](#)
 - [Manage CloudWatch metrics and alarms using an AWS SDK](#)
- [Cross-service examples for CloudWatch using AWS SDKs](#)
 - [Monitor performance of Amazon DynamoDB using an AWS SDK](#)

Actions for CloudWatch using AWS SDKs

The following code examples demonstrate how to perform individual CloudWatch actions with AWS SDKs. These excerpts call the CloudWatch API and are code excerpts from larger programs that must be run in context. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the [Amazon CloudWatch API Reference](#).

Examples

- [Use DeleteAlarms with an AWS SDK or CLI](#)
- [Use DeleteAnomalyDetector with an AWS SDK or CLI](#)
- [Use DeleteDashboards with an AWS SDK or CLI](#)
- [Use DescribeAlarmHistory with an AWS SDK or CLI](#)
- [Use DescribeAlarms with an AWS SDK or CLI](#)
- [Use DescribeAlarmsForMetric with an AWS SDK or CLI](#)
- [Use DescribeAnomalyDetectors with an AWS SDK or CLI](#)
- [Use DisableAlarmActions with an AWS SDK or CLI](#)
- [Use EnableAlarmActions with an AWS SDK or CLI](#)
- [Use GetDashboard with an AWS SDK or CLI](#)
- [Use GetMetricData with an AWS SDK or CLI](#)
- [Use GetMetricStatistics with an AWS SDK or CLI](#)
- [Use GetMetricWidgetImage with an AWS SDK or CLI](#)
- [Use ListDashboards with an AWS SDK or CLI](#)
- [Use ListMetrics with an AWS SDK or CLI](#)
- [Use PutAnomalyDetector with an AWS SDK or CLI](#)
- [Use PutDashboard with an AWS SDK or CLI](#)
- [Use PutMetricAlarm with an AWS SDK or CLI](#)
- [Use PutMetricData with an AWS SDK or CLI](#)

Use DeleteAlarms with an AWS SDK or CLI

The following code examples show how to use DeleteAlarms.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with alarms](#)
- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <iostream>
```

Delete the alarm.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::DeleteAlarmsRequest request;
request.AddAlarmNames(alarm_name);

auto outcome = cw.DeleteAlarms(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to delete CloudWatch alarm:" <<
        outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
        << std::endl;
}
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an alarm

The following example uses the `delete-alarms` command to delete the Amazon CloudWatch alarm named "myalarm":

```
aws cloudwatch delete-alarms --alarm-names myalarm
```

Output:

```
This command returns to the prompt if successful.
```

- For API details, see [DeleteAlarms](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
```



```
public class DeleteAlarm {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <alarmName>

            Where:
                alarmName - An alarm name to delete (for example, MyAlarm).
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_2;
        CloudWatchClient cw = CloudWatchClient.builder()
            .region(region)
            .build();

        deleteCWAlarm(cw, alarmName);
        cw.close();
    }

    public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
        try {
            DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
                .alarmNames(alarmName)
                .build();

            cw.deleteAlarms(request);
            System.out.printf("Successfully deleted alarm %s", alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [DeleteAlarms](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DeleteAlarms](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
```

```
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
        metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
  lo_cwt->deletealarms(  
    it_alarmnames = it_alarm_names  
  ).  
  MESSAGE 'Alarms deleted.' TYPE 'I'.  
CATCH /aws1/cx_cwtresourcenotfound .  
  MESSAGE 'Resource being accessed is not found.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [DeleteAlarms](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DeleteAnomalyDetector with an AWS SDK or CLI

The following code examples show how to use DeleteAnomalyDetector.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>  
/// Delete a single metric anomaly detector.  
/// </summary>  
/// <param name="anomalyDetector">The anomaly detector to delete.</param>
```

```
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var deleteAnomalyDetectorResponse = await
    _amazonCloudWatch.DeleteAnomalyDetectorAsync(
        new DeleteAnomalyDetectorRequest()
        {
            SingleMetricAnomalyDetector = anomalyDetector
        });

    return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [DeleteAnomalyDetector](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
```

```
        .metricName(customMetricName)
        .namespace(customMetricNamespace)
        .stat("Maximum")
        .build();

        DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
        .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
        .build();

        cw.deleteAnomalyDetector(request);
        System.out.println("Successfully deleted the Anomaly Detector.");

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- For API details, see [DeleteAnomalyDetector](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
}
```



```
val singleMetricAnomalyDetectorVal =
    SingleMetricAnomalyDetector {
        metricName = customMetricName
        namespace = customMetricNamespace
        stat = "Maximum"
    }

val request =
    DeleteAnomalyDetectorRequest {
        singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
    }

CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    cwClient.deleteAnomalyDetector(request)
    println("Successfully deleted the Anomaly Detector.")
}
}
```

- For API details, see [DeleteAnomalyDetector](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DeleteDashboards with an AWS SDK or CLI

The following code examples show how to use DeleteDashboards.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Delete a list of CloudWatch dashboards.
/// </summary>
/// <param name="dashboardNames">List of dashboard names to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteDashboards(List<string> dashboardNames)
{
    var deleteDashboardsResponse = await
        _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

    return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [DeleteDashboards](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
    try {
        DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
            .dashboardNames(dashboardName)
            .build();
        cw.deleteDashboards(dashboardsRequest);
        System.out.println(dashboardName + " was successfully deleted.");
    } catch (CloudWatchException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [DeleteDashboards](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}
```

- For API details, see [DeleteDashboards](#) in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Deletes the specified dashboard, promoting for confirmation before proceeding. To bypass confirmation add the `-Force` switch to the command.

```
Remove-CWDashboard -DashboardName Dashboard1
```

- For API details, see [DeleteDashboards](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeAlarmHistory with an AWS SDK or CLI

The following code examples show how to use `DescribeAlarmHistory`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>  
/// Describe the history of an alarm for a number of days in the past.  
/// </summary>
```

```
/// <param name="alarmName">The name of the alarm.</param>
/// <param name="historyDays">The number of days in the past.</param>
/// <returns>The list of alarm history data.</returns>
public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
{
    List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
    var paginatedAlarmHistory =
    _amazonCloudWatch.Paginators.DescribeAlarmHistory(
        new DescribeAlarmHistoryRequest()
        {
            AlarmName = alarmName,
            EndDateUtc = DateTime.UtcNow,
            HistoryItemType = HistoryItemType.StateUpdate,
            StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
        });

    await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
    {
        alarmHistory.Add(data);
    }
    return alarmHistory;
}
```

- For API details, see [DescribeAlarmHistory](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve history for an alarm

The following example uses the `describe-alarm-history` command to retrieve history for the Amazon CloudWatch alarm named "myalarm":

```
aws cloudwatch describe-alarm-history --alarm-name "myalarm" --history-item-type
StateUpdate
```

Output:

```
{
```

```

"AlarmHistoryItems": [
  {
    "Timestamp": "2014-04-09T18:59:06.442Z",
    "HistoryItemType": "StateUpdate",
    "AlarmName": "myalarm",
    "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue\": \"ALARM\", \"stateReason\": \"testing purposes\"}, \"newState\":{\"stateValue\": \"OK\", \"stateReason\": \"Threshold Crossed: 2 datapoints were not greater than the threshold (70.0). The most recent datapoints: [38.958, 40.292].\", \"stateReasonData\":{\"version\":\"1.0\", \"queryDate\": \"2014-04-09T18:59:06.419+0000\", \"startDate\": \"2014-04-09T18:44:00.000+0000\", \"statistic\": \"Average\", \"period\": 300, \"recentDatapoints\": [38.958, 40.292], \"threshold\": 70.0}}}",
    "HistorySummary": "Alarm updated from ALARM to OK"
  },
  {
    "Timestamp": "2014-04-09T18:59:05.805Z",
    "HistoryItemType": "StateUpdate",
    "AlarmName": "myalarm",
    "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue\": \"OK\", \"stateReason\": \"Threshold Crossed: 2 datapoints were not greater than the threshold (70.0). The most recent datapoints: [38.839999999999996, 39.714].\", \"stateReasonData\":{\"version\": \"1.0\", \"queryDate\": \"2014-03-11T22:45:41.569+0000\", \"startDate\": \"2014-03-11T22:30:00.000+0000\", \"statistic\": \"Average\", \"period\": 300, \"recentDatapoints\": [38.839999999999996, 39.714], \"threshold\": 70.0}}, \"newState\": {\"stateValue\": \"ALARM\", \"stateReason\": \"testing purposes\"}}",
    "HistorySummary": "Alarm updated from OK to ALARM"
  }
]
}

```

- For API details, see [DescribeAlarmHistory](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String alarmName = rootNode.findValue("exampleAlarmName").asText();

        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();
        DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
            .startDate(start)
            .endDate(endDate)
            .alarmName(alarmName)
            .historyItemType(HistoryItemType.ACTION)
            .build();

        DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
        List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
        if (historyItems.isEmpty()) {
            System.out.println("No alarm history data found for " + alarmName
+ ".");
        } else {
            for (AlarmHistoryItem item : historyItems) {
                System.out.println("History summary: " +
item.historySummary());
                System.out.println("Time stamp: " + item.timestamp());
            }
        }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [DescribeAlarmHistory](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest =
        DescribeAlarmHistoryRequest {
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDateVal)
            alarmName = alarmNameVal
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            }
        }
    }
}
```



```
        } else {
            for (item in historyItems) {
                println("History summary ${item.historySummary}")
                println("Time stamp: ${item.timestamp}")
            }
        }
    }
}
```

- For API details, see [DescribeAlarmHistory](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeAlarms with an AWS SDK or CLI

The following code examples show how to use DescribeAlarms.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with alarms](#)
- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Describe the current alarms, optionally filtered by state.
/// </summary>
```

```
/// <param name="stateValue">Optional filter for alarm state.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
{
    List<MetricAlarm> alarms = new List<MetricAlarm>();
    var paginatedDescribeAlarms =
    _amazonCloudWatch.Paginators.DescribeAlarms(
        new DescribeAlarmsRequest()
        {
            StateValue = stateValue
        });

    await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
    {
        alarms.Add(data);
    }
    return alarms;
}
```

- For API details, see [DescribeAlarms](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list information about an alarm

The following example uses the `describe-alarms` command to provide information about the alarm named "myalarm":

```
aws cloudwatch describe-alarms --alarm-names "myalarm"
```

Output:

```
{
  "MetricAlarms": [
    {
      "EvaluationPeriods": 2,
      "AlarmArn": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:myalarm",
```

```

    "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
    "AlarmConfigurationUpdatedTimestamp": "2012-12-27T00:49:54.032Z",
    "ComparisonOperator": "GreaterThanThreshold",
    "AlarmActions": [
      "arn:aws:sns:us-east-1:123456789012:myHighCpuAlarm"
    ],
    "Namespace": "AWS/EC2",
    "AlarmDescription": "CPU usage exceeds 70 percent",
    "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\\\"2014-04-09T18:59:06.419+0000\\\",\\\"startDate\\\":\\\"2014-04-09T18:44:00.000+0000\\\",
\\\"statistic\\\":\\\"Average\\\",\\\"period\\\":300,\\\"recentDatapoints\\\":[38.958,40.292],
\\\"threshold\\\":70.0}\",
    "Period": 300,
    "StateValue": "OK",
    "Threshold": 70.0,
    "AlarmName": "myalarm",
    "Dimensions": [
      {
        "Name": "InstanceId",
        "Value": "i-0c986c72"
      }
    ],
    "Statistic": "Average",
    "StateReason": "Threshold Crossed: 2 datapoints were not greater than
the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
    "InsufficientDataActions": [],
    "OKActions": [],
    "ActionsEnabled": true,
    "MetricName": "CPUUtilization"
  }
]
}

```

- For API details, see [DescribeAlarms](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void describeAlarms(CloudWatchClient cw) {
    try {
        List<AlarmType> typeList = new ArrayList<>();
        typeList.add(AlarmType.METRIC_ALARM);

        DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
            .alarmTypes(typeList)
            .maxRecords(10)
            .build();

        DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
        List<MetricAlarm> alarmList = response.metricAlarms();
        for (MetricAlarm alarm : alarmList) {
            System.out.println("Alarm name: " + alarm.alarmName());
            System.out.println("Alarm description: " +
alarm.alarmDescription());
        }
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [DescribeAlarms](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(alarmsRequest)
        response.metricAlarms?.forEach { alarm ->
            println("Alarm name: ${alarm.alarmName}")
            println("Alarm description: ${alarm.alarmDescription}")
        }
    }
}
```

- For API details, see [DescribeAlarms](#) in *AWS SDK for Kotlin API reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require "aws-sdk-cloudwatch"

# Lists the names of available Amazon CloudWatch alarms.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   list_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def list_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts alarm.alarm_name
    end
  else
    puts "No alarms found."
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end
```

- For API details, see [DescribeAlarms](#) in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.
    oo_result = lo_cwt->describealarms(           " oo_result is
returned for testing purposes. "
    it_alarmnames = it_alarm_names
    ).
    MESSAGE 'Alarms retrieved.' TYPE 'I'.
CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
```

```
DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-  
>av_err_msg }|.
MESSAGE lv_error TYPE 'E'.
ENDTRY.
```

- For API details, see [DescribeAlarms](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeAlarmsForMetric with an AWS SDK or CLI

The following code examples show how to use `DescribeAlarmsForMetric`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Describe the current alarms for a specific metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The name of the metric.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
```

```
{
    var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
        new DescribeAlarmsForMetricRequest()
        {
            Namespace = metricNamespace,
            MetricName = metricName
        });

    return alarmsResult.MetricAlarms;
}
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

Describe the alarms.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::DescribeAlarmsRequest request;
request.SetMaxRecords(1);

bool done = false;
```



```
bool header = false;
while (!done)
{
    auto outcome = cw.DescribeAlarms(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to describe CloudWatch alarms:" <<
            outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header)
    {
        std::cout << std::left <<
            std::setw(32) << "Name" <<
            std::setw(64) << "Arn" <<
            std::setw(64) << "Description" <<
            std::setw(20) << "LastUpdated" <<
            std::endl;
        header = true;
    }

    const auto &alarms = outcome.GetResult().GetMetricAlarms();
    for (const auto &alarm : alarms)
    {
        std::cout << std::left <<
            std::setw(32) << alarm.GetAlarmName() <<
            std::setw(64) << alarm.GetAlarmArn() <<
            std::setw(64) << alarm.GetAlarmDescription() <<
            std::setw(20) <<
            alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
                SIMPLE_DATE_FORMAT_STR) <<
            std::endl;
    }

    const auto &next_token = outcome.GetResult().GetNextToken();
    request.SetNextToken(next_token);
    done = next_token.empty();
}
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To display information about alarms associated with a metric

The following example uses the `describe-alarms-for-metric` command to display information about any alarms associated with the Amazon EC2 CPUUtilization metric and the instance with the ID `i-0c986c72`:

```
aws cloudwatch describe-alarms-for-metric --metric-name CPUUtilization --
namespace AWS/EC2 --dimensions Name=InstanceId,Value=i-0c986c72
```

Output:

```
{
  "MetricAlarms": [
    {
      "EvaluationPeriods": 10,
      "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm2",
      "StateUpdatedTimestamp": "2013-10-30T03:03:51.479Z",
      "AlarmConfigurationUpdatedTimestamp": "2013-10-30T03:03:50.865Z",
      "ComparisonOperator": "GreaterThanOrEqualToThreshold",
      "AlarmActions": [
        "arn:aws:sns:us-east-1:111122223333:NotifyMe"
      ],
      "Namespace": "AWS/EC2",
      "AlarmDescription": "CPU usage exceeds 70 percent",
      "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2013-10-30T03:03:51.479+0000\",\"startDate\":\"2013-10-30T02:08:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":
[40.698,39.612,42.432,39.796,38.816,42.28,42.854,40.088,40.760000000000005,41.316],
\"threshold\":70.0}",
      "Period": 300,
      "StateValue": "OK",
      "Threshold": 70.0,
      "AlarmName": "myHighCpuAlarm2",
      "Dimensions": [
        {
          "Name": "InstanceId",
          "Value": "i-0c986c72"
        }
      ]
    }
  ]
}
```

```

    ],
    "Statistic": "Average",
    "StateReason": "Threshold Crossed: 10 datapoints were not
greater than or equal to the threshold (70.0). The most recent datapoints:
[40.7600000000000005, 41.316].",
    "InsufficientDataActions": [],
    "OKActions": [],
    "ActionsEnabled": true,
    "MetricName": "CPUUtilization"
  },
  {
    "EvaluationPeriods": 2,
    "AlarmArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:myHighCpuAlarm",
    "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
    "AlarmConfigurationUpdatedTimestamp": "2014-04-09T22:26:05.958Z",
    "ComparisonOperator": "GreaterThanThreshold",
    "AlarmActions": [
      "arn:aws:sns:us-east-1:111122223333:HighCPUAlarm"
    ],
    "Namespace": "AWS/EC2",
    "AlarmDescription": "CPU usage exceeds 70 percent",
    "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
    "Period": 300,
    "StateValue": "OK",
    "Threshold": 70.0,
    "AlarmName": "myHighCpuAlarm",
    "Dimensions": [
      {
        "Name": "InstanceId",
        "Value": "i-0c986c72"
      }
    ],
    "Statistic": "Average",
    "StateReason": "Threshold Crossed: 2 datapoints were not greater than
the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
    "InsufficientDataActions": [],
    "OKActions": [],
    "ActionsEnabled": false,
    "MetricName": "CPUUtilization"
  }
}

```

```
]
}
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        boolean hasAlarm = false;
        int retries = 10;

        DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .build();

        while (!hasAlarm && retries > 0) {
            DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
            hasAlarm = response.hasMetricAlarms();
            retries--;
            Thread.sleep(20000);
        }
    }
}
```

```
        System.out.println(".");
    }
    if (!hasAlarm)
        System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
    else
        System.out.println("Alarm state found for " + customMetricName +
".");

    } catch (CloudWatchException | IOException | InterruptedException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
    const command = new DescribeAlarmsCommand({
        AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
        CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    });

    try {
        return await client.send(command);
    } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

```
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
            retries--
            delay(20000)
            println(".")
        }
    }
}
```

```

        if (!hasAlarm) {
            println("No Alarm state found for $customMetricName after 10
retries.")
        } else {
            println("Alarm state found for $customMetricName.")
        }
    }
}

```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()

```



```
    logger.info("Got alarms for metric %s.%s.", metric_namespace,
metric_name)
    return alarm_iter
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   describe_metric_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def describe_metric_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms

  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts "-" * 16
      puts "Name:           " + alarm.alarm_name
      puts "State value:      " + alarm.state_value
      puts "State reason:     " + alarm.state_reason
      puts "Metric:           " + alarm.metric_name
      puts "Namespace:        " + alarm.namespace
      puts "Statistic:        " + alarm.statistic
      puts "Period:           " + alarm.period.to_s
      puts "Unit:             " + alarm.unit.to_s
      puts "Eval. periods:    " + alarm.evaluation_periods.to_s
      puts "Threshold:        " + alarm.threshold.to_s
      puts "Comp. operator:   " + alarm.comparison_operator
```

```
    if alarm.key?(:ok_actions) && alarm.ok_actions.count.positive?
      puts "OK actions:"
      alarm.ok_actions.each do |a|
        puts "  " + a
      end
    end

    if alarm.key?(:alarm_actions) && alarm.alarm_actions.count.positive?
      puts "Alarm actions:"
      alarm.alarm_actions.each do |a|
        puts "  " + a
      end
    end

    if alarm.key?(:insufficient_data_actions) &&
      alarm.insufficient_data_actions.count.positive?
      puts "Insufficient data actions:"
      alarm.insufficient_data_actions.each do |a|
        puts "  " + a
      end
    end

    puts "Dimensions:"
    if alarm.key?(:dimensions) && alarm.dimensions.count.positive?
      alarm.dimensions.each do |d|
        puts "  Name: " + d.name + ", Value: " + d.value
      end
    else
      puts "  None for this alarm."
    end
  end
else
  puts "No alarms found."
end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end

# Example usage:
def run_me
  region = ""

  # Print usage information and then stop.
  if ARGV[0] == "--help" || ARGV[0] == "-h"
```

```
puts "Usage:  ruby cw-ruby-example-show-alarms.rb REGION"
puts "Example: ruby cw-ruby-example-show-alarms.rb us-east-1"
exit 1
# If no values are specified at the command prompt, use these default values.
elsif ARGV.count.zero?
  region = "us-east-1"
# Otherwise, use the values as specified at the command prompt.
else
  region = ARGV[0]
end

cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
puts "Available alarms:"
describe_metric_alarms(cloudwatch_client)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see [DescribeAlarmsForMetric](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeAnomalyDetectors with an AWS SDK or CLI

The following code examples show how to use DescribeAnomalyDetectors.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Describe anomaly detectors for a metric and namespace.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The metric of the anomaly detectors.</param>
/// <returns>The list of detectors.</returns>
public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
{
    List<AnomalyDetector> detectors = new List<AnomalyDetector>();
    var paginatedDescribeAnomalyDetectors =
    _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
        new DescribeAnomalyDetectorsRequest()
        {
            MetricName = metricName,
            Namespace = metricNamespace
        });

    await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
    {
        detectors.Add(data);
    }

    return detectors;
}
```

- For API details, see [DescribeAnomalyDetectors](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
            .maxResults(10)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .build();

        DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
        List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
        for (AnomalyDetector detector : anomalyDetectorList) {
            System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
            System.out.println("State: " + detector.stateValue());
        }
    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

```
    }  
  }  
}
```

- For API details, see [DescribeAnomalyDetectors](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun describeAnomalyDetectors(fileName: String) {  
    // Read values from the JSON file.  
    val parser = JsonFactory().createParser(File(fileName))  
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)  
    val customMetricNamespace =  
rootNode.findValue("customMetricNamespace").asText()  
    val customMetricName = rootNode.findValue("customMetricName").asText()  
  
    val detectorsRequest =  
        DescribeAnomalyDetectorsRequest {  
            maxResults = 10  
            metricName = customMetricName  
            namespace = customMetricNamespace  
        }  
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->  
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)  
        response.anomalyDetectors?.forEach { detector ->  
            println("Metric name:  
${detector.singleMetricAnomalyDetector?.metricName}")  
            println("State: ${detector.stateValue}")  
        }  
    }  
}
```

- For API details, see [DescribeAnomalyDetectors](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DisableAlarmActions` with an AWS SDK or CLI

The following code examples show how to use `DisableAlarmActions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Disable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableAlarmActions(List<string> alarmNames)
{
    var disableAlarmActionsResult = await
        _amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            }
        );
}
```

```
    return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

Disable the alarm actions.

```
Aws::CloudWatch::CloudWatchClient cw;

Aws::CloudWatch::Model::DisableAlarmActionsRequest
disableAlarmActionsRequest;
disableAlarmActionsRequest.AddAlarmNames(alarm_name);

auto disableAlarmActionsOutcome =
cw.DisableAlarmActions(disableAlarmActionsRequest);
if (!disableAlarmActionsOutcome.IsSuccess())
{
    std::cout << "Failed to disable actions for alarm " << alarm_name <<
        ": " << disableAlarmActionsOutcome.GetError().GetMessage() <<
        std::endl;
}
else
```



```
{
    std::cout << "Successfully disabled actions for alarm " <<
        alarm_name << std::endl;
}
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To disable actions for an alarm

The following example uses the `disable-alarm-actions` command to disable all actions for the alarm named `myalarm`:

```
aws cloudwatch disable-alarm-actions --alarm-names myalarm
```

This command returns to the prompt if successful.

- For API details, see [DisableAlarmActions](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
    software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
```

```
* environment, including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
*/
public class DisableAlarmActions {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <alarmName>

                Where:
                alarmName - An alarm name to disable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
            .region(region)
            .build();

        disableActions(cw, alarmName);
        cw.close();
    }

    public static void disableActions(CloudWatchClient cw, String alarmName) {
        try {
            DisableAlarmActionsRequest request =
DisableAlarmActionsRequest.builder()
                .alarmNames(alarmName)
                .build();

            cw.disableAlarmActions(request);
            System.out.printf("Successfully disabled actions on alarm %s",
alarmName);
        }
    }
}
```

```
        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
    const command = new DisableAlarmActionsCommand({
        AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
        CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    });

    try {
        return await client.send(command);
    } catch (err) {
        console.error(err);
    }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [DisableAlarmActions](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).

- For API details, see [DisableAlarmActions](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun disableActions(alarmName: String) {
    val request =
        DisableAlarmActionsRequest {
            alarmNames = listOf(alarmName)
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm $alarmName")
    }
}
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""
```

```
def __init__(self, cloudwatch_resource):
    """
    :param cloudwatch_resource: A Boto3 CloudWatch resource.
    """
    self.cloudwatch_resource = cloudwatch_resource

def enable_alarm_actions(self, alarm_name, enable):
    """
    Enables or disables actions on the specified alarm. Alarm actions can be
    used to send notifications or automate responses when an alarm enters a
    particular state.

    :param alarm_name: The name of the alarm.
    :param enable: When True, actions are enabled for the alarm. Otherwise,
they
                    disabled.
    """
    try:
        alarm = self.cloudwatch_resource.Alarm(alarm_name)
        if enable:
            alarm.enable_actions()
        else:
            alarm.disable_actions()
        logger.info(
            "%s actions for alarm %s.",
            "Enabled" if enable else "Disabled",
            alarm_name,
        )
    except ClientError:
        logger.exception(
            "Couldn't %s actions alarm %s.",
            "enable" if enable else "disable",
            alarm_name,
        )
        raise
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Disables an alarm in Amazon CloudWatch.
#
# Prerequisites.
#
# - The alarm to disable.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm to disable.
# @return [Boolean] true if the alarm was disabled; otherwise, false.
# @example
#   exit 1 unless alarm_actions_disabled?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket'
#   )
def alarm_actions_disabled?(cloudwatch_client, alarm_name)
  cloudwatch_client.disable_alarm_actions(alarm_names: [alarm_name])
  return true
rescue StandardError => e
  puts "Error disabling alarm actions: #{e.message}"
  return false
end

# Example usage:
def run_me
  alarm_name = "ObjectsInBucket"
  alarm_description = "Objects exist in this bucket for more than 1 day."
  metric_name = "NumberOfObjects"
  # Notify this Amazon Simple Notification Service (Amazon SNS) topic when
  # the alarm transitions to the ALARM state.
  alarm_actions = ["arn:aws:sns:us-
east-1:111111111111:Default_CloudWatch_Alarms_Topic"]
```

```
namespace = "AWS/S3"
statistic = "Average"
dimensions = [
  {
    name: "BucketName",
    value: "doc-example-bucket"
  },
  {
    name: "StorageType",
    value: "AllStorageTypes"
  }
]
period = 86_400 # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
unit = "Count"
evaluation_periods = 1 # More than one day.
threshold = 1 # One object.
comparison_operator = "GreaterThanThreshold" # More than one object.
# Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
region = "us-east-1"

cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

if alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  puts "Alarm '#{alarm_name}' created or updated."
else
  puts "Could not create or update alarm '#{alarm_name}'."
end

if alarm_actions_disabled?(cloudwatch_client, alarm_name)
  puts "Alarm '#{alarm_name}' disabled."
```



```
else
  puts "Could not disable alarm '#{alarm_name}'."
end
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
"Disables actions on the specified alarm. "
TRY.
  lo_cwt->disablealarmactions(
    it_alarmnames = it_alarm_names
  ).
  MESSAGE 'Alarm actions disabled.' TYPE 'I'.
CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
  DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
  MESSAGE lv_error TYPE 'E'.
ENDTRY.
```

- For API details, see [DisableAlarmActions](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use EnableAlarmActions with an AWS SDK or CLI

The following code examples show how to use EnableAlarmActions.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).


```
/// <summary>
/// Enable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableAlarmActions(List<string> alarmNames)
{
    var enableAlarmActionsResult = await
        _amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

    return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Enable the alarm actions.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::PutMetricAlarmRequest request;
request.SetAlarmName(alarm_name);
request.SetComparisonOperator(
    Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
request.SetEvaluationPeriods(1);
request.SetMetricName("CPUUtilization");
request.SetNamespace("AWS/EC2");
request.SetPeriod(60);
request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
request.SetThreshold(70.0);
request.SetActionsEnabled(false);
request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
request.AddAlarmActions(actionArn);

Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("InstanceId");
dimension.SetValue(instanceId);
request.AddDimensions(dimension);
```

```
auto outcome = cw.PutMetricAlarm(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch alarm:" <<
        outcome.GetError().GetMessage() << std::endl;
    return;
}

Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
enable_request.AddAlarmNames(alarm_name);

auto enable_outcome = cw.EnableAlarmActions(enable_request);
if (!enable_outcome.IsSuccess())
{
    std::cout << "Failed to enable alarm actions:" <<
        enable_outcome.GetError().GetMessage() << std::endl;
    return;
}

std::cout << "Successfully created alarm " << alarm_name <<
    " and enabled actions on it." << std::endl;
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To enable all actions for an alarm

The following example uses the `enable-alarm-actions` command to enable all actions for the alarm named `myalarm`:


```
aws cloudwatch enable-alarm-actions --alarm-names myalarm
```

This command returns to the prompt if successful.

- For API details, see [EnableAlarmActions](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
    software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class EnableAlarmActions {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <alarmName>

                Where:
                alarmName - An alarm name to enable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarm = args[0];
```

```
Region region = Region.US_EAST_1;
CloudWatchClient cw = CloudWatchClient.builder()
    .region(region)
    .build();

enableActions(cw, alarm);
cw.close();
}

public static void enableActions(CloudWatchClient cw, String alarm) {
    try {
        EnableAlarmActionsRequest request =
        EnableAlarmActionsRequest.builder()
            .alarmNames(alarm)
            .build();

        cw.enableAlarmActions(request);
        System.out.printf("Successfully enabled actions on alarm %s", alarm);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
```

```
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [EnableAlarmActions](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```


- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [EnableAlarmActions](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun enableActions(alarm: String) {
    val request =
        EnableAlarmActionsRequest {
            alarmNames = listOf(alarm)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println("Successfully enabled actions on alarm $alarm")
    }
}
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
```

```
"""Encapsulates Amazon CloudWatch functions."""

def __init__(self, cloudwatch_resource):
    """
    :param cloudwatch_resource: A Boto3 CloudWatch resource.
    """
    self.cloudwatch_resource = cloudwatch_resource

def enable_alarm_actions(self, alarm_name, enable):
    """
    Enables or disables actions on the specified alarm. Alarm actions can be
    used to send notifications or automate responses when an alarm enters a
    particular state.

    :param alarm_name: The name of the alarm.
    :param enable: When True, actions are enabled for the alarm. Otherwise,
they
                    disabled.
    """
    try:
        alarm = self.cloudwatch_resource.Alarm(alarm_name)
        if enable:
            alarm.enable_actions()
        else:
            alarm.disable_actions()
        logger.info(
            "%s actions for alarm %s.",
            "Enabled" if enable else "Disabled",
            alarm_name,
        )
    except ClientError:
        logger.exception(
            "Couldn't %s actions alarm %s.",
            "enable" if enable else "disable",
            alarm_name,
        )
        raise
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
"Enable actions on the specified alarm."
TRY.
  lo_cwt->enablealarmactions(
    it_alarmnames = it_alarm_names
  ).
  MESSAGE 'Alarm actions enabled.' TYPE 'I'.
CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
  DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
  MESSAGE lv_error TYPE 'E'.
ENDTRY.
```

- For API details, see [EnableAlarmActions](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetDashboard with an AWS SDK or CLI

The following code examples show how to use GetDashboard.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- For API details, see [GetDashboard](#) in *AWS SDK for .NET API Reference*.

PowerShell

Tools for PowerShell

Example 1: Returns the arn the body of the specified dashboard.

```
Get-CWDashboard -DashboardName Dashboard1
```

Output:

```
DashboardArn
```

```
DashboardBody
```

```
-----  
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- For API details, see [GetDashboard](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetMetricData with an AWS SDK or CLI

The following code examples show how to use `GetMetricData`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>  
/// Get data for CloudWatch metrics.  
/// </summary>  
/// <param name="minutesOfData">The number of minutes of data to include.</  
param>  
/// <param name="useDescendingTime">True to return the data descending by  
time.</param>  
/// <param name="endDateUtc">The end date for the data, in UTC.</param>  
/// <param name="maxDataPoints">The maximum data points to include.</param>  
/// <param name="dataQueries">Optional data queries to include.</param>  
/// <returns>A list of the requested metric data.</returns>
```

```
public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
bool useDescendingTime, DateTime? endDateUtc = null,
    int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
{
    var metricData = new List<MetricDataResult>();
    // If no end time is provided, use the current time for the end time.
    endDateUtc ??= DateTime.UtcNow;
    var timeZoneOffset =
    TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
    var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
    // The timezone string should be in the format +0000, so use the timezone
    offset to format it correctly.
    var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
    var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
        new GetMetricDataRequest()
        {
            StartTimeUtc = startTimeUtc,
            EndTimeUtc = endDateUtc.Value,
            LabelOptions = new LabelOptions { Timezone = timeZoneString },
            ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
            ScanBy.TimestampAscending,
            MaxDatapoints = maxDataPoints,
            MetricDataQueries = dataQueries,
        });
    await foreach (var data in paginatedMetricData.MetricDataResults)
    {
        metricData.Add(data);
    }
    return metricData;
}
```

- For API details, see [GetMetricData](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        // Set the date.
        Instant nowDate = Instant.now();

        long hours = 1;
        long minutes = 30;
        Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
ChronoUnit.MINUTES);

        Metric met = Metric.builder()
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .build();

        MetricStat metStat = MetricStat.builder()
            .stat("Maximum")
            .period(1)
            .metric(met)
            .build();
```

```
    MetricDataQuery dataQuery = MetricDataQuery.builder()
        .metricStat(metStat)
        .id("foo2")
        .returnData(true)
        .build();

    List<MetricDataQuery> dq = new ArrayList<>();
    dq.add(dataQuery);

    GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
        .maxDatapoints(10)
        .scanBy(ScanBy.TIMESTAMP_DESCENDING)
        .startTime(nowDate)
        .endTime(date2)
        .metricDataQueries(dq)
        .build();

    GetMetricDataResponse response = cw.getMetricData(getMetReq);
    List<MetricDataResult> data = response.metricDataResults();
    for (MetricDataResult item : data) {
        System.out.println("The label is " + item.label());
        System.out.println("The status code is " +
item.statusCode().toString());
    }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [GetMetricData](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).


```
suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
        MetricStat {
            stat = "Maximum"
            period = 1
            metric = met
        }

    val dataQuery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQuery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
        }
}
```

```
        scanBy = ScanBy.TimestampDescending
        startTime =
            aws.smithy.kotlin.runtime.time
                .Instant(nowDate)
        endTime =
            aws.smithy.kotlin.runtime.time
                .Instant(date2)
        metricDataQueries = dq
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}
```

- For API details, see [GetMetricData](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetMetricStatistics with an AWS SDK or CLI

The following code examples show how to use GetMetricStatistics.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get billing statistics using a call to a wrapper class.
/// </summary>
/// <returns>A collection of billing statistics.</returns>
private static async Task<List<Datapoint>> SetupBillingStatistics()
{
    // Make a request for EstimatedCharges with a period of one day for the
    // past seven days.
    var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
        "AWS/Billing",
        "EstimatedCharges",
        new List<string>() { "Maximum" },
        new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
        7,
        86400);

    billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

    return billingStatistics;
}

/// <summary>
/// Wrapper to get statistics for a specific CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The name of the metric.</param>
/// <param name="statistics">The list of statistics to include.</param>
/// <param name="dimensions">The list of dimensions to include.</param>
/// <param name="days">The number of days in the past to include.</param>
/// <param name="period">The period for the data.</param>
/// <returns>A list of DataPoint objects for the statistics.</returns>
```

```
public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
    string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
{
    var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
        new GetMetricStatisticsRequest()
        {
            Namespace = metricNamespace,
            MetricName = metricName,
            Dimensions = dimensions,
            Statistics = statistics,
            StartTimeUtc = DateTime.UtcNow.AddDays(-days),
            EndTimeUtc = DateTime.UtcNow,
            Period = period
        });

    return metricStatistics.Datapoints;
}
```

- For API details, see [GetMetricStatistics](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To get the CPU utilization per EC2 instance

The following example uses the `get-metric-statistics` command to get the CPU utilization for an EC2 instance with the ID `i-abcdef`.

```
aws cloudwatch get-metric-statistics --metric-name CPUUtilization --start-time
2014-04-08T23:18:00Z --end-time 2014-04-09T23:18:00Z --period 3600 --namespace
AWS/EC2 --statistics Maximum --dimensions Name=InstanceId,Value=i-abcdef
```

Output:

```
{
  "Datapoints": [
    {
      "Timestamp": "2014-04-09T11:18:00Z",
```

```
    "Maximum": 44.79,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T20:18:00Z",  
    "Maximum": 47.92,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T19:18:00Z",  
    "Maximum": 50.85,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T09:18:00Z",  
    "Maximum": 47.92,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T03:18:00Z",  
    "Maximum": 76.84,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T21:18:00Z",  
    "Maximum": 48.96,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T14:18:00Z",  
    "Maximum": 47.92,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T08:18:00Z",  
    "Maximum": 47.92,  
    "Unit": "Percent"  
  },  
  {  
    "Timestamp": "2014-04-09T16:18:00Z",  
    "Maximum": 45.55,  
    "Unit": "Percent"  
  },  
  {
```

```
    "Timestamp": "2014-04-09T06:18:00Z",
    "Maximum": 47.92,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T13:18:00Z",
    "Maximum": 45.08,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T05:18:00Z",
    "Maximum": 47.92,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T18:18:00Z",
    "Maximum": 46.88,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T17:18:00Z",
    "Maximum": 52.08,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T07:18:00Z",
    "Maximum": 47.92,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T02:18:00Z",
    "Maximum": 51.23,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-09T12:18:00Z",
    "Maximum": 47.67,
    "Unit": "Percent"
  },
  {
    "Timestamp": "2014-04-08T23:18:00Z",
    "Maximum": 46.88,
    "Unit": "Percent"
  },
},
```

```

    {
      "Timestamp": "2014-04-09T10:18:00Z",
      "Maximum": 51.91,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2014-04-09T04:18:00Z",
      "Maximum": 47.13,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2014-04-09T15:18:00Z",
      "Maximum": 48.96,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2014-04-09T00:18:00Z",
      "Maximum": 48.16,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2014-04-09T01:18:00Z",
      "Maximum": 49.18,
      "Unit": "Percent"
    }
  ],
  "Label": "CPUUtilization"
}

```

Specifying multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space. If a single metric includes multiple dimensions, you must specify a value for every defined dimension.

For more examples using the `get-metric-statistics` command, see [Get Statistics for a Metric](#) in the *Amazon CloudWatch Developer Guide*.

```

aws cloudwatch get-metric-statistics --metric-name Buffers --
namespace MyNameSpace --dimensions Name=InstanceID,Value=i-abcdef

```

```
Name=InstanceType,Value=m1.small --start-time 2016-10-15T04:00:00Z --end-time
2016-10-19T07:00:00Z --statistics Average --period 60
```

- For API details, see [GetMetricStatistics](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
namespace, String metVal,
        String metricOption, String date, Dimension myDimension) {
    try {
        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();

        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
        .endTime(endDate)
        .startTime(start)
        .dimensions(myDimension)
        .metricName(metVal)
        .namespace(namespace)
        .period(86400)
        .statistics(Statistic.fromValue(metricOption))
        .build();

        GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
        List<Datapoint> data = response.datapoints();
        if (!data.isEmpty()) {
            for (Datapoint datapoint : data) {
                System.out
                    .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
            }
        }
    }
}
```



```
        } else {
            System.out.println("The returned data list is empty");
        }

    } catch (CloudWatchException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [GetMetricStatistics](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getAndDisplayMetricStatistics(
    namespaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
        }
}
```

```

        namespace = nameSpaceVal
        period = 86400
        statistics = listOf(Statistic.fromValue(metricOption))
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}

```

- For API details, see [GetMetricStatistics](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

```

```

def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
    """
    Gets statistics for a metric within a specified time span. Metrics are
    grouped
    into the specified period.

    :param namespace: The namespace of the metric.
    :param name: The name of the metric.
    :param start: The UTC start time of the time span to retrieve.
    :param end: The UTC end time of the time span to retrieve.
    :param period: The period, in seconds, in which to group metrics. The
    period
                    must match the granularity of the metric, which depends on
                    the metric's age. For example, metrics that are older than
                    three hours have a one-minute granularity, so the period
    must
                    be at least 60 and must be a multiple of 60.
    :param stat_types: The type of statistics to retrieve, such as average
    value
                    or maximum value.
    :return: The retrieved statistics for the metric.
    """
    try:
        metric = self.cloudwatch_resource.Metric(namespace, name)
        stats = metric.get_statistics(
            StartTime=start, EndTime=end, Period=period,
            Statistics=stat_types
        )
        logger.info(
            "Got %s statistics for %s.", len(stats["Datapoints"]),
            stats["Label"]
        )
    except ClientError:
        logger.exception("Couldn't get statistics for %s.%s.", namespace,
            name)
        raise
    else:
        return stats

```

- For API details, see [GetMetricStatistics](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetMetricWidgetImage with an AWS SDK or CLI

The following code examples show how to use GetMetricWidgetImage.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get an image for a metric graphed over time.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metric">The name of the metric.</param>
/// <param name="stat">The name of the stat to chart.</param>
/// <param name="period">The period to use for the chart.</param>
/// <returns>A memory stream for the chart image.</returns>
public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
{
    var metricImageWidget = new
    {
        title = "Example Metric Graph",
        view = "timeSeries",
```

```
        stacked = false,
        period = period,
        width = 1400,
        height = 600,
        metrics = new List<List<object>>
            { new() { metricNamespace, metric, new { stat } } }
    };

    var metricImageWidgetString =
    JsonSerializer.Serialize(metricImageWidget);
    var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
        new GetMetricWidgetImageRequest()
        {
            MetricWidget = metricImageWidgetString
        });

    return imageResponse.MetricWidgetImage;
}

/// <summary>
/// Save a metric image to a file.
/// </summary>
/// <param name="memoryStream">The MemoryStream for the metric image.</param>
/// <param name="metricName">The name of the metric.</param>
/// <returns>The path to the file.</returns>
public string SaveMetricImage(MemoryStream memoryStream, string metricName)
{
    var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
    using var sr = new StreamReader(memoryStream);
    // Writes the memory stream to a file.
    File.WriteAllBytes(metricFileName, memoryStream.ToArray());
    var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
        metricFileName);
    return filePath;
}
```

- For API details, see [GetMetricWidgetImage](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
    System.out.println("Getting Image data for custom metric.");
    try {
        String myJSON = "{\n" +
            "  \"title\": \"Example Metric Graph\",\n" +
            "  \"view\": \"timeSeries\",\n" +
            "  \"stacked\": false,\n" +
            "  \"period\": 10,\n" +
            "  \"width\": 1400,\n" +
            "  \"height\": 600,\n" +
            "  \"metrics\": [\n" +
            "    [\n" +
            "      \"AWS/Billing\",\n" +
            "      \"EstimatedCharges\",\n" +
            "      \"Currency\",\n" +
            "      \"USD\"\n" +
            "    ]\n" +
            "  ]\n" +
            "}";

        GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
            .metricWidget(myJSON)
            .build();

        GetMetricWidgetImageResponse response =
cw.getMetricWidgetImage(imageRequest);
        SdkBytes sdkBytes = response.metricWidgetImage();
        byte[] bytes = sdkBytes.asByteArray();
        File outputFile = new File(fileName);
```

```
        try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
            outputStream.write(bytes);
        }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [GetMetricWidgetImage](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
                "AWS/Billing",
                "EstimatedCharges",
                "Currency",
                "USD"
            ]
        ]
    }"""
}
```

```
val imageRequest =
    GetMetricWidgetImageRequest {
        metricWidget = myJSON
    }

CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    val response = cwClient.getMetricWidgetImage(imageRequest)
    val bytes = response.metricWidgetImage
    if (bytes != null) {
        File(fileName).writeBytes(bytes)
    }
}
println("You have successfully written data to $fileName")
}
```

- For API details, see [GetMetricWidgetImage](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ListDashboards with an AWS SDK or CLI

The following code examples show how to use ListDashboards.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get a list of dashboards.
/// </summary>
```



```
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}
```

- For API details, see [ListDashboards](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void listDashboards(CloudWatchClient cw) {
    try {
        ListDashboardsIterable listRes = cw.listDashboardsPaginator();
        listRes.stream()
            .flatMap(r -> r.dashboardEntries().stream())
            .forEach(entry -> {
                System.out.println("Dashboard name is: " +
                    entry.dashboardName());
                System.out.println("Dashboard ARN is: " +
                    entry.dashboardArn());
            });
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

```
        System.exit(1);
    }
}
```

- For API details, see [ListDashboards](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}
```

- For API details, see [ListDashboards](#) in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Returns the collection of dashboards for your account.

```
Get-CWDashboardList
```

Output:

```
DashboardArn DashboardName LastModified      Size
-----
arn:...      Dashboard1      7/6/2017 8:14:15 PM 252
```

Example 2: Returns the collection of dashboards for your account whose names start with the prefix 'dev'.

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- For API details, see [ListDashboards](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ListMetrics with an AWS SDK or CLI


The following code examples show how to use ListMetrics.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
```

```
/// List metrics available, optionally within a namespace.
/// </summary>
/// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
/// <param name="filter">Optional dimension filter.</param>
/// <param name="metricName">Optional metric name filter.</param>
/// <returns>The list of metrics.</returns>
public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
{
    var results = new List<Metric>();
    var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
        new ListMetricsRequest
        {
            Namespace = metricNamespace,
            Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
            MetricName = metricName
        });
    // Get the entire list using the paginator.
    await foreach (var metric in paginateMetrics.Metrics)
    {
        results.Add(metric);
    }

    return results;
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

List the metrics.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::ListMetricsRequest request;

if (argc > 1)
{
    request.SetMetricName(argv[1]);
}

if (argc > 2)
{
    request.SetNamespace(argv[2]);
}

bool done = false;
bool header = false;
while (!done)
{
    auto outcome = cw.ListMetrics(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to list CloudWatch metrics:" <<
            outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header)
    {
        std::cout << std::left << std::setw(48) << "MetricName" <<
            std::setw(32) << "Namespace" << "DimensionNameValuePairs" <<
            std::endl;
        header = true;
    }
}
```

```

const auto &metrics = outcome.GetResult().GetMetrics();
for (const auto &metric : metrics)
{
    std::cout << std::left << std::setw(48) <<
        metric.GetMetricName() << std::setw(32) <<
        metric.GetNamespace();
    const auto &dimensions = metric.GetDimensions();
    for (auto iter = dimensions.cbegin();
        iter != dimensions.cend(); ++iter)
    {
        const auto &dimkv = *iter;
        std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
        if (iter + 1 != dimensions.cend())
        {
            std::cout << ", ";
        }
    }
    std::cout << std::endl;
}

const auto &next_token = outcome.GetResult().GetNextToken();
request.SetNextToken(next_token);
done = next_token.empty();
}

```

- For API details, see [ListMetrics](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list the metrics for Amazon SNS

The following `list-metrics` example displays the metrics for Amazon SNS.

```
aws cloudwatch list-metrics \
    --namespace "AWS/SNS"
```

Output:

```
{
  "Metrics": [
```

```
{
  "Namespace": "AWS/SNS",
  "Dimensions": [
    {
      "Name": "TopicName",
      "Value": "NotifyMe"
    }
  ],
  "MetricName": "PublishSize"
},
{
  "Namespace": "AWS/SNS",
  "Dimensions": [
    {
      "Name": "TopicName",
      "Value": "CF0"
    }
  ],
  "MetricName": "PublishSize"
},
{
  "Namespace": "AWS/SNS",
  "Dimensions": [
    {
      "Name": "TopicName",
      "Value": "NotifyMe"
    }
  ],
  "MetricName": "NumberOfNotificationsFailed"
},
{
  "Namespace": "AWS/SNS",
  "Dimensions": [
    {
      "Name": "TopicName",
      "Value": "NotifyMe"
    }
  ],
  "MetricName": "NumberOfNotificationsDelivered"
},
{
  "Namespace": "AWS/SNS",
  "Dimensions": [
    {
```

```
        "Name": "TopicName",
        "Value": "NotifyMe"
    }
],
"MetricName": "NumberOfMessagesPublished"
},
{
    "Namespace": "AWS/SNS",
    "Dimensions": [
        {
            "Name": "TopicName",
            "Value": "CF0"
        }
    ],
    "MetricName": "NumberOfMessagesPublished"
},
{
    "Namespace": "AWS/SNS",
    "Dimensions": [
        {
            "Name": "TopicName",
            "Value": "CF0"
        }
    ],
    "MetricName": "NumberOfNotificationsDelivered"
},
{
    "Namespace": "AWS/SNS",
    "Dimensions": [
        {
            "Name": "TopicName",
            "Value": "CF0"
        }
    ],
    "MetricName": "NumberOfNotificationsFailed"
}
]
}
```

- For API details, see [ListMetrics](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class ListMetrics {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <namespace>\s

                Where:
                namespace - The namespace to filter against (for example, AWS/
                EC2).\s

                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String namespace = args[0];
Region region = Region.US_EAST_1;
CloudWatchClient cw = CloudWatchClient.builder()
    .region(region)
    .build();

listMets(cw, namespace);
cw.close();
}

public static void listMets(CloudWatchClient cw, String namespace) {
    boolean done = false;
    String nextToken = null;

    try {
        while (!done) {

            ListMetricsResponse response;
            if (nextToken == null) {
                ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

                response = cw.listMetrics(request);
            } else {
                ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .nextToken(nextToken)
                    .build();

                response = cw.listMetrics(request);
            }

            for (Metric metric : response.metrics()) {
                System.out.printf("Retrieved metric %s",
metric.metricName());
                System.out.println();
            }

            if (response.nextToken() == null) {
                done = true;
            } else {
                nextToken = response.nextToken();
            }
        }
    }
}
```

```
        }
    }

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { ListMetricsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = () => {
    // Use the AWS console to see available namespaces and metric names. Custom
    // metrics can also be created.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
    // viewing_metrics_with_cloudwatch.html
    const command = new ListMetricsCommand({
        Dimensions: [
            {
                Name: "LogGroupName",
            },
        ],
        MetricName: "IncomingLogEvents",
        Namespace: "AWS/Logs",
    });
```

```
return client.send(command);
};
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListMetrics](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};
```

```
cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListMetrics](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}
```

- For API details, see [ListMetrics](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
            three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {"Namespace": namespace, "MetricName": name}
            if recent:
                kwargs["RecentlyActive"] = "PT3H" # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
```

```
        raise
    else:
        return metric_iter
```

- For API details, see [ListMetrics](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Lists available metrics for a metric namespace in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric.
# @example
#   list_metrics_for_namespace(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC'
#   )
def list_metrics_for_namespace(cloudwatch_client, metric_namespace)
  response = cloudwatch_client.list_metrics(namespace: metric_namespace)

  if response.metrics.count.positive?
    response.metrics.each do |metric|
      puts " Metric name: #{metric.metric_name}"
      if metric.dimensions.count.positive?
        puts "   Dimensions:"
        metric.dimensions.each do |dimension|
          puts "     Name: #{dimension.name}, Value: #{dimension.value}"
        end
      else
        puts "No dimensions found."
      end
    end
  end
end
```

```
    end
  else
    puts "No metrics found for namespace '#{metric_namespace}'. " \
        "Note that it could take up to 15 minutes for recently-added metrics " \
        "to become available."
  end
end
end

# Example usage:
def run_me
  metric_namespace = "SITE/TRAFFIC"
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = "us-east-1"

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Add three datapoints.
  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisitors",
    "SiteName",
    "example.com",
    5_885.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "UniqueVisits",
    "SiteName",
    "example.com",
    8_628.0,
    "Count"
  )

  puts "Continuing..." unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    "PageViews",
    "PageURL",
    "example.html",
    18_057.0,
```



```

    "Count"
  )

  puts "Metrics for namespace '#{metric_namespace}':"
  list_metrics_for_namespace(cloudwatch_client, metric_namespace)
end

run_me if $PROGRAM_NAME == __FILE__

```

- For API details, see [ListMetrics](#) in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

"The following list-metrics example displays the metrics for Amazon
CloudWatch."
TRY.
    oo_result = lo_cwt->listmetrics(          " oo_result is returned for
testing purposes. "
    iv_namespace = iv_namespace
    ).
    DATA(lt_metrics) = oo_result->get_metrics( ).
    MESSAGE 'Metrics retrieved.' TYPE 'I'.
CATCH /aws1/cx_cwtinvparamvalueex .
    MESSAGE 'The specified argument was not valid.' TYPE 'E'.
ENDTRY.

```

- For API details, see [ListMetrics](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutAnomalyDetector with an AWS SDK or CLI

The following code examples show how to use PutAnomalyDetector.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Add an anomaly detector for a single metric.
/// </summary>
/// <param name="anomalyDetector">A single metric anomaly detector.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var putAlarmDetectorResult = await
    _amazonCloudWatch.PutAnomalyDetectorAsync(
        new PutAnomalyDetectorRequest()
        {
            SingleMetricAnomalyDetector = anomalyDetector
        });

    return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [PutAnomalyDetector](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .stat("Maximum")
            .build();

        PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
            .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
            .build();

        cw.putAnomalyDetector(anomalyDetectorRequest);
        System.out.println("Added anomaly detector for metric " +
customMetricName + ".");
    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
    }
}
```

```
        System.exit(1);
    }
}
```

- For API details, see [PutAnomalyDetector](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
        rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}
```

- For API details, see [PutAnomalyDetector](#) in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutDashboard with an AWS SDK or CLI

The following code examples show how to use PutDashboard.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Get started with metrics, dashboards, and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string
dashboardName)
{
    // Get the dashboard model from configuration.
```

```
var newDashboard = new DashboardModel();
_configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

// Add a new metric to the dashboard.
newDashboard.Widgets.Add(new Widget
{
    Height = 8,
    Width = 8,
    Y = 8,
    X = 0,
    Type = "metric",
    Properties = new Properties
    {
        Metrics = new List<List<object>>
            { new() { customMetricNamespace, customMetricName } },
        View = "timeSeries",
        Region = "us-east-1",
        Stat = "Sum",
        Period = 86400,
        YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
        Title = "Custom Metric Widget",
        LiveData = true,
        Sparkline = true,
        Trend = true,
        Stacked = false,
        SetPeriodToTimeRange = false
    }
});

var newDashboardString = JsonSerializer.Serialize(newDashboard,
    new JsonSerializerOptions
    { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
var validationMessages =
    await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

return validationMessages;
}

/// <summary>
/// Wrapper to create or add to a dashboard with metrics.
/// </summary>
/// <param name="dashboardName">The name for the dashboard.</param>
```

```
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
        // was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages;
    }
}
```

- For API details, see [PutDashboard](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void createDashboardWithMetrics(CloudWatchClient cw, String dashboardName, String fileName) {
    try {
        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(readFileAsString(fileName))
            .build();
    }
}
```

```
        PutDashboardResponse response = cw.putDashboard(dashboardRequest);
        System.out.println(dashboardName + " was successfully created.");
        List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
        if (messages.isEmpty()) {
            System.out.println("There are no messages in the new Dashboard");
        } else {
            for (DashboardValidationMessage message : messages) {
                System.out.println("Message is: " + message.message());
            }
        }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [PutDashboard](#) in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
```



```
val response = cwClient.putDashboard(dashboardRequest)
println("$dashboardNameVal was successfully created.")
val messages = response.dashboardValidationMessages
if (messages != null) {
    if (messages.isEmpty()) {
        println("There are no messages in the new Dashboard")
    } else {
        for (message in messages) {
            println("Message is: ${message.message}")
        }
    }
}
}
```

- For API details, see [PutDashboard](#) in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Creates or updates the dashboard named 'Dashboard1' to include two metric widgets side by side.

```
$dashBody = @"
{
  "widgets":[
    {
      "type":"metric",
      "x":0,
      "y":0,
      "width":12,
      "height":6,
      "properties":{"
        "metrics":[
          [
            "AWS/EC2",
            "CPUUtilization",
            "InstanceId",
            "i-012345"
          ]
        ]
      },
    ],
  ],
}
```

```

        "period":300,
        "stat":"Average",
        "region":"us-east-1",
        "title":"EC2 Instance CPU"
    }
},
{
    "type":"metric",
    "x":12,
    "y":0,
    "width":12,
    "height":6,
    "properties":{
        "metrics":[
            [
                "AWS/S3",
                "BucketSizeBytes",
                "BucketName",
                "MyBucketName"
            ]
        ],
        "period":86400,
        "stat":"Maximum",
        "region":"us-east-1",
        "title":"MyBucketName bytes"
    }
}
]
}
"@

```

```
Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

Example 2: Creates or updates the dashboard, piping the content describing the dashboard into the cmdlet.

```

$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1

```

- For API details, see [PutDashboard](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutMetricAlarm with an AWS SDK or CLI

The following code examples show how to use PutMetricAlarm.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with alarms](#)
- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Add a metric alarm to send an email when the metric passes a threshold.
/// </summary>
/// <param name="alarmDescription">A description of the alarm.</param>
/// <param name="alarmName">The name for the alarm.</param>
/// <param name="comparison">The type of comparison to use.</param>
/// <param name="metricName">The name of the metric for the alarm.</param>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="threshold">The threshold value for the alarm.</param>
/// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
/// <returns>True if successful.</returns>
```

```
public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
    string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
{
    try
    {
        var putEmailAlarmResponse = await
        _amazonCloudWatch.PutMetricAlarmAsync(
            new PutMetricAlarmRequest()
            {
                AlarmActions = alarmActions,
                AlarmDescription = alarmDescription,
                AlarmName = alarmName,
                ComparisonOperator = comparison,
                Threshold = threshold,
                Namespace = metricNamespace,
                MetricName = metricName,
                EvaluationPeriods = 1,
                Period = 10,
                Statistic = new Statistic("Maximum"),
                DatapointsToAlarm = 1,
                TreatMissingData = "ignore"
            });
        return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (LimitExceededException lex)
    {
        _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
    }

    return false;
}

/// <summary>
/// Add specific email actions to a list of action strings for a CloudWatch
alarm.
/// </summary>
/// <param name="accountId">The AccountId for the alarm.</param>
/// <param name="region">The region for the alarm.</param>
/// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
```

```
/// <param name="alarmActions">Optional list of existing alarm actions to
append to.</param>
/// <returns>A list of string actions for an alarm.</returns>
public List<string> AddEmailAlarmAction(string accountId, string region,
    string emailTopicName, List<string>? alarmActions = null)
{
    alarmActions ??= new List<string>();
    var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
    alarmActions.Add(snsAlarmAction);
    return alarmActions;
}
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Create the alarm to watch the metric.

```
Aws::CloudWatch::CloudWatchClient cw;
Aws::CloudWatch::Model::PutMetricAlarmRequest request;
request.SetAlarmName(alarm_name);
request.SetComparisonOperator(
    Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
```

```

request.SetEvaluationPeriods(1);
request.SetMetricName("CPUUtilization");
request.SetNamespace("AWS/EC2");
request.SetPeriod(60);
request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
request.SetThreshold(70.0);
request.SetActionsEnabled(false);
request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("InstanceId");
dimension.SetValue(instanceId);

request.AddDimensions(dimension);

auto outcome = cw.PutMetricAlarm(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to create CloudWatch alarm:" <<
        outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully created CloudWatch alarm " << alarm_name
        << std::endl;
}

```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To send an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent

The following example uses the `put-metric-alarm` command to send an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent:

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
when CPU exceeds 70 percent" --metric-name CPUUtilization --namespace AWS/
```

```
EC2 --statistic Average --period 300 --threshold 70 --comparison-operator
  GreaterThanThreshold --dimensions "Name=InstanceId,Value=i-12345678" --
evaluation-periods 2 --alarm-actions arn:aws:sns:us-east-1:111122223333:MyTopic
--unit Percent
```

This command returns to the prompt if successful. If an alarm with the same name already exists, it will be overwritten by the new alarm.

To specify multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space:

```
aws cloudwatch put-metric-alarm --alarm-name "Default_Test_Alarm3" --alarm-
description "The default example alarm" --namespace "CW EXAMPLE METRICS" --
metric-name Default_Test --statistic Average --period 60 --evaluation-periods 3
--threshold 50 --comparison-operator GreaterThanOrEqualToThreshold --dimensions
Name=key1,Value=value1 Name=key2,Value=value2
```

- For API details, see [PutMetricAlarm](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String createAlarm(CloudWatchClient cw, String fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
```

```
String customMetricName =
rootNode.findValue("customMetricName").asText();
String alarmName = rootNode.findValue("exampleAlarmName").asText();
String emailTopic = rootNode.findValue("emailTopic").asText();
String accountId = rootNode.findValue("accountId").asText();
String region = rootNode.findValue("region").asText();

// Create a List for alarm actions.
List<String> alarmActions = new ArrayList<>();
alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
    .alarmActions(alarmActions)
    .alarmDescription("Example metric alarm")
    .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
    .threshold(100.00)
    .metricName(customMetricName)
    .namespace(customMetricNamespace)
    .evaluationPeriods(1)
    .period(10)
    .statistic("Maximum")
    .datapointsToAlarm(1)
    .treatMissingData("ignore")
    .build();

cw.putMetricAlarm(alarmRequest);
System.out.println(alarmName + " was successfully created!");
return alarmName;

} catch (CloudWatchException | IOException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
return "";
}
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
        the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
}
```

```
    }  
  };  
  
  export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
  
export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js  
var AWS = require("aws-sdk");  
// Set the region  
AWS.config.update({ region: "REGION" });  
  
// Create CloudWatch service object  
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });  
  
var params = {  
  AlarmName: "Web_Server_CPU_Utilization",  
  ComparisonOperator: "GreaterThanThreshold",  
  EvaluationPeriods: 1,  
  MetricName: "CPUUtilization",  
  Namespace: "AWS/EC2",  
  Period: 60,  
  Statistic: "Average",  
  Threshold: 70.0,  
  ActionsEnabled: false,
```

```
AlarmDescription: "Alarm when server CPU exceeds 70%",
Dimensions: [
  {
    Name: "InstanceId",
    Value: "INSTANCE_ID",
  },
],
Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricAlarm](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun putMetricAlarm(
    alarmNameVal: String,
    instanceIdVal: String,
) {
    val dimension0b =
        Dimension {
            name = "InstanceId"
            value = instanceIdVal
        }
}
```

```
val request =
    PutMetricAlarmRequest {
        alarmName = alarmNameVal
        comparisonOperator = ComparisonOperator.GreaterThanThreshold
        evaluationPeriods = 1
        metricName = "CPUUtilization"
        namespace = "AWS/EC2"
        period = 60
        statistic = Statistic.fromValue("Average")
        threshold = 70.0
        actionsEnabled = false
        alarmDescription = "An Alarm created by the Kotlin SDK when server
CPU utilization exceeds 70%"
        unit = StandardUnit.fromValue("Seconds")
        dimensions = listOf(dimension0b)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created an alarm with name $alarmNameVal")
    }
}
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
```

```
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def create_metric_alarm(
        self,
        metric_namespace,
        metric_name,
        alarm_name,
        stat_type,
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
            statistics.
        :param eval_periods: The number of periods that the metric must be over
the
            alarm threshold before the alarm is set into an
alarmed
            state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
            against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
```

```
        EvaluationPeriods=eval_periods,
        Threshold=threshold,
        ComparisonOperator=comparison_op,
    )
    logger.info(
        "Added alarm %s to track metric %s.%s.",
        alarm_name,
        metric_namespace,
        metric_name,
    )
except ClientError:
    logger.exception(
        "Couldn't add alarm %s to metric %s.%s",
        alarm_name,
        metric_namespace,
        metric_name,
    )
    raise
else:
    return alarm
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
# Creates or updates an alarm in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm.
# @param alarm_description [String] A description about the alarm.
# @param metric_name [String] The name of the metric associated with the alarm.
```

```

# @param alarm_actions [Array] A list of Strings representing the
#   Amazon Resource Names (ARNs) to execute when the alarm transitions to the
#   ALARM state.
# @param namespace [String] The namespace for the metric to alarm on.
# @param statistic [String] The statistic for the metric.
# @param dimensions [Array] A list of dimensions for the metric, specified as
#   Aws::CloudWatch::Types::Dimension.
# @param period [Integer] The number of seconds before re-evaluating the metric.
# @param unit [String] The unit of measure for the statistic.
# @param evaluation_periods [Integer] The number of periods over which data is
#   compared to the specified threshold.
# @param threshold [Float] The value against which the specified statistic is
#   compared.
# @param comparison_operator [String] The arithmetic operation to use when
#   comparing the specified statistic and threshold.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
# @example
#   exit 1 unless alarm_created_or_updated?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket',
#     'Objects exist in this bucket for more than 1 day.',
#     'NumberOfObjects',
#     ['arn:aws:sns:us-east-1:111111111111:Default_CloudWatch_Alarms_Topic'],
#     'AWS/S3',
#     'Average',
#     [
#       {
#         name: 'BucketName',
#         value: 'doc-example-bucket'
#       },
#       {
#         name: 'StorageType',
#         value: 'AllStorageTypes'
#       }
#     ],
#     86_400,
#     'Count',
#     1,
#     1,
#     'GreaterThanThreshold'
#   )
def alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,

```

```
alarm_description,  
metric_name,  
alarm_actions,  
namespace,  
statistic,  
dimensions,  
period,  
unit,  
evaluation_periods,  
threshold,  
comparison_operator  
)  
cloudwatch_client.put_metric_alarm(  
  alarm_name: alarm_name,  
  alarm_description: alarm_description,  
  metric_name: metric_name,  
  alarm_actions: alarm_actions,  
  namespace: namespace,  
  statistic: statistic,  
  dimensions: dimensions,  
  period: period,  
  unit: unit,  
  evaluation_periods: evaluation_periods,  
  threshold: threshold,  
  comparison_operator: comparison_operator  
)  
return true  
rescue StandardError => e  
  puts "Error creating alarm: #{e.message}"  
  return false  
end
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
  lo_cwt->putmetricalarm(  
    iv_alarmname           = iv_alarm_name  
    iv_comparisonoperator  = iv_comparison_operator  
    iv_evaluationperiods   = iv_evaluation_periods  
    iv_metricname         = iv_metric_name  
    iv_namespace          = iv_namespace  
    iv_statistic           = iv_statistic  
    iv_threshold           = iv_threshold  
    iv_actionsenabled      = iv_actions_enabled  
    iv_alarmdescription    = iv_alarm_description  
    iv_unit                = iv_unit  
    iv_period              = iv_period  
    it_dimensions         = it_dimensions  
  ).  
  MESSAGE 'Alarm created.' TYPE 'I'.  
CATCH /aws1/cx_cwtlimitexceededfault.  
  MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.  
ENDTRY.
```

- For API details, see [PutMetricAlarm](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutMetricData with an AWS SDK or CLI

The following code examples show how to use PutMetricData.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Get started with metrics, dashboards, and alarms](#)
- [Manage metrics and alarms](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Add some metric data using a call to a wrapper class.
/// </summary>
/// <param name="customMetricName">The metric name.</param>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <returns></returns>
private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
    string customMetricNamespace)
{
    List<MetricDatum> customData = new List<MetricDatum>();
    Random rnd = new Random();

    // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
    var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
    for (int i = 0; i < 10; i++)
    {
        var metricValue = rnd.Next(0, 100);
        customData.Add(
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = metricValue,
```

```
        TimestampUtc = utcNowMinus15.AddMinutes(i)
    }
    );
}

    await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
    return customData;
}

/// <summary>
/// Wrapper to add metric data to a CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricData">A data object for the metric data.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricData(string metricNamespace,
    List<MetricDatum> metricData)
{
    var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
        new PutMetricDataRequest()
        {
            MetricData = metricData,
            Namespace = metricNamespace,
        });

    return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see [PutMetricData](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

Put data into the metric.

```
Aws::CloudWatch::CloudWatchClient cw;

Aws::CloudWatch::Model::Dimension dimension;
dimension.SetName("UNIQUE_PAGES");
dimension.SetValue("URLS");

Aws::CloudWatch::Model::MetricDatum datum;
datum.SetMetricName("PAGES_VISITED");
datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
datum.SetValue(data_point);
datum.AddDimensions(dimension);

Aws::CloudWatch::Model::PutMetricDataRequest request;
request.SetNamespace("SITE/TRAFFIC");
request.AddMetricData(datum);

auto outcome = cw.PutMetricData(request);
if (!outcome.IsSuccess())
{
    std::cout << "Failed to put sample metric data:" <<
        outcome.GetError().GetMessage() << std::endl;
}
else
{
    std::cout << "Successfully put sample metric data" << std::endl;
}
```

- For API details, see [PutMetricData](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI**To publish a custom metric to Amazon CloudWatch**

The following example uses the `put-metric-data` command to publish a custom metric to Amazon CloudWatch:

```
aws cloudwatch put-metric-data --namespace "Usage Metrics" --metric-data file://metric.json
```

The values for the metric itself are stored in the JSON file, `metric.json`.

Here are the contents of that file:

```
[
  {
    "MetricName": "New Posts",
    "Timestamp": "Wednesday, June 12, 2013 8:28:20 PM",
    "Value": 0.50,
    "Unit": "Count"
  }
]
```

For more information, see *Publishing Custom Metrics* in the *Amazon CloudWatch Developer Guide*.

To specify multiple dimensions

The following example illustrates how to specify multiple dimensions. Each dimension is specified as a `Name=Value` pair. Multiple dimensions are separated by a comma.:

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit Bytes --value 231434333 --dimensions InstanceID=1-23456789,InstanceType=m1.small
```

- For API details, see [PutMetricData](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        // Set an Instant object.
        String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
        Instant instant = Instant.parse(time);

        MetricDatum datum = MetricDatum.builder()
            .metricName(customMetricName)
            .unit(StandardUnit.NONE)
            .value(1001.00)
            .timestamp(instant)
            .build();

        MetricDatum datum2 = MetricDatum.builder()
            .metricName(customMetricName)
            .unit(StandardUnit.NONE)
            .value(1002.00)
            .timestamp(instant)
            .build();
```

```
List<MetricDatum> metricDataList = new ArrayList<>();
metricDataList.add(datum);
metricDataList.add(datum2);

PutMetricDataRequest request = PutMetricDataRequest.builder()
    .namespace(customMetricNamespace)
    .metricData(metricDataList)
    .build();

cw.putMetricData(request);
System.out.println("Added metric values for for metric " +
customMetricName);

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

- For API details, see [PutMetricData](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Import the SDK and client modules and call the API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
    // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_PutMetricData.html#API_PutMetricData_RequestParameters
    // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html
```

```
// for more information about the parameters in this command.
const command = new PutMetricDataCommand({
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
});

try {
  return await client.send(command);
} catch (err) {
  console.error(err);
}
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricData](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [PutMetricData](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
        rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
        ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
```

```

        aws.smithy.kotlin.runtime.time
            .Instant(instant)
    }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}

```

- For API details, see [PutMetricData](#) in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: Creates a new MetricDatum object, and writes it to Amazon Web Services CloudWatch Metrics.

```

### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50


### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric

```

- For API details, see [PutMetricData](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
        given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
                             name)
            raise
```

Put a set of data into a CloudWatch metric.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
        set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
        counts.
        contains a list of values and a list of corresponding
        counts.
        The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            ),
```

```
    )
    logger.info("Put data set for metric %s.%s.", namespace, name)
except ClientError:
    logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
    raise
```

- For API details, see [PutMetricData](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require "aws-sdk-cloudwatch"

# Adds a datapoint to a metric in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric to add the
#   datapoint to.
# @param metric_name [String] The name of the metric to add the datapoint to.
# @param dimension_name [String] The name of the dimension to add the
#   datapoint to.
# @param dimension_value [String] The value of the dimension to add the
#   datapoint to.
# @param metric_value [Float] The value of the datapoint.
# @param metric_unit [String] The unit of measurement for the datapoint.
# @return [Boolean]
# @example
#   exit 1 unless datapoint_added_to_metric?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC',
#     'UniqueVisitors',
```

```
#   'SiteName',
#   'example.com',
#   5_885.0,
#   'Count'
# )
def datapoint_added_to_metric?(
  cloudwatch_client,
  metric_namespace,
  metric_name,
  dimension_name,
  dimension_value,
  metric_value,
  metric_unit
)
  cloudwatch_client.put_metric_data(
    namespace: metric_namespace,
    metric_data: [
      {
        metric_name: metric_name,
        dimensions: [
          {
            name: dimension_name,
            value: dimension_value
          }
        ],
        value: metric_value,
        unit: metric_unit
      }
    ]
  )
  puts "Added data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}'."
  return true
rescue StandardError => e
  puts "Error adding data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}': #{e.message}"
  return false
end
```

- For API details, see [PutMetricData](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudWatch using AWS SDKs

The following code examples show you how to implement common scenarios in CloudWatch with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudWatch. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- [Get started with CloudWatch alarms using an AWS SDK](#)
- [Get started with CloudWatch metrics, dashboards, and alarms using an AWS SDK](#)
- [Manage CloudWatch metrics and alarms using an AWS SDK](#)

Get started with CloudWatch alarms using an AWS SDK

The following code example shows how to:

- Create an alarm.
- Disable alarm actions.
- Describe an alarm.
- Delete an alarm.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).


```

DATA lt_alarmnames TYPE /aws1/cl_cwtalarmnames_w=>tt_alarmnames.
DATA lo_alarmname TYPE REF TO /aws1/cl_cwtalarmnames_w.

"Create an alarm"
TRY.
  lo_cwt->putmetricalarm(
    iv_alarmname           = iv_alarm_name
    iv_comparisonoperator  = iv_comparison_operator
    iv_evaluationperiods   = iv_evaluation_periods
    iv_metricname          = iv_metric_name
    iv_namespace           = iv_namespace
    iv_statistic            = iv_statistic
    iv_threshold            = iv_threshold
    iv_actionsenabled      = iv_actions_enabled
    iv_alarmdescription    = iv_alarm_description
    iv_unit                 = iv_unit
    iv_period               = iv_period
    it_dimensions           = it_dimensions
  ).
  MESSAGE 'Alarm created' TYPE 'I'.
CATCH /aws1/cx_cwtlimitexceededfault.
  MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.

"Create an ABAP internal table for the created alarm."
CREATE OBJECT lo_alarmname EXPORTING iv_value = iv_alarm_name.
INSERT lo_alarmname INTO TABLE lt_alarmnames.

"Disable alarm actions."
TRY.
  lo_cwt->disablealarmactions(
    it_alarmnames          = lt_alarmnames
  ).
  MESSAGE 'Alarm actions disabled' TYPE 'I'.
CATCH /aws1/cx_rt_service_generic INTO DATA(lo_disablealarm_exception).
  DATA(lv_disablealarm_error) = |"{ lo_disablealarm_exception-
>av_err_code }" - { lo_disablealarm_exception->av_err_msg }|.
  MESSAGE lv_disablealarm_error TYPE 'E'.
ENDTRY.

"Describe alarm using the same ABAP internal table."
TRY.
  oo_result = lo_cwt->describealarms(
    " oo_result is
returned for testing purpose "

```

```
        it_alarmnames          = lt_alarmnames
    ).
    MESSAGE 'Alarms retrieved' TYPE 'I'.
    CATCH /aws1/cx_rt_service_generic INTO DATA(lo_describealarms_exception).
    DATA(lv_describealarms_error) = |"{ lo_describealarms_exception-
>av_err_code }" - { lo_describealarms_exception->av_err_msg }|.
    MESSAGE lv_describealarms_error TYPE 'E'.
ENDTRY.

"Delete alarm."
TRY.
    lo_cwt->deletealarms(
        it_alarmnames = lt_alarmnames
    ).
    MESSAGE 'Alarms deleted' TYPE 'I'.
    CATCH /aws1/cx_cwtresourcenotfound .
    MESSAGE 'Resource being access is not found.' TYPE 'E'.
ENDTRY.
```

- For API details, see the following topics in *AWS SDK for SAP ABAP API reference*.
 - [DeleteAlarms](#)
 - [DescribeAlarms](#)
 - [DisableAlarmActions](#)
 - [PutMetricAlarm](#)

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Get started with CloudWatch metrics, dashboards, and alarms using an AWS SDK

The following code examples show how to:

- List CloudWatch namespaces and metrics.
- Get statistics for a metric and for estimated billing.
- Create and update a dashboard.
- Create and add data to a metric.

- Create and trigger an alarm, then view alarm history.
- Add an anomaly detector.
- Get a metric image, then clean up resources.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
public class CloudWatchScenario
{
    /*
        Before running this .NET code example, set up your development environment,
        including your credentials.

        To enable billing metrics and statistics for this example, make sure billing
        alerts are enabled for your account:
        https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
        monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

        This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
        10. Describe current CloudWatch alarms.
        11. Get recent data for the custom metric.
        12. Add data to the custom metric to trigger the alarm.
        13. Wait for an alarm state.
```

```
14. Get history for the CloudWatch alarm.
15. Add an anomaly detector.
16. Describe current anomaly detectors.
17. Get and display a metric image.
18. Clean up resources.
*/

private static ILogger logger = null!;
private static CloudWatchWrapper _cloudWatchWrapper = null!;
private static IConfiguration _configuration = null!;
private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
private static SingleMetricAnomalyDetector? anomalyDetector = null!;

static async Task Main(string[] args)
{
    // Set up dependency injection for the Amazon service.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
                .AddTransient<CloudWatchWrapper>()
        )
        .Build();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally, load local settings.
        .Build();

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<CloudWatchScenario>();

    _cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();

    Console.WriteLine(new string('-', 80));
```

```
Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
Console.WriteLine(new string('-', 80));

try
{
    var selectedNamespace = await SelectNamespace();
    var selectedMetric = await SelectMetric(selectedNamespace);
    await GetAndDisplayMetricStatistics(selectedNamespace,
selectedMetric);
    await GetAndDisplayEstimatedBilling();
    await CreateDashboardWithMetrics();
    await ListDashboards();
    await CreateNewCustomMetric();
    await AddMetricToDashboard();
    await CreateMetricAlarm();
    await DescribeAlarms();
    await GetCustomMetricData();
    await AddMetricDataForAlarm();
    await CheckForMetricAlarm();
    await GetAlarmHistory();
    anomalyDetector = await AddAnomalyDetector();
    await DescribeAnomalyDetectors();
    await GetAndOpenMetricImage();
    await CleanupResources();
}
catch (Exception ex)
{
    logger.LogError(ex, "There was a problem executing the scenario.");
    await CleanupResources();
}

}

/// <summary>
/// Select a namespace.
/// </summary>
/// <returns>The selected namespace.</returns>
private static async Task<string> SelectNamespace()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
    var metrics = await _cloudWatchWrapper.ListMetrics();
    // Get a distinct list of namespaces.
```

```
var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
for (int i = 0; i < namespaces.Count; i++)
{
    Console.WriteLine($"{i + 1}. {namespaces[i]}");
}

var namespaceChoiceNumber = 0;
while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
{
    Console.WriteLine(
list:");
        "Select a namespace by entering a number from the preceding
    var choice = Console.ReadLine();
    Int32.TryParse(choice, out namespaceChoiceNumber);
}

var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

Console.WriteLine(new string('-', 80));

return selectedNamespace;
}

/// <summary>
/// Select a metric from a namespace.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <returns>The metric name.</returns>
private static async Task<Metric> SelectMetric(string metricNamespace)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

    var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

    for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
    {
        var dimensionsWithValues = namespaceMetrics[i].Dimensions
            .Where(d => !string.Equals("None", d.Value));
        Console.WriteLine($"{i + 1}. {namespaceMetrics[i].MetricName} " +
            $"{string.Join(", :", dimensionsWithValues.Select(d
=> d.Value))}");
    }
}
```

```
    }

    var metricChoiceNumber = 0;
    while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
    {
        Console.WriteLine(
            "Select a metric by entering a number from the preceding list:");
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out metricChoiceNumber);
    }

    var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

    Console.WriteLine(new string('-', 80));

    return selectedMetric;
}

/// <summary>
/// Get and display metric statistics for a specific metric.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <param name="metric">The CloudWatch metric.</param>
/// <returns>Async task.</returns>
private static async Task GetAndDisplayMetricStatistics(string
metricNamespace, Metric metric)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"3. Get CloudWatch metric statistics for the last
day.");

    for (int i = 0; i < _statTypes.Count; i++)
    {
        Console.WriteLine($"{i + 1}. {_statTypes[i]}");
    }

    var statisticChoiceNumber = 0;
    while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
    {
        Console.WriteLine(
            "Select a metric statistic by entering a number from the
preceding list:");
```

```
        var choice = Console.ReadLine();
        Int32.TryParse(choice, out statisticChoiceNumber);
    }

    var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
    var statisticsList = new List<string> { selectedStatistic };

    var metricStatistics = await
        _cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
        statisticsList, metric.Dimensions, 1, 60);

    if (!metricStatistics.Any())
    {
        Console.WriteLine($"No {selectedStatistic} statistics found for
        {metric} in namespace {metricNamespace}.");
    }

    metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
    for (int i = 0; i < metricStatistics.Count && i < 10; i++)
    {
        var metricStat = metricStatistics[i];
        var statValue =
        metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
        Console.WriteLine($"\\t{i + 1}. Timestamp
        {metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get and display estimated billing statistics.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <param name="metric">The CloudWatch metric.</param>
/// <returns>Async task.</returns>
private static async Task GetAndDisplayEstimatedBilling()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"4. Get CloudWatch estimated billing for the last
    week.");

    var billingStatistics = await SetupBillingStatistics();
```



```
        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the
        past seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
    }
}
```

```
var newDashboardString = JsonSerializer.Serialize(
    newDashboard,
    new JsonSerializerOptions
    {
        DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
    });
var validationMessages =
    await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

Console.WriteLine(validationMessages.Any() ? $"{\tValidation messages:" :
null);
for (int i = 0; i < validationMessages.Count; i++)
{
    Console.WriteLine($"{\t{i + 1}. {validationMessages[i].Message}");
}
Console.WriteLine($"{\tDashboard {dashboardName} was created.");
Console.WriteLine(new string('-', 80));
}

/// <summary>
/// List dashboards.
/// </summary>
/// <returns>Async task.</returns>
private static async Task ListDashboards()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"{6. List the CloudWatch dashboards in the current
account.");

    var dashboards = await _cloudWatchWrapper.ListDashboards();

    for (int i = 0; i < dashboards.Count; i++)
    {
        Console.WriteLine($"{\t{i + 1}. {dashboards[i].DashboardName}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Create and add data for a new custom metric.
/// </summary>
/// <returns>Async task.</returns>
```

```
private static async Task CreateNewCustomMetric()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"7. Create and add data for a new custom metric.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

    var valuesString = string.Join(',', customData.Select(d => d.Value));
    Console.WriteLine($"\\tAdded metric values for for metric
{customMetricName}: \\n\\t{valuesString}");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add some metric data using a call to a wrapper class.
/// </summary>
/// <param name="customMetricName">The metric name.</param>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <returns></returns>
private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
    string customMetricNamespace)
{
    List<MetricDatum> customData = new List<MetricDatum>();
    Random rnd = new Random();

    // Add 10 random values up to 100, starting with a timestamp 15 minutes
in the past.
    var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
    for (int i = 0; i < 10; i++)
    {
        var metricValue = rnd.Next(0, 100);
        customData.Add(
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = metricValue,
                TimestampUtc = utcNowMinus15.AddMinutes(i)
            }
        );
    }
}
```

```
        }
    );
}

    await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);
    return customData;
}

/// <summary>
/// Add the custom metric to the dashboard.
/// </summary>
/// <returns>Async task.</returns>
private static async Task AddMetricToDashboard()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"8. Add the new custom metric to the dashboard.");

    var dashboardName = _configuration["dashboardName"];

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

    Console.WriteLine(validationMessages.Any() ? $"{\tValidation messages:" :
null);
    for (int i = 0; i < validationMessages.Count; i++)
    {
        Console.WriteLine($"{\t{i + 1}. {validationMessages[i].Message}");
    }
    Console.WriteLine($"{\tDashboard {dashboardName} updated with metric
{customMetricName}.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
```

```
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string
dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
                { new() { customMetricNamespace, customMetricName } },
            View = "timeSeries",
            Region = "us-east-1",
            Stat = "Sum",
            Period = 86400,
            YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
            Title = "Custom Metric Widget",
            LiveData = true,
            Sparkline = true,
            Trend = true,
            Stacked = false,
            SetPeriodToTimeRange = false
        }
    });

    var newDashboardString = JsonSerializer.Serialize(newDashboard,
        new JsonSerializerOptions
        { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
    var validationMessages =
        await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

    return validationMessages;
}
```

```
/// <summary>
/// Create a CloudWatch alarm for the new metric.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CreateMetricAlarm()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var alarmName = _configuration["exampleAlarmName"];
    var accountId = _configuration["accountId"];
    var region = _configuration["region"];
    var emailTopic = _configuration["emailTopic"];
    var alarmActions = new List<string>();

    if (GetYesNoResponse(
        $"{alarmName}? (y/n)"))
    {
        _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
        alarmActions);
    }

    await _cloudWatchWrapper.PutMetricEmailAlarm(
        "Example metric alarm",
        alarmName,
        ComparisonOperator.GreaterThanOrEqualToThreshold,
        customMetricName,
        customMetricNamespace,
        100,
        alarmActions);

    Console.WriteLine($"{alarmName} added for metric
    {customMetricName}.");
    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Describe Alarms.
/// </summary>
```

```
/// <returns>Async task.</returns>
private static async Task DescribeAlarms()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

    var alarms = await _cloudWatchWrapper.DescribeAlarms();
    alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

    for (int i = 0; i < alarms.Count && i < 10; i++)
    {
        var alarm = alarms[i];
        Console.WriteLine($"\\t{i + 1}. {alarm.AlarmName}");
        Console.WriteLine($"\\tState: {alarm.StateValue} for
{alarm.MetricName} {alarm.ComparisonOperator} {alarm.Threshold}");
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get the recent data for the metric.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetCustomMetricData()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"11. Get current data for new custom metric.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];
    var accountId = _configuration["accountId"];

    var query = new List<MetricDataQuery>
    {
        new MetricDataQuery
        {
            AccountId = accountId,
            Id = "m1",
            Label = "Custom Metric Data",
            MetricStat = new MetricStat
            {
                Metric = new Metric
```

```
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
        },
        Period = 1,
        Stat = "Maximum"
    }
}
};

var metricData = await _cloudWatchWrapper.GetMetricData(
    20,
    true,
    DateTime.UtcNow.AddMinutes(1),
    20,
    query);

for (int i = 0; i < metricData.Count; i++)
{
    for (int j = 0; j < metricData[i].Values.Count; j++)
    {
        Console.WriteLine(
            $"{\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
    }
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Add metric data to trigger an alarm.
/// </summary>
/// <returns>Async task.</returns>
private static async Task AddMetricDataForAlarm()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"12. Add metric data to the custom metric to trigger
an alarm.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];
    var nowUtc = DateTime.UtcNow;
    List<MetricDatum> customData = new List<MetricDatum>
```



```
    {
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc.AddMinutes(-2)
        },
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc.AddMinutes(-1)
        },
        new MetricDatum
        {
            MetricName = customMetricName,
            Value = 101,
            TimestampUtc = nowUtc
        }
    };
    var valuesString = string.Join(',', customData.Select(d => d.Value));
    Console.WriteLine($"\\tAdded metric values for for metric
{customMetricName}: \\n\\t{valuesString}");
    await _cloudWatchWrapper.PutMetricData(customMetricNamespace,
customData);

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Check for a metric alarm using the DescribeAlarmsForMetric action.
/// </summary>
/// <returns>Async task.</returns>
private static async Task CheckForMetricAlarm()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"13. Checking for an alarm state.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];
    var hasAlarm = false;
    var retries = 10;
    while (!hasAlarm && retries > 0)
    {
```

```
        var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
        hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
        retries--;
        Thread.Sleep(20000);
    }

    Console.WriteLine(hasAlarm
        ? $"{"\tAlarm state found for {customMetricName}."
        : $"{"\tNo Alarm state found for {customMetricName} after 10
retries."});

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get history for an alarm.
/// </summary>
/// <returns>Async task.</returns>
private static async Task GetAlarmHistory()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"14. Get alarm history.");

    var exampleAlarmName = _configuration["exampleAlarmName"];

    var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

    for (int i = 0; i < alarmHistory.Count; i++)
    {
        var history = alarmHistory[i];
        Console.WriteLine($"{"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
    }
    if (!alarmHistory.Any())
    {
        Console.WriteLine($"{"\tNo alarm history data found for
{exampleAlarmName}."});
    }

    Console.WriteLine(new string('-', 80));
}
```

```
/// <summary>
/// Add an anomaly detector.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"15. Add an anomaly detector.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var detector = new SingleMetricAnomalyDetector
    {
        MetricName = customMetricName,
        Namespace = customMetricNamespace,
        Stat = "Maximum"
    };
    await _cloudWatchWrapper.PutAnomalyDetector(detector);
    Console.WriteLine($"\\tAdded anomaly detector for metric
{customMetricName}.");

    Console.WriteLine(new string('-', 80));
    return detector;
}

/// <summary>
/// Describe anomaly detectors.
/// </summary>
/// <returns>Async task.</returns>
private static async Task DescribeAnomalyDetectors()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

    var customMetricNamespace = _configuration["customMetricNamespace"];
    var customMetricName = _configuration["customMetricName"];

    var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);
```

```
        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state
{detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"{i + 1}. Getting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
        _cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
        customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream,
        "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"{i + 1}. File saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"{i + 1}. Press enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }
}
```

```
/// <summary>
/// Clean up created resources.
/// </summary>
/// <param name="metricNamespace">The namespace for metrics.</param>
/// <param name="metric">The CloudWatch metric.</param>
/// <returns>Async task.</returns>
private static async Task CleanupResources()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"18. Clean up resources.");

    var dashboardName = _configuration["dashboardName"];
    if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
    {
        Console.WriteLine($" \tDeleting dashboard.");
        var dashboardList = new List<string> { dashboardName };
        await _cloudWatchWrapper.DeleteDashboards(dashboardList);
    }

    var alarmName = _configuration["exampleAlarmName"];
    if (GetYesNoResponse($" \tDelete alarm {alarmName}? (y/n)"))
    {
        Console.WriteLine($" \tCleaning up alarms.");
        var alarms = new List<string> { alarmName };
        await _cloudWatchWrapper.DeleteAlarms(alarms);
    }

    if (GetYesNoResponse($" \tDelete anomaly detector? (y/n)") &&
        anomalyDetector != null)
    {
        Console.WriteLine($" \tCleaning up anomaly detector.");

        await _cloudWatchWrapper.DeleteAnomalyDetector(
            anomalyDetector);
    }

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Get a yes or no response from the user.
/// </summary>
```

```

    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}

```

Wrapper methods used by the scenario for CloudWatch actions.

```

/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
        ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
    listing metrics.</param>

```

```

    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,

```

```

        Dimensions = dimensions,
        Statistics = statistics,
        StartTimeUtc = DateTime.UtcNow.AddDays(-days),
        EndTimeUtc = DateTime.UtcNow,
        Period = period
    });

    return metricStatistics.Datapoints;
}

/// <summary>
/// Wrapper to create or add to a dashboard with metrics.
/// </summary>
/// <param name="dashboardName">The name for the dashboard.</param>
/// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
/// <returns>A list of validation messages for the dashboard.</returns>
public async Task<List<DashboardValidationMessage>> PutDashboard(string
dashboardName,
    string dashboardBody)
{
    // Updating a dashboard replaces all contents.
    // Best practice is to include a text widget indicating this dashboard
was created programmatically.
    var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
        new PutDashboardRequest()
        {
            DashboardName = dashboardName,
            DashboardBody = dashboardBody
        });

    return dashboardResponse.DashboardValidationMessages;
}

/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()

```



```
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}

/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}

/// <summary>
/// Wrapper to add metric data to a CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricData">A data object for the metric data.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricData(string metricNamespace,
    List<MetricDatum> metricData)
{
    var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
        new PutMetricDataRequest()
        {
            MetricData = metricData,
            Namespace = metricNamespace,
        });

    return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

```

/// <summary>
/// Get an image for a metric graphed over time.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metric">The name of the metric.</param>
/// <param name="stat">The name of the stat to chart.</param>
/// <param name="period">The period to use for the chart.</param>
/// <returns>A memory stream for the chart image.</returns>
public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
{
    var metricImageWidget = new
    {
        title = "Example Metric Graph",
        view = "timeSeries",
        stacked = false,
        period = period,
        width = 1400,
        height = 600,
        metrics = new List<List<object>>
            { new() { metricNamespace, metric, new { stat } } }
    };

    var metricImageWidgetString =
JsonSerializer.Serialize(metricImageWidget);
    var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
        new GetMetricWidgetImageRequest()
        {
            MetricWidget = metricImageWidgetString
        });

    return imageResponse.MetricWidgetImage;
}

/// <summary>
/// Save a metric image to a file.
/// </summary>
/// <param name="memoryStream">The MemoryStream for the metric image.</param>
/// <param name="metricName">The name of the metric.</param>
/// <returns>The path to the file.</returns>
public string SaveMetricImage(MemoryStream memoryStream, string metricName)
{
    var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
}

```

```

        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
        bool useDescendingTime, DateTime? endDateUtc = null,
            int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
            TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
        offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
                ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });
    }

```

```
        await foreach (var data in paginatedMetricData.MetricDataResults)
        {
            metricData.Add(data);
        }
        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                }
            );
        }
    }
};
```

```

        return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
    }
    catch (LimitExceededException lex)
    {
        _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
    }

    return false;
}

/// <summary>
/// Add specific email actions to a list of action strings for a CloudWatch
alarm.
/// </summary>
/// <param name="accountId">The AccountId for the alarm.</param>
/// <param name="region">The region for the alarm.</param>
/// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
/// <param name="alarmActions">Optional list of existing alarm actions to
append to.</param>
/// <returns>A list of string actions for an alarm.</returns>
public List<string> AddEmailAlarmAction(string accountId, string region,
    string emailTopicName, List<string>? alarmActions = null)
{
    alarmActions ??= new List<string>();
    var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
    alarmActions.Add(snsAlarmAction);
    return alarmActions;
}

/// <summary>
/// Describe the current alarms, optionally filtered by state.
/// </summary>
/// <param name="stateValue">Optional filter for alarm state.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
{
    List<MetricAlarm> alarms = new List<MetricAlarm>();
    var paginatedDescribeAlarms =
_amazonCloudWatch.Paginators.DescribeAlarms(
    new DescribeAlarmsRequest()

```

```
        {
            StateValue = stateValue
        });

    await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
    {
        alarms.Add(data);
    }
    return alarms;
}

/// <summary>
/// Describe the current alarms for a specific metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The name of the metric.</param>
/// <returns>The list of alarm data.</returns>
public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
{
    var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
        new DescribeAlarmsForMetricRequest()
        {
            Namespace = metricNamespace,
            MetricName = metricName
        });

    return alarmsResult.MetricAlarms;
}

/// <summary>
/// Describe the history of an alarm for a number of days in the past.
/// </summary>
/// <param name="alarmName">The name of the alarm.</param>
/// <param name="historyDays">The number of days in the past.</param>
/// <returns>The list of alarm history data.</returns>
public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
{
    List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
    var paginatedAlarmHistory =
    _amazonCloudWatch.Paginators.DescribeAlarmHistory(
        new DescribeAlarmHistoryRequest()
        {
```

```
        AlarmName = alarmName,
        EndDateUtc = DateTime.UtcNow,
        HistoryItemType = HistoryItemType.StateUpdate,
        StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
    });

    await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
    {
        alarmHistory.Add(data);
    }
    return alarmHistory;
}

/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Disable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableAlarmActions(List<string> alarmNames)
{
    var disableAlarmActionsResult = await
    _amazonCloudWatch.DisableAlarmActionsAsync(
        new DisableAlarmActionsRequest()
        {
            AlarmNames = alarmNames
        });

    return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

```
}

/// <summary>
/// Enable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> EnableAlarmActions(List<string> alarmNames)
{
    var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
    new EnableAlarmActionsRequest()
    {
        AlarmNames = alarmNames
    });

    return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Add an anomaly detector for a single metric.
/// </summary>
/// <param name="anomalyDetector">A single metric anomaly detector.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
    new PutAnomalyDetectorRequest()
    {
        SingleMetricAnomalyDetector = anomalyDetector
    });

    return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Describe anomaly detectors for a metric and namespace.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricName">The metric of the anomaly detectors.</param>
/// <returns>The list of detectors.</returns>
```



```
public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
{
    List<AnomalyDetector> detectors = new List<AnomalyDetector>();
    var paginatedDescribeAnomalyDetectors =
    _amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
        new DescribeAnomalyDetectorsRequest()
        {
            MetricName = metricName,
            Namespace = metricNamespace
        });

    await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
    {
        detectors.Add(data);
    }

    return detectors;
}

/// <summary>
/// Delete a single metric anomaly detector.
/// </summary>
/// <param name="anomalyDetector">The anomaly detector to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
{
    var deleteAnomalyDetectorResponse = await
    _amazonCloudWatch.DeleteAnomalyDetectorAsync(
        new DeleteAnomalyDetectorRequest()
        {
            SingleMetricAnomalyDetector = anomalyDetector
        });

    return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a list of CloudWatch dashboards.
/// </summary>
/// <param name="dashboardNames">List of dashboard names to delete.</param>
/// <returns>True if successful.</returns>
```

```
public async Task<bool> DeleteDashboards(List<string> dashboardNames)
{
    var deleteDashboardsResponse = await
    _amazonCloudWatch.DeleteDashboardsAsync(
        new DeleteDashboardsRequest()
        {
            DashboardNames = dashboardNames
        });

    return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
 - [DeleteAlarms](#)
 - [DeleteAnomalyDetector](#)
 - [DeleteDashboards](#)
 - [DescribeAlarmHistory](#)
 - [DescribeAlarms](#)
 - [DescribeAlarmsForMetric](#)
 - [DescribeAnomalyDetectors](#)
 - [GetMetricData](#)
 - [GetMetricStatistics](#)
 - [GetMetricWidgetImage](#)
 - [ListMetrics](#)
 - [PutAnomalyDetector](#)
 - [PutDashboard](#)
 - [PutMetricAlarm](#)
 - [PutMetricData](#)

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.AlarmHistoryItem;
import software.amazon.awssdk.services.cloudwatch.model.AlarmType;
import software.amazon.awssdk.services.cloudwatch.model.AnomalyDetector;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ComparisonOperator;
import
    software.amazon.awssdk.services.cloudwatch.model.DashboardValidationMessage;
import software.amazon.awssdk.services.cloudwatch.model.Datapoint;
import software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmsRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.DeleteAnomalyDetectorRequest;
import software.amazon.awssdk.services.cloudwatch.model.DeleteDashboardsRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmHistoryResponse;
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsForMetricResponse;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsRequest;
import software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmsResponse;
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsRequest;
```

```
import
    software.amazon.awssdk.services.cloudwatch.model.DescribeAnomalyDetectorsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.GetMetricDataResponse;
import
    software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.GetMetricStatisticsResponse;
import
    software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageRequest;
import
    software.amazon.awssdk.services.cloudwatch.model.GetMetricWidgetImageResponse;
import software.amazon.awssdk.services.cloudwatch.model.HistoryItemType;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsResponse;
import software.amazon.awssdk.services.cloudwatch.model.Metric;
import software.amazon.awssdk.services.cloudwatch.model.MetricAlarm;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataQuery;
import software.amazon.awssdk.services.cloudwatch.model.MetricDataResult;
import software.amazon.awssdk.services.cloudwatch.model.MetricDatum;
import software.amazon.awssdk.services.cloudwatch.model.MetricStat;
import
    software.amazon.awssdk.services.cloudwatch.model.PutAnomalyDetectorRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutDashboardResponse;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricDataRequest;
import software.amazon.awssdk.services.cloudwatch.model.ScanBy;
import
    software.amazon.awssdk.services.cloudwatch.model.SingleMetricAnomalyDetector;
import software.amazon.awssdk.services.cloudwatch.model.StandardUnit;
import software.amazon.awssdk.services.cloudwatch.model.Statistic;
import
    software.amazon.awssdk.services.cloudwatch.paginators.ListDashboardsIterable;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.Instant;
```

```
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To enable billing metrics and statistics for this example, make sure billing
 * alerts are enabled for your account:
 * https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/monitor\_estimated\_charges\_with\_cloudwatch.html#turning\_on\_billing\_metrics
 *
 * This Java code example performs the following tasks:
 *
 * 1. List available namespaces from Amazon CloudWatch.
 * 2. List available metrics within the selected Namespace.
 * 3. Get statistics for the selected metric over the last day.
 * 4. Get CloudWatch estimated billing for the last week.
 * 5. Create a new CloudWatch dashboard with metrics.
 * 6. List dashboards using a paginator.
 * 7. Create a new custom metric by adding data for it.
 * 8. Add the custom metric to the dashboard.
 * 9. Create an alarm for the custom metric.
 * 10. Describe current alarms.
 * 11. Get current data for the new custom metric.
 * 12. Push data into the custom metric to trigger the alarm.
 * 13. Check the alarm state using the action DescribeAlarmsForMetric.
 * 14. Get alarm history for the new alarm.
 * 15. Add an anomaly detector for the custom metric.
 * 16. Describe current anomaly detectors.
 * 17. Get a metric image for the custom metric.
 * 18. Clean up the Amazon CloudWatch resources.
 */
public class CloudWatchScenario {
```

```
public static final String DASHES = new String(new char[80]).replace("\0",
"-");

public static void main(String[] args) throws IOException {
    final String usage = ""

        Usage:
        <myDate> <costDateWeek> <dashboardName> <dashboardJson>
<dashboardAdd> <settings> <metricImage> \s

        Where:
        myDate - The start date to use to get metric statistics. (For
example, 2023-01-11T18:35:24.00Z.)\s
        costDateWeek - The start date to use to get AWS/Billinget
statistics. (For example, 2023-01-11T18:35:24.00Z.)\s
        dashboardName - The name of the dashboard to create.\s
        dashboardJson - The location of a JSON file to use to create a
dashboard. (See Readme file.)\s
        dashboardAdd - The location of a JSON file to use to update a
dashboard. (See Readme file.)\s
        settings - The location of a JSON file from which various
values are read. (See Readme file.)\s
        metricImage - The location of a BMP file that is used to create
a graph.\s

        """;

    if (args.length != 7) {
        System.out.println(usage);
        System.exit(1);
    }

    Region region = Region.US_EAST_1;
    String myDate = args[0];
    String costDateWeek = args[1];
    String dashboardName = args[2];
    String dashboardJson = args[3];
    String dashboardAdd = args[4];
    String settings = args[5];
    String metricImage = args[6];

    Double dataPoint = Double.parseDouble("10.0");
    Scanner sc = new Scanner(System.in);
    CloudWatchClient cw = CloudWatchClient.builder()
        .region(region)
```

```
        .credentialsProvider(ProfileCredentialsProvider.create())
        .build();

System.out.println(DASHES);
System.out.println("Welcome to the Amazon CloudWatch example scenario.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println(
    "1. List at least five available unique namespaces from Amazon
CloudWatch. Select one from the list.");
ArrayList<String> list = listNameSpaces(cw);
for (int z = 0; z < 5; z++) {
    int index = z + 1;
    System.out.println("    " + index + ". " + list.get(z));
}

String selectedNamespace = "";
String selectedMetrics = "";
int num = Integer.parseInt(sc.nextLine());
if (1 <= num && num <= 5) {
    selectedNamespace = list.get(num - 1);
} else {
    System.out.println("You did not select a valid option.");
    System.exit(1);
}
System.out.println("You selected " + selectedNamespace);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("2. List available metrics within the selected
namespace and select one from the list.");
ArrayList<String> metList = listMets(cw, selectedNamespace);
for (int z = 0; z < 5; z++) {
    int index = z + 1;
    System.out.println("    " + index + ". " + metList.get(z));
}
num = Integer.parseInt(sc.nextLine());
if (1 <= num && num <= 5) {
    selectedMetrics = metList.get(num - 1);
} else {
    System.out.println("You did not select a valid option.");
    System.exit(1);
}
```

```
System.out.println("You selected " + selectedMetrics);
Dimension myDimension = getSpecificMet(cw, selectedNamespace);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. Get statistics for the selected metric over the
last day.");
String metricOption = "";
ArrayList<String> statTypes = new ArrayList<>();
statTypes.add("SampleCount");
statTypes.add("Average");
statTypes.add("Sum");
statTypes.add("Minimum");
statTypes.add("Maximum");

for (int t = 0; t < 5; t++) {
    System.out.println("    " + (t + 1) + ". " + statTypes.get(t));
}
System.out.println("Select a metric statistic by entering a number from
the preceding list:");
num = Integer.parseInt(sc.nextLine());
if (1 <= num && num <= 5) {
    metricOption = statTypes.get(num - 1);
} else {
    System.out.println("You did not select a valid option.");
    System.exit(1);
}
System.out.println("You selected " + metricOption);
getAndDisplayMetricStatistics(cw, selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("4. Get CloudWatch estimated billing for the last
week.");
getMetricStatistics(cw, costDateWeek);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("5. Create a new CloudWatch dashboard with metrics.");
createDashboardWithMetrics(cw, dashboardName, dashboardJson);
System.out.println(DASHES);

System.out.println(DASHES);
```



```
System.out.println("6. List dashboards using a paginator.");
listDashboards(cw);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("7. Create a new custom metric by adding data to
it.");
createNewCustomMetric(cw, dataPoint);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("8. Add an additional metric to the dashboard.");
addMetricToDashboard(cw, dashboardAdd, dashboardName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("9. Create an alarm for the custom metric.");
String alarmName = createAlarm(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("10. Describe ten current alarms.");
describeAlarms(cw);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("11. Get current data for new custom metric.");
getCustomMetricData(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("12. Push data into the custom metric to trigger the
alarm.");
addMetricDataForAlarm(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("13. Check the alarm state using the action
DescribeAlarmsForMetric.");
checkForMetricAlarm(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("14. Get alarm history for the new alarm.");
```

```
getAlarmHistory(cw, settings, myDate);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("15. Add an anomaly detector for the custom metric.");
addAnomalyDetector(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("16. Describe current anomaly detectors.");
describeAnomalyDetectors(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("17. Get a metric image for the custom metric.");
getAndOpenMetricImage(cw, metricImage);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("18. Clean up the Amazon CloudWatch resources.");
deleteDashboard(cw, dashboardName);
deleteCWAlarm(cw, alarmName);
deleteAnomalyDetector(cw, settings);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("The Amazon CloudWatch example scenario is
complete.");
System.out.println(DASHES);
cw.close();
}

public static void deleteAnomalyDetector(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
```

```
        SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .stat("Maximum")
            .build();

        DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
            .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
            .build();

        cw.deleteAnomalyDetector(request);
        System.out.println("Successfully deleted the Anomaly Detector.");

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void deleteCWAlarm(CloudWatchClient cw, String alarmName) {
    try {
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
            .alarmNames(alarmName)
            .build();

        cw.deleteAlarms(request);
        System.out.println("Successfully deleted alarm " + alarmName);

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteDashboard(CloudWatchClient cw, String dashboardName)
{
    try {
        DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
```

```

        .dashboardNames(dashboardName)
        .build();
    cw.deleteDashboards(dashboardsRequest);
    System.out.println(dashboardName + " was successfully deleted.");

} catch (CloudWatchException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}

public static void getAndOpenMetricImage(CloudWatchClient cw, String
fileName) {
    System.out.println("Getting Image data for custom metric.");
    try {
        String myJSON = "{\n" +
            "  \"title\": \"Example Metric Graph\",\n" +
            "  \"view\": \"timeSeries\",\n" +
            "  \"stacked\": false,\n" +
            "  \"period\": 10,\n" +
            "  \"width\": 1400,\n" +
            "  \"height\": 600,\n" +
            "  \"metrics\": [\n" +
            "    [\n" +
            "      \"AWS/Billing\",\n" +
            "      \"EstimatedCharges\",\n" +
            "      \"Currency\",\n" +
            "      \"USD\"\n" +
            "    ]\n" +
            "  ]\n" +
            "}";

        GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
            .metricWidget(myJSON)
            .build();

        GetMetricWidgetImageResponse response =
cw.getMetricWidgetImage(imageRequest);
        SdkBytes sdkBytes = response.metricWidgetImage();
        byte[] bytes = sdkBytes.asByteArray();
        File outputFile = new File(fileName);
        try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {

```

```
        outputStream.write(bytes);
    }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void describeAnomalyDetectors(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
            .maxResults(10)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .build();

        DescribeAnomalyDetectorsResponse response =
cw.describeAnomalyDetectors(detectorsRequest);
        List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
        for (AnomalyDetector detector : anomalyDetectorList) {
            System.out.println("Metric name: " +
detector.singleMetricAnomalyDetector().metricName());
            System.out.println("State: " + detector.stateValue());
        }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

```
public static void addAnomalyDetector(CloudWatchClient cw, String fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .stat("Maximum")
            .build();

        PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
            .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
            .build();

        cw.putAnomalyDetector(anomalyDetectorRequest);
        System.out.println("Added anomaly detector for metric " +
customMetricName + ".");
    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void getAlarmHistory(CloudWatchClient cw, String fileName,
String date) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String alarmName = rootNode.findValue("exampleAlarmName").asText();
```

```
        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();
        DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
        .startDate(start)
        .endDate(endDate)
        .alarmName(alarmName)
        .historyItemType(HistoryItemType.ACTION)
        .build();

        DescribeAlarmHistoryResponse response =
cw.describeAlarmHistory(historyRequest);
        List<AlarmHistoryItem> historyItems = response.alarmHistoryItems();
        if (historyItems.isEmpty()) {
            System.out.println("No alarm history data found for " + alarmName
+ ".");
        } else {
            for (AlarmHistoryItem item : historyItems) {
                System.out.println("History summary: " +
item.historySummary());
                System.out.println("Time stamp: " + item.timestamp());
            }
        }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void checkForMetricAlarm(CloudWatchClient cw, String fileName)
{
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        boolean hasAlarm = false;
        int retries = 10;
```

```
        DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
        .metricName(customMetricName)
        .namespace(customMetricNamespace)
        .build();

        while (!hasAlarm && retries > 0) {
            DescribeAlarmsForMetricResponse response =
cw.describeAlarmsForMetric(metricRequest);
            hasAlarm = response.hasMetricAlarms();
            retries--;
            Thread.sleep(20000);
            System.out.println(".");
        }
        if (!hasAlarm)
            System.out.println("No Alarm state found for " + customMetricName
+ " after 10 retries.");
        else
            System.out.println("Alarm state found for " + customMetricName +
".");

    } catch (CloudWatchException | IOException | InterruptedException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void addMetricDataForAlarm(CloudWatchClient cw, String
fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();

        // Set an Instant object.
        String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
```



```
Instant instant = Instant.parse(time);

MetricDatum datum = MetricDatum.builder()
    .metricName(customMetricName)
    .unit(StandardUnit.NONE)
    .value(1001.00)
    .timestamp(instant)
    .build();

MetricDatum datum2 = MetricDatum.builder()
    .metricName(customMetricName)
    .unit(StandardUnit.NONE)
    .value(1002.00)
    .timestamp(instant)
    .build();

List<MetricDatum> metricDataList = new ArrayList<>();
metricDataList.add(datum);
metricDataList.add(datum2);

PutMetricDataRequest request = PutMetricDataRequest.builder()
    .namespace(customMetricNamespace)
    .metricData(metricDataList)
    .build();

cw.putMetricData(request);
System.out.println("Added metric values for for metric " +
customMetricName);

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void getCustomMetricData(CloudWatchClient cw, String fileName)
{
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
```

```
String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
String customMetricName =
rootNode.findValue("customMetricName").asText();

// Set the date.
Instant nowDate = Instant.now();

long hours = 1;
long minutes = 30;
Instant date2 = nowDate.plus(hours, ChronoUnit.HOURS).plus(minutes,
ChronoUnit.MINUTES);

Metric met = Metric.builder()
    .metricName(customMetricName)
    .namespace(customMetricNamespace)
    .build();

MetricStat metStat = MetricStat.builder()
    .stat("Maximum")
    .period(1)
    .metric(met)
    .build();

MetricDataQuery dataQuery = MetricDataQuery.builder()
    .metricStat(metStat)
    .id("foo2")
    .returnData(true)
    .build();

List<MetricDataQuery> dq = new ArrayList<>();
dq.add(dataQuery);

GetMetricDataRequest getMetReq = GetMetricDataRequest.builder()
    .maxDatapoints(10)
    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
    .startTime(nowDate)
    .endTime(date2)
    .metricDataQueries(dq)
    .build();

GetMetricDataResponse response = cw.getMetricData(getMetReq);
List<MetricDataResult> data = response.metricDataResults();
for (MetricDataResult item : data) {
```

```
        System.out.println("The label is " + item.label());
        System.out.println("The status code is " +
item.statusCode().toString());
    }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void describeAlarms(CloudWatchClient cw) {
    try {
        List<AlarmType> typeList = new ArrayList<>();
        typeList.add(AlarmType.METRIC_ALARM);

        DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
            .alarmTypes(typeList)
            .maxRecords(10)
            .build();

        DescribeAlarmsResponse response = cw.describeAlarms(alarmsRequest);
        List<MetricAlarm> alarmList = response.metricAlarms();
        for (MetricAlarm alarm : alarmList) {
            System.out.println("Alarm name: " + alarm.alarmName());
            System.out.println("Alarm description: " +
alarm.alarmDescription());
        }
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static String createAlarm(CloudWatchClient cw, String fileName) {
    try {
        // Read values from the JSON file.
        JsonParser parser = new JsonFactory().createParser(new
File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
```

```
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        String alarmName = rootNode.findValue("exampleAlarmName").asText();
        String emailTopic = rootNode.findValue("emailTopic").asText();
        String accountId = rootNode.findValue("accountId").asText();
        String region = rootNode.findValue("region").asText();

        // Create a List for alarm actions.
        List<String> alarmActions = new ArrayList<>();
        alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);
        PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
            .alarmActions(alarmActions)
            .alarmDescription("Example metric alarm")
            .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
            .threshold(100.00)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .evaluationPeriods(1)
            .period(10)
            .statistic("Maximum")
            .datapointsToAlarm(1)
            .treatMissingData("ignore")
            .build();

        cw.putMetricAlarm(alarmRequest);
        System.out.println(alarmName + " was successfully created!");
        return alarmName;

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
    return "";
}

public static void addMetricToDashboard(CloudWatchClient cw, String fileName,
String dashboardName) {
    try {
        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(readFileAsString(fileName))
```

```
        .build();

        cw.putDashboard(dashboardRequest);
        System.out.println(dashboardName + " was successfully updated.");

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void createNewCustomMetric(CloudWatchClient cw, Double
dataPoint) {
    try {
        Dimension dimension = Dimension.builder()
            .name("UNIQUE_PAGES")
            .value("URLS")
            .build();

        // Set an Instant object.
        String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
        Instant instant = Instant.parse(time);

        MetricDatum datum = MetricDatum.builder()
            .metricName("PAGES_VISITED")
            .unit(StandardUnit.NONE)
            .value(dataPoint)
            .timestamp(instant)
            .dimensions(dimension)
            .build();

        PutMetricDataRequest request = PutMetricDataRequest.builder()
            .namespace("SITE/TRAFFIC")
            .metricData(datum)
            .build();

        cw.putMetricData(request);
        System.out.println("Added metric values for for metric
PAGES_VISITED");

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

```
    }
}

public static void listDashboards(CloudWatchClient cw) {
    try {
        ListDashboardsIterable listRes = cw.listDashboardsPaginator();
        listRes.stream()
            .flatMap(r -> r.dashboardEntries().stream())
            .forEach(entry -> {
                System.out.println("Dashboard name is: " +
entry.dashboardName());
                System.out.println("Dashboard ARN is: " +
entry.dashboardArn());
            });

    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void createDashboardWithMetrics(CloudWatchClient cw, String
dashboardName, String fileName) {
    try {
        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(readFileAsString(fileName))
            .build();

        PutDashboardResponse response = cw.putDashboard(dashboardRequest);
        System.out.println(dashboardName + " was successfully created.");
        List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
        if (messages.isEmpty()) {
            System.out.println("There are no messages in the new Dashboard");
        } else {
            for (DashboardValidationMessage message : messages) {
                System.out.println("Message is: " + message.message());
            }
        }
    }

    } catch (CloudWatchException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

```
    }
}

public static String readFileAsString(String file) throws IOException {
    return new String(Files.readAllBytes(Paths.get(file)));
}

public static void getMetricStatistics(CloudWatchClient cw, String
costDateWeek) {
    try {
        Instant start = Instant.parse(costDateWeek);
        Instant endDate = Instant.now();
        Dimension dimension = Dimension.builder()
            .name("Currency")
            .value("USD")
            .build();

        List<Dimension> dimensionList = new ArrayList<>();
        dimensionList.add(dimension);
        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .metricName("EstimatedCharges")
            .namespace("AWS/Billing")
            .dimensions(dimensionList)
            .statistics(Statistic.MAXIMUM)
            .startTime(start)
            .endTime(endDate)
            .period(86400)
            .build();

        GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
        List<Datapoint> data = response.datapoints();
        if (!data.isEmpty()) {
            for (Datapoint datapoint : data) {
                System.out
                    .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
            }
        } else {
            System.out.println("The returned data list is empty");
        }
    } catch (CloudWatchException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void getAndDisplayMetricStatistics(CloudWatchClient cw, String
namespace, String metVal,
        String metricOption, String date, Dimension myDimension) {
    try {
        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();

        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .endTime(endDate)
            .startTime(start)
            .dimensions(myDimension)
            .metricName(metVal)
            .namespace(nameSpace)
            .period(86400)
            .statistics(Statistic.fromValue(metricOption))
            .build();

        GetMetricStatisticsResponse response =
cw.getMetricStatistics(statisticsRequest);
        List<Datapoint> data = response.datapoints();
        if (!data.isEmpty()) {
            for (Datapoint datapoint : data) {
                System.out
                    .println("Timestamp: " + datapoint.timestamp() + "
Maximum value: " + datapoint.maximum());
            }
        } else {
            System.out.println("The returned data list is empty");
        }

    } catch (CloudWatchException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static Dimension getSpecificMet(CloudWatchClient cw, String namespace)
{
```



```
    try {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsResponse response = cw.listMetrics(request);
        List<Metric> myList = response.metrics();
        Metric metric = myList.get(0);
        return metric.dimensions().get(0);
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

public static ArrayList<String> listMets(CloudWatchClient cw, String
namespace) {
    try {
        ArrayList<String> metList = new ArrayList<>();
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsIterable listRes = cw.listMetricsPaginator(request);
        listRes.stream()
            .flatMap(r -> r.metrics().stream())
            .forEach(metrics -> metList.add(metrics.metricName()));

        return metList;
    } catch (CloudWatchException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

public static ArrayList<String> listNameSpaces(CloudWatchClient cw) {
    try {
        ArrayList<String> nameSpaceList = new ArrayList<>();
        ListMetricsRequest request = ListMetricsRequest.builder()
            .build();
```

```
ListMetricsIterable listRes = cw.listMetricsPaginator(request);
listRes.stream()
    .flatMap(r -> r.metrics().stream())
    .forEach(metrics -> {
        String data = metrics.namespace();
        if (!nameSpaceList.contains(data)) {
            nameSpaceList.add(data);
        }
    });

    return nameSpaceList;
} catch (CloudWatchException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return null;
}
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.

- [DeleteAlarms](#)
- [DeleteAnomalyDetector](#)
- [DeleteDashboards](#)
- [DescribeAlarmHistory](#)
- [DescribeAlarms](#)
- [DescribeAlarmsForMetric](#)
- [DescribeAnomalyDetectors](#)
- [GetMetricData](#)
- [GetMetricStatistics](#)
- [GetMetricWidgetImage](#)
- [ListMetrics](#)
- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**
```

```
Before running this Kotlin code example, set up your development environment, including your credentials.
```

```
For more information, see the following documentation topic:
```

```
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
```

```
To enable billing metrics and statistics for this example, make sure billing alerts are enabled for your account:
```

```
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/monitor\_estimated\_charges\_with\_cloudwatch.html#turning\_on\_billing\_metrics
```

```
This Kotlin code example performs the following tasks:
```

1. List available namespaces from Amazon CloudWatch. Select a namespace from the list.
2. List available metrics within the selected namespace.
3. Get statistics for the selected metric over the last day.
4. Get CloudWatch estimated billing for the last week.
5. Create a new CloudWatch dashboard with metrics.
6. List dashboards using a paginator.
7. Create a new custom metric by adding data for it.
8. Add the custom metric to the dashboard.
9. Create an alarm for the custom metric.
10. Describe current alarms.
11. Get current data for the new custom metric.
12. Push data into the custom metric to trigger the alarm.
13. Check the alarm state using the action `DescribeAlarmsForMetric`.
14. Get alarm history for the new alarm.
15. Add an anomaly detector for the custom metric.
16. Describe current anomaly detectors.

17. Get a metric image for the custom metric.
 18. Clean up the Amazon CloudWatch resources.
- */

```
val DASHES: String? = String(CharArray(80)).replace("\u0000", "-")
```

```
suspend fun main(args: Array<String>) {
    val usage = ""
        Usage:
            <myDate> <costDateWeek> <dashboardName> <dashboardJson>
<dashboardAdd> <settings> <metricImage>
```

Where:

myDate - The start date to use to get metric statistics. (For example, 2023-01-11T18:35:24.00Z.)

costDateWeek - The start date to use to get AWS Billing and Cost Management statistics. (For example, 2023-01-11T18:35:24.00Z.)

dashboardName - The name of the dashboard to create.

dashboardJson - The location of a JSON file to use to create a dashboard. (See Readme file.)

dashboardAdd - The location of a JSON file to use to update a dashboard. (See Readme file.)

settings - The location of a JSON file from which various values are read. (See Readme file.)

metricImage - The location of a BMP file that is used to create a graph.

```
""
```

```
if (args.size != 7) {
    println(usage)
    System.exit(1)
}
```

```
val myDate = args[0]
val costDateWeek = args[1]
val dashboardName = args[2]
val dashboardJson = args[3]
val dashboardAdd = args[4]
val settings = args[5]
var metricImage = args[6]
val dataPoint = "10.0".toDouble()
val in0b = Scanner(System.`in`)
```

```
println(DASHES)
```

```
println("Welcome to the Amazon CloudWatch example scenario.")
println(DASHES)

println(DASHES)
println("1. List at least five available unique namespaces from Amazon
CloudWatch. Select a CloudWatch namespace from the list.")
val list: ArrayList<String> = listNameSpaces()
for (z in 0..4) {
    println("    ${z + 1}. ${list[z]}")
}

var selectedNamespace: String
var selectedMetrics = ""
var num = inOb.nextLine().toInt()
println("You selected $num")

if (1 <= num && num <= 5) {
    selectedNamespace = list[num - 1]
} else {
    println("You did not select a valid option.")
    exitProcess(1)
}
println("You selected $selectedNamespace")
println(DASHES)

println(DASHES)
println("2. List available metrics within the selected namespace and select
one from the list.")
val metList = listMets(selectedNamespace)
for (z in 0..4) {
    println("    ${ z + 1}. ${metList?.get(z)}")
}
num = inOb.nextLine().toInt()
if (1 <= num && num <= 5) {
    selectedMetrics = metList!![num - 1]
} else {
    println("You did not select a valid option.")
    System.exit(1)
}
println("You selected $selectedMetrics")
val myDimension = getSpecificMet(selectedNamespace)
if (myDimension == null) {
    println("Error - Dimension is null")
    exitProcess(1)
}
```

```
}
println(DASHES)

println(DASHES)
println("3. Get statistics for the selected metric over the last day.")
val metricOption: String
val statTypes = ArrayList<String>()
statTypes.add("SampleCount")
statTypes.add("Average")
statTypes.add("Sum")
statTypes.add("Minimum")
statTypes.add("Maximum")

for (t in 0..4) {
    println("    ${t + 1}. ${statTypes[t]}")
}
println("Select a metric statistic by entering a number from the preceding
list:")
num = in0b.nextLine().toInt()
if (1 <= num && num <= 5) {
    metricOption = statTypes[num - 1]
} else {
    println("You did not select a valid option.")
    exitProcess(1)
}
println("You selected $metricOption")
getAndDisplayMetricStatistics(selectedNamespace, selectedMetrics,
metricOption, myDate, myDimension)
println(DASHES)

println(DASHES)
println("4. Get CloudWatch estimated billing for the last week.")
getMetricStatistics(costDateWeek)
println(DASHES)

println(DASHES)
println("5. Create a new CloudWatch dashboard with metrics.")
createDashboardWithMetrics(dashboardName, dashboardJson)
println(DASHES)

println(DASHES)
println("6. List dashboards using a paginator.")
listDashboards()
println(DASHES)
```

```
println(DASHES)
println("7. Create a new custom metric by adding data to it.")
createNewCustomMetric(dataPoint)
println(DASHES)

println(DASHES)
println("8. Add an additional metric to the dashboard.")
addMetricToDashboard(dashboardAdd, dashboardName)
println(DASHES)

println(DASHES)
println("9. Create an alarm for the custom metric.")
val alarmName: String = createAlarm(settings)
println(DASHES)

println(DASHES)
println("10. Describe 10 current alarms.")
describeAlarms()
println(DASHES)

println(DASHES)
println("11. Get current data for the new custom metric.")
getCustomMetricData(settings)
println(DASHES)

println(DASHES)
println("12. Push data into the custom metric to trigger the alarm.")
addMetricDataForAlarm(settings)
println(DASHES)

println(DASHES)
println("13. Check the alarm state using the action
DescribeAlarmsForMetric.")
checkForMetricAlarm(settings)
println(DASHES)

println(DASHES)
println("14. Get alarm history for the new alarm.")
getAlarmHistory(settings, myDate)
println(DASHES)

println(DASHES)
println("15. Add an anomaly detector for the custom metric.")
```

```
addAnomalyDetector(settings)
println(DASHES)

println(DASHES)
println("16. Describe current anomaly detectors.")
describeAnomalyDetectors(settings)
println(DASHES)

println(DASHES)
println("17. Get a metric image for the custom metric.")
getAndOpenMetricImage(metricImage)
println(DASHES)

println(DASHES)
println("18. Clean up the Amazon CloudWatch resources.")
deleteDashboard(dashboardName)
deleteAlarm(alarmName)
deleteAnomalyDetector(settings)
println(DASHES)

println(DASHES)
println("The Amazon CloudWatch example scenario is complete.")
println(DASHES)
}

suspend fun deleteAnomalyDetector(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }
}
```



```
CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    cwClient.deleteAnomalyDetector(request)
    println("Successfully deleted the Anomaly Detector.")
}
}

suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}

suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}

suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting Image data for custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
                "AWS/Billing",
                "EstimatedCharges",
                "Currency",
            ]
        ]
    }"""
}
```

```
        "USD"
    ]
}""

val imageRequest =
    GetMetricWidgetImageRequest {
        metricWidget = myJSON
    }

CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    val response = cwClient.getMetricWidgetImage(imageRequest)
    val bytes = response.metricWidgetImage
    if (bytes != null) {
        File(fileName).writeBytes(bytes)
    }
}
println("You have successfully written data to $fileName")
}

suspend fun describeAnomalyDetectors(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
        rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val detectorsRequest =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(detectorsRequest)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
${detector.singleMetricAnomalyDetector?.metricName}")
            println("State: ${detector.stateValue}")
        }
    }
}
```

```
suspend fun addAnomalyDetector(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            metricName = customMetricName
            namespace = customMetricNamespace
            stat = "Maximum"
        }

    val anomalyDetectorRequest =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(anomalyDetectorRequest)
        println("Added anomaly detector for metric $customMetricName.")
    }
}

suspend fun getAlarmHistory(
    fileName: String,
    date: String,
) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val start = Instant.parse(date)
    val endDateVal = Instant.now()

    val historyRequest =
        DescribeAlarmHistoryRequest {
            startDate =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            endDate =
                aws.smithy.kotlin.runtime.time
```

```

        .Instant(endDateVal)
        alarmName = alarmNameVal
        historyItemType = HistoryItemType.Action
    }

    CloudWatchClient {
        credentialsProvider = EnvironmentCredentialsProvider()
        region = "us-east-1"
    }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(historyRequest)
        val historyItems = response.alarmHistoryItems
        if (historyItems != null) {
            if (historyItems.isEmpty()) {
                println("No alarm history data found for $alarmNameVal.")
            } else {
                for (item in historyItems) {
                    println("History summary ${item.historySummary}")
                    println("Time stamp: ${item.timestamp}")
                }
            }
        }
    }
}

suspend fun checkForMetricAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
        rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    var hasAlarm = false
    var retries = 10

    val metricRequest =
        DescribeAlarmsForMetricRequest {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        while (!hasAlarm && retries > 0) {
            val response = cwClient.describeAlarmsForMetric(metricRequest)
            if (response.metricAlarms?.count()!! > 0) {
                hasAlarm = true
            }
        }
    }
}

```

```
        }
        retries--
        delay(20000)
        println(".")
    }
    if (!hasAlarm) {
        println("No Alarm state found for $customMetricName after 10
retries.")
    } else {
        println("Alarm state found for $customMetricName.")
    }
}
}

suspend fun addMetricDataForAlarm(fileName: String?) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set an Instant object.
    val time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1001.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }

    val datum2 =
        MetricDatum {
            metricName = customMetricName
            unit = StandardUnit.None
            value = 1002.00
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
        }
}
```

```
    }

    val metricDataList = ArrayList<MetricDatum>()
    metricDataList.add(datum)
    metricDataList.add(datum2)

    val request =
        PutMetricDataRequest {
            namespace = customMetricNamespace
            metricData = metricDataList
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric $customMetricName")
    }
}

suspend fun getCustomMetricData(fileName: String) {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val customMetricNamespace =
        rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()

    // Set the date.
    val nowDate = Instant.now()
    val hours: Long = 1
    val minutes: Long = 30
    val date2 =
        nowDate.plus(hours, ChronoUnit.HOURS).plus(
            minutes,
            ChronoUnit.MINUTES,
        )

    val met =
        Metric {
            metricName = customMetricName
            namespace = customMetricNamespace
        }

    val metStat =
        MetricStat {
```

```
        stat = "Maximum"
        period = 1
        metric = met
    }

    val dataQuery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = ArrayList<MetricDataQuery>()
    dq.add(dataQuery)
    val getMetReq =
        GetMetricDataRequest {
            maxDatapoints = 10
            scanBy = ScanBy.TimestampDescending
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(nowDate)
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(date2)
            metricDataQueries = dq
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(getMetReq)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}

suspend fun describeAlarms() {
    val typeList = ArrayList<AlarmType>()
    typeList.add(AlarmType.MetricAlarm)
    val alarmsRequest =
        DescribeAlarmsRequest {
            alarmTypes = typeList
            maxRecords = 10
        }
}
```

```
CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    val response = cwClient.describeAlarms(alarmsRequest)
    response.metricAlarms?.forEach { alarm ->
        println("Alarm name: ${alarm.alarmName}")
        println("Alarm description: ${alarm.alarmDescription}")
    }
}

suspend fun createAlarm(fileName: String): String {
    // Read values from the JSON file.
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode: JsonNode = ObjectMapper().readTree(parser)
    val customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText()
    val customMetricName = rootNode.findValue("customMetricName").asText()
    val alarmNameVal = rootNode.findValue("exampleAlarmName").asText()
    val emailTopic = rootNode.findValue("emailTopic").asText()
    val accountId = rootNode.findValue("accountId").asText()
    val region2 = rootNode.findValue("region").asText()

    // Create a List for alarm actions.
    val alarmActionObs: MutableList<String> = ArrayList()
    alarmActionObs.add("arn:aws:sns:$region2:$accountId:$emailTopic")
    val alarmRequest =
        PutMetricAlarmRequest {
            alarmActions = alarmActionObs
            alarmDescription = "Example metric alarm"
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanOrEqualToThreshold
            threshold = 100.00
            metricName = customMetricName
            namespace = customMetricNamespace
            evaluationPeriods = 1
            period = 10
            statistic = Statistic.Maximum
            datapointsToAlarm = 1
            treatMissingData = "ignore"
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(alarmRequest)
        println("$alarmNameVal was successfully created!")
    }
}
```



```
        return alarmNameVal
    }
}

suspend fun addMetricToDashboard(
    fileNameVal: String,
    dashboardNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully updated.")
    }
}

suspend fun createNewCustomMetric(dataPoint: Double) {
    val dimension =
        Dimension {
            name = "UNIQUE_PAGES"
            value = "URLS"
        }

    // Set an Instant object.
    val time =
        ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT)
    val instant = Instant.parse(time)
    val datum =
        MetricDatum {
            metricName = "PAGES_VISITED"
            unit = StandardUnit.None
            value = dataPoint
            timestamp =
                aws.smithy.kotlin.runtime.time
                    .Instant(instant)
            dimensions = listOf(dimension)
        }

    val request =
        PutMetricDataRequest {
```

```
        namespace = "SITE/TRAFFIC"
        metricData = listOf(datum)
    }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for for metric PAGES_VISITED")
    }
}

suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}

suspend fun createDashboardWithMetrics(
    dashboardNameVal: String,
    fileNameVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = readFileAsString(fileNameVal)
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}
```

```
    }
  }
}

fun readFileAsString(file: String): String =
  String(Files.readAllBytes(Paths.get(file)))

suspend fun getMetricStatistics(costDateWeek: String?) {
  val start = Instant.parse(costDateWeek)
  val endDate = Instant.now()
  val dimension =
    Dimension {
      name = "Currency"
      value = "USD"
    }

  val dimensionList: MutableList<Dimension> = ArrayList()
  dimensionList.add(dimension)

  val statisticsRequest =
    GetMetricStatisticsRequest {
      metricName = "EstimatedCharges"
      namespace = "AWS/Billing"
      dimensions = dimensionList
      statistics = listOf(Statistic.Maximum)
      startTime =
        aws.smithy.kotlin.runtime.time
          .Instant(start)
      endTime =
        aws.smithy.kotlin.runtime.time
          .Instant(endDate)
      period = 86400
    }

  CloudWatchClient { region = "us-east-1" }.use { cwClient ->
    val response = cwClient.getMetricStatistics(statisticsRequest)
    val data: List<Datapoint>? = response.datapoints
    if (data != null) {
      if (!data.isEmpty()) {
        for (datapoint in data) {
          println("Timestamp: ${datapoint.timestamp} Maximum value:
            ${datapoint.maximum}")
        }
      } else {
        println("The returned data list is empty")
      }
    }
  }
}
```

```
    }
  }
}

suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}
```

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}

suspend fun getSpecificMet(namespaceVal: String?): Dimension? {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(request)
        val myList = response.metrics
        if (myList != null) {
            return myList[0].dimensions?.get(0)
        }
    }
    return null
}

suspend fun listNameSpaces(): ArrayList<String> {
    val nameSpaceList = ArrayList<String>()
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.listMetrics(ListMetricsRequest {})
        response.metrics?.forEach { metrics ->
            val data = metrics.namespace
            if (!nameSpaceList.contains(data)) {
                nameSpaceList.add(data!!)
            }
        }
    }
}
```

```
    }  
    return nameSpaceList  
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
 - [DeleteAlarms](#)
 - [DeleteAnomalyDetector](#)
 - [DeleteDashboards](#)
 - [DescribeAlarmHistory](#)
 - [DescribeAlarms](#)
 - [DescribeAlarmsForMetric](#)
 - [DescribeAnomalyDetectors](#)
 - [GetMetricData](#)
 - [GetMetricStatistics](#)
 - [GetMetricWidgetImage](#)
 - [ListMetrics](#)
 - [PutAnomalyDetector](#)
 - [PutDashboard](#)
 - [PutMetricAlarm](#)
 - [PutMetricData](#)

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Manage CloudWatch metrics and alarms using an AWS SDK

The following code example shows how to:

- Create an alarm to watch a CloudWatch metric.
- Put data into a metric and trigger the alarm.
- Get data from the alarm.
- Delete the alarm.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create a class that wraps CloudWatch operations.

```
from datetime import datetime, timedelta
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource

    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
        set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
```

```

    :param data_set: The set of data to send. This set is a dictionary that
                    contains a list of values and a list of corresponding
counts.
                    The value and count lists must be the same length.
    """
    try:
        metric = self.cloudwatch_resource.Metric(namespace, name)
        metric.put_data(
            Namespace=namespace,
            MetricData=[
                {
                    "MetricName": name,
                    "Timestamp": timestamp,
                    "Values": data_set["values"],
                    "Counts": data_set["counts"],
                    "Unit": unit,
                }
            ],
        )
        logger.info("Put data set for metric %s.%s.", namespace, name)
    except ClientError:
        logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
        raise

def create_metric_alarm(
    self,
    metric_namespace,
    metric_name,
    alarm_name,
    stat_type,
    period,
    eval_periods,
    threshold,
    comparison_op,
):
    """
    Creates an alarm that watches a metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    :param alarm_name: The name of the alarm.
    :param stat_type: The type of statistic the alarm watches.

```



```

        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                               alarm threshold before the alarm is set into an
alarmed
                               state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                               against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm

    def put_metric_data(self, namespace, name, value, unit):

```

```

"""
Sends a single data value to CloudWatch for a metric. This metric is
given
a timestamp of the current UTC time.

:param namespace: The namespace of the metric.
:param name: The name of the metric.
:param value: The value of the metric.
:param unit: The unit of the metric.
"""
try:
    metric = self.cloudwatch_resource.Metric(namespace, name)
    metric.put_data(
        Namespace=namespace,
        MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
    )
    logger.info("Put data for metric %s.%s", namespace, name)
except ClientError:
    logger.exception("Couldn't put data for metric %s.%s", namespace,
name)
    raise

def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
    """
    Gets statistics for a metric within a specified time span. Metrics are
grouped
into the specified period.

:param namespace: The namespace of the metric.
:param name: The name of the metric.
:param start: The UTC start time of the time span to retrieve.
:param end: The UTC end time of the time span to retrieve.
:param period: The period, in seconds, in which to group metrics. The
period
must match the granularity of the metric, which depends on
the metric's age. For example, metrics that are older than
three hours have a one-minute granularity, so the period
must
be at least 60 and must be a multiple of 60.
:param stat_types: The type of statistics to retrieve, such as average
value
or maximum value.

```

```
        :return: The retrieved statistics for the metric.
        """
    try:
        metric = self.cloudwatch_resource.Metric(namespace, name)
        stats = metric.get_statistics(
            StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
        )
        logger.info(
            "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
        )
    except ClientError:
        logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
        raise
    else:
        return stats

def get_metric_alarms(self, metric_namespace, metric_name):
    """
    Gets the alarms that are currently watching the specified metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    :returns: An iterator that yields the alarms.
    """
    metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
    alarm_iter = metric.alarms.all()
    logger.info("Got alarms for metric %s.%s.", metric_namespace,
metric_name)
    return alarm_iter

def delete_metric_alarms(self, metric_namespace, metric_name):
    """
    Deletes all of the alarms that are currently watching the specified
metric.

    :param metric_namespace: The namespace of the metric.
    :param metric_name: The name of the metric.
    """
    try:
```

```
        metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
        metric.alarms.delete()
        logger.info(
            "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
        )
    except ClientError:
        logger.exception(
            "Couldn't delete alarms for metric %s.%s.",
            metric_namespace,
            metric_name,
        )
        raise
```

Use the wrapper class to put data in a metric, trigger an alarm that watches the metric, and get data from the alarm.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    cw_wrapper = CloudWatchWrapper(boto3.resource("cloudwatch"))

    minutes = 20
    metric_namespace = "doc-example-metric"
    metric_name = "page_views"
    start = datetime.utcnow() - timedelta(minutes=minutes)
    print(
        f"Putting data into metric {metric_namespace}.{metric_name} spanning the
"
        f"last {minutes} minutes."
    )
    for offset in range(0, minutes):
        stamp = start + timedelta(minutes=offset)
        cw_wrapper.put_metric_data_set(
            metric_namespace,
            metric_name,
```

```
        stamp,
        "Count",
        {
            "values": [
                random.randint(bound, bound * 2)
                for bound in range(offset + 1, offset + 11)
            ],
            "counts": [random.randint(1, offset + 1) for _ in range(10)],
        },
    ),

alarm_name = "high_page_views"
period = 60
eval_periods = 2
print(f"Creating alarm {alarm_name} for metric {metric_name}.")
alarm = cw_wrapper.create_metric_alarm(
    metric_namespace,
    metric_name,
    alarm_name,
    "Maximum",
    period,
    eval_periods,
    100,
    "GreaterThanThreshold",
)
print(f"Alarm ARN is {alarm.alarm_arn}.")
print(f"Current alarm state is: {alarm.state_value}.")

print(
    f"Sending data to trigger the alarm. This requires data over the
    threshold "
    f"for {eval_periods} periods of {period} seconds each."
)
while alarm.state_value == "INSUFFICIENT_DATA":
    print("Sending data for the metric.")
    cw_wrapper.put_metric_data(
        metric_namespace, metric_name, random.randint(100, 200), "Count"
    )
    alarm.load()
    print(f"Current alarm state is: {alarm.state_value}.")
    if alarm.state_value == "INSUFFICIENT_DATA":
        print(f"Waiting for {period} seconds...")
        time.sleep(period)
    else:
```

```
        print("Wait for a minute for eventual consistency of metric data.")
        time.sleep(period)
        if alarm.state_value == "OK":
            alarm.load()
            print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Getting data for metric {metric_namespace}.{metric_name} during
timespan "
        f"of {start} to {datetime.utcnow()} (times are UTC)."
    )
    stats = cw_wrapper.get_metric_statistics(
        metric_namespace,
        metric_name,
        start,
        datetime.utcnow(),
        60,
        ["Average", "Minimum", "Maximum"],
    )
    print(
        f"Got {len(stats['Datapoints'])} data points for metric "
        f"{metric_namespace}.{metric_name}."
    )
    pprint(sorted(stats["Datapoints"], key=lambda x: x["Timestamp"]))

    print(f"Getting alarms for metric {metric_name}.")
    alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
    for alarm in alarms:
        print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

    print(f"Deleting alarms for metric {metric_name}.")
    cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)

    print("Thanks for watching!")
    print("-" * 88)
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [DeleteAlarms](#)
 - [DescribeAlarmsForMetric](#)

- [DisableAlarmActions](#)
- [EnableAlarmActions](#)
- [GetMetricStatistics](#)
- [ListMetrics](#)
- [PutMetricAlarm](#)
- [PutMetricData](#)

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for CloudWatch using AWS SDKs

The following sample applications use AWS SDKs to combine CloudWatch with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples

- [Monitor performance of Amazon DynamoDB using an AWS SDK](#)

Monitor performance of Amazon DynamoDB using an AWS SDK

The following code example shows how to configure an application's use of DynamoDB to monitor performance.

Java

SDK for Java 2.x

This example shows how to configure a Java application to monitor the performance of DynamoDB. The application sends metric data to CloudWatch where you can monitor the performance.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- CloudWatch
- DynamoDB

For a complete list of AWS SDK developer guides and code examples, see [Using CloudWatch with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon CloudWatch

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to CloudWatch, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon CloudWatch. It shows you how to configure Amazon CloudWatch to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudWatch resources.

Contents

- [Data protection in Amazon CloudWatch](#)
- [Identity and access management for Amazon CloudWatch](#)
- [Compliance validation for Amazon CloudWatch](#)
- [Resilience in Amazon CloudWatch](#)
- [Infrastructure security in Amazon CloudWatch](#)
- [AWS Security Hub](#)
- [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints](#)
- [Security considerations for Synthetics canaries](#)

Data protection in Amazon CloudWatch

The AWS [shared responsibility model](#) applies to data protection in Amazon CloudWatch. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with CloudWatch or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption in transit

CloudWatch uses end-to-end encryption of data in transit.

Identity and access management for Amazon CloudWatch

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CloudWatch resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon CloudWatch works with IAM](#)
- [Identity-based policy examples for Amazon CloudWatch](#)
- [Troubleshooting Amazon CloudWatch identity and access](#)
- [CloudWatch dashboard permissions update](#)
- [AWS managed \(predefined\) policies for CloudWatch](#)
- [Customer managed policy examples](#)
- [CloudWatch updates to AWS managed policies](#)
- [Using condition keys to limit access to CloudWatch namespaces](#)
- [Using condition keys to limit Contributor Insights users' access to log groups](#)
- [Using condition keys to limit alarm actions](#)
- [Using service-linked roles for CloudWatch](#)
- [Using service-linked roles for CloudWatch RUM](#)
- [Using service-linked roles for CloudWatch Application Insights](#)
- [AWS managed policies for Amazon CloudWatch Application Insights](#)
- [Amazon CloudWatch permissions reference](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in CloudWatch.

Service user – If you use the CloudWatch service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more CloudWatch features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in CloudWatch, see [Troubleshooting Amazon CloudWatch identity and access](#).

Service administrator – If you're in charge of CloudWatch resources at your company, you probably have full access to CloudWatch. It's your job to determine which CloudWatch features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with CloudWatch, see [How Amazon CloudWatch works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to CloudWatch. To view example CloudWatch identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon CloudWatch](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permission sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most

policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon CloudWatch works with IAM

Before you use IAM to manage access to CloudWatch, learn what IAM features are available to use with CloudWatch.

IAM features you can use with Amazon CloudWatch

IAM feature	CloudWatch support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how CloudWatch and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for CloudWatch

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for CloudWatch

To view examples of CloudWatch identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

Resource-based policies within CloudWatch

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant

the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for CloudWatch

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CloudWatch actions, see [Actions defined by Amazon CloudWatch](#) in the *Service Authorization Reference*.

Policy actions in CloudWatch use the following prefix before the action:

```
cloudwatch
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
  "cloudwatch:action1",  
  "cloudwatch:action2"  
]
```

To view examples of CloudWatch identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

Policy resources for CloudWatch

Supports policy resources	Yes
---------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*" 
```

To see a list of CloudWatch resource types and their ARNs, see [Resources defined by Amazon CloudWatch](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon CloudWatch](#).

To view examples of CloudWatch identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

Policy condition keys for CloudWatch

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of CloudWatch condition keys, see [Condition keys for Amazon CloudWatch](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon CloudWatch](#).

To view examples of CloudWatch identity-based policies, see [Identity-based policy examples for Amazon CloudWatch](#).

ACLs in CloudWatch

Supports ACLs

No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CloudWatch

Supports ABAC (tags in policies)

Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using temporary credentials with CloudWatch

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for CloudWatch

Supports forward access sessions (FAS)	Yes
--	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for CloudWatch

Supports service roles	Yes
------------------------	-----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break CloudWatch functionality. Edit service roles only when CloudWatch provides guidance to do so.

Identity-based policy examples for Amazon CloudWatch

By default, users and roles don't have permission to create or modify CloudWatch resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Creating IAM policies](#) in the *IAM User Guide*.

For details about actions and resource types defined by CloudWatch, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon CloudWatch](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the CloudWatch console](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CloudWatch resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the CloudWatch console

To access the Amazon CloudWatch console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CloudWatch resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the CloudWatch console, also attach the CloudWatch *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Permissions needed for CloudWatch console

The full set of permissions required to work with the CloudWatch console are listed below. These permissions provide full write and read access to the CloudWatch console.

- application-autoscaling:DescribeScalingPolicies
- autoscaling:DescribeAutoScalingGroups
- autoscaling:DescribePolicies
- cloudtrail:DescribeTrails
- cloudwatch>DeleteAlarms
- cloudwatch:DescribeAlarmHistory
- cloudwatch:DescribeAlarms
- cloudwatch:GetMetricData
- cloudwatch:GetMetricStatistics
- cloudwatch:ListMetrics
- cloudwatch:PutMetricAlarm
- cloudwatch:PutMetricData
- ec2:DescribeInstances

- ec2:DescribeTags
- ec2:DescribeVolumes
- es:DescribeElasticsearchDomain
- es:ListDomainNames
- events:DeleteRule
- events:DescribeRule
- events:DisableRule
- events:EnableRule
- events:ListRules
- events:PutRule
- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStream
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration
- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- logs:CancelExportTask
- logs:CreateExportTask
- logs:CreateLogGroup

- logs:CreateLogStream
- logs>DeleteLogGroup
- logs>DeleteLogStream
- logs>DeleteMetricFilter
- logs>DeleteRetentionPolicy
- logs>DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter
- s3:CreateBucket
- s3:ListBucket
- sns:CreateTopic
- sns:GetTopicAttributes
- sns:ListSubscriptions
- sns:ListTopics

- sns:SetTopicAttributes
- sns:Subscribe
- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs:ListQueues
- sqs:SetQueueAttributes
- swf:CreateAction
- swf:DescribeAction
- swf:ListActionTemplates
- swf:RegisterAction
- swf:RegisterDomain
- swf:UpdateAction

Additionally, to view the X-Ray Trace Map, you need `AWSXrayReadOnlyAccess`

Troubleshooting Amazon CloudWatch identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CloudWatch and IAM.

Topics

- [I am not authorized to perform an action in CloudWatch](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my CloudWatch resources](#)

I am not authorized to perform an action in CloudWatch

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional `my-example-widget` resource but doesn't have the fictional `cloudwatch:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
cloudwatch:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the `my-example-widget` resource by using the `cloudwatch:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CloudWatch.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CloudWatch. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my CloudWatch resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CloudWatch supports these features, see [How Amazon CloudWatch works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

CloudWatch dashboard permissions update

On May 1, 2018, AWS changed the permissions required to access CloudWatch dashboards. Dashboard access in the CloudWatch console now requires permissions that were introduced in 2017 to support dashboard API operations:

- **cloudwatch:GetDashboard**
- **cloudwatch:ListDashboards**
- **cloudwatch:PutDashboard**
- **cloudwatch>DeleteDashboards**

To access CloudWatch dashboards, you need one of the following:

- The **AdministratorAccess** policy.
- The **CloudWatchFullAccess** policy.
- A custom policy that includes one or more of these specific permissions:
 - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards
 - `cloudwatch:PutDashboard` to be able to create or modify dashboards
 - `cloudwatch>DeleteDashboards` to be able to delete dashboards

For more information about using policies to change permissions for an IAM user, see [Changing Permissions for an IAM user](#).

For more information about CloudWatch permissions, see [Amazon CloudWatch permissions reference](#).

For more information about dashboard API operations, see [PutDashboard](#) in the Amazon CloudWatch API Reference.

AWS managed (predefined) policies for CloudWatch

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see [AWS managed policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to CloudWatch.

Topics

- [CloudWatchFullAccessV2](#)
- [CloudWatchFullAccess](#)
- [CloudWatchReadOnlyAccess](#)
- [CloudWatchActionsEC2Access](#)
- [CloudWatchAutomaticDashboardsAccess](#)
- [CloudWatchAgentServerPolicy](#)
- [CloudWatchAgentAdminPolicy](#)
- [AWS managed \(predefined\) policies for CloudWatch cross-account observability](#)
- [AWS managed \(predefined\) policies for CloudWatch Application Signals](#)
- [AWS managed \(predefined\) policies for CloudWatch Synthetics](#)
- [AWS managed \(predefined\) policies for Amazon CloudWatch RUM](#)
- [AWS managed \(predefined\) policies for CloudWatch Evidently](#)
- [AWS managed policy for AWS Systems Manager Incident Manager](#)

CloudWatchFullAccessV2

AWS recently added the **CloudWatchFullAccessV2** managed IAM policy. This policy grants full access to CloudWatch actions and resources and also more properly scopes the permissions granted for other services such as Amazon SNS and Amazon EC2 Auto Scaling. We recommend

that you begin using this policy instead of using **CloudWatchFullAccess**. AWS plans to deprecate **CloudWatchFullAccess** in the near future.

It includes `application-signals`: permissions so that users can access all the functionality from the CloudWatch console under Application Signals. It includes some `autoscaling:Describe` permissions so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes some `sns` permissions so that users with this policy can retrieve create Amazon SNS topics and associate them with CloudWatch alarms. It includes IAM permissions so that users with this policy can view information about service-linked roles associated with CloudWatch. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have full access to CloudWatch Synthetics, AWS X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

The contents of **CloudWatchFullAccessV2** are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchFullAccessPermissions",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalingPolicies",
        "application-signals:*",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribePolicies",
        "cloudwatch:*",
        "logs:*",
        "sns:CreateTopic",
        "sns:ListSubscriptions",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics",
        "sns:Subscribe",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "oam:ListSinks",
        "rum:*",

```

```

        "synthetics:*",
        "xray:*"
    ],
    "Resource": "*"
  },
  {
    "Sid": "CloudWatchApplicationSignalsServiceLinkedRolePermissions",
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "application-
signals.cloudwatch.amazonaws.com"
      }
    }
  },
  {
    "Sid": "EventsServicePermissions",
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "events.amazonaws.com"
      }
    }
  },
  {
    "Sid": "OAMReadPermissions",
    "Effect": "Allow",
    "Action": [
      "oam:ListAttachedLinks"
    ],
    "Resource": "arn:aws:oam:*:*:sink/*"
  }
]
}

```

CloudWatchFullAccess

The **CloudWatchFullAccess** policy is on the path to deprecation. We recommend that you stop using it, and use [CloudWatchFullAccessV2](#) instead.

The contents of **CloudWatchFullAccess** are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:*",
        "logs:*",
        "sns:*",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "oam:ListSinks"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/AWSServiceRoleForCloudWatchEvents*",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "events.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "oam:ListAttachedLinks"
      ],
      "Resource": "arn:aws:oam::*:sink/*"
    }
  ]
}
```

```
}
```

CloudWatchReadOnlyAccess

The **CloudWatchReadOnlyAccess** policy grants read-only access to CloudWatch.

The policy includes some `logs:permissions`, so that users with this policy can use the console to view CloudWatch Logs information and CloudWatch Logs Insights queries. It includes `autoscaling:Describe*`, so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes the `application-signals:permissions` so that users can use Application Signals to monitor the health of their services. It includes `application-autoscaling:DescribeScalingPolicies`, so that users with this policy can access information about Application Auto Scaling policies. It includes `sns:Get*` and `sns:List*`, so that users with this policy can retrieve information about the Amazon SNS topics that receive notifications about CloudWatch alarms. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions, so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability. It includes the `iam:GetRole` permissions so that users can check if CloudWatch Application Signals have been set up.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have read-only access to CloudWatch Synthetics, AWS X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

The following is the content of the **CloudWatchReadOnlyAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchReadOnlyAccessPermissions",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalingPolicies",
        "application-signals:BatchGet*",
        "application-signals:Get*",
        "application-signals:List*",
        "autoscaling:Describe*",
        "cloudwatch:BatchGet*",
        "cloudwatch:Describe*",
        "cloudwatch:GenerateQuery",
        "cloudwatch:Get*",

```

```

        "cloudwatch:List*",
        "logs:Get*",
        "logs:List*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:Describe*",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "oam:ListSinks",
        "sns:Get*",
        "sns:List*",
        "rum:BatchGet*",
        "rum:Get*",
        "rum:List*",
        "synthetics:Describe*",
        "synthetics:Get*",
        "synthetics:List*",
        "xray:BatchGet*",
        "xray:Get*"
    ],
    "Resource": "*"
},
{
    "Sid": "OAMReadPermissions",
    "Effect": "Allow",
    "Action": [
        "oam:ListAttachedLinks"
    ],
    "Resource": "arn:aws:oam:*:*:sink/*"
},
{
    "Sid": "CloudWatchReadOnlyGetRolePermissions",
    "Effect": "Allow",
    "Action": "iam:GetRole",
    "Resource": "arn:aws:iam:*:*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
}
]
}

```

CloudWatchActionsEC2Access

The **CloudWatchActionsEC2Access** policy grants read-only access to CloudWatch alarms and metrics in addition to Amazon EC2 metadata. It also grants access to the Stop, Terminate, and Reboot API actions for EC2 instances.

The following is the content of the **CloudWatchActionsEC2Access** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Describe*",
        "ec2:Describe*",
        "ec2:RebootInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

CloudWatchAutomaticDashboardsAccess

The **CloudWatch-CrossAccountAccess** managed policy is used by the **CloudWatch-CrossAccountSharingRole** IAM role. This role and policy enable users of cross-account dashboards to view automatic dashboards in each account that is sharing dashboards.

The following is the content of **CloudWatchAutomaticDashboardsAccess**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "cloudfront:GetDistribution",
        "cloudfront:ListDistributions",
        "dynamodb:DescribeTable",
        "dynamodb:ListTables",

```

```

    "ec2:DescribeInstances",
    "ec2:DescribeVolumes",
    "ecs:DescribeClusters",
    "ecs:DescribeContainerInstances",
    "ecs:ListClusters",
    "ecs:ListContainerInstances",
    "ecs:ListServices",
    "elasticache:DescribeCacheClusters",
    "elasticbeanstalk:DescribeEnvironments",
    "elasticfilesystem:DescribeFileSystems",
    "elasticloadbalancing:DescribeLoadBalancers",
    "kinesis:DescribeStream",
    "kinesis:ListStreams",
    "lambda:GetFunction",
    "lambda:ListFunctions",
    "rds:DescribeDBClusters",
    "rds:DescribeDBInstances",
    "resource-groups:ListGroupResources",
    "resource-groups:ListGroups",
    "route53:GetHealthCheck",
    "route53:ListHealthChecks",
    "s3:ListAllMyBuckets",
    "s3:ListBucket",
    "sns:ListTopics",
    "sqs:GetQueueAttributes",
    "sqs:GetQueueUrl",
    "sqs:ListQueues",
    "synthetics:DescribeCanariesLastRun",
    "tag:GetResources"
  ],
  "Effect": "Allow",
  "Resource": "*"
},
{
  "Action": [
    "apigateway:GET"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:apigateway:*::/restapis*"
  ]
}
]

```

CloudWatchAgentServerPolicy

The **CloudWatchAgentServerPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances to allow the CloudWatch agent to read information from the instance and write it to CloudWatch. Its contents are as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CWACloudWatchServerPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "ec2:DescribeVolumes",
        "ec2:DescribeTags",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CWASSMServerPermissions",
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
    }
  ]
}
```


CloudWatchAgentAdminPolicy

The **CloudWatchAgentAdminPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances. This policy allows the CloudWatch agent to read information from the instance and write it to CloudWatch, and also to write information to Parameter Store. Its contents are as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CWACloudWatchPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "ec2:DescribeTags",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CWASSMPermissions",
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter",
        "ssm:PutParameter"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-*"
    }
  ]
}
```

Note

You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for CloudWatch actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

AWS managed (predefined) policies for CloudWatch cross-account observability

The policies in this section grant permissions related to CloudWatch cross-account observability. For more information, see [CloudWatch cross-account observability](#).

CloudWatchCrossAccountSharingConfiguration

The **CloudWatchCrossAccountSharingConfiguration** policy grants access to create, manage, and view Observability Access Manager links for sharing CloudWatch resources between accounts. For more information, see [CloudWatch cross-account observability](#). The contents are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Link",
        "oam:ListLinks"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "oam>DeleteLink",
        "oam:GetLink",
        "oam:TagResource"
      ],
      "Resource": "arn:aws:oam:*:*:link/*"
    }
  ],
}
```

```
{
  "Effect": "Allow",
  "Action": [
    "oam:CreateLink",
    "oam:UpdateLink"
  ],
  "Resource": [
    "arn:aws:oam:*:*:link/*",
    "arn:aws:oam:*:*:sink/*"
  ]
}
```

OAMFullAccess

The **OAMFullAccess** policy grants access to create, manage, and view Observability Access Manager sinks and links, which are used for CloudWatch cross-account observability.

The **OAMFullAccess** policy by itself does not permit you to share observability data across links. To create a link to share CloudWatch metrics, you also need either **CloudWatchFullAccess** or **CloudWatchCrossAccountSharingConfiguration**. To create a link to share CloudWatch Logs log groups, you also need either **CloudWatchLogsFullAccess** or **CloudWatchLogsCrossAccountSharingConfiguration**. To create a link to share X-Ray traces, you also need either **AWSXRayFullAccess** or **AWSXRayCrossAccountSharingConfiguration**.

For more information, see [CloudWatch cross-account observability](#). The contents are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "oam:*"
      ],
      "Resource": "*"
    }
  ]
}
```

OAMReadOnlyAccess

The **OAMReadOnlyAccess** policy grants read-only access to Observability Access Manager resources, which are used for CloudWatch cross-account observability. For more information, see [CloudWatch cross-account observability](#). The contents are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "oam:Get*",
        "oam:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed (predefined) policies for CloudWatch Application Signals

The policies in this section grant permissions related to CloudWatch Application Signals. For more information, see [Application Signals](#).

CloudWatchApplicationSignalsReadOnlyAccess

AWS has added the **CloudWatchApplicationSignalsReadOnlyAccess** managed IAM policy. This policy grants read only access to actions and resources available to users in the CloudWatch console under Application Signals. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It includes an `iam:GetRole` policy to allow users to retrieve information about an IAM role. It includes `logs:` policies to start and stop queries, retrieve the configuration for a metruc filter, and obtain query results. It includes `cloudwatch:` policies so that users can obtain information about a CloudWatch alarm or metrics. It includes `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data, and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries.

The following are the contents of the **CloudWatchApplicationSignalsReadOnlyAccess** policy.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsReadOnlyAccessPermissions",
      "Effect": "Allow",
      "Action": [
        "application-signals:BatchGetServiceLevelObjectiveBudgetReport",
        "application-signals:GetService",
        "application-signals:GetServiceLevelObjective",
        "application-signals:ListServiceLevelObjectives",
        "application-signals:ListServiceDependencies",
        "application-signals:ListServiceDependents",
        "application-signals:ListServiceOperations",
        "application-signals:ListServices",
        "application-signals:ListTagsForResource"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
      "Effect": "Allow",
      "Action": "iam:GetRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
      "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:StartQuery"
      ],
      "Resource": "arn:aws:logs::*:log-group:/aws/application-signals/data:*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsLogsPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:StopQuery",
        "logs:GetQueryResults"
      ],
      "Resource": "*"
    }
  ],
}

```

```
{
  "Sid": "CloudWatchApplicationSignalsAlarmsReadPermissions",
  "Effect": "Allow",
  "Action": [
    "cloudwatch:DescribeAlarms"
  ],
  "Resource": "*"
},
{
  "Sid": "CloudWatchApplicationSignalsMetricsReadPermissions",
  "Effect": "Allow",
  "Action": [
    "cloudwatch:GetMetricData",
    "cloudwatch:ListMetrics"
  ],
  "Resource": "*"
},
{
  "Sid": "CloudWatchApplicationSignalsSyntheticsReadPermissions",
  "Effect": "Allow",
  "Action": [
    "synthetics:DescribeCanaries",
    "synthetics:DescribeCanariesLastRun",
    "synthetics:GetCanaryRuns"
  ],
  "Resource": "*"
},
{
  "Sid": "CloudWatchApplicationSignalsRumReadPermissions",
  "Effect": "Allow",
  "Action": [
    "rum:BatchGetRumMetricDefinitions",
    "rum:GetAppMonitor",
    "rum:GetAppMonitorData",
    "rum:ListAppMonitors"
  ],
  "Resource": "*"
},
{
  "Sid": "CloudWatchApplicationSignalsXrayReadPermissions",
  "Effect": "Allow",
  "Action": [
    "xray:GetTraceSummaries"
  ],
}
```

```
"Resource": "*"
}
]
}
```

CloudWatchApplicationSignalsFullAccess

AWS has added the **CloudWatchApplicationSignalsFullAccess** managed IAM policy. This policy grants access to all actions and resources available to users in the CloudWatch console. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It uses `cloudwatch:` policies to retrieve data from metrics and alarms. It uses `logs:` policies to manage queries and filters. It uses `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries. It includes `arn:aws:cloudwatch:*:*:alarm:` policies so that users can retrieve information about a service level objective (SLO) alarm. It includes `iam:` policies to manage IAM roles. It uses `sns:` policies to create, list, and subscribe to an Amazon SNS topic.

The following are the contents of the **CloudWatchApplicationSignalsFullAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchApplicationSignalsFullAccessPermissions",
      "Effect": "Allow",
      "Action": "application-signals:*",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsAlarmsPermissions",
      "Effect": "Allow",
      "Action": "cloudwatch:DescribeAlarms",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsMetricsPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics"
      ]
    }
  ]
}
```

```
    ],
    "Resource": "*"
  },
  {
    "Sid": "CloudWatchApplicationSignalsLogGroupPermissions",
    "Effect": "Allow",
    "Action": [
      "logs:StartQuery"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
  },
  {
    "Sid": "CloudWatchApplicationSignalsLogsPermissions",
    "Effect": "Allow",
    "Action": [
      "logs:StopQuery",
      "logs:GetQueryResults"
    ],
    "Resource": "*"
  },
  {
    "Sid": "CloudWatchApplicationSignalsSyntheticsPermissions",
    "Effect": "Allow",
    "Action": [
      "synthetics:DescribeCanaries",
      "synthetics:DescribeCanariesLastRun",
      "synthetics:GetCanaryRuns"
    ],
    "Resource": "*"
  },
  {
    "Sid": "CloudWatchApplicationSignalsRumPermissions",
    "Effect": "Allow",
    "Action": [
      "rum:BatchCreateRumMetricDefinitions",
      "rum:BatchDeleteRumMetricDefinitions",
      "rum:BatchGetRumMetricDefinitions",
      "rum:GetAppMonitor",
      "rum:GetAppMonitorData",
      "rum:ListAppMonitors",
      "rum:PutRumMetricsDestination",
      "rum:UpdateRumMetricDefinition"
    ],
    "Resource": "*"
  }
}
```



```

    },
    {
      "Sid": "CloudWatchApplicationSignalsXrayPermissions",
      "Effect": "Allow",
      "Action": "xray:GetTraceSummaries",
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsPutMetricAlarmPermissions",
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricAlarm",
      "Resource": [
        "arn:aws:cloudwatch:*:*:alarm:SLO-AttainmentGoalAlarm-*",
        "arn:aws:cloudwatch:*:*:alarm:SLO-WarningAlarm-*",
        "arn:aws:cloudwatch:*:*:alarm:SLI-HealthAlarm-*"
      ]
    },
    {
      "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam:*:*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
      "Effect": "Allow",
      "Action": "iam:GetRole",
      "Resource": "arn:aws:iam:*:*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
      "Sid": "CloudWatchApplicationSignalsSnsWritePermissions",
      "Effect": "Allow",
      "Action": [
        "sns:CreateTopic",
        "sns:Subscribe"
      ],
      "Resource": "arn:aws:sns:*:*:cloudwatch-application-signals-*"
    }
  ]
}

```

```

    },
    {
      "Sid": "CloudWatchApplicationSignalsSnsReadPermissions",
      "Effect": "Allow",
      "Action": "sns:ListTopics",
      "Resource": "*"
    }
  ]
}

```

AWS managed (predefined) policies for CloudWatch Synthetics

The **CloudWatchSyntheticsFullAccess** and **CloudWatchSyntheticsReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch Synthetics. The following additional policies are also relevant:

- **AmazonS3ReadOnlyAccess** and **CloudWatchReadOnlyAccess** – These are necessary to be able to read all Synthetics data in the CloudWatch console.
- **AWSLambdaReadOnlyAccess** – To be able to view the source code used by canaries.
- **CloudWatchSyntheticsFullAccess** enables you to create canaries. Additionally, to create and delete canaries that have a new IAM role created for them, you also need the following inline policy statement:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:CreatePolicy",
        "iam>DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
      ],
      "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
        "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
      ]
    }
  ]
}

```

```
]
}
```

Important

Granting a user the `iam:CreateRole`, `iam>DeleteRole`, `iam:CreatePolicy`, `iam>DeletePolicy`, `iam:AttachRolePolicy`, and `iam:DetachRolePolicy` permissions gives that user full administrative access to create, attach, and delete roles and policies that have ARNs that match `arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*` and `arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*`. For example, a user with these permissions can create a policy that has full permissions for all resources, and attach that policy to any role that matches that ARN pattern. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see [Changing Permissions for an IAM User](#) and [To embed an inline policy for a user or role](#).

CloudWatchSyntheticsFullAccess

The following is the content of the **CloudWatchSyntheticsFullAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "synthetics:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:PutEncryptionConfiguration"
      ],
      "Resource": [
```

```

        "arn:aws:s3:::cw-syn-results-*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "iam:ListRoles",
        "s3:ListAllMyBuckets",
        "xray:GetTraceSummaries",
        "xray:BatchGetTraces",
        "apigateway:GET"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation"
    ],
    "Resource": "arn:aws:s3:::*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:ListBucket"
    ],
    "Resource": "arn:aws:s3:::cw-syn-*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObjectVersion"
    ],
    "Resource": "arn:aws:s3:::aws-synthetics-library-*"
},
{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ],
}

```

```

    "Condition": {
      "StringEquals": {
        "iam:PassedToService": [
          "lambda.amazonaws.com",
          "synthetics.amazonaws.com"
        ]
      }
    },
  ],
  {
    "Effect": "Allow",
    "Action": [
      "iam:GetRole",
      "iam:ListAttachedRolePolicies"
    ],
    "Resource": [
      "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetMetricData",
      "cloudwatch:GetMetricStatistics"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricAlarm",
      "cloudwatch>DeleteAlarms"
    ],
    "Resource": [
      "arn:aws:cloudwatch::*:alarm:Synthetics-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:DescribeAlarms"
    ],
    "Resource": [
      "arn:aws:cloudwatch::*:alarm:*"
    ]
  }

```

```

    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "lambda:CreateFunction",
      "lambda:AddPermission",
      "lambda:PublishVersion",
      "lambda:UpdateFunctionCode",
      "lambda:UpdateFunctionConfiguration",
      "lambda:GetFunctionConfiguration",
      "lambda>DeleteFunction"
    ],
    "Resource": [
      "arn:aws:lambda:*:*:function:cwsyn-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "lambda:GetLayerVersion",
      "lambda:PublishLayerVersion",
      "lambda>DeleteLayerVersion"
    ],
    "Resource": [
      "arn:aws:lambda:*:*:layer:cwsyn-*",
      "arn:aws:lambda:*:*:layer:Synthetics:*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeVpcs",
      "ec2:DescribeSubnets",
      "ec2:DescribeSecurityGroups"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "sns:ListTopics"
    ]
  }

```

```
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "sns:CreateTopic",
      "sns:Subscribe",
      "sns:ListSubscriptionsByTopic"
    ],
    "Resource": [
      "arn:*:sns:*:*:Synthetics-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:ListAliases"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:DescribeKey"
    ],
    "Resource": "arn:aws:kms:*:*:key/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:*:*:key/*",
    "Condition": {
      "StringLike": {
        "kms:ViaService": [
          "s3.*.amazonaws.com"
        ]
      }
    }
  }
}
```

```
]
}
```

CloudWatchSyntheticsReadOnlyAccess

The following is the content of the **CloudWatchSyntheticsReadOnlyAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "synthetics:Describe*",
        "synthetics:Get*",
        "synthetics:List*",
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed (predefined) policies for Amazon CloudWatch RUM

The **AmazonCloudWatchRUMFullAccess** and **AmazonCloudWatchRUMReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch RUM.

AmazonCloudWatchRUMFullAccess

The following are the contents of the **AmazonCloudWatchRUMFullAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rum:*"
      ],
      "Resource": "*"
    },
    {
```



```
    "Effect": "Allow",
    "Action": [
      "iam:GetRole",
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/aws-service-role/rum.amazonaws.com/
AWSServiceRoleForRealUserMonitoring"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/RUM-Monitor*"
    ],
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": [
          "cognito-identity.amazonaws.com"
        ]
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetMetricData",
      "cloudwatch:GetMetricStatistics",
      "cloudwatch:ListMetrics"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "cloudwatch:DescribeAlarms"
    ],
    "Resource": "arn:aws:cloudwatch::*:alarm:*"
  },
  {
    "Effect": "Allow",
```

```

    "Action": [
      "cognito-identity:CreateIdentityPool",
      "cognito-identity:ListIdentityPools",
      "cognito-identity:DescribeIdentityPool",
      "cognito-identity:GetIdentityPoolRoles",
      "cognito-identity:SetIdentityPoolRoles"
    ],
    "Resource": "arn:aws:cognito-identity:*:*:identitypool/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs>DeleteLogGroup",
      "logs:PutRetentionPolicy",
      "logs>CreateLogStream"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:*RUMService*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogDelivery",
      "logs:GetLogDelivery",
      "logs:UpdateLogDelivery",
      "logs>DeleteLogDelivery",
      "logs:ListLogDeliveries",
      "logs:DescribeResourcePolicies"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:DescribeLogGroups"
    ],
    "Resource": "arn:aws:logs:*:*:log-group::log-stream:*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "synthetics:describeCanaries",
      "synthetics:describeCanariesLastRun"
    ],
  },

```

```

        "Resource": "arn:aws:synthetics:*:*:canary:*"
    }
]
}

```

AmazonCloudWatchRUMReadOnlyAccess

The following are the contents of the **AmazonCloudWatchRUMReadOnlyAccess** policy.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rum:GetAppMonitor",
        "rum:GetAppMonitorData",
        "rum:ListAppMonitors",
        "rum:ListRumMetricsDestinations",
        "rum:BatchGetRumMetricDefinitions"
      ],
      "Resource": "*"
    }
  ]
}

```

AmazonCloudWatchRUMServiceRolePolicy

You can't attach **AmazonCloudWatchRUMServiceRolePolicy** to your IAM entities. This policy is attached to a service-linked role that allows CloudWatch RUM to publish monitoring data to other relevant AWS services. For more information about this service linked role, see [Using service-linked roles for CloudWatch RUM](#).

The complete contents of **AmazonCloudWatchRUMServiceRolePolicy** are as follows.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments"
      ]
    }
  ]
}

```

```

    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": "cloudwatch:PutMetricData",
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "cloudwatch:namespace": [
          "RUM/CustomMetrics/*",
          "AWS/RUM"
        ]
      }
    }
  }
]
}

```

AWS managed (predefined) policies for CloudWatch Evidently

The **CloudWatchEvidentlyFullAccess** and **CloudWatchEvidentlyReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch Evidently.

CloudWatchEvidentlyFullAccess

The following are the contents of the **CloudWatchEvidentlyFullAccess** policy.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "evidently:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [

```

```

        "iam:ListRoles"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "iam:GetRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/service-role/CloudWatchRUMevidentlyRole-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation",
        "s3:ListAllMyBuckets"
    ],
    "Resource": "arn:aws:s3:::*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:ListTagsForResource"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:DescribeAlarms",
        "cloudwatch:TagResource",
        "cloudwatch:UntagResource"
    ],
    "Resource": [
        "arn:aws:cloudwatch::*:alarm:*"
    ]
},
{

```

```
    "Effect": "Allow",
    "Action": [
        "cloudtrail:LookupEvents"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricAlarm"
    ],
    "Resource": [
        "arn:aws:cloudwatch:*:*:alarm:Evidently-Alarm-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns:ListTopics"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "sns:CreateTopic",
        "sns:Subscribe",
        "sns:ListSubscriptionsByTopic"
    ],
    "Resource": [
        "arn:*:sns:*:*:Evidently-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "*"
    ]
}
```

```
]
}
```

CloudWatchEvidentlyReadOnlyAccess

The following are the contents of the **CloudWatchEvidentlyReadOnlyAccess** policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "evidently:GetExperiment",
        "evidently:GetFeature",
        "evidently:GetLaunch",
        "evidently:GetProject",
        "evidently:GetSegment",
        "evidently:ListExperiments",
        "evidently:ListFeatures",
        "evidently:ListLaunches",
        "evidently:ListProjects",
        "evidently:ListSegments",
        "evidently:ListSegmentReferencs"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy for AWS Systems Manager Incident Manager

The **AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy** policy is attached to a service-linked role that allows CloudWatch to start incidents in AWS Systems Manager Incident Manager on your behalf. For more information, see [Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions](#).

The policy has the following permission:

- `ssm-incidents:StartIncident`

Customer managed policy examples

In this section, you can find example user policies that grant permissions for various CloudWatch actions. These policies work when you are using the CloudWatch API, AWS SDKs, or the AWS CLI.

Examples

- [Example 1: Allow user full access to CloudWatch](#)
- [Example 2: Allow read-only access to CloudWatch](#)
- [Example 3: Stop or terminate an Amazon EC2 instance](#)

Example 1: Allow user full access to CloudWatch

To grant a user full access to CloudWatch, you can use grant them the **CloudWatchFullAccess** managed policy instead of creating a customer-managed policy. The contents of the **CloudWatchFullAccess** are listed in [CloudWatchFullAccess](#).

Example 2: Allow read-only access to CloudWatch

The following policy allows a user read-only access to CloudWatch and view Amazon EC2 Auto Scaling actions, CloudWatch metrics, CloudWatch Logs data, and alarm-related Amazon SNS data.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
        "logs:Get*",
        "logs:Describe*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "sns:Get*"
      ]
    }
  ]
}
```



```
    "sns:List*"
  ],
  "Effect": "Allow",
  "Resource": "*"
}
]
```

Example 3: Stop or terminate an Amazon EC2 instance

The following policy allows an CloudWatch alarm action to stop or terminate an EC2 instance. In the sample below, the `GetMetricData`, `ListMetrics`, and `DescribeAlarms` actions are optional. It is recommended that you include these actions to ensure that you have correctly stopped or terminated the instance.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

}

CloudWatch updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
CloudWatchApplicationSignalReadOnlyAccess – New policy	<p>CloudWatch created a new policy named CloudWatchApplicationSignalReadOnlyAccess.</p> <p>This policy grants read only access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes <code>application-signal</code> policies so that users can use read only actions and resources available in the CloudWatch console under Application Signals. It contains an <code>iam</code> policy to manage IAM roles. It includes some <code>logs</code> policies to manage log queries and filters. It includes <code>cloudwatch</code> policies to retrieve information on CloudWatch alarms and metrics. It includes some <code>synthetic</code> policies to retrieve</p>	June 7, 2024

Change	Description	Date
	information about synthetics canaries. It includes <code>rum:</code> policies to manage RUM clients and jobs. It contains an <code>xray:</code> policy to obtain trace summaries.	

Change	Description	Date
CloudWatchApplicationSignalsFullAccess – New policy	<p>CloudWatch created a new policy named CloudWatchApplicationSignalsFullAccess.</p> <p>This policy grants full access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes application-signals: so that users can use Application Signals actions and resources. It includes some cloudwatch: policies to retrieve information on CloudWatch alarms and metrics. It includes some logs: policies to manage log queries. It includes some synthetics: policies to write and retrieve information about synthetics canaries. It includes rum: policies to manage RUM clients and jobs. It contains an xray: policy to obtain trace summaries. It includes some cloudwatch: policies to manage CloudWatch alarms. It contains some iam: policies to manage IAM roles. It includes some sns: policies to manage Amazon Simple</p>	June 7, 2024

Change	Description	Date
	Notification Service notifications.	
CloudWatchFullAccessV2 – Update to an existing policy	<p>CloudWatch updated the policy named CloudWatchFullAccessV2.</p> <p>The scope of the <code>CloudWatchFullAccessPermissions</code> policy was updated to add <code>application-signals:*</code> so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.</p>	May 20, 2024

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	<p>CloudWatch updated the policy named CloudWatchReadOnlyAccess.</p> <p>The scope of the <code>CloudWatchReadOnlyAccessPermissions</code> policy was updated to add <code>application-signals:BatchGet*</code>, <code>application-signals:List*</code>, and <code>application-signals:Get*</code> so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.</p> <p>The scope of <code>CloudWatchReadOnlyGetRolePermissions</code> was updated to add the <code>iam:GetRole</code> action so that users can check if CloudWatch Application Signals is set up.</p>	May 20, 2024

Change	Description	Date
<p>CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy</p>	<p>CloudWatch updated the policy named CloudWatchApplicationSignalsServiceRolePolicy.</p> <p>The scoping of the <code>logs:StartQuery</code> and <code>logs:GetQueryResults</code> permissions was changed to add the <code>arn:aws:logs:*:*:log-group:/aws/appsignals/*:*</code> and <code>arn:aws:logs:*:*:log-group:/aws/application-signals/data:*</code> ARNs to enable Application Signals on more architectures.</p>	<p>April 18, 2024</p>
<p>CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy</p>	<p>CloudWatch changed the scope of a permission in CloudWatchApplicationSignalsServiceRolePolicy.</p> <p>The scope of the <code>cloudwatch:GetMetricData</code> permission was changed to <code>*</code> so that Application Signals can retrieve metrics from sources in linked accounts.</p>	<p>April 08, 2024</p>

Change	Description	Date
CloudWatchAgentServerPolicy – Update to an existing policy	<p>CloudWatch added permissions to CloudWatchAgentServerPolicy.</p> <p>The <code>xray:PutTraceSegments</code> , <code>xray:PutTelemetryRecords</code> , <code>xray:GetSamplingRules</code> , <code>xray:GetSamplingTargets</code> , <code>xray:GetSamplingStatisticSummaries</code> and <code>logs:PutRetentionPolicy</code> permissions were added so that the CloudWatch agent can publish X-Ray traces and modify log group retention periods.</p>	February 12, 2024

Change	Description	Date
CloudWatchAgentAdminPolicy – Update to an existing policy	<p>CloudWatch added permissions to CloudWatchAgentAdminPolicy.</p> <p>The <code>xray:PutTraceSegments</code> , <code>xray:PutTelemetryRecords</code> , <code>xray:GetSamplingRules</code> , <code>xray:GetSamplingTargets</code> , <code>xray:GetSamplingStatisticSummaries</code> and <code>logs:PutRetentionPolicy</code> permissions were added so that the CloudWatch agent can publish X-Ray traces and modify log group retention periods.</p>	February 12, 2024

Change	Description	Date
<p>CloudWatchFullAccessV2 – Update to an existing policy</p>	<p>CloudWatch added permissions to CloudWatchFullAccessV2.</p> <p>Existing permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new permissions for CloudWatch Application Signals were added so that users with this policy can manage CloudWatch Application Signals.</p> <p>The permission to create the CloudWatch Application Signals service-linked role was added to allow CloudWatch Application Signals to discover telemetry data in logs, metrics, traces, and tags.</p>	<p>December 5, 2023</p>

Change	Description	Date
<p>CloudWatchReadOnlyAccess – Update to an existing policy</p>	<p>CloudWatch added permissions to CloudWatchReadOnlyAccess.</p> <p>Existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new read-only permissions for CloudWatch Application Signals were added so that users with this policy can triage and diagnose their service health issues as reported by CloudWatch Application Signals.</p> <p>The <code>cloudwatch:GenerateQuery</code> permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.</p>	<p>December 5, 2023</p>
<p>CloudWatchReadOnlyAccess – Update to an existing policy.</p>	<p>CloudWatch added a permission to CloudWatchReadOnlyAccess.</p> <p>The <code>cloudwatch:GenerateQuery</code> permission was added, so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.</p>	<p>December 01, 2023</p>

Change	Description	Date
CloudWatchApplicationSignalsServiceRolePolicy – New policy	<p>CloudWatch added a new policy CloudWatchApplicationSignalsServiceRolePolicy.</p> <p>The CloudWatchApplicationSignalsServiceRolePolicy grants an upcoming feature permissions to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data.</p>	November 9, 2023
AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy – New policy	<p>CloudWatch added a new policy AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy.</p> <p>The AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy grants permission to CloudWatch to fetch Performance Insights metrics from databases on your behalf.</p>	September 20, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	<p>CloudWatch added a permission to CloudWatchReadOnlyAccess.</p> <p>The <code>application-autoscaling:DescribeScalingPolicies</code> permission was added so that users with this policy can access information about Application Auto Scaling policies.</p>	September 14, 2023
CloudWatchFullAccessV2 – New policy	<p>CloudWatch added a new policy CloudWatchFullAccessV2.</p> <p>The CloudWatchFullAccessV2 grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling. For more information, see CloudWatchFullAccessV2.</p>	August 1, 2023

Change	Description	Date
<p>AWSServiceRoleForInternetMonitor – Update to an existing policy</p>	<p>Amazon CloudWatch Internet Monitor added new permissions to monitor Network Load Balancer resources.</p> <p>The <code>elasticloadbalancing:DescribeLoadBalancers</code> and <code>ec2:DescribeNetworkInterfaces</code> permissions are required so that Internet Monitor can monitor customers' Network Load Balancer traffic by analyzing flow logs for NLB resources.</p> <p>For more information, see Using Amazon CloudWatch Internet Monitor.</p>	<p>July 15, 2023</p>
<p>CloudWatchReadOnlyAccess – Update to an existing policy</p>	<p>CloudWatch added permissions to CloudWatchReadOnlyAccess.</p> <p>The <code>logs:StartLiveTail</code> and <code>logs:StopLiveTail</code> permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.</p>	<p>June 6, 2023</p>

Change	Description	Date
CloudWatchCrossAccountSharingConfiguration – New policy	<p>CloudWatch added a new policy to enable you to manage CloudWatch cross-account observability links that share CloudWatch metrics.</p> <p>For more information, see CloudWatch cross-account observability.</p>	November 27, 2022
OAMFullAccess – New policy	<p>CloudWatch added a new policy to enable you to fully manage CloudWatch cross-account observability links and sinks.</p> <p>For more information, see CloudWatch cross-account observability.</p>	November 27, 2022
OAMReadOnlyAccess – New policy	<p>CloudWatch added a new policy to enable you to view information about CloudWatch cross-account observability links and sinks.</p> <p>For more information, see CloudWatch cross-account observability.</p>	November 27, 2022

Change	Description	Date
CloudWatchFullAccess – Update to an existing policy	CloudWatch added permissions to CloudWatchFullAccess . The <code>oam:ListSinks</code> and <code>oam:ListAttachedLinks</code> permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to CloudWatchReadOnlyAccess . The <code>oam:ListSinks</code> and <code>oam:ListAttachedLinks</code> permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022

Change	Description	Date
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	<p>CloudWatch RUM updated a condition key in AmazonCloudWatchRUMServiceRolePolicy.</p> <p>The "Condition": { "StringEquals": { "cloudwatch:namespace": "AWS/RUM" } } condition key was changed to the following so that CloudWatch RUM can send custom metrics to custom metric namespaces.</p> <pre>"Condition": { "StringLike": { "cloudwatch:namespace": ["RUM/CustomMetrics/*", "AWS/RUM"] } }</pre>	February 2, 2023

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	<p>CloudWatch added permissions the AmazonCloudWatchRUMReadOnlyAccess policy.</p> <p>The <code>rum:ListRumMetricsDestinations</code> and <code>rum:BatchGetRumMetricsDefinitions</code> permissions were added so that CloudWatch RUM can send extended metrics to CloudWatch and Evidently.</p>	October 27, 2022
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	<p>CloudWatch RUM added permissions to AmazonCloudWatchRUMServiceRolePolicy.</p> <p>The <code>cloudwatch:PutMetricData</code> permission was added so that CloudWatch RUM can send extended metrics to CloudWatch.</p>	October 26, 2022

Change	Description	Date
<p>CloudWatchEvidentlyReadOnlyAccess – Update to an existing policy</p>	<p>CloudWatch Evidently added permissions to CloudWatchEvidentlyReadOnlyAccess.</p> <p>The <code>evidently:GetSegment</code> , <code>evidently:ListSegments</code> , and <code>evidently:ListSegmentReferences</code> permissions were added so that users with this policy can see Evidently audience segments that have been created.</p>	<p>August 12, 2022</p>
<p>CloudWatchSyntheticsFullAccess – Update to an existing policy</p>	<p>CloudWatch Synthetics added permissions to CloudWatchSyntheticsFullAccess.</p> <p>The <code>lambda:DeleteFunction</code> and <code>lambda:DeleteLayerVersion</code> permissions were added so that CloudWatch Synthetics can delete related resources when a canary is deleted.</p> <p>The <code>iam:ListAttachedRolePolicies</code> was added so that customers can view the policies that are attached to a canary's IAM role.</p>	<p>May 6, 2022</p>

Change	Description	Date
AmazonCloudWatchRUMFullAccess – New policy	<p>CloudWatch added a new policy to enable full management of CloudWatch RUM.</p> <p>CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.</p>	November 29, 2021
AmazonCloudWatchRUMReadOnlyAccess – New policy	<p>CloudWatch added a new policy to enable read-only access to CloudWatch RUM.</p> <p>CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.</p>	November 29, 2021
CloudWatchEvidentlyFullAccess – New policy	<p>CloudWatch added a new policy to enable full management of CloudWatch Evidently.</p> <p>CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently.</p>	November 29, 2021

Change	Description	Date
CloudWatchEvidentlyReadOnlyAccess – New policy	<p>CloudWatch added a new policy to enable read-only access to CloudWatch Evidently.</p> <p>CloudWatch Evidently allows you to perform A/B experiments of your web applications, and to roll them out gradually. For more information, see Perform launches and A/B experiments with CloudWatch Evidently.</p>	November 29, 2021
AWSServiceRoleForCloudWatchRUM – New managed policy	CloudWatch added a policy for a new service-linked role to allow CloudWatch RUM to publish monitoring data to other relevant AWS services.	November 29, 2021

Change	Description	Date
<p>CloudWatchSyntheticsFullAccess – Update to an existing policy</p>	<p>CloudWatch Synthetics added permissions to CloudWatchSyntheticsFullAccess, and also changed the scope of one permission.</p> <p>The <code>kms:ListAliases</code> permission was added so that users can list available AWS KMS keys that can be used to encrypt canary artifacts. The <code>kms:DescribeKey</code> permission was added so that users can see the details of keys that will be used to encrypt for canary artifacts. And the <code>kms:Decrypt</code> permission was added to enable users to decrypt canary artifacts. This decryption ability is limited to use on resources within Amazon S3 buckets.</p> <p>The Resource scope of the <code>s3:GetBucketLocation</code> permission was changed from <code>*</code> to <code>arn:aws:s3:::*</code>.</p>	<p>September 29, 2021</p>

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	<p>CloudWatch Synthetics added a permission to CloudWatchSyntheticsFullAccess.</p> <p>The <code>lambda:UpdateFunctionCode</code> permission was added so that users with this policy can change the runtime version of canaries.</p>	July 20, 2021
AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy – New managed policy	<p>CloudWatch added a new managed IAM policy to allow CloudWatch to create incidents in AWS Systems Manager Incident Manager.</p>	May 10, 2021
CloudWatchAutomaticDashboardsAccess – Update to an existing policy	<p>CloudWatch added a permission to the CloudWatchAutomaticDashboardsAccess managed policy. The <code>synthetics:DescribeCanariesLastRun</code> permission was added to this policy to enable cross-account dashboard users to see details about CloudWatch Synthetics canary runs.</p>	April 20, 2021
CloudWatch started tracking changes	<p>CloudWatch started tracking changes for its AWS managed policies.</p>	April 14, 2021

Using condition keys to limit access to CloudWatch namespaces

Use IAM condition keys to limit users to publishing metrics only in the CloudWatch namespaces that you specify.

Allowing publishing in one namespace only

The following policy limits the user to publishing metrics only in the namespace named `MyCustomNamespace`.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Resource": "*",
    "Action": "cloudwatch:PutMetricData",
    "Condition": {
      "StringEquals": {
        "cloudwatch:namespace": "MyCustomNamespace"
      }
    }
  }
}
```

Excluding publishing from a namespace

The following policy allows the user to publish metrics in any namespace except for `CustomNamespace2`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": "*",
      "Action": "cloudwatch:PutMetricData"
    },
    {
      "Effect": "Deny",
      "Resource": "*",
      "Action": "cloudwatch:PutMetricData",
      "Condition": {
```



```
        "StringEquals": {
            "cloudwatch:namespace": "CustomNamespace2"
        }
    }
}
]
```

Using condition keys to limit Contributor Insights users' access to log groups

To create a rule in Contributor Insights and see its results, a user must have the `cloudwatch:PutInsightRule` permission. By default, a user with this permission can create a Contributor Insights rule that evaluates any log group in CloudWatch Logs and then see the results. The results can contain contributor data for those log groups.

You can create IAM policies with condition keys to grant users the permission to write Contributor Insights rules for some log groups while preventing them from writing rules for and seeing this data from other log groups.

For more information about the `Condition` element in IAM policies, see [IAM JSON policy elements: Condition](#).

Allow access to write rules and view results for only certain log groups

The following policy allows the user access to write rules and view results for the log group named `AllowedLogGroup` and all log groups that have names that start with `AllowedWildcard`. It does not grant access to write rules or view rule results for any other log groups.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCertainLogGroups",
      "Effect": "Allow",
      "Action": "cloudwatch:PutInsightRule",
      "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
      "Condition": {
        "ForAllValues:StringEqualsIgnoreCase": {
          "cloudwatch:requestInsightRuleLogGroups": [
```


Using condition keys to limit alarm actions

When CloudWatch alarms change state, they can perform different actions such as stopping and terminating EC2 instances and performing Systems Manager actions. These actions can be initiated when the alarm changes to any state, including ALARM, OK, or INSUFFICIENT_DATA.

Use the `cloudwatch:AlarmActions` condition key to allow a user to create alarms that can only perform the actions you specify when the alarm state changes. For example, you can allow a user to create alarms that can only perform actions which are not EC2 actions.

Allow a user to create alarms that can only send Amazon SNS notifications or perform Systems Manager actions

The following policy limits the user to creating alarms that can only send Amazon SNS notifications and perform Systems Manager actions. The user can't create alarms that perform EC2 actions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricAlarm",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringLike": {
          "cloudwatch:AlarmActions": [
            "arn:aws:sns:*",
            "arn:aws:ssm:*"
          ]
        }
      }
    }
  ]
}
```

Using service-linked roles for CloudWatch

Amazon CloudWatch uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to CloudWatch. Service-linked

roles are predefined by CloudWatch and include all the permissions that the service requires to call other AWS services on your behalf.

One service-linked role in CloudWatch makes setting up CloudWatch alarms that can terminate, stop, or reboot an Amazon EC2 instance without requiring you to manually add the necessary permissions. Another service-linked role enables a monitoring account to access CloudWatch data from other accounts that you specify, to build cross-account cross-Region dashboards.

CloudWatch defines the permissions of these service-linked roles, and unless defined otherwise, only CloudWatch can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This restriction protects your CloudWatch resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch alarms EC2 actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchEvents** – CloudWatch uses this service-linked role to perform Amazon EC2 alarm actions.

The **AWSServiceRoleForCloudWatchEvents** service-linked role trusts the CloudWatch Events service to assume the role. CloudWatch Events invokes the terminate, stop, or reboot instance actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchEvents** service-linked role permissions policy allows CloudWatch Events to complete the following actions on Amazon EC2 instances:

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`
- `ec2:DescribeInstanceRecoveryAttribute`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceStatus`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

Service-linked role permissions for CloudWatch Application Signals

CloudWatch Application Signals uses the service-linked role named **AWSServiceRoleForCloudWatchApplicationSignals** – CloudWatch uses this service-linked role to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.

The **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role trusts CloudWatch Application Signals to assume the role. Application Signals collects the logs, traces, metrics, and tags data from your account.

The **AWSServiceRoleForCloudWatchApplicationSignals** has an IAM policy attached, and this policy is named **CloudWatchApplicationSignalsServiceRolePolicy**. This policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant AWS services. It includes permissions for Application Signals to complete the following actions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

The complete contents of **CloudWatchApplicationSignalsServiceRolePolicy** are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "XRayPermission",
      "Effect": "Allow",
      "Action": [
        "xray:GetServiceGraph"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "CWLogsPermission",
    "Effect": "Allow",
    "Action": [
        "logs:StartQuery",
        "logs:GetQueryResults"
    ],
    "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/appsignals/*:*",
        "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "CWListMetricsPermission",
    "Effect": "Allow",
    "Action": [
        "cloudwatch:ListMetrics"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "CWGetMetricDataPermission",

```

```

    "Effect": "Allow",
    "Action": [
      "cloudwatch:GetMetricData"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Sid": "TagsPermission",
    "Effect": "Allow",
    "Action": [
      "tag:GetResources"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:ResourceAccount": "${aws:PrincipalAccount}"
      }
    }
  }
]
}

```

Service-linked role permissions for CloudWatch alarms Systems Manager OpsCenter actions

CloudWatch uses the service-linked role named

AWSServiceRoleForCloudWatchAlarms_ActionSSM – CloudWatch uses this service-linked role to perform Systems Manager OpsCenter actions when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager OpsCenter actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm:CreateOpsItem`

Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions

CloudWatch uses the service-linked role named

AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – CloudWatch uses this service-linked role to start Incident Manager incidents when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager Incident Manager action when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm-incidents:StartIncident`

Service-linked role permissions for CloudWatch cross-account cross-Region

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchCrossAccount** – CloudWatch uses this role to access CloudWatch data in other AWS accounts that you specify. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the sharing account. It is the sharing role that provides access to data.

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

The **AWSServiceRoleForCloudWatchCrossAccount** service-linked role trusts the CloudWatch service to assume the role.

Service-linked role permissions for CloudWatch database Performance Insights

CloudWatch uses the service-linked role named

AWSServiceRoleForCloudWatchMetrics_DbPerfInsights. – CloudWatch uses this role to retrieve Performance Insights metrics for creating alarms and snapshotting.

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role has the **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy** IAM policy attached. The contents of that policy are as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "pi:GetResourceMetrics"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    }
  ]
}
```

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role trusts the CloudWatch service to assume the role.

Creating a service-linked role for CloudWatch

You do not need to manually create any of these service-linked roles. The first time you create an alarm in the AWS Management Console, the IAM CLI, or the IAM API, CloudWatch creates **AWSServiceRoleForCloudWatchEvents** and **AWSServiceRoleForCloudWatchAlarms_ActionSSM** for you.

The first time that you enable service and topology discovery, Application Signals creates **AWSServiceRoleForCloudWatchApplicationSignals** for you.

When you first enable an account to be a monitoring account for cross-account cross-Region functionality, CloudWatch creates **AWSServiceRoleForCloudWatchCrossAccount** for you.

When you first create an alarm that uses the `DB_PERF_INSIGHTS` metric math function, CloudWatch creates **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** for you.

For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*.

Editing a service-linked role for CloudWatch

CloudWatch does not allow you to edit the **AWSServiceRoleForCloudWatchEvents**, **AWSServiceRoleForCloudWatchAlarms_ActionSSM**, **AWSServiceRoleForCloudWatchCrossAccount**, or **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** roles. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following commands:

```
$ aws iam get-role --role-name role-name
```

Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **myrole**.

2. To update a service-linked role's description, use the following command:

```
$ aws iam update-role-description --role-name role-name --description description
```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

[GetRole](#)

2. To update a role's description, use the following command:

[UpdateRoleDescription](#)

Deleting a service-linked role for CloudWatch

If you no longer have alarms that automatically stop, terminate, or reboot EC2 instances, we recommend that you delete the `AWSServiceRoleForCloudWatchEvents` role.

If you no longer have alarms that perform Systems Manager OpsCenter actions, we recommend that you delete the `AWSServiceRoleForCloudWatchAlarms_ActionSSM` role.

If you delete all alarms that use the `DB_PERF_INSIGHTS` metric math function, we recommend that you delete the `AWSServiceRoleForCloudWatchMetrics_DbPerfInsights` service-linked role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the `AWSServiceRoleForCloudWatchEvents` role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

Note

If you are unsure whether CloudWatch is using the `AWSServiceRoleForCloudWatchEvents` role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not

met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call [DeleteServiceLinkedRole](#). In the request, specify the role name that you want to delete.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.

2. To check the status of the deletion, call [GetServiceLinkedRoleDeletionStatus](#). In the request, specify the `DeletionTaskId`.

The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

CloudWatch updates to AWS service-linked roles

View details about updates to AWS managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [CloudWatch Document history](#) page.

Change	Description	Date
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	CloudWatch add more log groups to the scope of the <code>logs:StartQuery</code> and <code>logs:GetQueryResults</code> permissions granted by this role.	April 24, 2024
AWSServiceRoleForCloudWatchApplicationSignals – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch Application Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.	November 9, 2023
AWSServiceRoleForCloudWatchMetrics_DbPerformanceInsights – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming and snapshotting. An IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 13, 2023
AWSServiceRoleForCloudWatchAlarms	CloudWatch added a new service-linked role to allow	April 26, 2021

Change	Description	Date
ActionSSMIncidents – New service-linked role	CloudWatch to create incidents in AWS Systems Manager Incident Manager.	
CloudWatch started tracking changes	CloudWatch started tracking changes for its service-linked roles.	April 26, 2021

Using service-linked roles for CloudWatch RUM

CloudWatch RUM uses a AWS Identity and Access Management (IAM) [service-linked role](#). A service-linked role is a unique type of IAM role that is linked directly to RUM. The service-linked role is predefined by RUM and includes all the permissions that the service requires to call other AWS services on your behalf.

RUM defines the permissions of the service-linked role, and unless defined otherwise, only RUM can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your RUM resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for RUM

RUM uses the service-linked role named **AWSServiceRoleForCloudWatchRUM** – this role allows RUM to send AWS X-Ray trace data into your account, for app monitors that you enable X-Ray tracing for.

The **AWSServiceRoleForCloudWatchRUM** service-linked role trusts the X-Ray service to assume the role. X-Ray sends the trace data to your account.

The **AWSServiceRoleForCloudWatchRUM** service-linked role has an IAM policy attached named **AmazonCloudWatchRUMServiceRolePolicy**. This policy grants permission to CloudWatch RUM to

publish monitoring data to other relevant AWS services. It includes permissions that allow RUM to complete the following actions:

- `xray:PutTraceSegments`
- `cloudwatch:PutMetricData`

The complete contents of **AmazonCloudWatchRUMServiceRolePolicy** are as follows.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "cloudwatch:namespace": [
            "RUM/CustomMetrics/*",
            "AWS/RUM"
          ]
        }
      }
    }
  ]
}
```

Creating a service-linked role for RUM

You do not need to manually create the service-linked role for CloudWatch RUM. The first time that you create an app monitor with X-Ray tracing enabled, or update an app monitor to use X-Ray tracing, RUM creates **AWSServiceRoleForCloudWatchRUM** for you.

For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*.

Editing a service-linked role for RUM

CloudWatch RUM does not allow you to edit the **AWSServiceRoleForCloudWatchRUM** role. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following commands:

```
$ aws iam get-role --role-name role-name
```

Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **myrole**.

2. To update a service-linked role's description, use the following command:

```
$ aws iam update-role-description --role-name role-name --description description
```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

[GetRole](#)

2. To update a role's description, use the following command:

[UpdateRoleDescription](#)

Deleting a service-linked role for RUM

If you no longer have app monitors with X-Ray enabled, we recommend that you delete the **AWSServiceRoleForCloudWatchRUM** role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the **AWSServiceRoleForCloudWatchRUM** role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

Note

If you are unsure whether RUM is using the **AWSServiceRoleForCloudWatchRUM** role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you

must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call [DeleteServiceLinkedRole](#). In the request, specify the role name that you want to delete.

Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the DeletionTaskId from the response to check the status of the deletion task.

2. To check the status of the deletion, call [GetServiceLinkedRoleDeletionStatus](#). In the request, specify the DeletionTaskId.

The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Using service-linked roles for CloudWatch Application Insights

CloudWatch Application Insights uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to CloudWatch Application Insights. Service-linked roles are predefined by CloudWatch Application Insights and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CloudWatch Application Insights easier because you don't have to manually add the necessary permissions. CloudWatch Application Insights defines the permissions of its service-linked roles, and unless defined otherwise, only CloudWatch Application Insights can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch Application Insights

CloudWatch Application Insights uses the service-linked role named **AWSServiceRoleForApplicationInsights**. Application Insights uses this role to perform operations such as analyzing the resource groups of the customer, creating CloudFormation stacks to create alarms on metrics, and configuring the CloudWatch Agent on EC2 instances. This service-linked role has an IAM policy attached to it named `CloudwatchApplicationInsightsServiceLinkedRolePolicy`. For updates to this policy, see [Application Insights updates to AWS managed policies](#).

The role permissions policy allows CloudWatch Application Insights to complete the following actions on resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch>DeleteAlarms",
        "cloudwatch:PutAnomalyDetector",
        "cloudwatch>DeleteAnomalyDetector",
        "cloudwatch:DescribeAnomalyDetectors"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",

```

```
    "logs:DescribeLogStreams",
    "logs:DescribeLogGroups"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "events:DescribeRule"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "cloudFormation:CreateStack",
    "cloudFormation:UpdateStack",
    "cloudFormation>DeleteStack",
    "cloudFormation:DescribeStackResources"
  ],
  "Resource": [
    "arn:aws:cloudformation:*:*:stack/ApplicationInsights-*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "cloudFormation:DescribeStacks",
    "cloudFormation:ListStackResources",
    "cloudFormation:ListStacks"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "tag:GetResources"
  ],
  "Resource": [
    "*"
  ]
}
```

```
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "resource-groups:ListGroupResources",
      "resource-groups:GetGroupQuery",
      "resource-groups:GetGroup"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "resource-groups:CreateGroup",
      "resource-groups>DeleteGroup"
    ],
    "Resource": [
      "arn:aws:resource-groups:*:*:group/ApplicationInsights-*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "elasticloadbalancing:DescribeLoadBalancers",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeTargetHealth"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "autoscaling:DescribeAutoScalingGroups"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

```

    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:PutParameter",
        "ssm>DeleteParameter",
        "ssm:AddTagsToResource",
        "ssm:RemoveTagsFromResource",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-ApplicationInsights-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:CreateAssociation",
        "ssm:UpdateAssociation",
        "ssm>DeleteAssociation",
        "ssm:DescribeAssociation"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:association/*",
        "arn:aws:ssm:*:*:managed-instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetOpsItem",
        "ssm:CreateOpsItem",
        "ssm:DescribeOpsItems",
        "ssm:UpdateOpsItem",
        "ssm:DescribeInstanceInformation"
      ],
      "Resource": [
        "*"
      ]
    },
    {

```



```
    "Effect": "Allow",
    "Action": [
        "ssm:AddTagsToResource"
    ],
    "Resource": "arn:aws:ssm:*:*:opsitem/*"
},
{
    "Effect": "Allow",
    "Action": [
        "ssm:ListCommandInvocations",
        "ssm:GetCommandInvocation"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": "ssm:SendCommand",
    "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-CheckPerformanceCounterSets",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AWSEC2-DetectWorkload",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeNatGateways"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
```

```
    "rds:DescribeDBInstances",
    "rds:DescribeDBClusters"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "lambda:ListFunctions",
    "lambda:GetFunctionConfiguration",
    "lambda:ListEventSourceMappings"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "events:PutRule",
    "events:PutTargets",
    "events:RemoveTargets",
    "events>DeleteRule"
  ],
  "Resource": [
    "arn:aws:events:*:*:rule/AmazonCloudWatch-ApplicationInsights-*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "xray:GetServiceGraph",
    "xray:GetTraceSummaries",
    "xray:GetTimeSeriesServiceStatistics",
    "xray:GetTraceGraph"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
```

```
"Action": [
  "dynamodb:ListTables",
  "dynamodb:DescribeTable",
  "dynamodb:DescribeContributorInsights",
  "dynamodb:DescribeTimeToLive"
],
"Resource": [
  "*"
]
},
{
  "Effect": "Allow",
  "Action": [
    "application-autoscaling:DescribeScalableTargets"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "s3:ListAllMyBuckets",
    "s3:GetMetricsConfiguration",
    "s3:GetReplicationConfiguration"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "states:ListStateMachines",
    "states:DescribeExecution",
    "states:DescribeStateMachine",
    "states:GetExecutionHistory"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
```

```

    "Action": [
      "apigateway:GET"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "ecs:DescribeClusters",
      "ecs:DescribeContainerInstances",
      "ecs:DescribeServices",
      "ecs:DescribeTaskDefinition",
      "ecs:DescribeTasks",
      "ecs:DescribeTaskSets",
      "ecs:ListClusters",
      "ecs:ListContainerInstances",
      "ecs:ListServices",
      "ecs:ListTasks"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "ecs:UpdateClusterSettings"
    ],
    "Resource": [
      "arn:aws:ecs:*:*:cluster/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "eks:DescribeCluster",
      "eks:DescribeFargateProfile",
      "eks:DescribeNodegroup",
      "eks:ListClusters",
      "eks:ListFargateProfiles",
      "eks:ListNodegroups",
      "fsx:DescribeFileSystems",

```

```
    "fsx:DescribeVolumes"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "sns:GetSubscriptionAttributes",
    "sns:GetTopicAttributes",
    "sns:GetSMSAttributes",
    "sns:ListSubscriptionsByTopic",
    "sns:ListTopics"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "sqs:ListQueues"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "logs>DeleteSubscriptionFilter"
  ],
  "Resource": [
    "arn:aws:logs:*:*:log-group:*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "logs:PutSubscriptionFilter"
  ],
  "Resource": [
    "arn:aws:logs:*:*:log-group:*",
    "arn:aws:logs:*:*:destination:AmazonCloudWatch-ApplicationInsights-LogIngestionDestination*"
  ]
}
```

```
]
},
{
  "Effect": "Allow",
  "Action": [
    "elasticfilesystem:DescribeFileSystems"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "route53:GetHostedZone",
    "route53:GetHealthCheck",
    "route53:ListHostedZones",
    "route53:ListHealthChecks",
    "route53:ListQueryLoggingConfigs"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "route53resolver:ListFirewallRuleGroupAssociations",
    "route53resolver:GetFirewallRuleGroup",
    "route53resolver:ListFirewallRuleGroups",
    "route53resolver:ListResolverEndpoints",
    "route53resolver:GetResolverQueryLogConfig",
    "route53resolver:ListResolverQueryLogConfigs",
    "route53resolver:ListResolverQueryLogConfigAssociations",
    "route53resolver:GetResolverEndpoint",
    "route53resolver:GetFirewallRuleGroupAssociation"
  ],
  "Resource": [
    "*"
  ]
}
]
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for CloudWatch Application Insights

You don't need to manually create a service-linked role. When you create a new Application Insights application in the AWS Management Console, CloudWatch Application Insights creates the service-linked role for you.

If you delete this service-linked role, and then want to create it again, you can use the same process to recreate the role in your account. When you create a new Application Insights application, CloudWatch Application Insights creates the service-linked role for you again.

Editing a service-linked role for CloudWatch Application Insights

CloudWatch Application Insights does not allow you to edit the `AWSServiceRoleForApplicationInsights` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for CloudWatch Application Insights

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you avoid having an unused entity that is not actively monitored or maintained. However, you must delete all applications in Application Insights before you can manually delete the role.

Note

If the CloudWatch Application Insights service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CloudWatch Application Insights resources used by the `AWSServiceRoleForApplicationInsights`

- Delete all of your CloudWatch Application Insights applications. For more information, see "Deleting Your Application(s)" in the CloudWatch Application Insights User Guide.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForApplicationInsights` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Supported Regions for CloudWatch Application Insights service-linked roles

CloudWatch Application Insights supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see [CloudWatch Application Insights Regions and Endpoints](#).

AWS managed policies for Amazon CloudWatch Application Insights

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

AWS managed policy: CloudWatchApplicationInsightsFullAccess

You can attach the `CloudWatchApplicationInsightsFullAccess` policy to your IAM identities.

This policy grants administrative permissions that allow full access to Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows full access to Application Insights functionality.
- `iam` – Allows Application Insights to create the service-linked role, `AWSServiceRoleForApplicationInsights`. This is required so that Application Insights can perform operations such as analyze the resource groups of a customer, create CloudFormation stacks to create alarms on metrics, and configure the CloudWatch Agent on EC2 instances. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "applicationinsights:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
```

```

    "sqs:ListQueues",
    "elasticloadbalancing:DescribeLoadBalancers",
    "elasticloadbalancing:DescribeTargetGroups",
    "elasticloadbalancing:DescribeTargetHealth",
    "autoscaling:DescribeAutoScalingGroups",
    "lambda:ListFunctions",
    "dynamodb:ListTables",
    "s3:ListAllMyBuckets",
    "sns:ListTopics",
    "states:ListStateMachines",
    "apigateway:GET",
    "ecs:ListClusters",
    "ecs:DescribeTaskDefinition",
    "ecs:ListServices",
    "ecs:ListTasks",
    "eks:ListClusters",
    "eks:ListNodegroups",
    "fsx:DescribeFileSystems",
    "logs:DescribeLogGroups",
    "elasticfilesystem:DescribeFileSystems"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": [
    "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
  ],
  "Condition": {
    "StringEquals": {
      "iam:AWSServiceName": "application-insights.amazonaws.com"
    }
  }
}
]
}

```

AWS managed policy: CloudWatchApplicationInsightsReadOnlyAccess

You can attach the `CloudWatchApplicationInsightsReadOnlyAccess` policy to your IAM identities.

This policy grants administrative permissions that allow read-only access to all Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows read-only access to Application Insights functionality.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "applicationinsights:Describe*",
        "applicationinsights:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: `CloudwatchApplicationInsightsServiceLinkedRolePolicy`

You can't attach `CloudwatchApplicationInsightsServiceLinkedRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Application Insights to monitor customer resources. For more information, see [Using service-linked roles for CloudWatch Application Insights](#).

Application Insights updates to AWS managed policies

View details about updates to AWS managed policies for Application Insights since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Application Insights [Document history](#) page.

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to list CloudFormation stacks.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to analyze and monitor AWS resources nested in the CloudFormation stack.</p>	April 24, 2023
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to get list of Amazon VPC and Route 53 resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to automatically set up best practice network monitoring with Amazon CloudWatch.</p>	January 23, 2023
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to get SSM command invocation results.</p> <p>These permissions are required for Amazon CloudWatch Application</p>	December 19, 2022

Change	Description	Date
	<p>Insights to automatically detect and monitor workloads running on Amazon EC2 instances.</p>	
<p>CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy</p>	<p>Application Insights added new permissions to describe Amazon VPC and Route 53 resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to read customer Amazon VPC and Route 53 resource configurations, and to help customers automatically set up best practice network monitoring with Amazon CloudWatch.</p>	<p>December 19, 2022</p>
<p>CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy</p>	<p>Application Insights added new permissions to describe EFS resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to read Amazon EFS customer resource configurations, and to help customers automatically set up best practices for EFS monitoring with CloudWatch.</p>	<p>October 3, 2022</p>

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to describe the EFS file system.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.</p>	October 3, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to retrieve information about FSx resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to monitor workloads by retrieving sufficient information about the underlying FSx volumes.</p>	September 12, 2022
AWS managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	<p>Application Insights added a new permission to describe log groups.</p> <p>This permissions is required for Amazon CloudWatch Application Insights to ensure that the correct permissions for monitoring log groups are in an account when creating a new application.</p>	January 24, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to create and delete CloudWatch Log Subscription Filters.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to create Subscription Filters to facilitate log monitoring of resources within configured applications.</p>	January 24, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to describe target groups and target health for Elastic Load Balancers.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.</p>	November 4, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to run the AmazonCloudWatch-ManagedAgent SSM document on Amazon EC2 instances.</p> <p>This permissions is required for Amazon CloudWatch Application Insights to clean up CloudWatch agent configuration files created by Application Insights.</p>	September 30, 2021

Change	Description	Date
<p>CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy</p>	<p>Application Insights added new permissions to support account-based application monitoring to onboard and monitor all supported resources in your account.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to query, tag resources, and create groups for these resources.</p> <p>Application Insights added new permissions to support monitoring of SNS topics.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to gather metadata from SNS resources to configure monitoring for SNS topics.</p>	<p>September 15, 2021</p>

Change	Description	Date
AWS managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	<p>Application Insights added new permissions to describe and list supported resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.</p>	September 15, 2021
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	<p>Application Insights added new permissions to describe FSx resources.</p> <p>These permissions are required for Amazon CloudWatch Application Insights to read customer FSx resource configurations, and to help customers automatically set up best practice FSx monitoring with CloudWatch.</p>	August 31, 2021

Change	Description	Date
<p>CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy</p>	<p>Application Insights added new permissions to describe and list ECS and EKS service resources.</p> <p>This permission is required for Amazon CloudWatch Application Insights to read customer container resources configuration, and to help customers automatically set up best practice container monitoring with CloudWatch.</p>	<p>May 18, 2021</p>
<p>CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy</p>	<p>Application Insights added new permissions to allow OpsCenter to tag OpsItems using the <code>ssm:AddTagsToResource</code> action on resources with the <code>opsitem</code> resource type.</p> <p>This permission is required by OpsCenter. Amazon CloudWatch Application Insights creates OpsItems so that the customer can resolve problems using AWS SSM OpsCenter.</p>	<p>April 13, 2021</p>
<p>Application Insights started tracking changes</p>	<p>Application Insights started tracking changes for its AWS managed policies.</p>	<p>April 13, 2021</p>

Amazon CloudWatch permissions reference

The following table lists each CloudWatch API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's Action field, and you specify a wildcard character (*) as the resource value in the policy's Resource field.

You can use AWS-wide condition keys in your CloudWatch policies to express conditions. For a complete list of AWS-wide keys, see [AWS Global and IAM Condition Context Keys](#) in the *IAM User Guide*.

Note

To specify an action, use the `cloudwatch:` prefix followed by the API operation name. For example: `cloudwatch:GetMetricData`, `cloudwatch:ListMetrics`, or `cloudwatch:*` (for all CloudWatch actions).

Topics

- [CloudWatch API operations and required permissions for actions](#)
- [CloudWatch Contributor Insights API operations and required permissions for actions](#)
- [CloudWatch Events API operations and required permissions for actions](#)
- [CloudWatch Logs API operations and required permissions for actions](#)
- [Amazon EC2 API operations and required permissions for actions](#)
- [Amazon EC2 Auto Scaling API operations and required permissions for actions](#)

CloudWatch API operations and required permissions for actions

CloudWatch API operations	Required permissions (API actions)
DeleteAlarms	<code>cloudwatch:DeleteAlarms</code> Required to delete an alarm.
DeleteDashboards	<code>cloudwatch:DeleteDashboards</code>

CloudWatch API operations	Required permissions (API actions)
DeleteMetricStream	<p>Required to delete a dashboard.</p> <p><code>cloudwatch:DeleteMetricStream</code></p> <p>Required to delete a metric stream.</p>
DescribeAlarmHistory	<p><code>cloudwatch:DescribeAlarmHistory</code></p> <p>Required to view alarm history. To retrieve information about composite alarms, your <code>cloudwatch:DescribeAlarmHistory</code> permission must have a <code>*</code> scope. You can't return information about composite alarms if your <code>cloudwatch:DescribeAlarmHistory</code> permission has a narrower scope.</p>
DescribeAlarms	<p><code>cloudwatch:DescribeAlarms</code></p> <p>Required to retrieve information about alarms.</p> <p>To retrieve information about composite alarms, your <code>cloudwatch:DescribeAlarms</code> permission must have a <code>*</code> scope. You can't return information about composite alarms if your <code>cloudwatch:DescribeAlarms</code> permission has a narrower scope.</p>
DescribeAlarmsForMetric	<p><code>cloudwatch:DescribeAlarmsForMetric</code></p> <p>Required to view alarms for a metric.</p>

CloudWatch API operations	Required permissions (API actions)
DisableAlarmActions	<code>cloudwatch:DisableAlarmActions</code> Required to disable an alarm action.
EnableAlarmActions	<code>cloudwatch:EnableAlarmActions</code> Required to enable an alarm action.
GetDashboard	<code>cloudwatch:GetDashboard</code> Required to display data about existing dashboards.
GetMetricData	<code>cloudwatch:GetMetricData</code> Required to graph metric data in the CloudWatch console, to retrieve large batches of metric data, and perform metric math on that data.
GetMetricStatistics	<code>cloudwatch:GetMetricStatistics</code> Required to view graphs in other parts of the CloudWatch console and in dashboard widgets.
GetMetricStream	<code>cloudwatch:GetMetricStream</code> Required to view information about a metric stream.

CloudWatch API operations	Required permissions (API actions)
GetMetricWidgetImage	<code>cloudwatch:GetMetricWidgetImage</code> Required to retrieve a snapshot graph of one or more CloudWatch metrics as a bitmap image.
ListDashboards	<code>cloudwatch:ListDashboards</code> Required to view the list of CloudWatch dashboards in your account.
ListMetrics	<code>cloudwatch:ListMetrics</code> Required to view or search metric names within the CloudWatch console and in the CLI. Required to select metrics on dashboard widgets.
ListMetricStreams	<code>cloudwatch:ListMetricStreams</code> Required to view or search the list of metric streams in the account.
PutCompositeAlarm	<code>cloudwatch:PutCompositeAlarm</code> Required to create a composite alarm. To create a composite alarm, your <code>cloudwatch:PutCompositeAlarm</code> permission must have a <code>*</code> scope. You can't return information about composite alarms if your <code>cloudwatch:PutCompositeAlarm</code> permission has a narrower scope.

CloudWatch API operations	Required permissions (API actions)
PutDashboard	<code>cloudwatch:PutDashboard</code> Required to create a dashboard or update an existing dashboard.
PutMetricAlarm	<code>cloudwatch:PutMetricAlarm</code> Required to create or update an alarm.
PutMetricData	<code>cloudwatch:PutMetricData</code> Required to create metrics.
PutMetricStream	<code>cloudwatch:PutMetricStream</code> Required to create a metric stream.
SetAlarmState	<code>cloudwatch:SetAlarmState</code> Required to manually set an alarm's state.
StartMetricStreams	<code>cloudwatch:StartMetricStreams</code> Required to start the flow of metrics in a metric stream.
StopMetricStreams	<code>cloudwatch:StopMetricStreams</code> Required to temporarily stop the flow of metrics in a metric stream.

CloudWatch API operations	Required permissions (API actions)
TagResource	<p><code>cloudwatch:TagResource</code></p> <p>Required to add or update tags on CloudWatch resources such as alarms and Contributor Insights rules.</p>
UntagResource	<p><code>cloudwatch:UntagResource</code></p> <p>Required to remove tags from CloudWatch resources .</p>

CloudWatch Contributor Insights API operations and required permissions for actions

Important

When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see [Using condition keys to limit Contributor Insights users' access to log groups](#).

CloudWatch Contributor Insights API operations	Required permissions (API actions)
DeleteInsightRules	<p><code>cloudwatch:DeleteInsightRules</code></p> <p>Required to delete Contributor Insights rules.</p>
DescribeInsightRules	<code>cloudwatch:DescribeInsightRules</code>

CloudWatch Contributor Insights API operations	Required permissions (API actions)
	Required to view the Contributor Insights rules in your account.
EnableInsightRules	cloudwatch:EnableInsightRules Required to enable Contributor Insights rules.
GetInsightRuleReport	cloudwatch:GetInsightRuleReport Required to retrieve time series data and other statistics collected by Contributor Insights rules.
PutInsightRule	cloudwatch:PutInsightRule Required to create Contributor Insights rules. See the Important note at the beginning of this table.

CloudWatch Events API operations and required permissions for actions

CloudWatch Events API operations	Required permissions (API actions)
DeleteRule	events:DeleteRule Required to delete a rule.
DescribeRule	events:DescribeRule Required to list the details about a rule.
DisableRule	events:DisableRule

CloudWatch Events API operations	Required permissions (API actions)
	Required to disable a rule.
EnableRule	events:EnableRule Required to enable a rule.
ListRuleNamesByTarget	events:ListRuleNamesByTarget Required to list rules associated with a target.
ListRules	events:ListRules Required to list all rules in your account.
ListTargetsByRule	events:ListTargetsByRule Required to list all targets associated with a rule.
PutEvents	events:PutEvents Required to add custom events that can be matched to rules.
PutRule	events:PutRule Required to create or update a rule.
PutTargets	events:PutTargets Required to add targets to a rule.

CloudWatch Events API operations	Required permissions (API actions)
RemoveTargets	<p>events:RemoveTargets</p> <p>Required to remove a target from a rule.</p>
TestEventPattern	<p>events:TestEventPattern</p> <p>Required to test an event pattern against a given event.</p>

CloudWatch Logs API operations and required permissions for actions

CloudWatch Logs API operations	Required permissions (API actions)
CancelExportTask	<p>logs:CancelExportTask</p> <p>Required to cancel a pending or running export task.</p>
CreateExportTask	<p>logs:CreateExportTask</p> <p>Required to export data from a log group to an Amazon S3 bucket.</p>
CreateLogGroup	<p>logs:CreateLogGroup</p> <p>Required to create a new log group.</p>
CreateLogStream	<p>logs:CreateLogStream</p> <p>Required to create a new log stream in a log group.</p>
DeleteDestination	<p>logs>DeleteDestination</p>

CloudWatch Logs API operations	Required permissions (API actions)
	<p>Required to delete a log destination and disables any subscription filters to it.</p>
<p>DeleteLogGroup</p>	<p><code>logs:DeleteLogGroup</code></p> <p>Required to delete a log group and any associated archived log events.</p>
<p>DeleteLogStream</p>	<p><code>logs:DeleteLogStream</code></p> <p>Required to delete a log stream and any associated archived log events.</p>
<p>DeleteMetricFilter</p>	<p><code>logs:DeleteMetricFilter</code></p> <p>Required to delete a metric filter associated with a log group.</p>
<p>DeleteQueryDefinition</p>	<p><code>logs:DeleteQueryDefinition</code></p> <p>Required to delete a saved query definition in CloudWatch Logs Insights.</p>
<p>DeleteResourcePolicy</p>	<p><code>logs:DeleteResourcePolicy</code></p> <p>Required to delete a CloudWatch Logs resource policy.</p>
<p>DeleteRetentionPolicy</p>	<p><code>logs:DeleteRetentionPolicy</code></p> <p>Required to delete a log group's retention policy.</p>

CloudWatch Logs API operations	Required permissions (API actions)
DeleteSubscriptionFilter	<code>logs:DeleteSubscriptionFilter</code> Required to delete the subscription filter associated with a log group.
DescribeDestinations	<code>logs:DescribeDestinations</code> Required to view all destinations associated with the account.
DescribeExportTasks	<code>logs:DescribeExportTasks</code> Required to view all export tasks associated with the account.
DescribeLogGroups	<code>logs:DescribeLogGroups</code> Required to view all log groups associated with the account.
DescribeLogStreams	<code>logs:DescribeLogStreams</code> Required to view all log streams associated with a log group.
DescribeMetricFilters	<code>logs:DescribeMetricFilters</code> Required to view all metrics associated with a log group.

CloudWatch Logs API operations	Required permissions (API actions)
DescribeQueryDefinitions	<code>logs:DescribeQueryDefinitions</code> Required to see the list of saved query definitions in CloudWatch Logs Insights.
DescribeQueries	<code>logs:DescribeQueries</code> Required to see the list of CloudWatch Logs Insights queries that are scheduled, executing, or have recently executed.
DescribeResourcePolicies	<code>logs:DescribeResourcePolicies</code> Required to view a list of CloudWatch Logs resource policies.
DescribeSubscriptionFilters	<code>logs:DescribeSubscriptionFilters</code> Required to view all subscription filters associated with a log group.
FilterLogEvents	<code>logs:FilterLogEvents</code> Required to sort log events by log group filter pattern.
GetLogEvents	<code>logs:GetLogEvents</code> Required to retrieve log events from a log stream.

CloudWatch Logs API operations	Required permissions (API actions)
GetLogGroupFields	<code>logs:GetLogGroupFields</code> Required to retrieve the list of fields that are included in the log events in a log group.
GetLogRecord	<code>logs:GetLogRecord</code> Required to retrieve the details from a single log event.
GetQueryResults	<code>logs:GetQueryResults</code> Required to retrieve the results of CloudWatch Logs Insights queries.
ListTagsLogGroup	<code>logs:ListTagsLogGroup</code> Required to list the tags associated with a log group.
PutDestination	<code>logs:PutDestination</code> Required to create or update a destination log stream (such as an Kinesis stream).
PutDestinationPolicy	<code>logs:PutDestinationPolicy</code> Required to create or update an access policy associated with an existing log destination.

CloudWatch Logs API operations	Required permissions (API actions)
PutLogEvents	<code>logs:PutLogEvents</code> Required to upload a batch of log events to a log stream.
PutMetricFilter	<code>logs:PutMetricFilter</code> Required to create or update a metric filter and associate it with a log group.
PutQueryDefinition	<code>logs:PutQueryDefinition</code> Required to save a query in CloudWatch Logs Insights.
PutResourcePolicy	<code>logs:PutResourcePolicy</code> Required to create a CloudWatch Logs resource policy.
PutRetentionPolicy	<code>logs:PutRetentionPolicy</code> Required to set the number of days to keep log events (retention) in a log group.
PutSubscriptionFilter	<code>logs:PutSubscriptionFilter</code> Required to create or update a subscription filter and associate it with a log group.

CloudWatch Logs API operations	Required permissions (API actions)
StartQuery	logs:StartQuery Required to start CloudWatch Logs Insights queries.
StopQuery	logs:StopQuery Required to stop a CloudWatch Logs Insights query that is in progress.
TagLogGroup	logs:TagLogGroup Required to add or update log group tags.
TestMetricFilter	logs:TestMetricFilter Required to test a filter pattern against a sampling of log event messages.

Amazon EC2 API operations and required permissions for actions

Amazon EC2 API operations	Required permissions (API actions)
DescribeInstanceStatus	ec2:DescribeInstanceStatus Required to view EC2 instance status details.
DescribeInstances	ec2:DescribeInstances Required to view EC2 instance details.
RebootInstances	ec2:RebootInstances

Amazon EC2 API operations	Required permissions (API actions)
	Required to reboot an EC2 instance.
StopInstances	ec2:StopInstances Required to stop an EC2 instance.
TerminateInstances	ec2:TerminateInstances Required to terminate an EC2 instance.

Amazon EC2 Auto Scaling API operations and required permissions for actions

Amazon EC2 Auto Scaling API operations	Required permissions (API actions)
Scaling	autoscaling:Scaling Required to scale an Auto Scaling group.
Trigger	autoscaling:Trigger Required to trigger an Auto Scaling action.

Compliance validation for Amazon CloudWatch

Third-party auditors assess the security and compliance of Amazon CloudWatch as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Amazon CloudWatch is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon CloudWatch

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Infrastructure security in Amazon CloudWatch

As a managed service, Amazon CloudWatch is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CloudWatch through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the Amazon Web Services Cloud. A subnet is a range of IP addresses in a VPC. You can deploy a variety of AWS resources in the subnets of your VPCs. For example, you can deploy Amazon EC2 instances, EMR clusters, and DynamoDB tables in subnets. For more information, see the [Amazon VPC User Guide](#).

To enable CloudWatch to communicate with resources in a VPC without going through the public internet, use AWS PrivateLink. For more information, see [Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints](#).

A private subnet is a subnet with no default route to the public internet. Deploying an AWS resource in a private subnet does not prevent Amazon CloudWatch from collecting built-in metrics from the resource.

If you need to publish custom metrics from an AWS resource in a private subnet, you can do so using a proxy server. The proxy server forwards those HTTPS requests to the public API endpoints for CloudWatch.

AWS Security Hub

Monitor your usage of CloudWatch as it relates to security best practices by using AWS Security Hub. Security Hub uses *security controls* to evaluate resource configurations and *security standards* to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate CloudWatch resources, see [Amazon CloudWatch controls](#) in the AWS Security Hub User Guide.

Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC, CloudWatch, and CloudWatch Synthetics. You can use these connections to enable CloudWatch and CloudWatch Synthetics to communicate with your resources on your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. To connect your VPC to CloudWatch or CloudWatch Synthetics, you define an *interface VPC endpoint* to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CloudWatch or CloudWatch Synthetics without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see [What Is Amazon VPC](#) in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see the [New – AWS PrivateLink for AWS Services](#) blog post.

The following steps are for users of Amazon VPC. For more information, see [Getting Started](#) in the *Amazon VPC User Guide*.

CloudWatch VPC endpoint

CloudWatch currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)

- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Middle East (UAE)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Creating a VPC endpoint for CloudWatch

To start using CloudWatch with your VPC, create an interface VPC endpoint for CloudWatch. The service name to choose is `com.amazonaws.region.monitoring`. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch. CloudWatch calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch, and you already have metrics flowing to CloudWatch from resources located on your VPC, these metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch. This policy allows users connecting to CloudWatch through the VPC to send metric data to CloudWatch and prevents them from performing other CloudWatch actions.

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To edit the VPC endpoint policy for CloudWatch

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch, choose **Create Endpoint**. Select **com.amazonaws.region.monitoring**, and then choose **Create endpoint**.
4. Select the **com.amazonaws.region.monitoring** endpoint, and then choose the **Policy** tab.
5. Choose **Edit Policy**, and then make your changes.

CloudWatch Synthetics VPC endpoint

CloudWatch Synthetics currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)

- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)

Creating a VPC endpoint for CloudWatch Synthetics

To start using CloudWatch Synthetics with your VPC, create an interface VPC endpoint for CloudWatch Synthetics. The service name to choose is `com.amazonaws.region.synthetics`. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Synthetics. CloudWatch Synthetics communicates with other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch Synthetics, and you already have an interface endpoint for Amazon S3, CloudWatch Synthetics begins communicating with Amazon S3 through the interface VPC endpoint by default.

Controlling access to your CloudWatch Synthetics VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies affect canaries that are managed privately by VPC. They are not needed for canaries that run on private subnets.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Synthetics. This policy enables users connecting to CloudWatch Synthetics through the VPC to view information about canaries and their runs, but not to create, modify, or delete canaries.

```
{
  "Statement": [
    {
      "Action": [
        "synthetics:DescribeCanaries",
        "synthetics:GetCanaryRuns"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    }
  ]
}
```

To edit the VPC endpoint policy for CloudWatch Synthetics

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch Synthetics, choose **Create Endpoint**. Select **com.amazonaws.region.synthetics** and then choose **Create endpoint**.
4. Select the **com.amazonaws.region.synthetics** endpoint and then choose the **Policy** tab.
5. Choose **Edit Policy**, and then make your changes.

Security considerations for Synthetics canaries

The following sections explain security issues that you should consider when creating and running canaries in Synthetics.

Use secure connections

Because canary code and the results from canary test runs can contain sensitive information, do not have your canary connect to endpoints over unencrypted connections. Always use encrypted connections, such as those that begin with `https://`.

Canary naming considerations

The Amazon Resource Name (ARN) of a canary is included in the user-agent header as a part of outbound calls made from the Puppeteer-driven Chromium browser that is included as a part of the CloudWatch Synthetics wrapper library. This helps identify CloudWatch Synthetics canary traffic and relate it back to the canaries that are making calls.

The canary ARN includes the canary name. Choose canary names that do not reveal proprietary information.

Additionally, be sure to point your canaries only at websites and endpoints that you control.

Secrets and sensitive information in canary code

If you pass your canary code directly into the canary using a zip file, the contents of the script can be seen in AWS CloudTrail logs.

If you have sensitive information or secrets (such as access keys or database credentials) in a canary script, we strongly recommend that you store the script as a versioned object in Amazon S3 and pass the Amazon S3 location into the canary, instead of passing the canary code by a zip file.

If you do use a zip file to pass the canary script, we strongly recommend that you don't include secrets or sensitive information in your canary source code. For more information about how to use AWS Secrets Manager to help keep your secrets safe, see [What is AWS Secrets Manager?](#).

Permissions considerations

We recommend that you restrict access to resources that are created or used by CloudWatch Synthetics. Use tight permissions on the Amazon S3 buckets where canaries store test run results and other artifacts, such as logs and screenshots.

Similarly, keep tight permissions on the locations where your canary source code is stored, so that no user accidentally or maliciously deletes the Lambda layers or Lambda functions used for the canary.

To help make sure you run the canary code you intend, you can use object versioning on the Amazon S3 bucket where your canary code is stored. Then when you specify this code to run as a canary, you can include the object `versionId` as part of the path, as in the following examples.

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id  
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id  
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

Stack traces and exception messages

By default, CloudWatch Synthetics canaries capture any exception thrown by your canary script, no matter whether the script is custom or is from a blueprint. CloudWatch Synthetics logs both the exception message and the stack trace to three locations:

- Back into the CloudWatch Synthetics service to speed up debugging when you describe test runs
- Into CloudWatch Logs according to the configuration that your Lambda functions are created with
- Into the Synthetics log file, which is a plaintext file that is uploaded to the Amazon S3 location specified by the value you set for the `resultsLocation` of the canary

If you want to send and store less information, you can capture exceptions before they return to the CloudWatch Synthetics wrapper library.

You can also have request URLs in your errors. CloudWatch Synthetics scans for any URLs in the error thrown by your script and redacts restricted URL parameters from them based on the `restrictedUrlParameters` configuration. If you are logging error messages in your script, you can use [getSanitizedErrorMessage](#) to redact URLs before logging.

Scope your IAM roles narrowly

We recommend that you do not configure your canary to visit potentially malicious URLs or endpoints. Pointing your Canary to untrusted or unknown websites or endpoints could expose your Lambda function code to malicious user's scripts. Assuming a malicious website can break out of Chromium, it could have access to your Lambda code in a similar way to if you connected to it using an internet browser.

Run your Lambda function with an IAM execution role that has scoped-down permissions. This way, if your Lambda function is compromised by a malicious script, it is limited in the actions it can take when running as your canary's AWS account.

When you use the CloudWatch console to create a canary, it is created with a scoped-down IAM execution role.

Sensitive data redaction

CloudWatch Synthetics captures URLs, status code, failure reason (if any), and headers and bodies of requests and responses. This enables a canary user to understand, monitor, and debug canaries.

The configurations described in the following sections can be set at any point in canary execution. You can also choose to apply different configurations to different synthetics steps.

Request URLs

By default, CloudWatch Synthetics logs request URLs, status codes, and the status reason for each URL in canary logs. Request URLs can also appear in canary execution reports, HAR files, and so on. Your request URL might contain sensitive query parameters, such as access tokens or passwords. You can redact sensitive information from being logged by CloudWatch Synthetics.

To redact sensitive information, set the configuration property **restrictedUrlParameters**. For more information, see [SyntheticsConfiguration class](#). This causes CloudWatch Synthetics to redact URL parameters, including path and query parameter values, based on **restrictedUrlParameters** before logging. If you are logging URLs in your script, you can use [getSanitizedUrl\(url, stepConfig = null\)](#) to redact URLs before logging. For more information, see [SyntheticsLogHelper class](#).

Headers

By default, CloudWatch Synthetics doesn't log request/response headers. For UI canaries, this is the default behavior for canaries using runtime version `syn-nodejs-puppeteer-3.2` and later.

If your headers don't contain sensitive information, you can enable headers in HAR file and HTTP reports by setting the **includeRequestHeaders** and **includeResponseHeaders** properties to `true`. You can enable all headers but choose to restrict values of sensitive header keys. For example, you can choose to only redact `Authorization` headers from artifacts produced by canaries.

Request and response body

By default, CloudWatch Synthetics doesn't log the request/response body in canary logs or reports. This information is particularly useful for API canaries. Synthetics captures all HTTP requests and can show headers, request and response bodies. For more information, see [executeHttpStep\(stepName, requestOptions, \[callback\], \[stepConfig\]\)](#). You can choose to enable request/response body by setting the **includeRequestBody** and **includeResponseBody** properties to `true`.

Logging Amazon CloudWatch API and console operations with AWS CloudTrail

Amazon CloudWatch and CloudWatch Synthetics are integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls made by or on behalf of your AWS account. The captured calls include calls from the console and code calls to API operations.

If you create a *trail*, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for CloudWatch. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to CloudWatch, the IP address from which the request was made, who made the request, when it was made, and other details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

For an ongoing record of events in your AWS account, including events for CloudWatch and CloudWatch Synthetics, create a trail. A trail enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. You can configure other AWS services to further analyze and act on the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)

- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

Note

For information about CloudWatch Logs API calls that are logged in CloudTrail, see [CloudWatch Logs information in CloudTrail](#).

Topics

- [CloudWatch information in CloudTrail](#)
- [CloudWatch data plane events in CloudTrail](#)
- [Query generation information in CloudTrail](#)
- [CloudWatch Internet Monitor in CloudTrail](#)
- [CloudWatch Synthetics information in CloudTrail](#)

CloudWatch information in CloudTrail

CloudWatch supports logging the following actions as events in CloudTrail log files:

- [DeleteAlarms](#)
- [DeleteAnomalyDetector](#)
- [DeleteDashboards](#)
- [DescribeAlarmHistory](#)
- [DescribeAlarms](#)
- [DescribeAlarmsForMetric](#)
- [DescribeAnomalyDetectors](#)
- [DisableAlarmActions](#)
- [EnableAlarmActions](#)
- [GetDashboard](#)
- [ListDashboards](#)

- [PutAnomalyDetector](#)
- [PutDashboard](#)
- [PutMetricAlarm](#)
- [SetAlarmState](#)

Example: CloudWatch log file entries

The following example shows a CloudTrail log entry that demonstrates the `PutMetricAlarm` action.

```
{
  "Records": [{
    "eventVersion": "1.01",
    "userIdentity": {
      "type": "Root",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::123456789012:root",
      "accountId": "123456789012",
      "accessKeyId": "EXAMPLE_KEY_ID"
    },
    "eventTime": "2014-03-23T21:50:34Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "PutMetricAlarm",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
    "requestParameters": {
      "threshold": 50.0,
      "period": 60,
      "metricName": "CloudTrail Test",
      "evaluationPeriods": 3,
      "comparisonOperator": "GreaterThanThreshold",
      "namespace": "AWS/CloudWatch",
      "alarmName": "CloudTrail Test Alarm",
      "statistic": "Sum"
    },
    "responseElements": null,
    "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
    "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
  },
  ..additional entries
}
```

```
]
}
```

The following log file entry shows that a user called the CloudWatch Events PutRule action.

```
{
  "eventVersion": "1.03",
  "userIdentity": {
    "type": "Root",
    "principalId": "123456789012",
    "arn": "arn:aws:iam::123456789012:root",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-11-17T23:56:15Z"
      }
    }
  },
  "eventTime": "2015-11-18T00:11:28Z",
  "eventSource": "events.amazonaws.com",
  "eventName": "PutRule",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "AWS Internal",
  "userAgent": "AWS CloudWatch Console",
  "requestParameters": {
    "description": "",
    "name": "cttest2",
    "state": "ENABLED",
    "eventPattern": "{\"source\": [\"aws.ec2\"], \"detail-type\": [\"EC2 Instance State-change Notification\"]}",
    "scheduleExpression": ""
  },
  "responseElements": {
    "ruleArn": "arn:aws:events:us-east-1:123456789012:rule/cttest2"
  },
  "requestID": "e9caf887-8d88-11e5-a331-3332aa445952",
  "eventID": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
  "eventType": "AwsApiCall",
  "apiVersion": "2015-10-07",
  "recipientAccountId": "123456789012"
}
```

The following log file entry shows that a user called the CloudWatch Logs `CreateExportTask` action.

```
{
  "eventVersion": "1.03",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/someuser",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "someuser"
  },
  "eventTime": "2016-02-08T06:35:14Z",
  "eventSource": "logs.amazonaws.com",
  "eventName": "CreateExportTask",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
  "requestParameters": {
    "destination": "yourdestination",
    "logGroupName": "yourloggroup",
    "to": 123456789012,
    "from": 0,
    "taskName": "yourtask"
  },
  "responseElements": {
    "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
  },
  "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
  "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
  "eventType": "AwsApiCall",
  "apiVersion": "20140328",
  "recipientAccountId": "123456789012"
}
```

CloudWatch data plane events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch data plane operations [GetMetricData](#) and [GetMetricWidgetImage](#).

[Data events](#), also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

To enable logging of the **GetMetricData** and **GetMetricWidgetImage** data events in CloudTrail files, you'll need to enable the logging of data plane API activity in CloudTrail. See [Logging data events for trails](#) for more information.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch **GetMetricData** or **GetMetricWidgetImage** APIs, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the **GetMetricData** and **GetMetricWidgetImage** APIs using CloudTrail helps you enable operational and risk auditing, governance, and compliance of your AWS account.

Note

When you view the **GetMetricData** events in CloudTrail, you might see more calls than the calls that you initiated. This is because CloudWatch logs events to CloudTrail for **GetMetricData** actions that are initiated by internal components. For example, you'll see **GetMetricData** calls initiated by CloudWatch dashboards to refresh widget data, and **GetMetricData** calls initiated by a monitoring account to retrieve data from a source account, in cross-account observability. These internally-initiated calls don't incur CloudWatch charges, but they do count toward the number of events logged in CloudTrail, and CloudTrail charges according to the number of events logged.

The following is an example of a CloudTrail event for a **GetMetricData** operation.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDA2NYTR2EPCTNY7AF3L",
    "arn": "arn:aws:iam::111122223333:user/admin",
    "accountId": "111122223333",
    "accessKeyId": "EXAMPLE1234567890",
    "userName": "admin"
  }
}
```

```
  },
  "eventTime": "2024-05-08T16:20:34Z",
  "eventSource": "monitoring.amazonaws.com",
  "eventName": "GetMetricData",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "99.45.3.7",
  "userAgent": "aws-cli/2.13.23 Python/3.11.5 Darwin/23.4.0 exe/x86_64 prompt/off
command/cloudwatch.get-metric-data",
  "requestParameters": {
    "metricDataQueries": [{
      "id": "e1",
      "expression": "m1 / m2",
      "label": "ErrorRate"
    }],
    {
      "id": "m1",
      "metricStat": {
        "metric": {
          "namespace": "CWAgent",
          "metricName": "disk_used_percent",
          "dimensions": [{
            "name": "LoadBalancerName",
            "value": "EXAMPLE4623a5cb6a7384c5229"
          }]
        },
        "period": 300,
        "stat": "Sum",
        "unit": "Count"
      },
      "returnData": false
    },
    {
      "id": "m2",
      "metricStat": {
        "metric": {
          "namespace": "CWAgent",
          "metricName": "disk_used_percent",
          "dimensions": [{
            "name": "LoadBalancerName",
            "value": "EXAMPLE4623a5cb6a7384c5229"
          }]
        },
        "period": 300,
        "stat": "Sum"
      }
    }
  ]
}
```

```
        },
        "returnData": true
    }
],
"startTime": "Apr 19, 2024, 4:00:00 AM",
"endTime": "May 8, 2024, 4:30:00 AM"
},
"responseElements": null,
"requestID": "EXAMPLE-57ac-47d5-938c-f5917c6799d5",
"eventID": "EXAMPLE-211c-404b-b13d-36d93c8b4fbf",
"readOnly": true,
"resources": [{
    "type": "AWS::CloudWatch::Metric"
}],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "111122223333",
"eventCategory": "Data",
"tlsDetails": {
    "tlsVersion": "TLSv1.3",
    "cipherSuite": "TLS_AES_128_GCM_SHA256",
    "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
}
}
```

Query generation information in CloudTrail

CloudTrail logging for Query generator console events is also supported. Query generator is currently supported for CloudWatch Metric Insights and CloudWatch Logs Insights. In these CloudTrail events, the `eventSource` is `monitoring.amazonaws.com`.

The following example shows a CloudTrail log entry that demonstrates the **GenerateQuery** action in CloudWatch Metrics Insights.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId": "123456789012",
        "userName": "SAMPLE_NAME"
      },
      "attributes": {
        "creationDate": "2020-04-08T21:43:24Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2020-04-08T23:06:30Z",
  "eventSource": "monitoring.amazonaws.com",
  "eventName": "GenerateQuery",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "exampleUserAgent",
  "requestParameters": {
    "query_ask": "****",
    "query_type": "MetricsInsights",
    "metrics_insights": {
      "aws_namespaces": [
        "AWS/S3",
        "AWS/Lambda",
        "AWS/DynamoDB"
      ]
    }
  },
  "include_description": true
},
"responseElements": null,
"requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
"eventID": "52723fd9-4a54-478c-ac55-1234567890",
"readOnly": true,
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "111122223333",
"eventCategory": "Management"
}
```

CloudWatch Internet Monitor in CloudTrail

CloudWatch Internet Monitor supports logging the following actions as events in CloudTrail log files.

- [CreateMonitor](#)
- [DeleteMonitor](#)
- [GetHealthEvent](#)
- [GetMonitor](#)
- [GetQueryResults](#)
- [GetQueryStatus](#)
- [ListHealthEvents](#)
- [ListMonitors](#)
- [ListTagsForResource](#)
- [StartQuery](#)
- [StopQuery](#)
- [UpdateMonitor](#)

Example: CloudWatch Internet Monitor log file entries

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `ListMonitors` action.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",
```



```

        "userName": "SAMPLE_NAME"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-10-11T17:25:41Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-10-11T17:30:18Z",
  "eventSource": "internetmonitor.amazonaws.com",
  "eventName": "ListMonitors",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "requestParameters": null,
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEebbbb",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}

```

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `CreateMonitor` action.

```

{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",

```

```

        "userName": "SAMPLE_NAME"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-10-11T17:25:41Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-10-11T17:30:08Z",
  "eventSource": "internetmonitor.amazonaws.com",
  "eventName": "CreateMonitor",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "requestParameters": {
    "MonitorName": "TestMonitor",
    "Resources": ["arn:aws:ec2:us-east-2:444455556666:vpc/vpc-febc0b95"],
    "ClientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
  },
  "responseElements": {
    "Arn": "arn:aws:internetmonitor:us-east-2:444455556666:monitor/ct-
onboarding-test",
    "Status": "PENDING"
  },
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbbb",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}

```

CloudWatch Synthetics information in CloudTrail

CloudWatch Synthetics supports logging the following actions as events in CloudTrail log files:

- [CreateCanary](#)
- [DeleteCanary](#)
- [DescribeCanaries](#)

- [DescribeCanariesLastRun](#)
- [DescribeRuntimeVersions](#)
- [GetCanary](#)
- [GetCanaryRuns](#)
- [ListTagsForResource](#)
- [StartCanary](#)
- [StopCanary](#)
- [TagResource](#)
- [UntagResource](#)
- [UpdateCanary](#)

Example: CloudWatch Synthetics log file entries

The following example shows a CloudTrail Synthetics log entry that demonstrates the DescribeCanaries action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId": "123456789012",
        "userName": "SAMPLE_NAME"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-04-08T21:43:24Z"
      }
    }
  }
}
```

```

    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "DescribeCanaries",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "201ed5f3-15db-4f87-94a4-123456789",
    "eventID": "73ddb81-3dd0-4ada-b246-123456789",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
  }
}

```

The following example shows a CloudTrail Synthetics log entry that demonstrates the UpdateCanary action.

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",
        "accountId": "123456789012",
        "userName": "SAMPLE_NAME"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-04-08T21:43:24Z"
      }
    }
  }
}

```

```

    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
      "Schedule": {
        "Expression": "rate(1 minute)"
      },
      "name": "sample_canary_name",
      "Code": {
        "Handler": "myOwnScript.handler",
        "ZipFile": "SAMPLE_ZIP_FILE"
      }
    },
    "responseElements": null,
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
  }
}

```

The following example shows a CloudTrail Synthetics log entry that demonstrates the GetCanaryRuns action.

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111222333444:role/Administrator",

```

```
    "accountId": "123456789012",
      "userName": "SAMPLE_NAME"
    },
    "webIdFederationData": {},
    "attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2020-04-08T21:43:24Z"
    }
  }
},
"eventTime": "2020-04-08T23:06:30Z",
"eventSource": "synthetics.amazonaws.com",
"eventName": "GetCanaryRuns",
"awsRegion": "us-east-1",
"sourceIPAddress": "127.0.0.1",
"userAgent": "aws-internal/3 aws-sdk-java/1.11.590
Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
java/1.8.0_212 vendor/Oracle_Corporation",
"requestParameters": {
  "Filter": "TIME_RANGE",
  "name": "sample_canary_name",
  "FilterValues": [
    "2020-04-08T23:00:00.000Z",
    "2020-04-08T23:10:00.000Z"
  ]
},
"responseElements": null,
"requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
"eventID": "52723fd9-4a54-478c-ac55-1234567890",
"readOnly": true,
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}
```

Tagging your Amazon CloudWatch resources

A *tag* is a custom attribute label that you or AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to a CloudWatch rule that you assign to an EC2 instance.

The following sections provide more information about tags for CloudWatch.

Supported resources in CloudWatch

The following resources in CloudWatch support tagging:

- Alarms – You can tag alarms using the [tag-resource](#) AWS CLI command and the [TagResource](#) API. You can also view and manage your alarm tags using the *Alarms* details page in the CloudWatch console.
- Canaries – You can tag canaries using the CloudWatch console. For more information, see [Creating a canary](#).
- Contributor Insights rules – You can tag Contributor Insights rules when you create them by using the [put-insight-rule](#) AWS CLI command and the [PutInsightRule](#) API. You can add tags to existing rules by using the [tag-resource](#) AWS CLI command and the [TagResource](#) API.
- Metric streams – You can tag metric streams when you create them by using the [put-metric-stream](#) AWS CLI command and the [PutMetricStream](#) API. You can add tags to existing metric streams by using the [tag-resource](#) AWS CLI command and the [TagResource](#) API.

For information about adding and managing tags, see [Managing tags](#).

Managing tags

Tags consist of the Key and Value properties on a resource. You can use the CloudWatch console, the AWS CLI, or the CloudWatch API to add, edit, or delete the values for these properties. For information about working with tags, see the following:

- [TagResource](#), [UntagResource](#), and [ListTagsForResource](#) in the *Amazon CloudWatch API Reference*
- [tag-resource](#), [untag-resource](#), and [list-tags-for-resource](#) in the *Amazon CloudWatch CLI Reference*
- [Working with Tag Editor](#) in the *Resource Groups User Guide*

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with CloudWatch resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: . : + = @ _ / - (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter` and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws :` prefix is prohibited for tags because it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags per resource limit.

Grafana integration

You can use Grafana version 6.5.0 and later to contextually advance through the CloudWatch console and query a dynamic list of metrics by using wildcards. This can help you monitor metrics for AWS resources, such as Amazon Elastic Compute Cloud instances or containers. When new instances are created as part of an Auto Scaling event, they appear in the graph automatically. You don't need to track the new instance IDs. Prebuilt dashboards help simplify the getting started experience for monitoring Amazon EC2, Amazon Elastic Block Store, and AWS Lambda resources.

You can use Grafana version 7.0 and later to perform CloudWatch Logs Insights queries on log groups in CloudWatch Logs. You can visualize your query results in bar, line, and stacked graphs and in a table format. For more information about CloudWatch Logs Insights, see [Analyzing Log Data with CloudWatch Logs Insights](#).

For more information about how to get started, see [Using AWS CloudWatch in Grafana](#) in the Grafana Labs documentation.

CloudWatch service quotas

CloudWatch has the following quotas for metrics, alarms, API requests, and alarm email notifications.

Note

For some AWS services including CloudWatch, you can use the CloudWatch usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information, see [Visualizing your service quotas and setting alarms](#).

Resource	Default quota
Alarm actions	5/alarm. This quota cannot be changed.
Alarm evaluation period	The maximum value, calculated by multiplying the alarm period by the number of evaluation periods used, is one day (86,400 seconds). This quota cannot be changed.
Alarms	10/month/customer for free. Additional alarms incur charges. No limit on the total number of alarms per account. Alarms based on metric math expressions can have up to 10 metrics. 200 Metrics Insights alarms per Region. You can request a quota increase .
Anomaly detection models	500 per Region, per account.
API requests	1,000,000/month/customer for free.
Canaries	200 per Region per account.

Resource	Default quota
	You can request a quota increase .
Contributor Insights API requests	<p>The following APIs have a quota of 20 transactions per second (TPS) and per Region.</p> <ul style="list-style-type: none">• DescribeInsightRules <p>The quota cannot be changed.</p> <ul style="list-style-type: none">• GetInsightRuleReport <p>You can request a quota increase.</p> <p>The following APIs have a quota of 5 TPS per Region. This quota cannot be changed.</p> <ul style="list-style-type: none">• DeleteInsightRules• PutInsightRule <p>The following APIs have a quota of 1 TPS per Region. This quota cannot be changed.</p> <ul style="list-style-type: none">• DisableInsightRules• EnableInsightRules
Contributor Insights rules	100 rules per Region per account. You can request a quota increase .
Custom metrics	No quota.

Resource	Default quota
Dashboards	<p>Up to 500 widgets per dashboard. Up to 500 metrics per dashboard widget. Up to 2500 metrics per dashboard, across all widgets.</p> <p>These quotas include all metrics retrieved for use in metric math functions, even if those metrics are not displayed on the graph.</p> <p>These quotas cannot be changed.</p>
DescribeAlarms	<p>9 transactions per second (TPS) per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
DeleteAlarms request DescribeAlarmHistory request DisableAlarmActions request EnableAlarmActions request SetAlarmState request	<p>3 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled.</p> <p>These quotas cannot be changed.</p>
DescribeAlarmsForMetric request	<p>9 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>This quotas cannot be changed.</p>
DeleteDashboards request GetDashboard request ListDashboards request PutDashboard request	<p>10 TPS per Region for each of these operations. The maximum number of operation requests you can make per second without being throttled.</p> <p>These quotas cannot be changed.</p>

Resource	Default quota
PutAnomalyDetector DescribeAnomalyDetectors	10 TPS per Region. The maximum number of operation requests you can make per second without being throttled.
DeleteAnomalyDetector	5 TPS per Region. The maximum number of operation requests you can make per second without being throttled.
Dimensions	30/metric. This quota cannot be changed.

Resource	Default quota
GetMetricData	<p>10 TPS per Region for operations that include Metrics Insights queries. For operations that do not include Metrics Insights queries, the quota is 50 TPS per Region. This is the maximum number of operation requests you can make per second without being throttled. You can request a quota increase.</p> <p>For <code>GetMetricData</code> operations that include a Metrics Insights query, the quota is 4,300,000 Datapoints Per Second (DPS) for the most recent 3 hours. This is calculated against the total number of data points scanned by the query (which can include no more than 10,000 metrics.)</p> <p>180,000 Datapoints Per Second (DPS) if the <code>StartTime</code> used in the API request is less than or equal to three hours from current time. 396,000 DPS if the <code>StartTime</code> is more than three hours from current time. This is the maximum number of datapoints you can request per second using one or more API calls without being throttled. This quota cannot be changed.</p> <p>The DPS is calculated based on estimated data points, not actual data points. The data point estimate is calculated using the requested time range, period, and retention period. This means that if the actual data points in the requested metrics are sparse or empty, throttling still occurs if the estimated data points exceed the quota. The DPS quota is per-Region.</p>

Resource	Default quota
GetMetricData	<p>A single <code>GetMetricData</code> call can include the following:</p> <ul style="list-style-type: none">• As many as 500 <code>MetricDataQuery</code> structures.• As many as 100 <code>SERVICE_QUOTA()</code> functions.• As many as 100 <code>SEARCH()</code> functions.• As many as 5 <code>LAMBDA()</code> functions. <p>These quotas can't be changed.</p>
GetMetricStatistics	<p>400 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
GetMetricWidgetImage	<p>Up to 500 metrics per image. This quota cannot be changed.</p> <p>20 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
ListMetrics	<p>25 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
Metric data values	<p>The value of a metric data point must be within the range of -2^{360} to 2^{360}. Special values (for example, NaN, +Infinity, -Infinity) are not supported. This quota cannot be changed.</p>

Resource	Default quota
MetricDatum items	1000/ PutMetricData request. A MetricDatum object can contain a single value or a StatisticSet object representing many values. This quota cannot be changed.
Metrics	10/month/customer for free.
Metrics Insights queries	<p>A single query can process no more than 10,000 metrics. This means that if the SELECT, FROM, and WHERE clauses would match more than 10,000 metrics, only the first 10,000 of these metrics that are found will be processed by the query.</p> <p>A single query can return no more than 500 time series.</p> <p>You can query only the most recent three hours of data</p>
Metrics Insights query generator requests	As many as five concurrent natural-language generated requests.
Observability Access Manager (OAM) API request rates.	<p>1 TPS per Region for PutSinkPolicy.</p> <p>10 TPS per Region for each other CloudWatch OAM API.</p> <p>These quotas reflect the maximum number of operation requests you can make per second without being throttled.</p> <p>These quotas can't be changed.</p>
OAM source account links	<p>Each source account can be linked to as many as 5 monitoring accounts</p> <p>This quota can't be changed.</p>
OAM sinks	<p>1 sink per Region per account</p> <p>This quota can't be changed.</p>

Resource	Default quota
PutCompositeAlarm request	<p>3 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
PutMetricAlarm request	<p>3 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>You can request a quota increase.</p>
PutMetricData request	<p>1MB for HTTP POST requests. PutMetricData can handle 500 transactions per second (TPS), which is the maximum number of operation requests that you can make per second without being throttled. PutMetricData can handle 1,000 metrics per request.</p> <p>You can request a quota increase.</p>
Amazon SNS email notifications	1,000/month/customer for free.
Synthetics Groups	<p>20 per account.</p> <p>This quota can't be changed.</p>
TagResource	<p>20 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>This quota can't be changed.</p>
UntagResource	<p>20 TPS per Region. The maximum number of operation requests you can make per second without being throttled.</p> <p>This quota can't be changed.</p>

Document history

The following table describes important changes in each release of the *Amazon CloudWatch User Guide*, beginning in June 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
CloudWatch Metrics Insights support for natural language query generation is generally available.	CloudWatch Metrics Insights supports natural language query to generate and update queries. For more information, see Use natural language to generate and update CloudWatch Metric Insights queries.	June 10, 2024
CloudWatch Application Signals is generally available	CloudWatch Application Signals is now generally available. Use Application Signals to instrument your applications on AWS so that you can monitor current application health, create service level objectives (SLOs), and track long-term application performance against your business objectives. For more information, see Application Signals.	June 10, 2024
New CloudWatchApplicationSignalsReadOnlyAccess IAM policy	CloudWatch added a CloudWatchApplicationSignalsReadOnlyAccess policy to add read only actions and resources available in the CloudWatch	June 7, 2024

	h console for Application Signals.	
New CloudWatchApplicationSignalsFullAccess policy	CloudWatch added a CloudWatchApplicationSignalsFullAccess policy to add actions and resources available in the CloudWatch console for Application Signals.	June 7, 2024
CloudTrail now captures API activities related to CloudWatch data plane operations.	CloudTrail now logs events in CloudTrail for GetMetricData and GetMetricWidgetImage API activities.	June 6, 2024
CloudWatch Application Signals service map supports canary, RUM clients and AWS service dependency groupings :	The Application Signals preview release has added default groupings in the service map for canaries, RUM clients, and AWS service dependencies of the same type. This change reduces the number of icons in the service map default view to make it easier to view and navigate.	May 21, 2024

[CloudWatchReadOnlyAccess policy updated](#)

CloudWatch changed the scope of a permission in **CloudWatchReadOnlyAccess**. The scope of the policy added the `application-signal` `s:BatchGet*` , `application-signal` `s:Get*` , and `application-signal` `s:List*` actions so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services. CloudWatch also added an `iam:GetRole` action so that users can check if Application Signals is set up.

May 17, 2024

[CloudWatchFullAccessV2 policy updated](#)

CloudWatch changed the scope of a permission in **CloudWatchFullAccessV2**. The scope of the policy added the `application-signal` `s:*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.

May 17, 2024

[Lambda Insights supports AWS GovCloud \(US-East\) and AWS GovCloud \(US-West\)](#)

CloudWatch Lambda Insights has added support for the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions.

April 29, 2024

[CloudWatch cross-account observability supports resource filters](#)

You can now create filters to specify which metric namespaces and log groups are shared from the source account to the monitoring account, when you create the link between the accounts.

April 26, 2024

[CloudWatch Application Signals updates](#)

The Application Signals preview release has added three features. Application Signals now supports Python applications. It offers a simpler enablement process for applications on Amazon EKS architectures. And it includes new configurations that you can use to manage the cardinality of metrics that are collected.

April 26, 2024

[CloudWatch Container Insights with enhanced observability for Amazon EKS can collect AWS Elastic Fabric Adapter \(EFA\) metrics](#)

You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect AWS Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters.

April 23, 2024

[Updated IAM policy](#)

CloudWatch updated the **CloudWatchApplicationSignalsServiceRolePolicy** policy. The scoping of the `logs:StartQuery` and `logs:GetQueryResults` permissions in this policy was changed to add `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` and `"arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"` to enable Application Signals on more architectures. This policy is attached to the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role.

April 18, 2024

[Internet Monitor provides a global internet weather map to authenticated AWS customers](#)

Amazon CloudWatch Internet Monitor now displays a global internet weather map that is available in the console to all authenticated AWS customers. To view the map, in the Amazon CloudWatch console, navigate to Internet Monitor.

April 16, 2024

[CloudWatch Container Insights with enhanced observability for Amazon EKS can collect AWS Neuron metrics](#)

You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect AWS Neuron metrics from Amazon EKS clusters.

April 16, 2024

[CloudWatch Application Signals adds a **Service overview** tab and more metrics to aid in diagnostics](#)

A new **Service overview** tab displays an overview of your service, including number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, and top operations and dependencies. You can also now view X-Ray traces that are correlated with issues including faults, errors, and latency problems.

April 16, 2024

[CloudWatch Container Insights with enhanced observability for Amazon EKS adds support for Windows](#)

You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect metrics from Windows worker nodes on Amazon EKS clusters.

April 10, 2024

[CloudWatchApplicationSignalsServiceRolePolicy IAM policy updated](#)

CloudWatch changed the scope of a permission in **CloudWatchApplicationSignalsServiceRolePolicy**. The scope of the `cloudwatch:GetMetricData` permission was changed to `*` so that Application Signals can retrieve metrics from sources in linked accounts.

April 8, 2024

[Amazon CloudWatch Internet Monitor now supports cross-account observability](#)

You can now use Internet Monitor cross-account observability to monitor your applications that span multiple AWS accounts within a single AWS Region.

March 29, 2024

[CloudWatchAgentServerPolicy and CloudWatchAgentAdminPolicy policies updated](#)

CloudWatch added permissions to both the CloudWatchAgentServerPolicy and CloudWatchAgentAdminPolicy policies to allow the CloudWatch agent to publish X-Ray traces and modify log group retention periods. In both policies, the `xray:PutTraceSegments` , `xray:PutTelemetryRecords` , `xray:GetSamplingRules` , `xray:GetSamplingTargets` , `xray:GetSamplingStatisticSummaries` and `logs:PutRetentionPolicy` permissions were added

February 12, 2024

[New service linked role and IAM policy for CloudWatch Network Monitor](#)

CloudWatch added a new service-linked role, called **AWSServiceRoleForNetworkMonitor**. CloudWatch added this new service-linked role to allow you to create monitors to fetch network metrics between source subnets and destination IP addresses. The new **CloudWatchNetworkMonitorServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch network metrics on your behalf.

December 22, 2023

[CloudWatch releases Amazon CloudWatch Network Monitor](#)

CloudWatch released a new feature, Amazon CloudWatch Network Monitor. This is a new active network monitoring service that identifies if a network issue exists within the AWS network or your own company network.

December 22, 2023

[CloudWatchReadOnlyAccess policy updated](#)

CloudWatch added existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM and new read-only permissions for CloudWatch Application Signals to **CloudWatchReadOnlyAccess** so that users with this policy can triage and diagnose service health issues as reported by CloudWatch Application Signals. The `cloudwatch:GenerateQuery` permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.

December 5, 2023

[CloudWatchFullAccessV2 policy updated](#)

CloudWatch added existing permissions to **CloudWatchFullAccessV2** for CloudWatch Synthetics, X-Ray, and CloudWatch RUM, and added new permissions for CloudWatch Application Signals so that users with this policy can fully manage Application Signals to triage and diagnose issues with service health.

December 5, 2023

[New service-linked role and new IAM policy](#)

CloudWatch added a new service-linked role, called **AWSServiceRoleForCloudWatchApplicationSignals**. CloudWatch added this new service-linked role to allow CloudWatch Application Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals. The new **CloudWatchApplicationSignalsServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant AWS services.

November 30, 2023

[CloudWatch launches Preview release of Application Signals](#)

CloudWatch Application Signals is in Preview. Use Application Signals to instrument your applications on AWS so that you can monitor current application health, create service level objectives (SLOs), and track long-term application performance against your business objectives. For more information, see [Application Signals](#).

November 30, 2023

[CloudWatch adds support for querying other data sources](#)

You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources. For more information, see [Querying metrics from other data sources](#).

November 26, 2023

[CloudWatch Metrics Insights supports natural language query generation](#)

CloudWatch Metrics Insights supports natural language query to generate and update queries. For more information, see [Use natural language to generate and update CloudWatch Metric Insights queries](#).

November 26, 2023

[CloudWatch releases Container Insights with enhanced observability for Amazon EKS](#)

CloudWatch released a new version of Container Insights. This version supports enhanced observability for Amazon EKS clusters and can collect more detailed metrics from clusters running Amazon EKS. After installation, it automatically collects detailed infrastructure telemetry and container logs for your Amazon EKS clusters. You can then use curated, immediately usable dashboards to drill down into application and infrastructure telemetry.

November 6, 2023

[CloudWatch metric streams adds quick partner setup](#)

CloudWatch metric streams now provides a quick partner setup option, which you can use to quickly set up a metric stream to some third-party providers.

October 17, 2023

[CloudWatch releases alarm recommendations](#)

CloudWatch Synthetics now provides alarm recommendations for metrics from other AWS services. These recommendations can help you identify the metrics that you should set alarms for to follow best practices for monitoring these services.

October 16, 2023

CloudWatch Synthetics releases runtime syn-nodejs-puppeteer-6.0	CloudWatch Synthetics released runtime syn-nodejs-puppeteer-6.0 .	September 26, 2023
Adds Amazon CloudWatch Application Insights support for cross-account applications	You can now share CloudWatch Application Insights applications across account boundaries.	September 26, 2023
New service-linked role and new IAM policy	CloudWatch added a new service-linked role, called AWSServiceRoleForCloudWatchMetrics_DbPerfInsights . CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming, anomaly detection, and snapshotting. The new AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 20, 2023
Adds new metric math function	CloudWatch added a new metric math function, DB_PERF_INSIGHTS , that you can use to fetch Performance Insights metrics from AWS database services for alarming, anomaly detection, and snapshotting.	September 20, 2023

[CloudWatchReadOnlyAccess policy updated](#)

CloudWatch added the `application-autoscaling:DescribeScalingPolicies` permission to **CloudWatchReadOnlyAccess** so that users with this policy can access information about Application Auto Scaling policies.

September 14, 2023

[CloudWatch agent added support for AL2023](#)

The CloudWatch agent supports AL2023.

August 8, 2023

[New managed IAM policy, CloudWatchFullAccessV2](#)

CloudWatch added a new policy **CloudWatchFullAccessV2**. This policy grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling.

August 1, 2023

[Updated service linked role for Amazon CloudWatch Internet Monitor – update to an existing policy](#)

Adds new permissions, `elasticloadbalancing:DescribeLoadBalancers` and `ec2:DescribeNetworkInterfaces`, to the service linked role for Internet Monitor, to support monitoring traffic for specific Network Load Balancer resources.

July 25, 2023

[Added support for Network Load Balancer resources in Amazon CloudWatch Internet Monitor](#)

Adds support for creating a monitor in Internet Monitor with specific Network Load Balancer resources, to provide more granular levels of observability for your application.

July 25, 2023

[Dashboard variables feature](#)

CloudWatch released *dashboard variables*, which you can use to create flexible dashboards that can quickly display different contents depending on how you set one input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different AWS Regions. For more information, see [Create flexible dashboards with dashboard variables](#).

June 28, 2023

[Internet Monitor now supports customizing the threshold for health events](#)

Internet Monitor added the ability to customize the threshold for when a global performance score or availability score triggers a health event. For more information, see [Tracking real-time performance and availability in Amazon CloudWatch Internet Monitor](#).

June 26, 2023

[Internet Monitor now supports all commercial Regions](#)

Internet Monitor added seven new AWS Regions and now supports all commercial Regions.

June 19, 2023

[New Lambda Insights extension versions](#)

CloudWatch added the 1.0.229.0 version of the Lambda Insights extension for both x86-64 platforms and ARM64 platforms. For more information, see [Available versions of the Lambda Insights extension](#).

June 12, 2023

[CloudWatchReadOnlyAccess policy updated](#)

CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `logs:StartLiveTail` and `logs:StopLiveTail` permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see [Use live tail to view logs in near real time](#).

June 6, 2023

[CloudWatch RUM adds support for custom metrics](#)

You can use CloudWatch RUM app monitors to create custom metrics and send them to CloudWatch and CloudWatch Evidently. This feature includes an update to the **AmazonCloudWatchRUMServiceRolePolicy** managed IAM policy. In that policy, a condition key was changed so that CloudWatch RUM can send custom metrics to custom metric namespaces.

February 9, 2023

[New and updated managed policies for CloudWatch](#)

To support CloudWatch cross-account observability, the `CloudWatchFullAccess` and `CloudWatchReadOnlyAccess` policies have been updated, and the following new managed policies have been added: `CloudWatchCrossAccountSharingConfiguration` , `OAMFullAccess` , and `OAMReadOnlyAccess` . For more information, see [CloudWatch updates to AWS managed policies](#).

February 7, 2023

[CloudWatch Application Insights service-linked role policy updates — update to an existing policy.](#)

CloudWatch Application Insights updated an existing AWS service-linked role policy.

December 19, 2022

Amazon CloudWatch Application Insights support for containerized applications and microservices from the Container Insights console.	You can display CloudWatch Application Insights detected problems for Amazon ECS and Amazon EKS on your Container Insights dashboard.	November 17, 2021
Amazon CloudWatch Application Insights monitoring for SAP HANA databases.	You can monitor SAP HANA databases with Application Insights.	November 15, 2021
Amazon CloudWatch Application Insights support for monitoring all resources in an account.	You can onboard and monitor all resources in an account.	September 15, 2021
Amazon CloudWatch Application Insights support for Amazon FSx.	You can monitor metrics retrieved from Amazon FSx.	August 31, 2021
SDK Metrics is no longer supported.	CloudWatch SDK Metrics is no longer supported.	August 25, 2021
Amazon CloudWatch Application Insights support for setting up container monitoring.	You can monitor containers using best practices with Amazon CloudWatch Application Insights.	May 18, 2021
Metric streams is generally available	You can use metric streams to continually stream CloudWatch metrics to a destination of your choice. For more information, see Metric streams in the <i>Amazon CloudWatch User Guide</i> .	March 31, 2021

Amazon CloudWatch Application Insights monitoring for Oracle databases on Amazon RDS and Amazon EC2.	You can monitor metrics and logs retrieved from Oracle with Amazon CloudWatch Application Insights.	January 16, 2021
Lambda Insights is generally available	CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. For more information, see Using Lambda Insights in the <i>Amazon CloudWatch User Guide</i> .	December 3, 2020
Amazon CloudWatch Application Insights monitoring for Prometheus JMX exporter metrics.	You can monitor metrics retrieved from Prometheus JMX exporter with Amazon CloudWatch Application Insights.	November 20, 2020
CloudWatch Synthetics releases new runtime version	CloudWatch Synthetics has released a new runtime version. For more information, see Canary Runtime Versions in the <i>Amazon CloudWatch User Guide</i> .	September 11, 2020
Amazon CloudWatch Application Insights monitoring for PostgreSQL on Amazon RDS and Amazon EC2.	You can monitor applications built with PostgreSQL running on Amazon RDS or Amazon EC2.	September 11, 2020

[CloudWatch supports dashboard sharing](#)

You can now share CloudWatch dashboards with people outside of your organization and AWS account. For more information, see [Sharing CloudWatch Dashboards](#) in the *Amazon CloudWatch User Guide*.

September 10, 2020

[Set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights](#)

You can use the documentation tutorial to help you to set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights.

August 19, 2020

[AWS CloudFormation support for Amazon CloudWatch Application Insights applications.](#)

You can add CloudWatch Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from AWS CloudFormation templates.

July 30, 2020

[Amazon CloudWatch Application Insights monitoring for Aurora for MySQL database clusters.](#)

You can monitor Aurora for MySQL database clusters (RDS Aurora) with Amazon CloudWatch Application Insights.

July 2, 2020

[CloudWatch Contributor Insights general availability](#)

CloudWatch Contributor Insights is now generally available. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see [Using Contributor Insights to Analyze High-Cardinality Data](#) in the *Amazon CloudWatch User Guide*.

April 2, 2020

[CloudWatch Synthetics public preview](#)

CloudWatch Synthetics is now in public preview. It enables you to create canaries to monitor your endpoints and APIs. For more information, see [Using Canaries](#) in the *Amazon CloudWatch User Guide*.

November 25, 2019

[CloudWatch Contributor Insights public preview](#)

CloudWatch Contributor Insights is now in public preview. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see [Using Contributor Insights to Analyze High-Cardinality Data](#) in the *Amazon CloudWatch User Guide*.

November 25, 2019

[CloudWatch launches ServiceLens feature](#)

ServiceLens enhances the observability of your services and applications by enabling you to integrate traces, metrics, logs, and alarms into one place. ServiceLens integrates CloudWatch with AWS X-Ray to provide an end-to-end view of your application.

November 21, 2019

[Use CloudWatch to proactively manage your AWS service quotas](#)

You can use CloudWatch to proactively manage your AWS service quotas. CloudWatch usage metrics provide visibility into your account's usage of resources and API operations. For more information, see [Service Quotas Integration and Usage Metrics](#) in the *Amazon CloudWatch User Guide*.

November 19, 2019

[CloudWatch sends events when alarms change state](#)

CloudWatch now sends an event to Amazon EventBridge when any CloudWatch alarm changes state. For more information, see [Alarm Events and EventBridge](#) in the *Amazon CloudWatch User Guide*.

October 8, 2019

[Container Insights](#)

CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see [Using Container Insights](#) in the *Amazon CloudWatch User Guide*.

August 30, 2019

[Updates for Container Insights preview metrics on Amazon EKS and Kubernetes](#)

The Container Insights on Amazon EKS and Kubernetes public preview has been updated. InstanceId is now included as a dimension to the cluster EC2 instances . This allows alarms that have been created on these metrics to trigger the following EC2 actions: Stop, Terminate, Reboot, or Recover. Additionally, pod and service metrics are now reported by Kubernetes namespace to simplify the monitoring and alarming on metrics by namespace.

August 19, 2019

[Updates for AWS Systems Manager OpsCenter integration](#)

Updates on how CloudWatch Application Insights integrate with Systems Manager OpsCenter.

August 7, 2019

[CloudWatch usage metrics](#)

CloudWatch usage metrics help you track the usage of your CloudWatch resources and stay within your service limits. For more information, see <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html>.

August 6, 2019

[CloudWatch Container Insights public preview](#)

CloudWatch Container Insights is now in public preview. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see [Using Container Insights](#) in the *Amazon CloudWatch User Guide*.

July 9, 2019

[CloudWatch Anomaly Detection public preview](#)

CloudWatch anomaly detection is now in public preview. CloudWatch applies machine-learning algorithms to a metric's past data to create a model of the metric's expected values. You can use this model for visualization and for setting alarms. For more information, see [Using CloudWatch Anomaly Detection](#) in the *Amazon CloudWatch User Guide*.

July 9, 2019

[CloudWatch Application Insights for .NET and SQL Server](#)

CloudWatch Application Insights for .NET and SQL Server facilitates observability for .NET and SQL Server applications. It can help you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications.

June 21, 2019

[CloudWatch agent section re-organized](#)

The CloudWatch agent documentation has been rewritten to improve clarity, especially for customers using the command line to install and configure the agent. For more information, see [Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent](#) in the *Amazon CloudWatch User Guide*.

March 28, 2019

[SEARCH function added to metric math expressions](#)

You can now use a SEARCH function in metric math expressions. This enables you to create dashboards that update automatically as new resources are created that match the search query. For more information, see [Using Search Expressions in Graphs](#) in the *Amazon CloudWatch User Guide*.

March 21, 2019

[AWS SDK Metrics for Enterprise Support](#)

SDK Metrics helps you assess the health of your AWS services and diagnose latency caused by reaching your account usage limits or by a service outage. For more information, see [Monitor Applications Using AWS SDK Metrics](#) in the *Amazon CloudWatch User Guide*.

December 11, 2018

Alarms on math expressions	CloudWatch supports creating alarms based on metric math expressions. For more information, see Alarms on Math Expressions in the <i>Amazon CloudWatch User Guide</i> .	November 20, 2018
New CloudWatch console home page	Amazon has created a new home page in the CloudWatch console, which automatically displays key metrics and alarms for all the AWS services you are using. For more information, see Getting Started with Amazon CloudWatch in the <i>Amazon CloudWatch User Guide</i> .	November 19, 2018
AWS CloudFormation templates for the CloudWatch Agent	Amazon has uploaded AWS CloudFormation templates that you can use to install and update the CloudWatch agent. For more information, see Install the CloudWatch Agent on New Instances Using AWS CloudFormation in the <i>Amazon CloudWatch User Guide</i> .	November 9, 2018

[Enhancements to the CloudWatch Agent](#)

The CloudWatch agent has been updated to work with both the StatsD and collectd protocols. It also has improved cross-account support. For more information, see [Retrieve Custom Metrics with StatsD](#), [Retrieve Custom Metrics with collectd](#), and [Sending Metrics and Logs to a Different AWS Account](#) in the *Amazon CloudWatch User Guide*.

September 28, 2018

[Support for Amazon VPC endpoints](#)

You can now establish a private connection between your VPC and CloudWatch. For more information, see [Using CloudWatch with Interface VPC Endpoints](#) in the *Amazon CloudWatch User Guide*.

June 28, 2018

The following table describes important changes to the *Amazon CloudWatch User Guide* before June 2018.

Change	Description	Release date
Metric math	You can now perform math expressions on CloudWatch metrics, producing new time series that you can add to graphs on your dashboard. For more information, see Use metric math .	April 4, 2018
"M out of N" alarms	You can now configure an alarm to trigger based on "M out of N" datapoints in any alarm evaluation period.	December 8, 2017

Change	Description	Release date
	n interval. For more information, see Evaluating an alarm .	
CloudWatch agent	A new unified CloudWatch agent was released. You can use the unified multi-platform agent to collect custom both system metrics and log files from Amazon EC2 instances and on-premises servers. The new agent supports both Windows and Linux and enables customization of metrics collected, including sub-resource metrics such as per-CPU core. For more information, see Collect metrics, logs, and traces with the CloudWatch agent .	September 7, 2017
NAT gateway metrics	Added metrics for Amazon VPC NAT gateway.	September 7, 2017
High-resolution metrics	You can now optionally set up custom metrics as high-resolution metrics, with a granularity of as low as one second. For more information, see High-resolution metrics .	July 26, 2017
Dashboard APIs	You can now create, modify, and delete dashboards using APIs and the AWS CLI. For more information, see Creating a CloudWatch dashboard .	July 6, 2017
AWS Direct Connect metrics	Added metrics for AWS Direct Connect.	June 29, 2017
Amazon VPC VPN metrics	Added metrics for Amazon VPC VPN.	May 15, 2017
AppStream 2.0 metrics	Added metrics for AppStream 2.0.	March 8, 2017

Change	Description	Release date
CloudWatch console color picker	You can now choose the color for each metric on your dashboard widgets. For more information, see Edit a graph on a CloudWatch dashboard .	February 27, 2017
Alarms on dashboards	Alarms can now be added to dashboards. For more information, see Add or remove an alarm widget from a CloudWatch dashboard .	February 15, 2017
Added metrics for Amazon Polly	Added metrics for Amazon Polly.	December 1, 2016
Added metrics for Amazon Managed Service for Apache Flink	Added metrics for Amazon Managed Service for Apache Flink.	December 1, 2016
Added support for percentile statistics	You can specify any percentile, using up to two decimal places (for example, p95.45). For more information, see Percentiles .	November 17, 2016
Added metrics for Amazon Simple Email Service	Added metrics for Amazon Simple Email Service.	November 2, 2016
Updated metrics retention	Amazon CloudWatch now retains metrics data for 15 months instead of 14 days.	November 1, 2016
Updated metrics console interface	The CloudWatch console is updated with improvements to existing functionality and new functionality.	November 1, 2016
Added metrics for Amazon Elastic Transcoder	Added metrics for Amazon Elastic Transcoder.	September 20, 2016

Change	Description	Release date
Added metrics for Amazon API Gateway	Added metrics for Amazon API Gateway.	September 9, 2016
Added metrics for AWS Key Management Service	Added metrics for AWS Key Management Service.	September 9, 2016
Added metrics for the new Application Load Balancers supported by Elastic Load Balancing	Added metrics for Application Load Balancers.	August 11, 2016
Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2	Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2.	March 23, 2016
Added new metrics for Amazon EC2 Spot fleet	Added new metrics for Amazon EC2 Spot fleet.	March 21, 2016
Added new CloudWatch Logs metrics	Added new CloudWatch Logs metrics.	March 10, 2016

Change	Description	Release date
Added Amazon OpenSearch Service and AWS WAF metrics and dimensions	Added Amazon OpenSearch Service and AWS WAF metrics and dimensions.	October 14, 2015
Added support for CloudWatch dashboards	Dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those that are spread out across different Regions. For more information, see Using Amazon CloudWatch dashboards .	October 8, 2015
Added AWS Lambda metrics and dimensions	Added AWS Lambda metrics and dimensions.	September 4, 2015
Added Amazon Elastic Container Service metrics and dimensions	Added Amazon Elastic Container Service metrics and dimensions.	August 17, 2015
Added Amazon Simple Storage Service metrics and dimensions	Added Amazon Simple Storage Service metrics and dimensions.	July 26, 2015
New feature: Reboot alarm action	Added the reboot alarm action and new IAM role for use with alarm actions. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance .	July 23, 2015
Added Amazon WorkSpaces metrics and dimensions	Added Amazon WorkSpaces metrics and dimensions.	April 30, 2015

Change	Description	Release date
Added Amazon Machine Learning metrics and dimensions	Added Amazon Machine Learning metrics and dimensions.	April 9, 2015
New feature: Amazon EC2 instance recovery alarm actions	Updated alarm actions to include new EC2 instance recovery action. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance .	March 12, 2015
Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions	Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions.	March 6, 2015
Added Amazon Simple Workflow Service metrics and dimensions	Added Amazon Simple Workflow Service metrics and dimensions.	May 9, 2014
Updated guide to add support for AWS CloudTrail	Added a new topic to explain how you can use AWS CloudTrail to log activity in Amazon CloudWatch. For more information, see Logging Amazon CloudWatch API and console operations with AWS CloudTrail .	April 30, 2014

Change	Description	Release date
Updated guide to use the new AWS Command Line Interface (AWS CLI)	<p>The AWS CLI is a cross-service CLI with a simplified installation, unified configuration, and consistent command line syntax. The AWS CLI is supported on Linux/Unix, Windows, and Mac. The CLI examples in this guide have been updated to use the new AWS CLI.</p> <p>For information about how to install and configure the new AWS CLI, see Getting Set Up with the AWS CLI Interface in the <i>AWS Command Line Interface User Guide</i>.</p>	February 21, 2014
Added Amazon Redshift and AWS OpsWorks metrics and dimensions	Added Amazon Redshift and AWS OpsWorks metrics and dimensions.	July 16, 2013
Added Amazon Route 53 metrics and dimensions	Added Amazon Route 53 metrics and dimensions.	June 26, 2013
New feature: Amazon CloudWatch Alarm Actions	Added a new section to document Amazon CloudWatch alarm actions, which you can use to stop or terminate an Amazon Elastic Compute Cloud instance. For more information, see Create alarms to stop, terminate, reboot, or recover an EC2 instance .	January 8, 2013
Updated EBS metrics	Updated the EBS metrics to include two new metrics for Provisioned IOPS volumes.	November 20, 2012

Change	Description	Release date
New billing alerts	You can now monitor your AWS charges using Amazon CloudWatch metrics and create alarms to notify you when you have exceeded the specified threshold. For more information, see Create a billing alarm to monitor your estimated AWS charges .	May 10, 2012
New metrics	You can now access six new Elastic Load Balancing metrics that provide counts of various HTTP response codes.	October 19, 2011
New feature	You can now access metrics from Amazon EMR.	June 30, 2011
New feature	You can now access metrics from Amazon Simple Notification Service and Amazon Simple Queue Service.	July 14, 2011
New Feature	Added information about using the <code>PutMetricData</code> API to publish custom metrics. For more information, see Publish custom metrics .	May 10, 2011
Updated metrics retention	Amazon CloudWatch now retains the history of an alarm for two weeks rather than six weeks. With this change, the retention period for alarms matches the retention period for metrics data.	April 7, 2011
New feature	Added ability to send Amazon Simple Notification Service or Auto Scaling notifications when a metric has crossed a threshold. For more information, see Alarms .	December 2, 2010
New feature	A number of CloudWatch actions now include the <code>MaxRecords</code> and <code>NextToken</code> parameters, which enable you to control pages of results to display.	December 2, 2010

Change	Description	Release date
New feature	This service now integrates with AWS Identity and Access Management (IAM).	December 2, 2010