User Guide

AWS DevOps Agent

AWS DevOps Agent: User Guide

Table of Contents

About AWS DevOps Agent

AWS DevOps Agent is a frontier agent that resolves and proactively prevents incidents, continuously improving reliability and performance.

AWS DevOps Agent investigates incidents and identifies operational improvements as an experienced DevOps engineer.

The agent works by:

- Learning your resources and their relationships.
- Working with your observability tools, skills, code repositories, and CI/CD pipelines.
- Correlating telemetry, code, and deployment data to understand relationships between your application resources.
- Supporting applications in multicloud and hybrid environments.

Key features

AWS DevOps Agent provides comprehensive incident response and prevention capabilities through the following features:

Always-on, autonomous incident response

AWS DevOps Agent autonomously investigates issues the moment they occur:

- **Automated incident investigation** – Begins investigating immediately when an alert or support ticket comes in
- **AWS DevOps Agent Chat** - Query your infrastructure, analyze system health, and guide investigations using natural language throughout the DevOps Agent Space web app. Chat provides context-aware responses based on the page you're viewing, whether asking about resources in Topology, steering an investigation, or filtering recommendations in Prevention.
- **Detailed mitigation plans** – Provides specific actions to resolve incidents, validate success, and revert changes if needed
- **Automated incident coordination** – Routes observations, findings, and mitigation steps through your preferred communication channels like Slack and ServiceNow

- **AWS Support integration** – Create AWS Support cases directly from an investigation with immediate context provided to AWS Support experts

Prevent future incidents

AWS DevOps Agent analyzes patterns across historical incidents to help you move from reactive firefighting to proactive operational improvement:

- **Targeted recommendations** – Delivers specific, actionable improvements that strengthen four key areas: observability (monitoring, alerting, logging), infrastructure optimization (autoscaling, capacity tuning), and deployment pipeline enhancement (testing, validation).
- **Continuous learning** – Refines recommendations based on your team's feedback

Get more from your DevOps tools

AWS DevOps Agent integrates with your existing tools without changing your workflows:

- **Application resource mapping** – Builds a topology graph of your application resources and their relationships
- **Built-in integrations** – Works with popular observability tools (Amazon CloudWatch, Dynatrace, Datadog, New Relic, and Splunk), code repositories, and CI/CD pipelines (GitHub Actions and repositories, GitLab workflows and repositories)
- **Custom tool integration** – Extend capabilities by connecting to your own Model Context Protocol (MCP) servers for additional tools
- **Conversational infrastructure queries** – Use natural language to query AWS resources, system metrics and alarm status without navigating multiple consoles. Chat understands context and maintains conversation history for follow-up questions.

How AWS DevOps Agent works

AWS DevOps Agent operates through a dual-console architecture. Administrators use the AWS Management Console to create and manage Agent Spaces, configure integrations, and set up access controls. Operations teams use the AWS DevOps Agent web app for day-to-day incident response and investigation activities. The web app is where operators can interact with agent investigations, browse cross-account application topology, and learn about preventative

improvements to observability, code, pipelines, and infrastructure architectures. To learn more, see the section called "Proactive incident prevention".

The service is organized around Agent Spaces, which are logical containers that define what AWS DevOps Agent can access and investigate. Each Agent Space contains your AWS account configurations, third-party tool integrations, and access permissions. To learn more, see the section called "What are DevOps Agent Spaces?".

AWS DevOps Agent automatically builds an application topology that maps your resources and their relationships. This topology helps the service understand your application architecture during investigations. To learn more, see the section called "What is a DevOps Agent Topology?".

Benefits

- **Reduce mean time to resolution (MTTR)** – Autonomous investigation starts immediately, accelerating incident resolution from hours to minutes
- **Prevent recurring incidents** – Targeted recommendations address root causes and strengthen system resilience
- **Improve operational efficiency** – Free your team from repetitive investigation tasks to focus on innovation
- **Work within existing workflows** – Integrates with your existing tools and processes without disruption

What is a DevOps Agent Web App?

AWS DevOps Agent uses a dual-console architecture that separates administrative functions from day-to-day operational activities. This design enables administrators to configure the service while operations teams focus on incident response and prevention.

Consoles

AWS DevOps Agent provides two distinct interfaces:

- **AWS Management Console** – Administrators use the AWS Management Console to set up and manage AWS DevOps Agent. In this console, you can the section called "Creating an Agent Space" connect AWS services and third-party tools, and manage access permissions for your organization.

- **DevOps Agent web app -** Operations teams use DevOps Agent Space web apps for daily incident response activities. This standalone application provides an interface where on-call engineers can launch investigations, interact with the agent through natural language chat, view application topologies, and review incident prevention recommendations.

Web app capabilities

The DevOps Agent web app provides the following primary capabilities:

- **Incident Response –** The page is where you create and track incident investigations as well as generate mitigation plans to resolve incidents.
- **Incident Prevention –** Found in the Prevention page, this is where you will find recommendations to improve your observability posture, delivery processes, and infrastructure architecture to prevent future incidents.
- **Topology –** The Topology page provides an interactive visual representation of the account resources and their relationships across all of the resources in the connected accounts. You can view the topology with different levels of detail using the "Show" dropdown to switch between System, Container, and Resource views.
- **Skills** – Modular instruction sets that extend AWS DevOps Agent with specialized capabilities. Skills contain domain knowledge, investigation methodologies, and tool configurations tailored to your infrastructure. Each skill enables specific tools and provides progressive disclosure of instructions only when relevant to the investigation.
- **Natural language Chat interface** – Available throughout the web app, Chat is an AI-powered conversational assistant that enables you to query your infrastructure, analyze system health, and work with investigations using natural language. Chat provides context-aware responses based on the page you're viewing.

Authentication

AWS DevOps Agent supports flexible authentication methods to accommodate different organizational requirements:

- **IAM Identity Center integration (User access)** – Organizations can use AWS Identity Center (IAM Identity Center) to centrally manage user access to the DevOps Agent Space web apps. IAM Identity Center can federate with external identity providers through standard OIDC and

SAML protocols, including providers like Okta, Ping Identity, and Microsoft Entra ID. This method supports multi-factor authentication from your identity provider.

- **External identity provider (IdP) authentication** – Organizations can connect an OIDC-compatible identity provider, such as Okta or Microsoft Entra ID, directly to the Agent Space web app without requiring IAM Identity Center. Users sign in with their corporate credentials through the IdP. For setup instructions, see the section called “Setting Up External Identity Provider (IdP) Authentication”.
- **IAM authentication link (Admin access)** – An alternative method provides direct access to the web app from the AWS Management Console using your existing console session. This option is useful before implementing full Identity Center integration, but sessions are limited to 10 minutes.

What are DevOps Agent Spaces?

A DevOps Agent Space is a logical container that defines the tools and infrastructure that AWS DevOps Agent has access to. Each Agent Space operates independently with its own AWS account access, third-party integrations, and user permissions.

An Agent Space represents the boundary of what AWS DevOps Agent can access and investigate during incident response. When you create an Agent Space, you define which AWS accounts the agent can access, which external tools it can connect to, and which users in your organization can interact with the agent.

Each Agent Space functions as an independent deployment of AWS DevOps Agent. You configure the Agent Space through the AWS Management Console, while your operations teams use the Agent Space’s web app to conduct investigations and review recommendations within that space.

How Agent Spaces are isolated

Agent Spaces maintain isolation to ensure security and prevent unintended access across different environments or teams:

- **AWS account isolation** – Each Agent Space uses dedicated IAM roles that grant access only to specific AWS accounts and resources. The agent cannot access AWS resources outside of those explicitly configured for the Agent Space.

- **User access isolation** – You control which users or groups can access each Agent Space. This allows you to align access permissions with your organizational structure, ensuring teams only interact with their designated Agent Spaces.
- **Data isolation** – Investigation data, incident history, and recommendations are maintained separately within each Agent Space. Information from one Agent Space is not visible or accessible from another Agent Space.
- **Chat data isolation** - Chat conversation history is also isolated within each Agent Space. Conversations and queries in one Agent Space are not visible or accessible from another Agent Space.

Agent Space Web App

Each Agent Space has a dedicated web app that is accessible outside of the AWS Management Console. See the section called “What is a DevOps Agent Web App?” to learn more about the web app.

When to use multiple Agent Spaces

Consider creating multiple Agent Spaces to support different organizational needs:

- **Team separation** – Create dedicated Agent Spaces for different application teams or business units to maintain clear ownership boundaries in the Agent Space.
- **Environment isolation** – Separate production and non-production environments into different Agent Spaces to prevent accidental cross-environment access.
- **Service boundaries** – Align Agent Spaces with specific services or application boundaries to keep investigations focused and relevant.
- **Compliance requirements** – Configure separate Agent Spaces with different access controls or data residency settings to meet regulatory requirements.

> **Note**
>
> When creating multiple Agent Spaces, you can use a dedicated AWS account as the primary account for an Agent Space and connect distinct application accounts as secondary accounts. This approach allows you to maintain granular access controls while ensuring that

each Agent Space can access only the resources specific to its intended scope, even when using automatic role creation.

What is a DevOps Agent Topology?

AWS DevOps Agent's automatically discovers and visualizes the resources and relationships within your applications and uses the resulting topology to understand your infrastructure during incident investigations and when making preventative recommendations.

How topology graphs are created

AWS DevOps Agent builds topology graphs through several automated processes:

- **Resource discovery** – The agent automatically scans your AWS accounts to identify resources like compute instances, storage services, networking components, and databases that are part of your applications.
- **Relationship detection** – The agent analyzes configuration data, CloudFormation stacks, and resource tags to determine how resources relate to one another.
- **Code and deployment mapping** – When connected to CI/CD pipelines, the agent links infrastructure resources back to their deployment processes and changed application and infrastructure code.
- **Observability behavior mapping** – Data from observability systems such as Amazon CloudWatch Application Signals and Dynatrace are used to identify observed behaviors that indicate relationships between resources.

Key capabilities

Resource mapping provides several capabilities that enhance incident investigation and prevention:

- **Interactive visualization** – Explore your application topology through an interactive graph in the Operator Web App. You can zoom and navigate the topology to understand complex relationships between resources. You can also use Chat to query topology information using natural language, such as 'Show me all Lambda functions connected to this DynamoDB table' or 'What resources are affected by this alarm?'.

- **Contextual investigation** – During incident investigations, AWS DevOps Agent is assisted by the resource topology to identify affected components, understand blast radius, and trace the impact path through your systems.
- **Root cause analysis** – The detailed understanding of resource relationships helps pinpoint where issues originate, even in complex distributed systems with many interdependencies.
- **Impact assessment** – When analyzing incidents, the agent can better determine which downstream services might be affected by identifying dependency chains in the topology.
- **Preventative recommendations** – The agent uses topology insights to make targeted recommendations for resilience improvements, suggesting changes that will have the most significant impact on system stability.

Topology views

The topology visualization in the Topology page in the Operator Web App offers multiple levels of detail:

- **Learned** – The default view, generated from the Agent Space Understanding skill. Displays a structured summary of your infrastructure organized by logical services and request paths.
- **System** – Shows high-level account and region boundaries.
- **Container** – Displays deployment stacks like CloudFormation stacks that contain related resources.
- **Components** – Shows individual components within containers and their relationships.
- **All Resources** – Shows the complete view with all discovered resources and their relationships.

Resource discovery

Resources are discovered through two methods:

- **CloudFormation stacks** – The agent lists all CloudFormation stacks and their resources in the primary AWS account and any connected secondary accounts. This is supported for any infrastructure-as-code tooling that uses CloudFormation for deployment, including AWS Cloud Development Kit (AWS CDK).
- **Resource Explorer** – For resources not deployed from CloudFormation, tagged resources are discovered from AWS Resource Explorer. The target AWS account must have Resource Explorer

enabled. This is useful for identifying application boundaries for resources deployed through the AWS Management Console, the AWS service APIs, or other infrastructure-as-code frameworks.

Investigation scope beyond topology

While the application topology provides important context during investigations, AWS DevOps Agent is not limited to investigating only the resources shown in the topology. The agent may use additional data sources, such as AWS service APIs or connected observability tools, to investigate resources that are not in the application topology.

To limit the resources the agent has access to, restrict the policy for the role assigned to the agent to access cross-account resources. For more information, see the section called “Limiting Agent Access in an AWS Account”.

Topology and the Agent Space Understanding skill

The topology graph feeds into the Agent Space Understanding learned skill, which encodes a structured summary of your infrastructure for use during investigations. When topology discovery completes for a new agent space, the system automatically generates the Agent Space Understanding skill. For more information about learned skills, see the section called “Learned Skills”.

DevOps Agent Skills

AWS DevOps Agent Skills are modular instruction sets that extend the agent's capabilities with specialized domain knowledge and investigation methodologies tailored to your infrastructure and operational workflows.

What are Skills

Skills are self-contained directories containing Markdown instructions that provide specialized capabilities to AWS DevOps Agent. AWS DevOps Agent supports a subset of the Agent Skills specification —an open standard for packaging agent instructions and resources—supporting only non-executable documents: Markdown instructions, PDFs, images, and data files.

Every skill requires a SKILL.md file containing instructions you want to provide for your AWS DevOps Agent. In addition to the required SKILL.md file, skills can include:

- **Investigation workflows** for specific scenarios or infrastructure types.

- **Reference materials** including architecture patterns and operational procedures.
- **Agent type targeting** – Skills can be targeted to specific agent types (Generic, On-demand, Incident Triage, Incident RCA, Incident Mitigation, Evaluation) to reduce context consumption and improve agent focus.

Why use Skills

Skills transform AWS DevOps Agent from a general-purpose assistant into a specialist for your infrastructure and operational workflows. Unlike one-time instructions provided in a chat message, Skills are reusable capabilities that load automatically when relevant to tasks performed by AWS DevOps Agent.

Key benefits:

- **Specialize your agent** – Tailor AWS DevOps Agent with investigation procedures, best practices, and organizational knowledge specific to your infrastructure and operational patterns.
- **Reduce repetition** – Create investigation workflows once and AWS DevOps Agent uses them automatically across all relevant investigations, eliminating the need to provide the same guidance repeatedly.
- **Compose capabilities** – Combine multiple Skills to build end-to-end investigation workflows. AWS DevOps Agent reads multiple skills during execution, such as a skill to retrieve deployments from your custom CI/CD pipeline and a skill to search your code repositories.
- **Amplify custom tools** – Create skills that guide AWS DevOps Agent in using your custom MCP server tools effectively. Skills can document when to invoke specific tools, what parameters to use for different scenarios, and how to interpret results to accomplish workflows specific to your infrastructure.

How Skills work

When AWS DevOps Agent encounters a relevant task, it loads the appropriate skills and follows the instructions to guide its investigation. For example, a "Database Performance Investigation" skill might include step-by-step procedures for analyzing RDS throttling issues, enabling the agent to systematically check alarm status, analyze connection metrics, and identify slow queries.

Skill structure

A skill is organized as a directory containing:

```
my-skill/
### SKILL.md              # Main skill instructions
### references/           # Optional: additional reference documentation
### assets/               # Optional: images, diagrams, data files
```

SKILL.md

The `SKILL.md` is the only mandatory file. It contains the core instructions written in Markdown format. This file should:

- Describe when and how to use the skill.
- Provide step-by-step investigation procedures.
- Include decision trees for different scenarios.
- Document expected outputs and success criteria.

Frontmatter

Frontmatter is the metadata block at the top of a `SKILL.md` file, enclosed between `---` delimiters. It contains the `name` and `description` fields that AWS DevOps Agent uses to determine when to activate the Skill during an investigation or task.

```
---
name: rds-performance-investigation
description: Investigation procedures for RDS performance issues including
  connection exhaustion, slow queries, replication lag, and storage capacity.
  Use this skill when investigating database latency, connection errors, or
  read/write performance degradation.
---
```

name – A unique identifier for the Skill. Use lowercase letters, numbers, and hyphens only (maximum 64 characters). Must not start or end with a hyphen.

description – A detailed explanation of when and why AWS DevOps Agent should use this Skill. AWS DevOps Agent evaluates this field to decide whether the Skill is relevant to the current task. A vague or missing description can cause the agent to skip the Skill entirely, even if the instructions are well-written.

Important – Write the description from the agent's perspective. Include the specific scenarios, services, error types, or symptoms that should trigger the Skill. For example, "Use this skill when

investigating database latency, connection errors, or query timeouts for Amazon RDS instances" is more effective than "RDS skill".

When you create a Skill in the UI, the system generates frontmatter automatically from the name and description you provide. Skills uploaded as zip files must include frontmatter in the `SKILL.md` file.

Example: Complete skill

The following example shows a complete, well-formed skill for investigating RDS performance issues. It demonstrates the directory structure, SKILL.md frontmatter, actionable investigation procedures, and a supplementary references file.

Directory structure:

```
rds-performance-investigation/
### SKILL.md
### references/
#   ### rds-metrics-reference.md
### assets/
    ### rds-investigation-flowchart.png
```

SKILL.md:

```
---
name: rds-performance-investigation
description: Investigation procedures for RDS performance issues including
  connection exhaustion, slow queries, replication lag, and storage capacity.
  Use this skill when investigating database latency, connection errors, or
  read/write performance degradation.
---

# RDS Performance Investigation

Use this skill when customers report database latency, connection errors,
query timeouts, or read/write performance degradation.


## Step 1: Check alarm status

Query CloudWatch for active alarms on the affected RDS instance. Look for:
```

```
- `DatabaseConnections` exceeding 80% of max_connections
- `ReadLatency` or `WriteLatency` above 20ms
- `FreeStorageSpace` below 20% of total storage
- `ReplicaLag` above 30 seconds (read replicas only)


## Step 2: Analyze connection metrics

Retrieve `DatabaseConnections` over the past hour. If connections are near
the max_connections limit, check for connection pool misconfiguration or
long-running idle connections.


## Step 3: Identify slow queries

Use Performance Insights (`pi:GetResourceMetrics`) to retrieve the top SQL
statements by average active sessions. Focus on queries with high `db.load`
contribution or frequent I/O waits.


## Step 4: Summarize findings

Provide a summary with:
1. Current performance status (healthy / degraded / critical)
2. Root cause hypothesis with supporting metrics
3. Recommended remediation steps ranked by priority
```

references/rds-metrics-reference.md:

```
# RDS CloudWatch Metrics Reference


| Metric | Normal Range | Investigation Threshold |
|---|---|---|
| DatabaseConnections | < 70% max_connections | > 80% max_connections |
| ReadLatency | < 5ms | > 20ms |
| WriteLatency | < 5ms | > 20ms |
| FreeStorageSpace | > 30% total storage | < 20% total storage |
| ReplicaLag | < 5 seconds | > 30 seconds |
| CPUUtilization | < 70% | > 85% |
```

Creating Skills

Before creating skills, you must have an Agent Space. For more information, see the section called "Creating an Agent Space".

You can create skills in two ways depending on your workflow preferences and skill complexity:

Creating a skill in the UI

Skills created in the AWS DevOps Agent Operator Web App contain a name, description, and instructions in a single SKILL.md file.

To create a skill in the UI:

- Navigate to the Skills page in your Agent Space Operator Web App.
- Click "Add skill".
- Select "Create skill" from the modal.
- Fill out the skill form:
  - **Name** – Lowercase letters, numbers, and hyphens only (maximum 64 characters). Must not start or end with a hyphen. Example: `rds-throttling-investigation`
  - **Description** – Brief explanation of when to use this skill (minimum 100 characters recommended, maximum 1,024 characters). This helps the agent determine when to activate the skill.
  - **Status** – Set to Active (default) or Inactive. Inactive skills are not used by the agent.
  - **Agent Type** – Select one or more agent types that can use this skill. **Generic** is selected by default and makes the skill available to all agent types. To target specific agents, deselect Generic and choose from: On-demand, Incident Triage, Incident RCA, Incident Mitigation, or Evaluation.
  - **Instructions** – Step-by-step procedures in Markdown format. Be specific and actionable.
- Click "Create" to save the skill.

The system automatically generates a SKILL.md file with the proper frontmatter structure.

To edit a skill created in the UI:

- Navigate to the skill in the Skills list and click the skill to open it.
- Click **Edit**.

- Modify the name, description, or instructions.
- Click **Save** to update the skill.

Uploading a skill

Skills uploaded as zip files contain a SKILL.md file plus additional resources such as reference materials or assets.

Skill structure:

```
my-skill.zip
### SKILL.md              # Required: main skill instructions
### references/           # Optional: reference documentation
#   ### architecture.md
#   ### troubleshooting.md
### assets/               # Optional: images, diagrams, data files
    ### topology.png
    ### metrics.csv
```

SKILL.md frontmatter requirements:

Skills uploaded as zip files must include frontmatter in SKILL.md with `name` and `description` fields. AWS DevOps Agent uses these fields to determine when to activate the Skill. For details on writing effective frontmatter, see the Frontmatter section earlier in this topic.

```
---
name: rds-performance-analysis
description: Comprehensive RDS performance investigation procedures
  for connection exhaustion, slow queries, and storage capacity issues.
  Use when investigating database latency or read/write degradation.
---


# RDS Performance Analysis


[Your skill instructions here...]
```

To create a skill via zip upload:

- Create a directory with your skill files following the structure above.

- Ensure SKILL.md includes proper frontmatter (name and description).
- Compress the directory into a .zip file.
- Navigate to the Skills page in your Agent Space Operator Web App.
- Click "Add skill".
- Select "Upload skill" from the modal.
- Drag and drop your .zip file or click to browse (ZIP files only, maximum 6 MB).
- Select one or more agent types that can use this skill (Generic is selected by default and applies to all agent types; deselect to target On-demand, Incident Triage, Incident RCA, Incident Mitigation, or Evaluation specifically).
- Review the zip file requirements and validation results.
- Click "Upload" to add the skill to your Agent Space.

Important restrictions for skills uploaded as zip files:

- **Scripts are currently not supported** – Skills containing scripts in the `scripts/` directory will be rejected during upload. Script execution will be enabled in a future release once agents have access to a secure coding environment.
- **Size limit** – Total zip file size must not exceed 6 MB (including all files).
- **SKILL.md required** – The zip file must contain a SKILL.md file with valid frontmatter.

Best practices for naming skills:

Use clear, descriptive names like "rds-throttling-investigation" rather than generic names. A good skill name reflects the specific scenario or service it addresses, making it easier to identify the right skill at a glance.

Managing Skills

AWS DevOps Agent provides comprehensive skill management capabilities through the Operator Web App:

Listing skills – View all Skills in your Agent Space. The Skills page displays skill name, Active or Inactive status, creation date, last updated date, and available actions.

Viewing skills – Click on any skill to see its detail view. Skills created in the UI display editable content where you can modify the name, description, or instructions directly in the UI and

click "Save" to update. Skills uploaded as zip files display a file tree showing SKILL.md and any additional directories like references/ and assets/. Click files in the tree to view their contents in read-only mode.

Selecting agents for a skill – Configure which agent types can use each skill when creating or editing it. In the Agent Type dropdown, select one or more agent types using the checkboxes: **Generic** (default — applies to all agent types), **On-demand** (conversational queries), **Incident Triage** (initial incident assessment), **Incident RCA** (root cause analysis), **Incident Mitigation** (automated incident response), or **Evaluation** (proactive recommendations). Generic is selected by default and makes the skill available to all agent types. Skills targeted to specific agents reduce context consumption and improve agent focus.

Activating and deactivating skills – Temporarily disable skills without deleting them using the Active/Inactive toggle. Open the skill detail view and toggle the switch to "Inactive" to prevent the agent from loading it for new investigations while preserving all content and configurations. In-progress investigations continue using the skill. Toggle back to "Active" to make the skill immediately available again.

Updating skills – Modify existing skills based on how they were created. For skills created in the UI, click "Edit" in the skill detail view, modify the name, description, or instructions, and click "Save" to update. For skills uploaded as zip files, modify the files locally, create a new zip file, and upload a new version.

Deleting skills – Permanently remove skills from your Agent Space. Open the skill list view, click the more options menu (⋮) and select "Delete," review the warning about permanent deletion, type the skill name to confirm, and click "Delete Skill." Deletion cannot be undone. In-progress investigations may be affected if they attempt to load the deleted skill. For skills uploaded as zip files, download the zip file before deleting as a backup. Consider deactivating the skill instead of deleting it if you may need it again.

Migrating from Runbooks

Existing Runbooks are automatically migrated to Skills with no customer action required. When your Agent Space transitions to the Skills model, all Runbooks are converted to Skills and appear in your Skills UI. After migration, you can:

- **Review migrated Skills** – Check that the automatic migration correctly converted your Runbooks.

- **Update as needed** – Edit Skills directly in the UI to refine instructions, update descriptions, or configure agent type targeting.
- **Expand with references** – For Skills that would benefit from additional reference materials or architecture diagrams, re-create them as zip upload skills with a references/ or assets/ directory.
- **Create new Skills** – Add new Skills for investigation workflows not previously covered by Runbooks.

Contact AWS Support if you encounter any issues with automatically migrated Skills or need assistance with post-migration updates.

Learned Skills

What Are Learned Skills?

Learned skills are structured knowledge files that the DevOps Agent generates from your Agent Space data. Each learned skill encodes a specific type of knowledge that the AWS DevOps Agent uses as it performs tasks. At launch, two learned skills are available: Agent Space Understanding and Tool Use Best Practices.

Agent Space Understanding

The Agent Space Understanding skill (`understanding-agent-space`) analyzes your connected cloud accounts, code repositories, and telemetry integrations to build a map of the resources and relationships in an Agent Space.

The skill produces a main `SKILL.md` file and a set of reference files. The main file contains a plain-language system overview with key domain concepts, the deployment environments (AWS account and region pairs, Azure subscriptions and regions, and so on), a container-level architecture diagram showing how logical services connect, the request paths that are central to your application with the components they traverse, and a mapping of code repositories to containers.

Each logical container receives a dedicated reference file describing its internal components (compute, data, messaging, network, and others) with resource types and physical identifiers such as ARNs, table names, and queue URLs. The reference file also captures observability coverage, including the alarms, dashboards, and monitors linked to each component. It also maps each

component to its associated code repositories, packages, and infrastructure-as-code definitions, providing a complete traceability chain from source code to deployed resources.

Each critical request path receives a dedicated reference file describing the full end-to-end request flow at component granularity, from the entry point through each intermediate service, data store, and external dependency. The file includes a sequenced flow diagram showing the order of operations and interaction mechanisms between components, along with the responsibility of each participant. It also catalogs the observability signals relevant to the path: log group patterns for each hop, key metrics (latency, error rates, throttling, token quotas) with their alarm names and dimensions, and distributed trace spans that can be correlated across services and accounts.

Tool Use Best Practices

The Tool Use Best Practices skill analyzes past investigation tool uses to extract effective usage patterns, common failure modes, and parameter guidance. This helps the DevOps Agent avoid known pitfalls and run investigations with fewer wasted steps. The skill produces a main file and a set of per-tool reference files. The main file serves as a routing index that lists each tool with the investigation scenarios it supports and links to the corresponding reference file.

Each per-tool reference file can include up to three sections:

- **Best Practices** — Investigation-driven techniques extracted from successful tool usage, such as CloudWatch Logs Insights query templates, environment-specific metric namespaces and dimensions, and CloudTrail event source filters. Each entry is organized around an investigation scenario and includes concrete parameter values and examples observed in past investigations.
- **Common Errors** — Recurring failure modes and their fixes. Each entry describes a specific error condition, such as querying an inaccessible account or constructing a malformed aggregation query, and provides a corrective action so the agent can avoid or recover from the error without wasting investigation steps.
- **Output Management** — Guidance for tool calls that tend to return large responses. Each entry describes a parameter change or processing strategy that reduces output size while preserving diagnostic value.

When live infrastructure access is available, the skill validates patterns against your environment before including them. Confirmed patterns are stated with confidence, unconfirmed patterns use cautious language, and disproved patterns are excluded. This keeps the skill aligned with the current state of your infrastructure.

Managing Learned Skills

Updates — The DevOps Agent automatically generates and updates learned skills based on activity in your Agent Space. The following describes when each skill is updated.

The DevOps Agent generates an updated **Tool Use Best Practices** skill every 30 investigations.

The **Agent Space Understanding** skill is generated by the learning agent, which runs whenever you add, update, or remove an Agent Space capability or integration.

To regenerate learned skills manually, choose the **Regenerate** button on the Topology page in the operator app, or chat with the agent and ask it to update the learned skills.

Deactivation — Learned skills are active by default. When active, the DevOps Agent loads them at the start of each DevOps Agent task. To stop a learned skill from being applied, deactivate it from the skill viewer in the operator app. Deactivating a skill does not delete it. The skill is retained and can be reactivated at any time. When a skill is deactivated, the DevOps Agent operates without that skill's knowledge.

Topology view — The Topology page in your Agent Space's web app uses the Agent Space Understanding Skill to visually display your Agent Space environment as logical containers and components . Click any container to see its components, resource identifiers, and telemetry.

Supported Regions

This topic describes the AWS Regions where you can use AWS DevOps Agent. For more information about AWS Regions, see Specify which AWS Regions your account can use in the *AWS Account Management Reference Guide*.

Cross-Region resource monitoring

AWS DevOps Agent can monitor and investigate resources in AWS accounts located in any AWS Region, regardless of which supported Region you create your Agent Space in. When you associate an AWS account with an Agent Space, the agent discovers and maps resources across all Regions within that account. This means you do not need an Agent Space in every Region where your workloads run.

Choose a supported Region based on your preferred data residency, proximity to your operations team, or organizational requirements.

Supported Regions

AWS DevOps Agent is available in the following AWS Regions.

Region Name	Region Code	Console Link
US East (N. Virginia)	us-east-1	Open console
US West (Oregon)	us-west-2	Open console
Asia Pacific (Sydney)	ap-southeast-2	Open console
Asia Pacific (Tokyo)	ap-northeast-1	Open console
Europe (Frankfurt)	eu-central-1	Open console
Europe (Ireland)	eu-west-1	Open console

Service endpoints

Region Name	Region Code	Endpoint	Protocol
US East (N. Virginia)	us-east-1	aidevops.us-east-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	aidevops.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	aidevops.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	aidevops.ap-northeast-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	aidevops.eu-central-1.amazonaws.com	HTTPS

Region Name	Region Code	Endpoint	Protocol
Europe (Ireland)	eu-west-1	aidevops.eu-west-1.amazonaws.com	HTTPS

Considerations

- **Agent Space Region selection** — An Agent Space and its data (investigations,

topology, recommendations) are stored in the Region where you create it. Choose a Region that meets your data residency requirements.

- **Cross-Region monitoring** — Resources in AWS accounts associated with an Agent

Space are monitored regardless of which Region those resources are deployed in. You do not need to create separate Agent Spaces in each Region where your workloads run.

- **Third-party integrations** — Connections to CI/CD providers (GitHub, GitLab),

observability tools (Dynatrace, Datadog, New Relic, Splunk), and MCP servers are configured per Agent Space and are not Region-dependent.

Getting started with AWS DevOps Agent

In this getting started guide, you'll create a basic Agent Space, configure minimal permissions, and conduct your first AI-powered investigation.

Topics:

Creating an Agent Space

An Agent Space defines the tools and infrastructure that AWS DevOps Agent has access to. This guide walks you through creating an Agent Space, configuring primary account access, and enabling the DevOps Agent Web App. See “What is an Agent Space” to learn more about the Agent Space concept.

Creating an Agent Space

Access the AWS DevOps Agent console

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console

Name the Agent Space

1. Click **Create Agent Space**

In the **Agent Space details** section, provide:

1. In the **Name** field, enter a name for your Agent Space
2. (Optional) In the **Description** field, add details about the Agent Space's purpose
3. (Optional) From the **Agent response language** dropdown, select the language the agent uses when generating responses, findings, and investigation output. Options include: Bahasa Indonesian, Chinese (Simplified/PRC), Chinese (Traditional/Taiwan), English (UK), French (France), German (Germany), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil), Spanish (Latin America), Turkish (Turkey), Arabic (Saudi Arabia), Thai (Thailand), and Vietnamese (Vietnam). If no language is selected, the agent responds in the language of the input.

Configuring primary account access

In the **Give this Agent Space AWS resource access** section, you will set up an IAM role to grant the Agent Space access to the primary AWS account. The primary account is the AWS account where you create your Agent Space. AWS DevOps Agent requires an IAM role to discover and access AWS resources in this account during investigations.

Choose a role configuration method. Select one of the following options:

Option 1: Auto-create a new AWS DevOps Agent role (recommended)

This option automatically creates a role with appropriate permissions for AWS DevOps Agent to investigate resources in your account.

> **Note**
>
> You must have IAM permissions to create new roles to use this option.

1. Select **Auto-create a new AWS DevOps Agent role**
2. (Optional) Update the Agent Space role name to be created

Option 2: Assign an existing role

Use this option when another administrator has previously created a role specifically for AWS DevOps Agent.

1. Select **Assign an existing role**

2. From the dropdown menu, select an existing role that has appropriate permissions

Option 3: Create a new AWS DevOps Agent role using a policy template

Use this option when you need to limit the services and resources the agent can access in the primary account.

1. Select **Create a new AWS DevOps Agent role using a policy template**
2. Follow the instructions to create the new role's trust policy and inline policy.

Enabling the Agent Space Web App

The Web App is where personnel interact with AWS DevOps Agent for incident investigations and reviewing recommendations. See AWS DevOps Agent Console Architecture[link] to learn more. When enabled, users can access the Agent Space Web App through an IAM authentication link from the AWS Management Console.

Select one of the following options:

Option 1: Auto-create a new AWS DevOps Agent role (recommended)

This option automatically creates a role with appropriate permissions for accessing the DevOps Agent Web App.

> **Note**
>
> You must have IAM permissions to create new roles to use this option.

1. Select **Auto-create a new AWS DevOps Agent role**
2. Review the permissions that will be granted to the role

Option 2: Assign an existing role

Use this option when another administrator has previously created an operator role.

1. Select **Assign an existing role**
2. From the dropdown menu, select an existing role that has appropriate permissions

Option 3: Create a new AWS DevOps Agent role using a policy template

Use this option when you need to customize permissions for web app access.

1. Select **Create a new AWS DevOps Agent role using a policy template**
2. Follow the instructions to create the new role's trust policy and inline policy.

Adding tags (optional)

You can add AWS tags to your Agent Space during creation. Tags are key-value pairs that help you organize and identify your resources. You can add up to 50 tags per Agent Space. To add tags, expand the **Tags** section on the Create Agent Space page and click **Add new tag**.

Complete agent space creation

Once all sections are filled out, click **Create**

Verifying your Agent Space setup

Once configured, the **Operator access** button will appear on the Agent Space details page. Clicking it will open the Web App in a new tab and authenticate successfully.

Next steps

After setting up your Agent Space, consider these next steps:

- Add secondary accounts if your applications span multiple AWS accounts
- Configure third-party integrations like observability tools or ticketing systems
- Set up AWS Identity Center authentication for production environments
- Explore your application resource mapping to help AWS DevOps Agent understand your infrastructure

AWS DevOps Agent CLI onboarding guide

Overview

With AWS DevOps Agent, you can monitor and manage your AWS infrastructure. This guide walks you through setting up AWS DevOps Agent by using the AWS Command Line Interface (AWS CLI).

You create IAM roles, set up an agent space, and associate your AWS account. You also enable the operator app and optionally connect third-party integrations. This guide takes approximately 20 minutes to complete.

AWS DevOps Agent is available in six AWS Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), and Europe (Ireland). For more information about supported Regions, see the section called “Supported Regions”.

Prerequisites

Before you begin, make sure that you have the following:

- AWS CLI version 2 installed and configured
- Authentication to your AWS monitoring account
- Permissions to create AWS Identity and Access Management (IAM) roles and attach policies
- An AWS account to use as the monitoring account
- Familiarity with the AWS CLI and JSON syntax

Throughout this guide, replace the following placeholder values with your own:

- `<MONITORING_ACCOUNT_ID>` — Your 12-digit AWS account ID for the monitoring (primary) account
- `<EXTERNAL_ACCOUNT_ID>` — The 12-digit AWS account ID of the secondary account to monitor (used in step 4)
- `<REGION>` — The AWS Region code for your agent space (for example, `us-east-1` or `eu-central-1`)
- `<AGENT_SPACE_ID>` — The agent space identifier that is returned by the `create-agent-space` command

IAM roles setup

1. Create the DevOps Agent space role

Create the IAM trust policy by running the following command:

```
cat > devops-agentspace-trust-policy.json << 'EOF'
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<MONITORING_ACCOUNT_ID>"
        },
        "ArnLike": {
          "aws:SourceArn":
 "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/*"
        }
      }
    }
  ]
}
EOF
```

Create the IAM role:

```
aws iam create-role \
  --region <REGION> \
  --role-name DevOpsAgentRole-AgentSpace \
  --assume-role-policy-document file://devops-agentspace-trust-policy.json
```

Save the role ARN by running the following command:

```
aws iam get-role --role-name DevOpsAgentRole-AgentSpace --query 'Role.Arn' --output
 text
```

Attach the AWS managed policy:

```
aws iam attach-role-policy \
  --role-name DevOpsAgentRole-AgentSpace \
  --policy-arn arn:aws:iam::aws:policy/AIDevOpsAgentAccessPolicy
```

Create and attach an inline policy to allow creation of the Resource Explorer service-linked role:

```
cat > devops-agentspace-additional-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateServiceLinkedRoles",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer"
      ]
    }
  ]
}
EOF

aws iam put-role-policy \
  --role-name DevOpsAgentRole-AgentSpace \
  --policy-name AllowCreateServiceLinkedRoles \
  --policy-document file://devops-agentspace-additional-policy.json
```

2. Create the operator app IAM role

Create the IAM trust policy by running the following command:

```
cat > devops-operator-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
```

```
          "aws:SourceAccount": "<MONITORING_ACCOUNT_ID>"
        },
        "ArnLike": {
          "aws:SourceArn":
 "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/*"
        }
      }
    }
  ]
}
EOF
```

Create the IAM role:

```
aws iam create-role \
  --role-name DevOpsAgentRole-WebappAdmin \
  --assume-role-policy-document file://devops-operator-trust-policy.json \
  --region <REGION>
```

Save the role ARN by running the following command:

```
aws iam get-role --role-name DevOpsAgentRole-WebappAdmin --query 'Role.Arn' --output
 text
```

Attach the AWS managed operator app policy:

```
aws iam attach-role-policy \
  --role-name DevOpsAgentRole-WebappAdmin \
  --policy-arn arn:aws:iam::aws:policy/AIDevOpsOperatorAppAccessPolicy
```

This managed policy grants the operator app permissions to access agent space features. These features include investigations, recommendations, knowledge management, chat, and AWS Support integration. The policy scopes access to the specific agent space by using the `aws:PrincipalTag/AgentSpaceId` condition. For more information about the full list of actions, see the section called “DevOps Agent IAM permissions”.

Onboarding steps

1. Create an agent space

Run the following command to create an agent space:

```
aws devops-agent create-agent-space \
  --name "MyAgentSpace" \
  --description "AgentSpace for monitoring my application" \
  --region <REGION>
```

Optionally, specify `--kms-key-arn` to use a customer managed AWS KMS key for encryption. You can also use `--tags` to add resource tags and `--locale` to set the language for agent responses.

Save the `agentSpaceId` from the response (located at `agentSpace.agentSpaceId`).

To list your agent spaces later, run the following command:

```
aws devops-agent list-agent-spaces \
  --region <REGION>
```

2. Associate your AWS account

Associate your AWS account to turn on topology discovery. Set the `accountType` to one of the following values:

- `monitor` — The primary account where the agent space exists. This account hosts the agent and is used for topology discovery.
- `source` — An additional account that the agent monitors. Use this type when you associate external accounts in step 4.

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id aws \
  --configuration '{
    "aws": {
      "assumableRoleArn": "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/DevOpsAgentRole-
AgentSpace",
      "accountId": "<MONITORING_ACCOUNT_ID>",
      "accountType": "monitor"
    }
  }' \
  --region <REGION>
```

3. Enable the operator app

Authentication flows can use IAM, IAM Identity Center (IDC), or an external identity provider (IdP). Run the following command to enable the operator app for your agent space:

```
aws devops-agent enable-operator-app \
  --agent-space-id <AGENT_SPACE_ID> \
  --auth-flow iam \
  --operator-app-role-arn "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/DevOpsAgentRole-
WebappAdmin" \
  --region <REGION>
```

For IAM Identity Center authentication, use `--auth-flow idc` and provide `--idc-instance-arn`. For an external identity provider, use `--auth-flow idp` and provide `--issuer-url`, `--idp-client-id`, and `--idp-client-secret`. For more information, see the section called "Setting Up IAM Identity Center Authentication" and the section called "Setting Up External Identity Provider (IdP) Authentication".

> **Note:** If you previously created an operator app role for another agent space in your account, you can reuse that role ARN.

4. (Optional) Associate additional source accounts

To monitor additional accounts with AWS DevOps Agent, create an IAM cross-account role.

Create the cross-account role in the external account

Switch to the external account and create the trust policy. The `MONITORING_ACCOUNT_ID` is the main account that hosts the agent space that you set up in step 2. This configuration allows the AWS DevOps Agent service to assume a role in the secondary source accounts on behalf of the monitoring account.

Run the following command to create the trust policy:

```
cat > devops-cross-account-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "<MONITORING_ACCOUNT_ID>",
          "sts:ExternalId":
 "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/<AGENT_SPACE_ID>"
        }
      }
    }
  ]
}
EOF
```

Create the cross-account IAM role:

```
aws iam create-role \
  --role-name DevOpsAgentCrossAccountRole \
  --assume-role-policy-document file://devops-cross-account-trust-policy.json
```

Save the role ARN by running the following command:

```
aws iam get-role --role-name DevOpsAgentCrossAccountRole --query 'Role.Arn' --output
 text
```

Attach the AWS managed policy:

```
aws iam attach-role-policy \
  --role-name DevOpsAgentCrossAccountRole \
  --policy-arn arn:aws:iam::aws:policy/AIDevOpsAgentAccessPolicy
```

Attach the inline policy to allow creation of the Resource Explorer service-linked role in the external account:

```
cat > devops-cross-account-additional-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateServiceLinkedRoles",
      "Effect": "Allow",
      "Action": [
```

```
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::<EXTERNAL_ACCOUNT_ID>:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer"
      ]
    }
  ]
}
EOF

aws iam put-role-policy \
  --role-name DevOpsAgentCrossAccountRole \
  --policy-name AllowCreateServiceLinkedRoles \
  --policy-document file://devops-cross-account-additional-policy.json
```

Associate the external account

Switch back to your monitoring account, and then run the following command to associate the external account:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id aws \
  --configuration '{
    "sourceAws": {
      "accountId": "<EXTERNAL_ACCOUNT_ID>",
      "accountType": "source",
      "assumableRoleArn": "arn:aws:iam::<EXTERNAL_ACCOUNT_ID>:role/
DevOpsAgentCrossAccountRole"
    }
  }' \
  --region <REGION>
```

5. (Optional) Associate GitHub

> **Note:** You must first register GitHub through the AWS DevOps Agent console by using the OAuth flow before you can associate it through the CLI.

For instructions on registering GitHub through the console, see the section called “Connecting to CI/CD pipelines”.

List the registered services:

```
aws devops-agent list-services \
  --region <REGION>
```

Save the <SERVICE_ID> for serviceType: github.

After you register GitHub in the console, associate GitHub repositories by running the following command:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "github": {
      "repoName": "<GITHUB_REPO_NAME>",
      "repoId": "<GITHUB_REPO_ID>",
      "owner": "<GITHUB_OWNER>",
      "ownerType": "organization"
    }
  }' \
  --region <REGION>
```

6. (Optional) Register and associate ServiceNow

First, register the ServiceNow service with OAuth credentials:

```
aws devops-agent register-service \
  --service servicenow \
  --service-details  '{
    "servicenow": {
      "instanceUrl": "<SERVICENOW_INSTANCE_URL>",
      "authorizationConfig": {
        "oAuthClientCredentials": {
            "clientName": "<SERVICENOW_CLIENT_NAME>",
            "clientId": "<SERVICENOW_CLIENT_ID>",
            "clientSecret": "<SERVICENOW_CLIENT_SECRET>"
        }
      }
    }
  }' \
  --region <REGION>
```

Save the returned <SERVICE_ID>, then associate ServiceNow:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "servicenow": {
      "instanceUrl": "<SERVICENOW_INSTANCE_URL>"
    }
  }' \
  --region <REGION>
```

7. (Optional) Register and associate Dynatrace

First, register the Dynatrace service with OAuth credentials:

```
aws devops-agent register-service \
  --service dynatrace \
  --service-details '{
  "dynatrace": {
    "accountUrn": "<DYNATRACE_ACCOUNT_URN>",
    "authorizationConfig": {
        "oAuthClientCredentials": {
            "clientName": "<DYNATRACE_CLIENT_NAME>",
            "clientId": "<DYNATRACE_CLIENT_ID>",
            "clientSecret": "<DYNATRACE_CLIENT_SECRET>"
        }
      }
    }
  }' \
  --region <REGION>
```

Save the returned <SERVICE_ID>, then associate Dynatrace. Resources are optional. The environment specifies which Dynatrace environment to associate with.

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "dynatrace": {
      "envId": "<DYNATRACE_ENVIRONMENT_ID>",
      "resources": [
        "<DYNATRACE_RESOURCE_1>",
        "<DYNATRACE_RESOURCE_2>"
```

```
      ]
    }
  }' \
  --region <REGION>
```

The response includes webhook information for integration. You can use this webhook to trigger an investigation from Dynatrace. For more information, see the section called “Connecting Dynatrace”.

8. (Optional) Register and associate Splunk

First, register the Splunk service with BearerToken credentials.

The endpoint uses the following format: `https://<XXX>.api.scs.splunk.com/<XXX>/mcp/v1/`

```
aws devops-agent register-service \
  --service mcpserversplunk \
  --service-details '{
  "mcpserversplunk": {
    "name": "<SPLUNK_NAME>",
    "endpoint": "<SPLUNK_ENDPOINT>",
    "authorizationConfig": {
        "bearerToken": {
            "tokenName": "<SPLUNK_TOKEN_NAME>",
            "tokenValue": "<SPLUNK_TOKEN_VALUE>"
        }
      }
    }
  }' \
  --region <REGION>
```

Save the returned <SERVICE_ID>, then associate Splunk:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "mcpserversplunk":  {
      "name": "<SPLUNK_NAME>",
      "endpoint": "<SPLUNK_ENDPOINT>"
    }
  }' \
```

```
  --region <REGION>
```

The response includes webhook information for integration. You can use this webhook to trigger an investigation from Splunk. For more information, see the section called “Connecting Splunk”.

9. (Optional) Register and associate New Relic

First, register the New Relic service with API key credentials.

Region: Either US or EU.

Optional fields: `applicationIds`, `entityGuids`, `alertPolicyIds`

```
aws devops-agent register-service \
  --service mcpservernewrelic \
  --service-details '{
    "mcpservernewrelic": {
      "authorizationConfig": {
        "apiKey": {
          "apiKey": "<YOUR_NEW_RELIC_API_KEY>",
          "accountId": "<YOUR_ACCOUNT_ID>",
          "region": "US",
          "applicationIds": ["<APP_ID_1>", "<APP_ID_2>"],
          "entityGuids": ["<ENTITY_GUID_1>"],
          "alertPolicyIds": ["<POLICY_ID_1>"]
        }
      }
    }
  }' \
  --region <REGION>
```

Save the returned `<SERVICE_ID>`, then associate New Relic:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "mcpservernewrelic":  {
      "accountId": "<YOUR_ACCOUNT_ID>",
      "endpoint": "https://mcp.newrelic.com/mcp/"
    }
  }' \
  --region <REGION>
```

The response includes webhook information for integration. You can use this webhook to trigger an investigation from New Relic. For more information, see the section called “Connecting New Relic”.

10. (Optional) Register and associate Datadog

You must first register Datadog through the AWS DevOps Agent console by using the OAuth flow before you can associate it through the CLI. For more information, see the section called “Connecting DataDog”.

List the registered services:

```
aws devops-agent list-services \
  --region <REGION>
```

Save the <SERVICE_ID> for serviceType: `mcpserverdatadog`.

Then associate Datadog:

```
aws devops-agent associate-service \
  --agent-space-id <AGENT_SPACE_ID> \
  --service-id <SERVICE_ID> \
  --configuration '{
    "mcpserverdatadog": {
      "name": "Datadog-MCP-Server",
      "endpoint": "<DATADOG_MCP_ENDPOINT>"
    }
  }' \
  --region <REGION>
```

The response includes webhook information for integration. You can use this webhook to trigger an investigation from Datadog. For more information, see the section called “Connecting DataDog”.

11. (Optional) Delete an agent space

Deleting an agent space removes all associations, configurations, and investigation data for that agent space. This action can't be undone.

To delete an agent space, run the following command:

```
aws devops-agent delete-agent-space \
```

```
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

Verification

To verify your setup, run the following commands:

```
# List your agent spaces
aws devops-agent list-agent-spaces \
  --region <REGION>

# Get details of a specific agent space
aws devops-agent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>

# List associations for an agent space
aws devops-agent list-associations \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

Next steps

- To connect additional integrations, see *Configuring capabilities for AWS DevOps Agent*.
- To learn about agent skills and capabilities, see the section called "DevOps Agent Skills".
- To understand the operator web app, see the section called "What is a DevOps Agent Web App?".

Notes

- Replace `<AGENT_SPACE_ID>`, `<MONITORING_ACCOUNT_ID>`, `<EXTERNAL_ACCOUNT_ID>`, `<REGION>`, and so on with your actual values.
- For a list of supported Regions, see the section called "Supported Regions".

Creating a test environment

This guide provides hands-on tests to validate AWS DevOps Agent's incident response functionality using sample architecture. Use this supplement if you want to test DevOps Agent before connecting your production systems.

Prerequisites

- AWS account with administrative access
- AWS DevOps Agent Space created with and configured using the Auto create DevOps Agent role flow

Cost and safety overview

Cost protection

- **EC2 test**: FREE (AWS Free Tier) or ~$0.02 for 2 hours
- **Lambda test**: FREE (1M requests/month free tier)
- **CloudWatch**: FREE (10 alarms, basic metrics included)
- **Expected estimated total cost**: $0.00 - $0.05 for complete testing

Safety features in these tests

- **Auto-termination**: Built-in automatic shutdown
- **Free Tier eligible**: Uses smallest instance types
- **Limited scope**: Minimal, isolated test resources
- **Easy cleanup**: Simple console steps to remove everything
- **No production impact**: Completely separate test environment

Set up your AWS account for testing

> **Important**
>
> Infrastructure resources need to be deployed in the AWS account where you created your DevOps Agent Space's primary cloud account. The specific region does not matter.

1. Log into AWS Console: https://console.aws.amazon.com
2. Ensure you're working in the same AWS account where your DevOps Agent Space is located

3. You can use any region for your testing resources

> **Note**
>
> The 1:1 mapping between your DevOps Agent's primary account and the test environment resources you are creating simplifies the test setup. You can easily extend your DevOps Agent Space to include secondary accounts and enable cross-account investigations.

Choose your test

You can run either test independently or both together:

Test option A: EC2 CPU capacity test

Purpose: Validate AWS DevOps Agent's ability to detect and investigate EC2 performance issues

Estimated time: 5 minutes setup + 10 minutes automatic execution

Difficulty: Fully automated (no manual steps required)

Test option B: Lambda error rate test

Purpose: Validate AWS DevOps Agent's ability to detect and investigate Lambda function errors

Estimated time: 10 minutes setup + 2 minutes to trigger

Difficulty: Very easy

Test option A: EC2 CPU capacity test

Step 1: Deploy CloudFormation stack for EC2 test

We'll use CloudFormation to create our test resources, which allows AWS DevOps Agent to properly track and investigate them.

1. **Navigate to CloudFormation**:
   a. In AWS Console, search for "CloudFormation" and click **CloudFormation**

b. Click **Create stack** > **With new resources (standard)**

2. **Upload template**:

a. Create a new local file called`AWS-DevOpsAgent-ec2-test.yaml`

b. Copy and paste this CloudFormation template into the file:

i.
```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'AWS DevOps Agent EC2 CPU Test Stack'
Parameters:
  MyIP:
    Type: String
    Description: Your current IP address for SSH access (find at https://
whatismyipaddress.com)
    Default: '0.0.0.0/0'
Resources:
  # Security Group for SSH access
  TestSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: AWS-DevOpsAgent-test-sg
      GroupDescription: AWS DevOps Agent beta testing security group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref MyIP
          Description: SSH access from your IP
      Tags:
        - Key: Name
          Value: AWS-DevOpsAgent-Test-SG
        - Key: Purpose
          Value: AWS-DevOpsAgent-Testing
  # Key Pair for SSH access
  TestKeyPair:
    Type: AWS::EC2::KeyPair
    Properties:
      KeyName: AWS-DevOpsAgent-test-key
      KeyType: rsa
      Tags:
        - Key: Name
          Value: AWS-DevOpsAgent-Test-Key
        - Key: Purpose
          Value: AWS-DevOpsAgent-Testing
```

```
  # EC2 Instance for CPU testing
  TestInstance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      ImageId: '{{resolve:ssm:/aws/service/ami-amazon-linux-latest/al2023-ami-
kernel-6.1-x86_64}}'
      KeyName: !Ref TestKeyPair
      SecurityGroupIds:
        - !Ref TestSecurityGroup
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          yum update -y
          yum install -y htop

          # Create the CPU stress test script
          cat > /home/ec2-user/cpu-stress-test.sh << 'EOF'
          #!/bin/bash
          echo "Starting AWS DevOpsAgent CPU Stress Test"
          echo "Time: $(date)"
          echo "Instance: $(curl -s http://169.254.169.254/latest/meta-data/
instance-id)"
          echo ""

          # Get number of CPU cores
          CORES=$(nproc)
          echo "CPU Cores: $CORES"
          echo ""

          echo "Starting stress test (5 minutes)..."
          echo "This will generate >70% CPU usage to trigger CloudWatch alarm"
          echo ""

          # Create CPU load using yes command
          echo "Starting CPU load processes..."
          for i in $(seq 1 $CORES); do
              (yes > /dev/null) &
              CPU_PID=$!
              echo "Started CPU load process $i (PID: $CPU_PID)"
              echo $CPU_PID >> /tmp/cpu_test_pids
          done

          # Auto-cleanup after 5 minutes
```

```
          (sleep 300 && echo "Stopping CPU load processes..." && kill $(cat /
tmp/cpu_test_pids 2>/dev/null) 2>/dev/null && rm -f /tmp/cpu_test_pids) &

          echo ""
          echo "CPU load processes started for 5 minutes"
          echo "Check CloudWatch for alarm trigger in 3-5 minutes"
          EOF

          chmod +x /home/ec2-user/cpu-stress-test.sh
          chown ec2-user:ec2-user /home/ec2-user/cpu-stress-test.sh

          # Create auto-shutdown script (safety mechanism)
          cat > /home/ec2-user/auto-shutdown.sh << 'SHUTDOWN_EOF'
          #!/bin/bash
          echo "Auto-shutdown scheduled for 2 hours from now: $(date)"
          sleep 7200
          echo "Auto-shutdown executing at: $(date)"
          sudo shutdown -h now
          SHUTDOWN_EOF

          chmod +x /home/ec2-user/auto-shutdown.sh
          nohup /home/ec2-user/auto-shutdown.sh > /home/ec2-user/auto-
shutdown.log 2>&1 &

          echo "AWS DevOpsAgent test setup completed at $(date)" > /home/ec2-
user/setup-complete.txt
      Tags:
        - Key: Name
          Value: AWS-DevOpsAgent-Test-Instance
        - Key: Purpose
          Value: AWS-DevOpsAgent-Testing
  # CloudWatch Alarm for CPU utilization
  CPUAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName: AWS-DevOpsAgent-EC2-CPU-Test
      AlarmDescription: AWS-DevOpsAgent beta test - EC2 CPU utilization alarm
      MetricName: CPUUtilization
      Namespace: AWS/EC2
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanThreshold
```

```
      Dimensions:
        - Name: InstanceId
          Value: !Ref TestInstance
      TreatMissingData: notBreaching
Outputs:
  InstanceId:
    Description: EC2 Instance ID for testing
    Value: !Ref TestInstance

  SecurityGroupId:
    Description: Security Group ID
    Value: !Ref TestSecurityGroup

  AlarmName:
    Description: CloudWatch Alarm Name
    Value: !Ref CPUAlarm

  SSHCommand:
    Description: SSH command to connect to instance
    Value: !Sub 'ssh -i "AWS-DevOpsAgent-test-key.pem" ec2-user@
${TestInstance.PublicDnsName}'
```

c. In the CloudFormation console, select **Upload a template file**

d. Click **Choose file**

e. Select the `AWS-DevOpsAgent-ec2-test.yaml` file

f. Click **Next**

3. **Configure stack**:

   a. **Stack name**:`AWS-DevOpsAgent-EC2-Test`

   b. **Parameters**:

      i. **MyIP**: Leave as default `0.0.0.0/0` (you can secure this later if needed)

   c. Click **Next**

4. **Configure stack options**:

   a. Leave defaults, click **Next**

5. **Review and create**:

   a. Check **I acknowledge that AWS CloudFormation might create IAM resources**

   b. Click **Submit**

6. **Wait for completion**:

a. Stack creation takes 3-5 minutes
b. Status will change from`CREATE_IN_PROGRESS`to`CREATE_COMPLETE`
c. **Important**: Your EC2 instance is now part of a CloudFormation stack that AWS DevOpsAgent can track!

Optional: Secure SSH access (only if you plan to connect to the instance)

Skip this step if you just want to run the automated test

1. **Navigate to EC2 Security Groups**:
   a. In AWS Console, go to **EC2** → **Security Groups**
   b. Find`AWS-DevOpsAgent-test-sg`
2. **Update SSH rule**:
   a. Select the security group → **Inbound rules** tab → **Edit inbound rules**
   b. Find the SSH rule (port 22)
   c. Change source from`0.0.0.0/0`to your IP:`[YOUR_IP]/32`
   d. Get your IP from https://whatismyipaddress.com
   e. Click **Save rules**

Step 2: Wait for automatic test execution

1. **Automatic test execution**:
   - The CPU stress test will **automatically start 5 minutes** after instance launch
   - No manual intervention required - just wait, the test runs completely in the background
2. **Monitor the test**:
   - Instance boots and prepares the test automatically
   - The script will run for 5 minutes and generate >70% CPU usage
   - CloudWatch alarm should trigger within 8-10 minutes total (5 min delay + 3-5 min for alarm)
3. **Optional: Manual re-run** (for additional testing):
   - Connect to your instance: EC2 console →`AWS-DevOpsAgent-Test-Instance`→ **Connect** → **Session Manager**
   - Run the stress test again:`./cpu-stress-test.sh`
   - Perfect for testing AWS DevOpsAgent's response multiple times

Test option B: Lambda error rate test

Step 1: Deploy CloudFormation stack for Lambda test

1. **Navigate to CloudFormation**:
   a. In AWS Console, go to **CloudFormation**
   b. Click **Create stack** → **With new resources (standard)**
2. **Upload template**:
   a. Create a new local file called`AWS-DevOpsAgent-lambda-test.yaml`
   b. Copy and paste this CloudFormation template into the file:
      i.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'AWS DevOpsAgent Lambda Error Test Stack'
Resources:
  # IAM Role for Lambda function
  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: AWS-DevOpsAgentLambdaTestRole
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      Tags:
        - Key: Name
          Value: AWS-DevOpsAgent-Lambda-Test-Role
        - Key: Purpose
          Value: AWS-DevOpsAgent-Testing
  # Lambda function that generates errors
  TestLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: AWS-DevOpsAgent-test-lambda
      Runtime: python3.12
      Handler: index.lambda_handler
      Role: !GetAtt LambdaExecutionRole.Arn
```

```
      Code:
        ZipFile: |
          import json
          import random
          import time
          from datetime import datetime
          def lambda_handler(event, context):
              print(f"AWS DevOpsAgent Test Lambda - {datetime.now()}")
              print(f"Event: {json.dumps(event)}")

              # Intentionally generate errors for testing
              error_scenarios = [
                  "Simulated database connection timeout",
                  "Test API rate limit exceeded",
                  "Intentional validation error for AWS DevOpsAgent testing"
              ]

              # Always throw an error for testing purposes
              error_message = random.choice(error_scenarios)
              print(f"Generating test error: {error_message}")

              # This will create a Lambda error that CloudWatch will detect
              raise Exception(f"AWS DevOpsAgent Test Error: {error_message}")
      Description: AWS DevOpsAgent beta test function - intentionally generates
errors
      Timeout: 30
      Tags:
        - Key: Name
          Value: AWS-DevOpsAgent-Test-Lambda
        - Key: Purpose
          Value: AWS-DevOpsAgent-Testing
  # CloudWatch Alarm for Lambda errors
  LambdaErrorAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName: AWS-DevOpsAgent-Lambda-Error-Test
      AlarmDescription: AWS-DevOpsAgent beta test - Lambda error rate alarm
      MetricName: Errors
      Namespace: AWS/Lambda
      Statistic: Sum
      Period: 60
      EvaluationPeriods: 1
      Threshold: 0
      ComparisonOperator: GreaterThanThreshold
```

```
      Dimensions:
        - Name: FunctionName
          Value: !Ref TestLambdaFunction
      TreatMissingData: notBreaching
Outputs:
  LambdaFunctionName:
    Description: Lambda Function Name for testing
    Value: !Ref TestLambdaFunction

  LambdaFunctionArn:
    Description: Lambda Function ARN
    Value: !GetAtt TestLambdaFunction.Arn

  AlarmName:
    Description: CloudWatch Alarm Name
    Value: !Ref LambdaErrorAlarm

  TestCommand:
    Description: AWS CLI command to test the function
    Value: !Sub 'aws lambda invoke --function-name ${TestLambdaFunction} --
payload "{\"test\":\"AWS DevOpsAgent validation\"}" response.json'
```

c. In the CloudFormation console, select **Upload a template file**

d. Click **Choose file**

e. Select the `AWS-DevOpsAgent-lambda-test.yaml` file

f. Click **Next**

3. **Configure stack**:

   a. **Stack name**:`AWS-DevOpsAgent-Lambda-Test`

   b. Click **Next**

4. **Configure stack options**:

   a. Leave defaults, click **Next**

5. **Review and create**:

   a. Check **I acknowledge that AWS CloudFormation might create IAM resources**

   b. Click **Submit**

6. **Wait for completion**:

   a. Stack creation takes 2-3 minutes

   b. Status will change to`CREATE_COMPLETE`

Step 2: Trigger Lambda errors

1. **Navigate to Lambda console**:
   a. Go to **AWS Lambda** console
   b. Find your function`AWS-DevOpsAgent-test-lambda`
2. **Test the function**:
   a. Click **Test** tab
   b. Click **Create new event**
   c. **Event name**:`AWS-DevOpsAgent-test-event`
   d. Use this JSON payload:
      i.
```
{
"test": "AWS DevOpsAgent validation",
"timestamp": "2024-01-01T00:00:00Z"
}
```
   e. Click **Save**
3. **Generate errors**:
   a. Click **Test** button 3 times (wait 10 seconds between each)
   b. Each test will generate an intentional error
   c. **CloudWatch alarm** should trigger within 2-3 minutes
   d. **AWS DevOpsAgent** should now be able to detect the alarm with an **Investigation** in the **Operator app** which you will set up next.

Validate AWS DevOps Agent detection

Step 1: Sanity check CloudWatch alarms (optional)

This step is for ensuring that the above tests are now in an alarm state.

For EC2 Test:

- In CloudWatch console, go to **Alarms**
- **Wait 3-5 minutes** after starting the stress test
- Your alarm should show **In alarm** state
- **If still "OK"**: Wait another 2-3 minutes (CloudWatch metrics can be delayed)

For Lambda Test:

- Check`AWS-DevOpsAgent-Lambda-Error-Test`alarm
- Should show **In alarm** within 2-3 minutes of running tests

Step 2: Start a AWS DevOps Agent Investigation

1. Open your **AWS DevOps Agent AgentSpace**
2. Click **Admin access**. This will open the DevOps Agent Space web app in a new window
3. Click the **Start Investigation** button on the right side of the screen
4. Complete the following form:
   a. **Investigation details:** Describe the investigation you'd like to run. Include any details you can about the investigation goals, areas to explore, or relevant information.
   b. **Investigation starting point**: Describe the information you'd like to start the investigation from. You can mention an alarm, metric, log snippet, or anything else to give DevOps Agent a starting point to work from. In this case, provide a summary of the alarms you just created.
   c. **Date and time of incident** (ISO 8601 preferred): YYYY-MM-DDTHH:MMZ
   d. **Name your investigation:** example: `Oncall_investigation_1:2025-10-27`
   e. **AWS Account ID** for the incident
   f. **Region** where the incident occurred
   g. **Priority** - AWS DevOpsAgent allows for two concurrent investigations. The Priority allows for you to define the order of execution of your investigations.
5. Click Investigate to launch the investigation.
6. Click on your Investigation listed in the dashboard. You will be taken to the Investigation Details screen where you can view the granular steps that DevOps Agent is taking.

Expected Results

EC2 test results:

- Detects EC2 CPU alarm
- Identifies root cause: "CPU stress testing workload"
- Shows timeline: Stress test → CPU spike → Alarm
- Provides recommendations for monitoring and scaling

Lambda test results:

- Detects Lambda error rate spike
- Identifies root cause: "Intentional test exceptions"
- Shows timeline: Function invocations → Errors → Alarm
- Provides recommendations for error handling and monitoring

Cleanup instructions

Cleanup test A (EC2 test)

Automatic cleanup

- Instance will auto-terminate after 2 hours (built into CloudFormation template)

Manual cleanup (immediate)

1. **Delete CloudFormation Stack**:
   a. Go to CloudFormation console
   b. Select`AWS-DevOpsAgent-EC2-Teststack`
   c. Click **Delete**
   d. Confirm deletion
   e. **This will automatically delete all resources**: EC2 instance, security group, key pair, and CloudWatch alarm

Cleanup test B (Lambda test)

1. **Delete CloudFormation Stack**:
   a. Go to CloudFormation console
   b. Select`AWS-DevOpsAgent-Lambda-Teststack`
   c. Click **Delete**
   d. Confirm deletion
   e. **This will automatically delete all resources**: Lambda function, IAM role, and CloudWatch alarm

Troubleshooting

Common issues

"Can't connect to EC2 instance"

- **Check Security Group**: Ensure SSH (port 22) is open to your IP
- **Check Key Permissions**: Run `chmod 400 AWS-DevOpsAgent-test-key.pem`
- **Verify Public IP**: Instance must have public IP assigned
- **Wait for Instance**: Ensure instance is in "Running" state

"Alarm not triggering"

- **Wait for Metrics**: CloudWatch metrics can take 2-5 minutes to appear
- **Check CPU Load**: SSH to instance and run `top` to verify CPU >70%
- **Verify Stress Test**: Run `ps aux | grep yes` to see if load processes are running
- **Extended Wait**: Sometimes takes up to 7-8 minutes for first alarm trigger

Test validation

Your AWS DevOp Agent testing is successful when:

Technical validation

- **Investigation Accuracy**: The results of the EC2 test should correctly indicate that the alarm was triggered due to CPU load. The result of the Lambda test should indicate that this was an intentional failure.
- **Timeline Accuracy**: Correct sequence of events shown
- **Recommendation Quality**: Actionable suggestions provided

Getting started with AWS DevOps Agent using AWS CDK

Overview

This guide shows you how to use the AWS Cloud Development Kit (AWS CDK) to create and deploy AWS DevOps Agent resources. The AWS CDK application automates the creation of an agent space,

AWS Identity and Access Management (IAM) roles, an operator app, and AWS account associations through AWS CloudFormation.

The AWS CDK approach automates the manual steps described in the CLI onboarding guide by defining all required resources as infrastructure as code.

AWS DevOps Agent is available in the following 6 AWS Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), and Europe (Ireland). For more information about supported Regions, see the section called "Supported Regions".

Prerequisites

Before you begin, make sure that you have the following:

- AWS Command Line Interface (AWS CLI) installed and configured with the appropriate credentials
- Node.js version 18 or later
- AWS CDK command line interface (CLI) installed globally. To install the AWS CDK CLI, run the following command:

```
npm install -g aws-cdk
```

- One AWS account for the monitoring (primary) account
- (Optional) A second AWS account if you want to set up cross-account monitoring

What this guide covers

This guide is divided into two parts:

- **Part 1** — Deploy an agent space with an operator app and an AWS association in your monitoring account. After you complete this part, the agent can monitor issues in that account.
- **Part 2 (Optional)** — Add a source AWS association for a service account and deploy a cross-account IAM role into that account. This configuration enables the agent space to monitor resources across accounts.

Resources created

Part 1: DevOpsAgentStack (monitoring account)

- **IAM role** (`DevOpsAgentRole-AgentSpace`) — Assumed by the DevOps Agent service to monitor the account. Includes the `AIDevOpsAgentAccessPolicy` managed policy and an inline policy that allows creation of the Resource Explorer service-linked role.
- **IAM role** (`DevOpsAgentRole-WebappAdmin`) — Operator app role with the `AIDevOpsOperatorAppAccessPolicy` managed policy for agent operations.
- **Agent space** (`MyCDKAgentSpace`) — The central agent space, created by using the `AWS::DevOpsAgent::AgentSpace` CloudFormation resource. Includes operator app configuration.
- **Association** (AWS monitor) — Links the monitoring account to the agent space by using the `AWS::DevOpsAgent::Association` CloudFormation resource.
- **Association** (AWS source) — (Optional) Links the service account to the agent space for cross-account monitoring.

Part 2: ServiceStack (service account, optional)

- **IAM role** (`DevOpsAgentRole-SecondaryAccount`) — Cross-account role with a fixed name. Trusted by the agent space in the monitoring account. Includes the `AIDevOpsAgentAccessPolicy` managed policy and an inline policy that allows creation of the Resource Explorer service-linked role.
- **Lambda function** (`echo-service`) — A simple example service that echoes back input events.

Setup

Step 1: Clone the sample repository

Run the following commands to clone the repository and change to the project directory:

```
git clone https://github.com/aws-samples/sample-aws-devops-agent-cdk.git
cd sample-aws-devops-agent-cdk
```

Step 2: Install dependencies

Run the following command to install the project dependencies:

```
npm install
```

Part 1: Deploy the agent space

In this section, you create the agent space, IAM roles, operator app, and an AWS association in your monitoring account.

Step 1: Configure the monitoring account ID

Open `lib/constants.ts` and set your monitoring account ID:

The following example shows the constant to update:

```
export const MONITORING_ACCOUNT_ID = "<YOUR_MONITORING_ACCOUNT_ID>";
```

Step 2: Bootstrap the AWS CDK environment

If you haven't bootstrapped the AWS CDK in your monitoring account, run the following command:

```
cdk bootstrap aws://<MONITORING_ACCOUNT_ID>/<REGION> --profile monitoring
```

Step 3: Build and deploy

Run the following commands to build the TypeScript code and deploy the stack:

```
npm run build
cdk deploy DevOpsAgentStack --profile monitoring
```

Step 4: Record the stack outputs

After deployment completes, the AWS CDK prints the stack outputs. Record these values for later use.

The following example shows the expected output:

```
Outputs:
```

```
DevOpsAgentStack.AgentSpaceArn = arn:aws:aidevops:<REGION>:123456789012:agentspace/
abc123
DevOpsAgentStack.AgentSpaceRoleArn = arn:aws:iam::123456789012:role/DevOpsAgentRole-
AgentSpace
DevOpsAgentStack.OperatorRoleArn = arn:aws:iam::123456789012:role/DevOpsAgentRole-
WebappAdmin
DevOpsAgentStack.AssociationId = assoc-xyz
```

If you plan to complete Part 2, save the `AgentSpaceArn` value. You need it to configure the service account stack.

Step 5: Verify the deployment

To verify that the agent space was created successfully, run the following AWS CLI command:

```
aws devopsagent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

At this point, your agent space is deployed with the operator app enabled and your monitoring account associated. The agent can monitor issues in this account.

Part 2 (Optional): Add cross-account monitoring

In this section, you extend the setup so that your agent space can monitor resources in a second AWS account (the service account). This involves two actions:

1. Adding a source AWS association in the DevOpsAgentStack that points to the service account.
2. Deploying the ServiceStack into the service account with an IAM role that trusts the agent space.

> **Important**
>
> You must complete Part 1 before you proceed. The ServiceStack requires the `AgentSpaceArn` from the DevOpsAgentStack deployment output.

Step 1: Configure the service account ID

Open `lib/constants.ts` and set your service account ID:

The following example shows the constant to update:

```
export const SERVICE_ACCOUNT_ID = "<YOUR_SERVICE_ACCOUNT_ID>";
```

The DevOpsAgentStack creates a source AWS association by using this account ID. If you deployed the DevOpsAgentStack before setting this value, redeploy to create the association:

Run the following commands to redeploy:

```
npm run build
cdk deploy DevOpsAgentStack --profile monitoring
```

Step 2: Set the agent space ARN

Copy the `AgentSpaceArn` value from the DevOpsAgentStack output (Part 1, Step 4) and set it in `lib/constants.ts`:

The following example shows the constant to update:

```
export const AGENT_SPACE_ARN =
 "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/<SPACE_ID>";
```

The ServiceStack uses this value to scope the trust policy on the secondary account role. The ServiceStack is only synthesized when this value is set.

Step 3: Bootstrap the service account

If you haven't bootstrapped the AWS CDK in your service account, run the following command:

```
cdk bootstrap aws://<SERVICE_ACCOUNT_ID>/<REGION> --profile service
```

Step 4: Deploy the ServiceStack

Run the following commands to build and deploy the ServiceStack by using credentials for the service account:

```
npm run build
cdk deploy ServiceStack --profile service
```

This creates the following resources in the service account:

- An IAM role (`DevOpsAgentRole-SecondaryAccount`) that trusts the agent space in the monitoring account
- An echo Lambda function (`echo-service`) as an example service

Step 5: Verify the deployment

To confirm that the Lambda function was deployed successfully, run the following commands to test the echo service:

```
aws lambda invoke \
  --function-name echo-service \
  --payload '{"test": "hello world"}' \
  --profile service \
  response.json
cat response.json
```

Troubleshooting

This section describes common issues and how to resolve them.

CloudFormation resource type not found

- Verify that you are deploying in a the section called “Supported Regions”.
- Confirm that your AWS CLI is configured with the appropriate permissions.

IAM role creation failed

- Verify that your deployment role has permissions to create IAM roles.
- Check that the trust policy conditions match your account ID.

Cross-account deployment fails with "Could not assume role in target account"

- Each stack must be deployed with credentials for the target account. Use the `--profile` flag to specify the correct AWS CLI profile.
- Verify that the AWS CDK has been bootstrapped in the target account.

IAM propagation delays

- IAM role changes can take a few minutes to propagate. If the agent space creation fails immediately after role creation, wait a few minutes and redeploy.

Cleanup

To remove all resources, destroy the stacks in reverse order.

Run the following commands to destroy the stacks:

```
# If you deployed the ServiceStack, destroy it first
cdk destroy ServiceStack --profile service
# Then destroy the DevOpsAgentStack
cdk destroy DevOpsAgentStack --profile monitoring
```

Warning: This action permanently deletes your agent space and all associated data. This action can't be undone. Make sure that you have backed up any important information before you proceed.

Security considerations

- The AWS CDK application creates IAM roles with trust policies that only allow the `aidevops.amazonaws.com` service principal to assume them.
- Trust policies include conditions that restrict access to your specific AWS account and agent space ARN.
- All policies follow the principle of least privilege. Review and customize the IAM policies based on your organization's security requirements.
- The cross-account role (`DevOpsAgentRole-SecondaryAccount`) uses a fixed name and is scoped to a specific agent space ARN.

Next steps

After you have deployed your AWS DevOps Agent by using the AWS CDK:

1. Learn about the full range of DevOps Agent capabilities in the AWS DevOps Agent User Guide.
2. Consider integrating the AWS CDK deployment into your CI/CD pipelines for automated infrastructure management.

Additional resources

- AWS DevOps Agent User Guide
- Sample CDK repository on the GitHub website
- CLI onboarding guide

Getting started with AWS DevOps Agent using AWS CloudFormation

Overview

This guide shows you how to use AWS CloudFormation templates to create and deploy AWS DevOps Agent resources. The templates automate the creation of an agent space, AWS Identity and Access Management (IAM) roles, an operator app, and AWS account associations as infrastructure as code.

The CloudFormation approach automates the manual steps described in the CLI onboarding guide by defining all required resources in declarative YAML templates.

AWS DevOps Agent is available in the following 6 AWS Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), and Europe (Ireland). For more information about supported Regions, see the section called “Supported Regions”.

Prerequisites

Before you begin, make sure that you have the following:

- AWS Command Line Interface (AWS CLI) installed and configured with the appropriate credentials
- Permissions to create IAM roles and CloudFormation stacks
- One AWS account for the monitoring (primary) account
- (Optional) A second AWS account if you want to set up cross-account monitoring

What this guide covers

This guide is divided into two parts:

- **Part 1** — Deploy an agent space with an operator app and an AWS association in your monitoring account. After you complete this part, the agent can monitor issues in that account.
- **Part 2 (Optional)** — Deploy a cross-account IAM role into a secondary account and add a source AWS association. This configuration enables the agent space to monitor resources across accounts.

Part 1: Deploy the agent space

In this section, you create a CloudFormation template that provisions the agent space, IAM roles, operator app, and an AWS association in your monitoring account.

Step 1: Create the CloudFormation template

Save the following template as `devops-agent-stack.yaml`:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS DevOps Agent - Agent Space with IAM roles, operator app, and AWS
 association

Parameters:
  AgentSpaceName:
    Type: String
    Default: MyCloudFormationAgentSpace
    Description: Name for the agent space
  AgentSpaceDescription:
    Type: String
    Default: Agent space deployed with CloudFormation
    Description: Description for the agent space

Resources:
  # IAM role assumed by the DevOps Agent service to monitor the account
  DevOpsAgentSpaceRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: DevOpsAgentRole-AgentSpace
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: aidevops.amazonaws.com
```

```
            Action: sts:AssumeRole
            Condition:
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
              ArnLike:
                aws:SourceArn: !Sub arn:aws:aidevops:${AWS::Region}:
${AWS::AccountId}:agentspace/*
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AIDevOpsAgentAccessPolicy
      Policies:
        - PolicyName: AllowCreateServiceLinkedRoles
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Sid: AllowCreateServiceLinkedRoles
                Effect: Allow
                Action:
                  - iam:CreateServiceLinkedRole
                Resource:
                  - !Sub arn:aws:iam::${AWS::AccountId}:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer

  # IAM role for the operator app interface
  DevOpsOperatorRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: DevOpsAgentRole-WebappAdmin
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: aidevops.amazonaws.com
            Action:
              - sts:AssumeRole
              - sts:TagSession
            Condition:
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
              ArnLike:
                aws:SourceArn: !Sub arn:aws:aidevops:${AWS::Region}:
${AWS::AccountId}:agentspace/*
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AIDevOpsOperatorAppAccessPolicy
```

```
  # The agent space resource
  AgentSpace:
    Type: AWS::DevOpsAgent::AgentSpace
    DependsOn:
      - DevOpsAgentSpaceRole
      - DevOpsOperatorRole
    Properties:
      Name: !Ref AgentSpaceName
      Description: !Ref AgentSpaceDescription
      OperatorApp:
        Iam:
          OperatorAppRoleArn: !GetAtt DevOpsOperatorRole.Arn

  # Association linking the monitoring account to the agent space
  MonitorAssociation:
    Type: AWS::DevOpsAgent::Association
    Properties:
      AgentSpaceId: !GetAtt AgentSpace.AgentSpaceId
      ServiceId: aws
      Configuration:
        Aws:
          AssumableRoleArn: !GetAtt DevOpsAgentSpaceRole.Arn
          AccountId: !Ref AWS::AccountId
          AccountType: monitor

Outputs:
  AgentSpaceId:
    Description: The agent space ID
    Value: !GetAtt AgentSpace.AgentSpaceId
  AgentSpaceArn:
    Description: The agent space ARN
    Value: !GetAtt AgentSpace.Arn
  AgentSpaceRoleArn:
    Description: The agent space IAM role ARN
    Value: !GetAtt DevOpsAgentSpaceRole.Arn
  OperatorRoleArn:
    Description: The operator app IAM role ARN
    Value: !GetAtt DevOpsOperatorRole.Arn
```

Step 2: Deploy the stack

Run the following command to deploy the stack. Replace <REGION> with a the section called "Supported Regions" (for example, us-east-1).

```
aws cloudformation deploy \
  --template-file devops-agent-stack.yaml \
  --stack-name DevOpsAgentStack \
  --capabilities CAPABILITY_NAMED_IAM \
  --region <REGION>
```

Step 3: Record the stack outputs

After deployment completes, run the following command to retrieve the stack outputs. Record these values for later use.

```
aws cloudformation describe-stacks \
  --stack-name DevOpsAgentStack \
  --query 'Stacks[0].Outputs' \
  --region <REGION>
```

The following example shows the expected output:

```
[
  {
    "OutputKey": "AgentSpaceId",
    "OutputValue": "abc123def456"
  },
  {
    "OutputKey": "AgentSpaceArn",
    "OutputValue": "arn:aws:aidevops:<REGION>:<ACCOUNT_ID>:agentspace/abc123def456"
  },
  {
    "OutputKey": "AgentSpaceRoleArn",
    "OutputValue": "arn:aws:iam::<ACCOUNT_ID>:role/DevOpsAgentRole-AgentSpace"
  },
  {
    "OutputKey": "OperatorRoleArn",
    "OutputValue": "arn:aws:iam::<ACCOUNT_ID>:role/DevOpsAgentRole-WebappAdmin"
  }
]
```

If you plan to complete Part 2, save the `AgentSpaceArn` value. You need it to configure the cross-account role.

Step 4: Verify the deployment

To verify that the agent space was created successfully, run the following AWS CLI command:

```
aws devops-agent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

At this point, your agent space is deployed with the operator app enabled and your monitoring account associated. The agent can monitor issues in this account.

Part 2 (Optional): Add cross-account monitoring

In this section, you extend the setup so that your agent space can monitor resources in a second AWS account (the service account). This involves two actions:

1. Deploying an IAM role in the service account that trusts the agent space.
2. Adding a source AWS association in the monitoring account that points to the service account.

> **Note:** You must complete Part 1 before you proceed. The service account template requires the `AgentSpaceArn` from the Part 1 stack outputs.

Step 1: Create the service account template

Save the following template as `devops-agent-service-account.yaml`. This template creates a cross-account IAM role in the secondary account.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS DevOps Agent - Cross-account IAM role for secondary account monitoring

Parameters:
  MonitoringAccountId:
    Type: String
    Description: The 12-digit AWS account ID of the monitoring account
  AgentSpaceArn:
    Type: String
    Description: The ARN of the agent space from the monitoring account
```

```
Resources:
  # Cross-account IAM role trusted by the agent space
  DevOpsSecondaryAccountRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: DevOpsAgentRole-SecondaryAccount
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: aidevops.amazonaws.com
            Action: sts:AssumeRole
            Condition:
              StringEquals:
                aws:SourceAccount: !Ref MonitoringAccountId
              ArnLike:
                aws:SourceArn: !Ref AgentSpaceArn
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AIDevOpsAgentAccessPolicy
      Policies:
        - PolicyName: AllowCreateServiceLinkedRoles
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Sid: AllowCreateServiceLinkedRoles
                Effect: Allow
                Action:
                  - iam:CreateServiceLinkedRole
                Resource:
                  - !Sub arn:aws:iam::${AWS::AccountId}:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer

Outputs:
  SecondaryAccountRoleArn:
    Description: The cross-account IAM role ARN
    Value: !GetAtt DevOpsSecondaryAccountRole.Arn
```

Step 2: Deploy the service account stack

Using credentials for the service account, run the following command:

```
aws cloudformation deploy \
```

```
  --template-file devops-agent-service-account.yaml \
  --stack-name DevOpsAgentServiceAccountStack \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameter-overrides \
    MonitoringAccountId=<MONITORING_ACCOUNT_ID> \
    AgentSpaceArn=<AGENT_SPACE_ARN> \
  --region <REGION>
```

Step 3: Add the source AWS association

Switch back to the monitoring account and create a source AWS association. You can do this by creating a separate stack or by updating the original template. The following example uses a standalone template.

Save the following template as `devops-agent-source-association.yaml`:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS DevOps Agent - Source AWS association for cross-account monitoring

Parameters:
  AgentSpaceId:
    Type: String
    Description: The agent space ID from the monitoring account stack
  ServiceAccountId:
    Type: String
    Description: The 12-digit AWS account ID of the service account
  ServiceAccountRoleArn:
    Type: String
    Description: The ARN of the DevOpsAgentRole-SecondaryAccount role in the service
 account

Resources:
  SourceAssociation:
    Type: AWS::DevOpsAgent::Association
    Properties:
      AgentSpaceId: !Ref AgentSpaceId
      ServiceId: aws
      Configuration:
        SourceAws:
          AccountId: !Ref ServiceAccountId
          AccountType: source
          AssumableRoleArn: !Ref ServiceAccountRoleArn
```

```
Outputs:
  SourceAssociationId:
    Description: The source association ID
    Value: !Ref SourceAssociation
```

Deploy the association stack using monitoring account credentials:

```
aws cloudformation deploy \
  --template-file devops-agent-source-association.yaml \
  --stack-name DevOpsAgentSourceAssociationStack \
  --parameter-overrides \
    AgentSpaceId=<AGENT_SPACE_ID> \
    ServiceAccountId=<SERVICE_ACCOUNT_ID> \
    ServiceAccountRoleArn=arn:aws:iam::<SERVICE_ACCOUNT_ID>:role/DevOpsAgentRole-
SecondaryAccount \
  --region <REGION>
```

Verification

Verify your setup by running the following AWS CLI commands:

```
# List your agent spaces
aws devops-agent list-agent-spaces \
  --region <REGION>

# Get details of a specific agent space
aws devops-agent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>

# List associations for an agent space
aws devops-agent list-associations \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

Troubleshooting

This section describes common issues and how to resolve them.

CloudFormation resource type not found

- Verify that you are deploying in a the section called “Supported Regions”.

- Confirm that your AWS CLI is configured with the appropriate permissions.

IAM role creation failed

- Verify that your deployment credentials have permissions to create IAM roles with custom names (`CAPABILITY_NAMED_IAM`).
- Check that the trust policy conditions match your account ID.

Cross-account deployment fails

- Each stack must be deployed with credentials for the target account. Use the `--profile` flag to specify the correct AWS CLI profile.
- Verify that the `AgentSpaceArn` parameter matches the exact ARN from the Part 1 stack outputs.

IAM propagation delays

- IAM role changes can take a few minutes to propagate. If the agent space creation fails immediately after role creation, wait a few minutes and redeploy.

Cleanup

To remove all resources, delete the stacks in reverse order.

Warning: This action permanently deletes your agent space and all associated data. This action can't be undone. Make sure that you have backed up any important information before you proceed.

Run the following commands to delete the stacks:

```
# If you deployed the source association stack, delete it first
aws cloudformation delete-stack \
  --stack-name DevOpsAgentSourceAssociationStack \
  --region <REGION>

aws cloudformation wait stack-delete-complete \
  --stack-name DevOpsAgentSourceAssociationStack \
  --region <REGION>
```

```
# If you deployed the service account stack, delete it next (using service account
 credentials)
aws cloudformation delete-stack \
  --stack-name DevOpsAgentServiceAccountStack \
  --region <REGION>

aws cloudformation wait stack-delete-complete \
  --stack-name DevOpsAgentServiceAccountStack \
  --region <REGION>

# Delete the main stack last
aws cloudformation delete-stack \
  --stack-name DevOpsAgentStack \
  --region <REGION>
```

Next steps

After you have deployed your AWS DevOps Agent by using AWS CloudFormation:

- To connect additional integrations, see *Configuring capabilities for AWS DevOps Agent*.
- To learn about agent skills and capabilities, see the section called "DevOps Agent Skills".
- To understand the operator web app, see the section called "What is a DevOps Agent Web App?".

Getting started with AWS DevOps Agent using Terraform

Overview

This guide shows you how to use Terraform to create and deploy AWS DevOps Agent resources. The Terraform configuration automates the creation of an agent space, IAM roles, an operator app, and AWS account associations.

The Terraform approach automates the manual steps described in the CLI onboarding guide by defining all required resources as infrastructure as code.

AWS DevOps Agent is available in the following 6 AWS Regions: US East (N. Virginia), US West (Oregon), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), and Europe (Ireland). For more information about supported Regions, see the section called "Supported Regions".

Prerequisites

Before you begin, make sure you have the following:

- Terraform >= 1.0 installed
- AWS CLI installed and configured with appropriate credentials
- One AWS account for the monitoring (primary) account
- (Optional) A second AWS account if you want to set up cross-account monitoring

What this guide covers

This guide is divided into two parts:

- **Part 1** — Deploy an agent space with an operator app and an AWS association in your monitoring account. After completing this part, the agent can monitor issues in that account.
- **Part 2 (Optional)** — Add a source AWS association for a service account and deploy a cross-account IAM role plus an echo Lambda into that account. This allows the agent space to monitor resources across accounts.

Resources created

Part 1: Monitoring account

- **IAM role** (`DevOpsAgentRole-AgentSpace-*`) — Assumed by the DevOps Agent service to monitor the account. Includes the `AIDevOpsAgentAccessPolicy` managed policy and an inline policy that allows creation of the Resource Explorer service-linked role.
- **IAM role** (`DevOpsAgentRole-WebappAdmin-*`) — Operator app role with the `AIDevOpsOperatorAppAccessPolicy` managed policy for agent operations.
- **Agent space** (configurable name) — The central agent space, created using the `awscc_devopsagent_agent_space` resource. Includes operator app configuration.
- **Association** (AWS monitor) — Links the monitoring account to the agent space using the `awscc_devopsagent_association` resource.
- **Association** (AWS source) — (Optional) Links the service account to the agent space for cross-account monitoring.

Part 2: Service account (optional)

- **IAM role** (`DevOpsAgentRole-SecondaryAccount-TF`) — Cross-account role with a fixed name. Trusted by the agent space in the monitoring account. Includes the `AIDevOpsAgentAccessPolicy` managed policy and an inline policy that allows creation of the Resource Explorer service-linked role.
- **Lambda function** (`echo-service-tf`) — A simple example service that echoes back input events.

Setup

Step 1: Clone the sample repository

```
git clone https://github.com/aws-samples/sample-aws-devops-agent-terraform.git
cd sample-aws-devops-agent-terraform
```

Step 2: Configure variables

Copy the example variables file and customize it for your environment:

```
cp terraform.tfvars.example terraform.tfvars
```

Edit `terraform.tfvars` with your agent space name and description:

```
agent_space_name        = "MyCompanyAgentSpace"
agent_space_description = "DevOps Agent Space for monitoring production workloads"
```

Part 1: Deploy the agent space

In this section, you create the agent space, IAM roles, operator app, and an AWS association in your monitoring account.

Step 1: Deploy with automation (recommended)

Use the provided deployment script for a streamlined setup:

```
./deploy.sh
```

This script automatically:

- Checks prerequisites (Terraform, AWS CLI, credentials)
- Creates `terraform.tfvars` from example if needed
- Initializes, validates, plans, and applies Terraform

Alternatively, if you prefer manual control:

```
terraform init
terraform plan
terraform apply
```

Type `yes` when prompted to confirm the deployment.

Step 2: Record the outputs

After deployment completes, Terraform prints the outputs. Record these values for later use:

```
Outputs:
agent_space_id              = "abc123"
agent_space_arn             =
 "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/abc123"
agent_space_name            = "MyCompanyAgentSpace"
devops_agentspace_role_arn  = "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/
DevOpsAgentRole-AgentSpace-a1b2c3d4"
devops_operator_role_arn    = "arn:aws:iam::<MONITORING_ACCOUNT_ID>:role/
DevOpsAgentRole-WebappAdmin-a1b2c3d4"
primary_account_id          = "<MONITORING_ACCOUNT_ID>"
primary_account_association_id = "assoc-xyz"
```

If you plan to complete Part 2, save the `agent_space_arn` value. You will need it to configure the service account resources.

Step 3: Verify the deployment

Run the post-deployment verification script:

```
./post-deploy.sh
```

Or use the AWS CLI to verify that the agent space was created successfully:

```
aws devops-agent get-agent-space \
  --agent-space-id <AGENT_SPACE_ID> \
  --region <REGION>
```

At this point, your agent space is deployed with the operator app enabled and your monitoring account associated. The agent can monitor issues in this account.

Part 2 (Optional): Add cross-account monitoring

In this section, you extend the setup so the agent space can monitor resources in a second AWS account (the service account). This involves two actions:

1. Adding a source AWS association that points to the service account.
2. Deploying a cross-account IAM role and an echo Lambda function into the service account.

> **Important**
>
> You must complete Part 1 before proceeding. The service account resources require the `agent_space_arn` from the Part 1 deployment output.

Step 1: Configure the service account ID

In `terraform.tfvars`, set your service account ID:

```
service_account_id = "<YOUR_SERVICE_ACCOUNT_ID>"
```

Step 2: Set the agent space ARN

Copy the `agent_space_arn` value from the Part 1 output (Step 2) and set it in `terraform.tfvars`:

```
agent_space_arn = "arn:aws:aidevops:<REGION>:<MONITORING_ACCOUNT_ID>:agentspace/
<SPACE_ID>"
```

The service account resources use this value to scope the trust policy on the secondary account role. These resources are only created when this value is set.

Step 3: Configure the `aws.service` provider

In `main.tf`, configure the `aws.service` provider alias with credentials for the service account. You can use either a named profile or an assume role:

Using a profile:

```
provider "aws" {
  alias   = "service"
  region  = var.aws_region
  profile = "your-service-account-profile"
}
```

Or using assume role:

```
provider "aws" {
  alias  = "service"
  region = var.aws_region
  assume_role {
    role_arn = "arn:aws:iam::<SERVICE_ACCOUNT_ID>:role/OrganizationAccountAccessRole"
  }
}
```

Step 4: Deploy

Apply the updated configuration:

```
terraform apply
```

This creates the following resources in the service account:

- An IAM role (`DevOpsAgentRole-SecondaryAccount-TF`) that trusts the agent space in the monitoring account
- An echo Lambda function (`echo-service-tf`) as an example service

It also creates a source AWS association in the monitoring account that links the service account.

Step 5: Verify the deployment

Test the echo service to confirm the Lambda function was deployed successfully:

```
aws lambda invoke \
  --function-name echo-service-tf \
  --payload '{"test": "hello world"}' \
  --profile <your-service-account-profile> \
  --region <REGION> \
  response.json
cat response.json
```

Troubleshooting

IAM propagation delays

- The configuration includes a 30-second `time_sleep` between IAM role creation and Agent Space creation. The DevOps Agent service validates the operator role's trust policy during Agent Space creation, and this can fail if IAM hasn't fully propagated. If you still see trust policy errors, wait a minute and run `terraform apply` again — the IAM roles will already exist and the apply will pick up where it left off.

Permission errors

- Verify that your AWS credentials have the necessary IAM permissions to create roles and policies.
- Check that the trust policy conditions match your account ID.

Cross-account deployment fails

- The `aws.service` provider must be configured with credentials for the service account. Use a named profile or an assume role block.
- Verify that the `agent_space_arn` value matches the ARN from the Part 1 output.

Terraform resource type not found

- Verify that you have the `awscc` provider version `~> 1.0` or later. The `awscc_devopsagent_agent_space` and `awscc_devopsagent_association` resources require the AWS Cloud Control provider.

Cleanup

To remove all resources, destroy in reverse order if you deployed Part 2:

```
./cleanup.sh
```

Or manually:

```
terraform destroy
```

Warning: This permanently deletes your agent space and all associated data. Make sure you have backed up any important information before proceeding.

Security considerations

- The Terraform configuration creates IAM roles with trust policies that only allow the `aidevops.amazonaws.com` service principal to assume them.
- Trust policies include conditions that restrict access to your specific AWS account and agent space ARN.
- All policies follow the principle of least privilege. Review and customize the IAM policies based on your organization's security requirements.
- The cross-account role (`DevOpsAgentRole-SecondaryAccount-TF`) uses a fixed name and is scoped to a specific agent space ARN.

Next steps

After you have deployed your AWS DevOps Agent using Terraform:

1. Learn about the full range of DevOps Agent capabilities in the AWS DevOps Agent User Guide.
2. Consider integrating the Terraform deployment into your CI/CD pipelines for automated infrastructure management.

Additional resources

- AWS DevOps Agent User Guide
- Sample Terraform repository

- CLI onboarding guide

Working with DevOps Agent

Working with DevOps Agent

AWS DevOps Agent works alongside your operations team across the full incident lifecycle — from detection through investigation, recovery, and prevention. The following topics describe how to use DevOps Agent to manage each phase of this lifecycle.

Autonomous incident response

When an incident is detected — whether through a built-in integration with your ticketing system, a webhook from your monitoring tools, or a manual trigger — DevOps Agent automatically begins an investigation. The agent analyzes metrics, logs, traces, code changes, and deployment history to determine a root cause and propose a mitigation plan. If you need additional help, you can escalate directly to AWS Support from the DevOps Agent Space web app, which automatically shares the investigation context with support engineers so you don't have to repeat what the agent already found. For more information, see the section called “Autonomous incident response”.

On-demand DevOps tasks

At any point during the incident lifecycle, you can interact with DevOps Agent through a conversational chat interface. Ask questions about your AWS resources, system health, alarm status, and deployment history using natural language. Chat is context-aware — when you're viewing a specific investigation, you can steer the agent to explore particular hypotheses, focus on specific logs, or update its root cause analysis. You can also query resource configurations, error trends, and investigation insights across your environment without navigating between consoles. For more information, see the section called “On Demand DevOps Tasks”.

Proactive incident prevention

After resolving incidents, DevOps Agent analyzes patterns across your investigation history to generate recommendations that prevent future incidents and reduce mean time to detection. Recommendations span four areas: observability posture, testing gaps, code changes, and infrastructure architecture. The agent runs evaluations weekly and updates recommendations as new incidents occur. You can accept, reject, or track recommendations, and the agent learns from

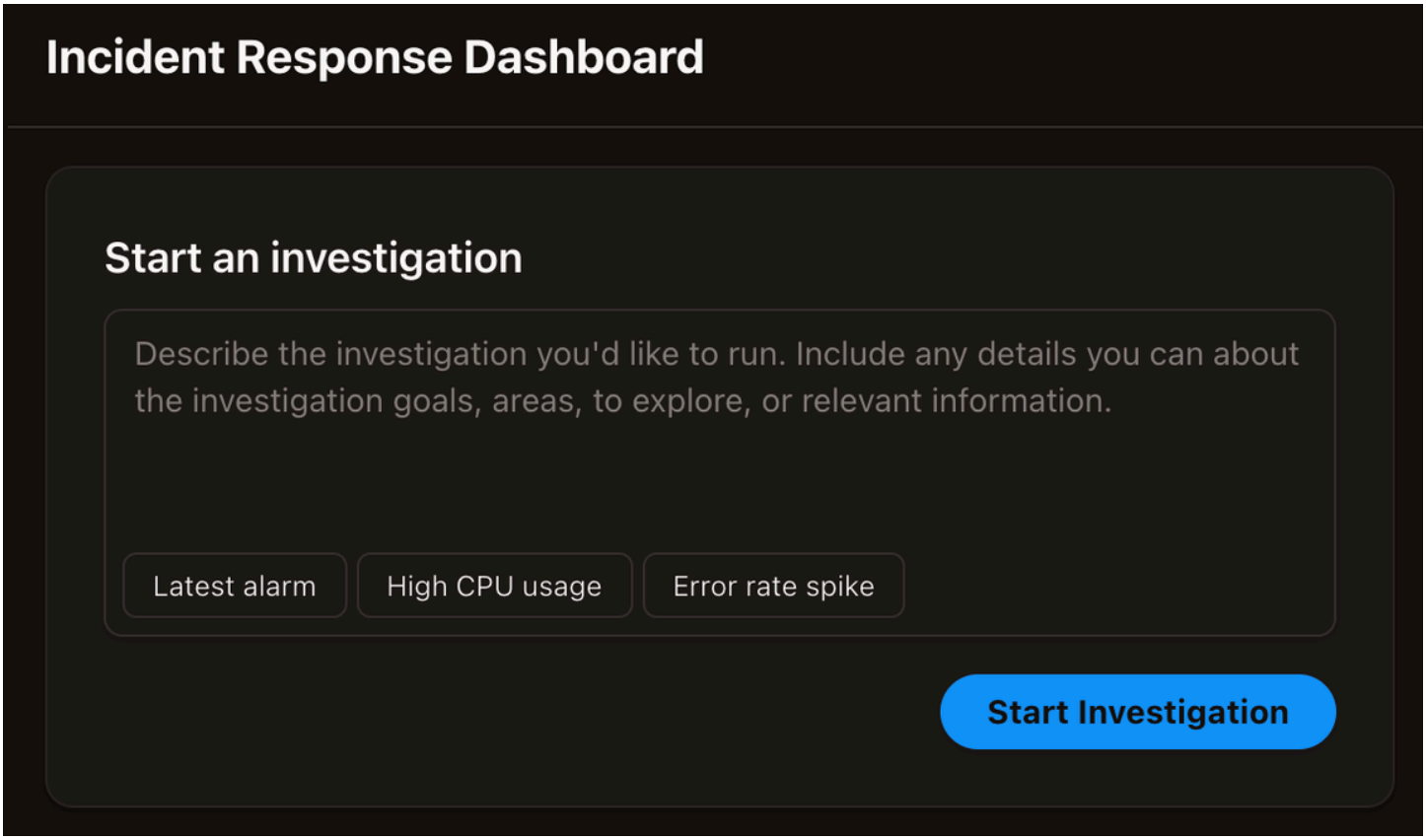
your feedback to refine future suggestions. For more information, see the section called “Proactive incident prevention”.

Autonomous incident response

Starting Investigations

Incident response investigations can be started in one of three ways.

- **Built-in integrations** - You can connect a DevOps Agent Space to ticketing systems like ServiceNow using built-in integrations. Once connected, DevOps Agent incident response investigations will be automatically triggered from support tickets, and your DevOps Agent will provide updates of its key findings, root cause analyses, and mitigation plans into the originating ticket.
- **Webhooks** - You can use webhooks to send events to AWS DevOps Agent. For example you can use webhooks to trigger incident response investigations from PagerDuty tickets or Grafana alarms.
- **Manually** - You can manually start incident response investigations from the Incident Response tab of any DevOps Agent Space web app. You can either enter free form text that describes the incident you want your DevOps Agent to investigate, and it will create an investigation plan, collect findings, determine a root cause, and offer to generate a mitigation plan. You can also choose from several pre-configured starting points to quickly begin your Investigation: Latest alarm to investigate your most recent triggered alarm and analyze the underlying metrics and logs to determine the root cause, High CPU usage to investigate high CPU utilization metrics across your compute resources and identify which processes or services are consuming excessive resources, or Error rate spike to investigate the recent increase in application error rates by analyzing metrics, application logs, and identifying the source of failures.



Once you click "Start Investigation" you'll be asked to provide some additional details to help the agent focus its work. The investigation dialog includes the following fields:

- **Investigation details** – Pre-filled with your description. You can edit this to refine the investigation scope.
- **Investigation starting point** – Optionally describe a specific alarm, metric, log snippet, or other starting point for the agent.
- **Date and time of incident** – Auto-filled with the current time in UTC format. Adjust if the incident occurred earlier.
- **Name your investigation** – Auto-generated with a timestamp. You can customize this (maximum 400 characters).
- **Priority** – Select the investigation priority from the dropdown (Medium is the default).

Review and adjust these fields as needed, then click "Start investigating..." to begin. You will then be taken to the investigation details page where you can see your DevOps Agent in action!

Incident triage

The triage phase is the first stage of AWS DevOps Agent's incident response system. When an external event triggers, such as an alarm from Datadog, an incident ticket from ServiceNow, or a problem from Dynatrace, AWS DevOps Agent automatically processes it within seconds to determine whether it should be investigated independently or linked to an existing investigation.

The primary function of the triage stage is incident correlation — identifying related incidents and consolidating them into a single investigation to avoid duplicate work and resource waste. When a new incident arrives, AWS DevOps Agent analyzes it alongside active investigations within a look-back window (typically 20 minutes). Using AI-powered analysis, it examines factors like component similarities, geographic region, and timing patterns to determine relationships between incidents.

AWS DevOps Agent makes one of two decisions:

- **Linked** – Correlates the incident to an existing investigation and sends a steering message to that investigation with context about the new incident.
- **Proceed** – Schedules a new independent investigation for the incident.

Viewing triage decisions

When incidents are linked, the primary investigation receives a steering message containing the linked incident's details and correlation reasoning. On your AWS DevOps Agent Space web app, you'll see a status of **LINKED** along with correlation reasoning explaining why incidents were linked. The primary investigation displays a list of all linked incidents, allowing you to see the full scope of related issues being investigated together. Your external ticket system (ServiceNow, PagerDuty, etc.) and communication channel (Slack) will receive a notification that the incident was linked along with correlation reasoning.

Unlinking incidents and custom correlation rules

If AWS DevOps Agent incorrectly correlates incidents, you can manually unlink them through the AWS DevOps Agent Space web app. This will reschedule the unlinked incident as an independent investigation. You can also provide custom correlation rules to guide AWS DevOps Agent by creating an AWS DevOps Agent Skill containing your correlation logic and associating it with the triage stage.

Ask for human support

AWS DevOps Agent can connect directly with AWS Support to streamline your incident response process. When you need additional help from AWS Support, from your DevOps Agent Space web app you can create support cases that automatically share investigation context with AWS Support engineers, reducing the time needed to explain your issue.

How it works

When investigating an incident, AWS DevOps Agent builds a comprehensive log of its analysis, including:

- Root cause investigation findings
- Metrics, logs, and traces analyzed
- Code changes and deployment history reviewed
- Remediation actions recommended
- Timeline of events and system behavior

You can escalate your investigation to AWS Support directly from the AWS DevOps Agent Space web app. When you do, AWS DevOps Agent automatically passes its investigation log to AWS Support, providing the support engineer with full context about your investigation without requiring you to manually gather and explain the details.

Chatting with AWS Support

Once you create a support case, you can communicate with AWS Support in a separate chat window within your AWS DevOps Agent Space web app. This allows you to:

- Discuss your issue with AWS Support engineers alongside your AWS DevOps Agent's investigation timeline
- View both AWS DevOps Agent's automated analysis and AWS Support's expert guidance in the same interface
- Seamlessly share additional information or clarification as needed

The chat experience keeps your AWS DevOps Agent investigation and AWS Support conversation readily accessible, enabling faster collaboration and resolution.

Support plan requirements

Your ability to create and interact with support cases through AWS DevOps Agent depends on your AWS Support plan. Please refer to the Support Plans user guide to learn more about your entitlements.

Note Basic Support customers cannot create technical support cases and therefore cannot escalate AWS DevOps Agent investigations to AWS Support **Developer Support customers** can create cases through AWS DevOps Agent, but must visit the AWS Support Center to correspond with Support engineers, as Developer Support does not include chat-based support **All other plans** can use the integrated chat experience within AWS DevOps Agent. For complete details about support plan entitlements, including response times and available case severities, see the AWS Support Plans User Guide.

What information is shared with AWS Support

When you create a support case from AWS DevOps Agent Space web app, the following information is automatically shared with AWS Support:

- **Investigation timeline**: Chronological record of AWS DevOps Agent's analysis
- **Resource information**: Affected AWS resources
- **Observability data**: Relevant metrics, logs, and traces from your integrated monitoring tools
- **Recent changes**: Code deployments, infrastructure changes, and configuration updates
- **Remediation attempts**: Actions AWS DevOps Agent recommended
- **Impact assessment**: Scope and severity of the incident

All data shared with AWS Support follows your existing AWS data residency and security configurations. AWS DevOps Agent shares only information related to your specific investigation and respects your organization's data governance policies.

Getting started

To use AWS DevOps Agent's AWS Support integration:

1. Ensure you have an active AWS Support plan.
2. Verify your AWS DevOps Agent’s IAM permissions include support case creation (support:CreateCase, support:DescribeCases).

3. When AWS DevOps Agent is investigating an issue and you need AWS Support assistance, choose **Ask for human support** from your DevOps Agent Space web app.
4. Review the investigation summary that will be shared with AWS Support.
5. Select the appropriate case severity based on your support plan entitlements.
6. Submit the case - AWS DevOps Agent automatically includes your investigation log.

The chat window opens automatically, allowing you to begin collaborating with AWS Support immediately.

Proactive incident prevention

AWS DevOps Agent analyzes patterns across your incident investigations to deliver targeted recommendations that continuously improve your operational posture and prevent future incidents. Access proactive incident prevention through the Ops Backlog page in the Operator Web App.

How proactive incident prevention works

AWS DevOps Agent evaluates recent incident investigations to identify lasting improvements to prevent future incidents and quicken the mean time to detection (MTTD). The agent analyzes multiple incidents to identify recommendations that may prevent whole classes of incidents in the future, focusing on the most impactful recommendations to ensure they are actionable.

By default, the agent automatically runs evaluations weekly. You can pause the schedule if you prefer to run evaluations only on demand. Manual evaluations are always available, which is useful when a recent investigation warrants a quick turnaround on recommended improvements.

The agent identifies improvements across four categories, shown in the Recommendation Categorization chart on the Ops Backlog page:

- **Observability** – Recommendations to enhance monitoring, alerting, logging, and system visibility to detect issues quicker and more accurately.
- **Infrastructure** – Recommendations to optimize resource configurations, capacity tuning, and architectural resilience.
- **Governance** – Recommendations to strengthen deployment processes, pipeline improvements, testing practices, and operational controls.

- **Code optimization** – Recommendations to improve application code quality, error handling, and code resilience.

This categorization helps you understand where your operational improvements are most needed and allows you to prioritize recommendations based on your team's focus areas.

Benefits

- **Prevent recurring incidents** – Address root causes systematically rather than repeatedly responding to the same types of issues
- **Reduce operational toil** – Free your team from repetitive firefighting to focus on innovation and strategic improvements
- **Improve system resilience** – Strengthen your infrastructure, observability, and deployment processes based on real incident data
- **Learn from historical patterns** – Leverage insights from past incidents to make targeted improvements that have the greatest impact

Agent summary

The Agent Summary in the Ops Backlog page of the Web App provides a description of the outcomes from the last evaluation of recent incidents. The summary explains the number of incident investigations analyzed, which incidents are similar to past ones, and which recommendations were created or updated with new information.

The summary helps you quickly understand what the agent discovered during its most recent evaluation and highlights the most notable recommendations that could have the greatest impact on your operational posture.

Controlling evaluations

You can control when AWS DevOps Agent evaluates incidents and generates recommendations:

- **Running evaluations manually** – Click the **Run Now** button in the Ops Backlog page to start an evaluation immediately. This is useful when a recent investigation warrants a quick turnaround on recommended improvements.
- **Stopping active evaluations** – Click the **Stop Evaluation** button in the Ops Backlog page to halt an evaluation that is currently in progress.

Managing recommendations

AWS DevOps Agent provides recommendations in the Ops Backlog page where you can review and manage them:

- **Viewing recommendation details** – Click on a recommendation to open the recommendation details page, where you can see more information about the suggested improvement including the incidents that informed the recommendation, the expected impacts, and next steps. For recommendations with code changes, you can also view the agent-ready specification that can be handed to a coding agent for implementation.
- **Keep** – Click 'Keep' to retain a recommendation in your backlog for tracking. This allows you to monitor which improvements you plan to implement and track their progress.
- **Discard** – Click 'Discard' to remove a recommendation from your backlog. When you discard a recommendation, you can provide a natural language explanation of why it doesn't meet your needs. The agent learns from this feedback and uses it to inform future recommendations, ensuring they become more aligned with your operational priorities and requirements over time.
- **Implemented** – Click 'Implemented' to mark a recommendation as completed. This helps you track which improvements have been applied and allows the agent to measure the effectiveness of its recommendations over time.
- **Automatic removal** – Recommendations that have not been marked as Keep or Implemented may be removed after approximately 6 weeks if no new incidents would have been prevented by implementing the recommendation. This ensures the Ops Backlog page focuses on the most relevant improvements for your operational challenges.
- **Recommendation updates** – Existing recommendations are updated when newer incidents are found that would have been prevented by the recommendation. Updates may change the recommendation's priority or refine the recommendation based on new insights.

Agent-ready specifications

For recommendations that involve code or configuration changes, AWS DevOps Agent can generate an agent-ready specification. This specification provides a structured document that can be handed directly to a coding agent for implementation.

The specification includes:

- **Problem statement** – A summary of the issue and its root cause

- **Solution summary** – A high-level description of the recommended approach
- **Target repositories** – The specific repositories where changes need to be made
- **Code changes** – Detailed descriptions of what needs to change and why, with specific file paths and implementation considerations
- **Test requirements** – What scenarios need to be tested
- **Implementation plan** – A phased approach to implementing the changes

Agent-ready specifications accelerate implementation by providing coding agents with the context they need to make production-ready changes without requiring extensive back-and-forth with engineers.

Implementing recommendations

To maximize the value of proactive incident prevention recommendations, consider the following practices for acting on them:

- **Using agent-ready specifications** – For recommendations with code changes, use the generated specification to accelerate implementation by handing it to a coding agent or using it as a detailed guide for manual implementation.
- **Adding recommendations to your ticket backlog** – Copy recommendations to your team's ticketing system or project management tool to ensure they are prioritized alongside other engineering work.
- **Prioritizing recommendations based on impact** – Focus first on recommendations that address the most frequent or severe incident types, or those that affect critical systems.
- **Tracking implementation progress** – Monitor which recommendations have been implemented and measure their effectiveness by observing whether similar incidents decrease over time.
- **Coordinating with development teams** – Share recommendations with the appropriate teams who own the affected systems, ensuring they have the context and resources needed to implement improvements.

On Demand DevOps Tasks

AWS DevOps Agent On Demand Tasks is a generative artificial intelligence (AI) powered conversational assistant that enables operations teams to query their application architecture, analyze system health, and access investigation insights using natural language. You can ask

questions about your AWS resources, system metrics, alarm status, deployment history, and incident patterns. Chat provides immediate answers grounded in your actual infrastructure and operations data, eliminating the need to navigate between multiple AWS consoles or monitoring tools.

Chat is integrated throughout the DevOps Agent Space web app and provides context-aware responses based on the page you are viewing. The interface maintains conversation history, enabling you to continue previous discussions and build on earlier queries.

Tasks capabilities

AWS DevOps Agent On Demand Tasks provides comprehensive capabilities to help you manage and understand your infrastructure:

Resource queries – Ask about AWS resources in your Agent Space, including Lambda functions, DynamoDB tables, EKS deployments, certificates, and infrastructure configurations. Chat can filter and analyze resources based on attributes like runtime versions, capacity settings, or deployment status. For example, ask "How many Lambdas are using Python 3.8?" or "Do I have any certificates about to expire?"

System health analysis – Query current and historical system health metrics, including alarm status, error rates, CPU utilization, and service availability. Chat can generate health summaries covering specific time periods and identify trends in system behavior. Ask questions like "Which alarms fired in the last 24 hours?" or "Any 5xx errors in the last hour?"

Investigation insights – Access information from completed and in-progress investigations, including root cause analysis, hypotheses explored, logs reviewed, and resolution patterns. Chat can identify common incident causes and provide recommendations based on historical data. Query "What is the most common cause of incidents last month?" or "What's the average resolution time for completed investigations?"

Investigation steering – When viewing an investigation detail page, guide the investigation by directing the agent to focus on specific logs, explore particular hypotheses, or update root cause analysis. Provide steering input like "Focus on logs for the payment service and update your RCA" or "Explore the hypothesis that DynamoDB throttling caused the issue."

Chat artifacts – Generate structured reports and documents, such as operational health summaries, error reports, and incident analyses. Artifacts appear in a dedicated panel and support versioned editing within the conversation.

Recommendation filtering – Query incident prevention recommendations with specific criteria, such as recommendations related to particular services or operational concerns. Chat explains the impact and implementation considerations for each recommendation. For example, "Show me recommendations that will prevent incidents involving DynamoDB" or "Which recommendations would help me detect request latency issues quicker?"

Accessing Chat

Chat is available as a persistent panel on the left side of the DevOps Agent Space web app. The left sidebar includes a **+ New chat** button, a **Pages** section for navigating to Incidents, Ops Backlog, and Topology, and a **Chats** section that displays your recent conversations. Choose **View all** to see your full conversation history.

Chat provides context-aware responses based on where you access it:

Topology – Ask general questions about your Agent Space resources, architecture, and operational health. Chat has full visibility into all connected accounts and services. From this context, you can query resource configurations, deployment history, topology information, and observability tool integrations.

Incident Response – When viewing the incident response page, ask questions about investigation trends, resolution times, and incident patterns across your Agent Space. Chat can analyze historical investigation data to identify common causes and improvement opportunities.

Investigation Detail – While viewing a specific investigation, Chat provides context-aware responses about that investigation. Ask about logs reviewed, hypotheses explored, root cause conclusions, and mitigation plans. You can also provide steering input to guide the investigation focus.

Prevention – From the prevention page, query recommendations with filters, understand why recommendations were made, and explore implementation approaches. Chat helps you prioritize and understand the impact of incident prevention recommendations.

The chat interface remains available as you switch between pages, but the context changes to provide relevant information for your current view. When you start a new conversation, it begins without prior context. When you continue an existing conversation, Chat maintains the full conversation history for follow-up questions.

Context-aware responses

Chat adapts its responses based on the page you are viewing in the DevOps Agent Space web app. This context awareness ensures you receive relevant information without needing to specify which investigation or resource scope you are asking about.

When viewing an investigation detail page, Chat automatically understands you are asking about that specific investigation. Questions like "What logs did you look at?" or "Which hypotheses did you explore?" refer to the investigation currently displayed. When you provide steering input, Chat applies it to the active investigation and creates a new root cause version if appropriate.

On the prevention page, Chat understands you are interested in incident prevention recommendations. Queries automatically filter and analyze recommendations within your Agent Space context. The system recognizes whether you are asking about general recommendations or specific recommendation details.

When accessing Chat from the Topology page, Chat provides broad visibility across all resources, metrics, and historical data in your Agent Space. You can ask about any resource, service, or operational concern without specifying the investigation or recommendation context.

This context awareness eliminates the need to repeatedly specify which investigation, recommendation, or resource scope you are referencing, creating a more natural conversational flow.

Managing conversations

Chat maintains conversation history to enable you to continue previous discussions and reference earlier queries.

Creating new conversations – Click the "New session" button in the chat panel to start a fresh conversation without prior context. New conversations do not carry over information from previous chats, allowing you to ask unrelated questions without confusion.

Accessing conversation history – Click "History" to view all previous conversations within your Agent Space. Conversations are organized chronologically with timestamps and preview text. Conversation history is retained for 90 days and is private to your user account within the Agent Space.

Continuing conversations – Select any conversation from your history to resume where you left off. Chat maintains the full context of previous messages, enabling you to ask follow-up

questions that reference earlier parts of the conversation. When you switch pages while viewing a conversation, the conversation context remains but page-specific context updates based on your current location.

Note that conversation history is isolated within each Agent Space. Conversations in one Agent Space are not visible or accessible from other Agent Spaces. This isolation ensures that sensitive information remains compartmentalized according to your organizational boundaries.

Generating artifacts

AWS DevOps Agent supports chat artifacts — structured, versioned documents generated by the agent during a conversation. Artifacts provide a dedicated, interactive panel in the chat UI for reviewing and editing AI-generated content, such as operational reports, error summaries, and health assessments.

You can request artifacts from any page in the DevOps Agent Space web app. Chat uses the current page context to scope the artifact content.

How artifacts work

When you ask Chat to create or update content, Chat generates an artifact — typically a formatted document — and displays it in the artifact panel alongside the conversation.

Generate – Send a natural language request to create a report or document. For example, ask "Generate a weekly operational health report for my Agent Space" or "Show me a report for my 4xx errors from last week".

Review – The artifact appears in a dedicated panel alongside the conversation. You can review the full content while continuing to interact with Chat.

Edit – Request changes to the artifact through Chat. For example, ask "Add a section on Lambda cold starts" or "Update the report to include last month's data". Chat creates a new version of the artifact with your requested changes.

Sample queries

The following examples demonstrate the types of questions you can ask Chat. These examples are organized by use case and context.

Artifact generation queries

From any page in the DevOps Agent Space web app:

- Generate a weekly operational health summary for my Agent Space
- Create a report of all 4xx errors from last week
- Build an incident summary report for the last 30 days
- Create a summary of alarm activity for the payment service this week
- Generate a deployment history report for the last 7 days
- Summarize all open recommendations into a report

Resource information queries

From any page in the DevOps Agent Space web app:

- How many Lambda functions are using Python 3.8?
- Do I have any certificates about to expire?
- List all DynamoDB tables with on-demand billing
- Show me EKS clusters in production
- Which Lambda functions have not been deployed in the last 90 days?
- List S3 buckets without versioning enabled
- What RDS instances are running database version X?

System health queries

From Topology or Incident Response pages:

- Which alarms fired in the last 24 hours?
- Any 5xx errors in the last hour?
- Show me Lambda error trends for the payment service
- What is the CPU utilization for my ECS cluster?
- Are there any unhealthy targets in my load balancers?
- Show me API Gateway throttling events from yesterday
- Which services had the highest error rate last week?
- Give me an overall health report covering the last 24 hours

Observability tool queries

From Topology:

- List Splunk log groups
- Show me Prometheus metrics and their alarm thresholds
- What Datadog monitors are configured for this service?
- List New Relic alert policies
- Show me Dynatrace dashboard configurations

Investigation insights queries

From Incident Response page:

- What is the most common cause of incidents last month?
- What's the average resolution time for completed investigations?
- Summarize investigations from last week and their RCA
- How many incidents were caused by DynamoDB throttling?
- Show me investigation trends over the past quarter
- Which services have the most frequent incidents?

Investigation detail queries

From Investigation Detail page:

- What logs did you look at?
- Which hypotheses did you explore?
- How risky is the mitigating action you propose?
- What was the timeline of events during this incident?
- Why did you conclude this was the root cause?
- What evidence supports your root cause analysis?
- Who provided steering during your investigation?
- Give me a summary of this incident investigation

Investigation steering queries

From Investigation Detail page:

- Focus on logs for the payment service between 14:00-15:00 UTC and update your RCA
- Explore the hypothesis that DynamoDB throttling caused the issue
- Check the ECS cluster configuration to see if that caused the alarm
- Only check logs for the last 2 hours, not the full day
- Investigate the spike in errors at 3 PM
- Look at the API Gateway logs instead of Lambda logs

Prevention recommendation queries

From Prevention page:

- What are my top 3 incident prevention recommendations?
- Show me recommendations that will prevent incidents involving DynamoDB
- Which recommendations would help me detect request latency issues quicker?
- List observability improvements that could prevent similar incidents
- Show me infrastructure recommendations for the payment service
- Which recommendations have the highest impact on system resilience?

Enabling Chat in your Agent Space

Chat is available in all DevOps Agent Space web apps. The setup process depends on whether you have a new or existing Agent Space.

New Agent Spaces

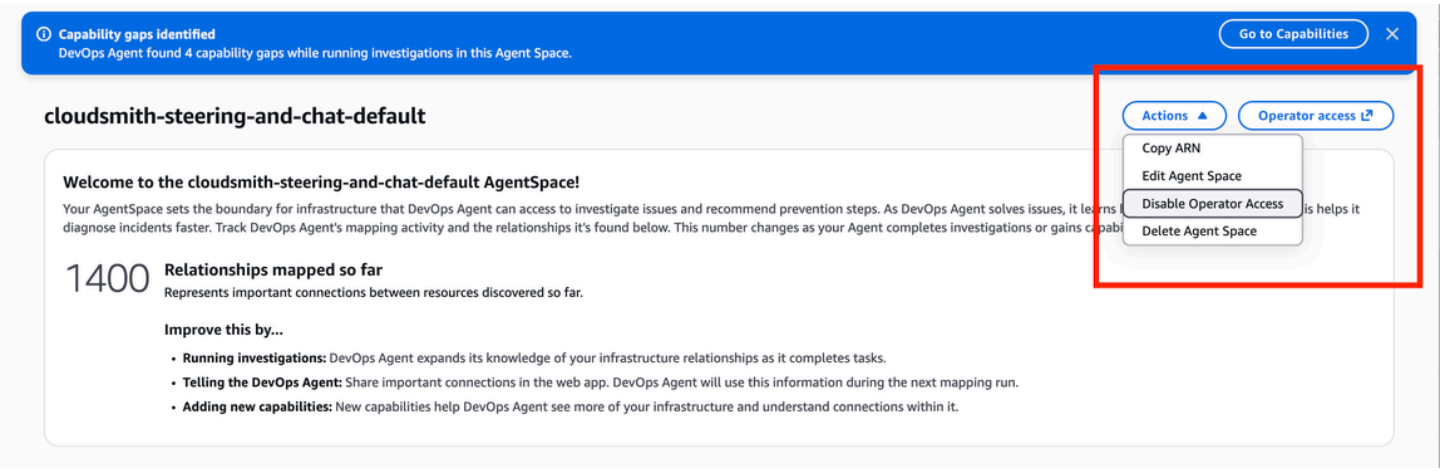
Chat is **automatically enabled** when you create a new Agent Space. No additional configuration or IAM permissions setup is required. After you configure your DevOps Agent Space web app, Chat is immediately available as a persistent panel on the left side of any page.

Existing Agent Spaces

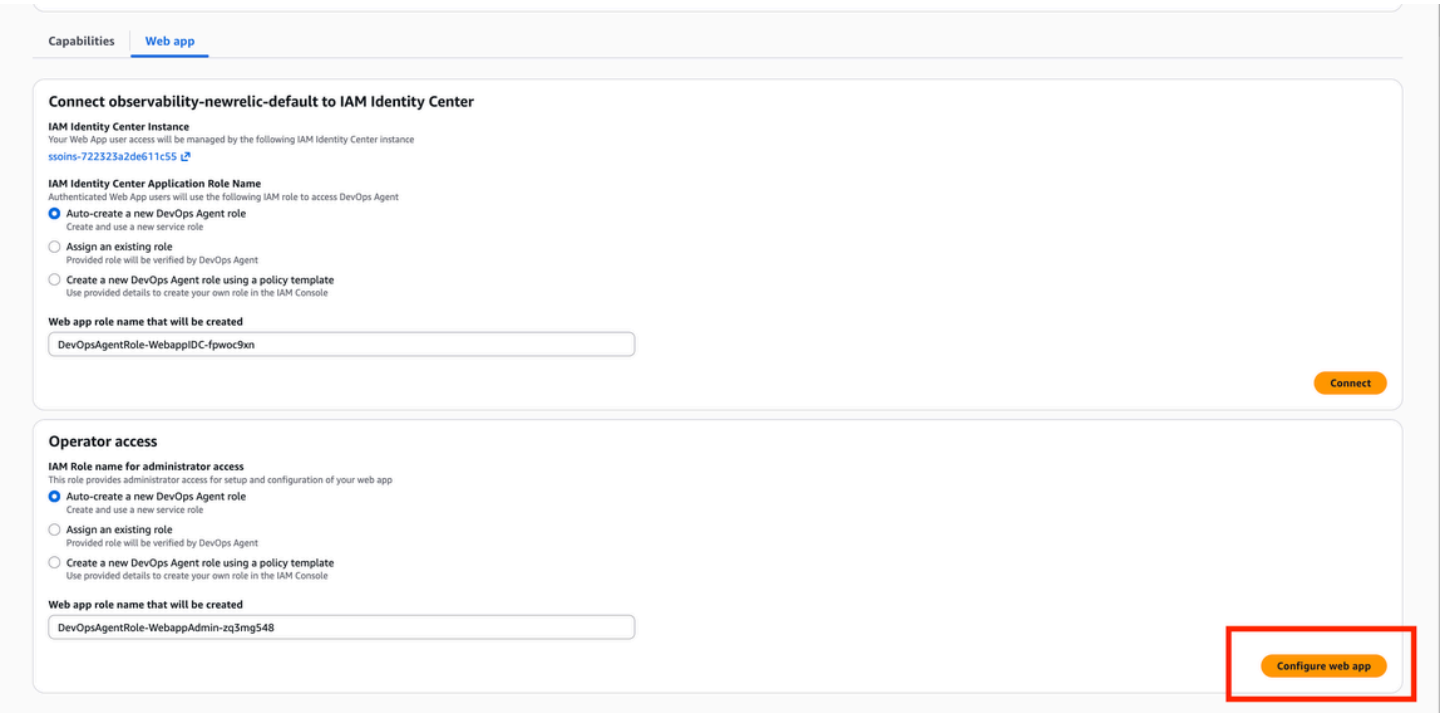
If you created your Agent Space before Chat was released, you must enable the required IAM permissions. You have two options:

Option 1: Revoke and re-enable operator app access

Navigate to the AWS DevOps Agent Admin Console, locate the Action dropdown in the top right corner, and disable the current operator access configuration.



Then enable the auto-create option for operator access.



This automatically applies the required IAM permissions for Chat along with all other current operator permissions.

Option 2: Add IAM permissions manually

Add the following IAM permissions to your existing operator access role:

- `aidevops:ListChats` – View chat conversation history
- `aidevops:CreateChat` – Create new chat conversations
- `aidevops:SendMessage` – Send messages and receive responses

Navigate to the AWS IAM console, locate your DevOps Agent operator role, and add these permissions to the role policy. Chat becomes available immediately after the permissions are added.

After completing either option, refresh your DevOps Agent Space web app and the chat panel appears on the left side of any page.

Configuring capabilities for AWS DevOps Agent

AWS DevOps Agent capabilities extend your agent's functionality by connecting it to your existing tools and infrastructure. Configure these capabilities to enable comprehensive incident investigation, automated response workflows, and seamless integration with your DevOps ecosystem.

The following capabilities help you maximize your DevOps Agent's effectiveness:

- **AWS EKS Access Setup** - Enable introspection of Kubernetes clusters, pod logs, and cluster events for both public and private EKS environments
- **Azure Integration** - Connect Azure subscriptions and Azure DevOps organizations to investigate Azure resources and correlate Azure DevOps deployments with incidents
- **CI/CD Pipeline Integration** - Connect GitHub and GitLab pipelines to correlate deployments with incidents and track code changes during investigations
- **MCP Server Connections** - Extend investigation capabilities by connecting external observability tools and custom monitoring systems through Model Context Protocol
- **Multi-Account AWS Access** - Configure secondary AWS accounts to investigate resources across your entire organization during incident response
- **Telemetry Source Integration** - Connect monitoring platforms like Datadog, Dynatrace, Grafana, New Relic, and Splunk for comprehensive observability data access
- **Ticketing and Chat Integration** - Connect ServiceNow, PagerDuty, and Slack to automate incident response workflows and enable team collaboration
- **Webhook Configuration** - Allow external systems to automatically trigger DevOps Agent investigations through HTTP requests
- **Amazon EventBridge Integration** - Incorporate AWS DevOps Agent into event-driven applications by routing investigation and mitigation lifecycle events to Amazon EventBridge targets

You can configure each capability independently based on your team's specific needs and existing tool stack. Start with the integrations most critical to your incident response workflow, then expand to additional capabilities as needed.

Migrating from public preview to general availability

If you used AWS DevOps Agent during public preview, you must update your IAM roles before the GA release. This guide walks through updating the monitoring roles and operator roles in your accounts.

What's changing

1. On-demand chat histories during preview are no longer accessible
2. New managed policies replace policies available during preview
3. Agent Spaces may have an outdated IAM Identity Center application access scope

On-demand chat history from public preview

The GA release introduces additional security measures to harden access controls for chat histories. As a result of these changes, on-demand chat histories from the public preview period (before March 30, 2026) are no longer accessible. Investigation journals and findings created during public preview are not affected. **This change applies only to on-demand chat conversations.**

New Managed Policies

For GA, AWS provides new managed policies that replace the preview-era policies:

Role type	Remove	Add
Monitoring	`AIOpsAssistantPolicy` managed policy	`AIDevOpsAgentAccessPolicy` managed policy
Operator (IAM and IDC)	Inline policy	`AIDevOpsOperatorAppAccessPolicy` managed policy

In addition, operator roles require updated trust policies, and IDC operator roles require a new inline policy.

Prerequisites

- Access to the AWS accounts where your DevOps Agent roles are configured (primary and all secondary accounts)
- IAM permissions to modify roles, policies, and trust relationships
- Your Agent Space ID, AWS account ID, and Region (visible in the DevOps Agent console)

Step 1: Update monitoring roles

Update the monitoring role in your primary account and in each secondary account. These are the Primary/Secondary source roles configured under the **Capabilities** tab in your agent space (example primary/secondary role: `DevOpsAgentRole-AgentSpace-3xj2396z`).

1. In the DevOps Agent console, go to your Agent Space and choose the **Capabilities** tab.
2. Find the monitoring role for your Primary/Secondary Sources (for example, `DevOpsAgentRole-AgentSpace-3xj2396z`) and choose **Edit**.
3. Under **Permissions policies**, remove the `AIOpsAssistantPolicy` AWS managed policy.
4. Choose **Add permissions**, **Attach policies**, and attach the `AIDevOpsAgentAccessPolicy` managed policy.
5. Edit the inline policy and replace its contents with the following, substituting your account ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateServiceLinkedRoles",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": [
                "arn:aws:iam::<account-id>:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer"
            ]
        }
    ]
}
```

1. The trust policy for the monitoring role does not require changes. Verify it matches the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "aidevops.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account-id>"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:aidevops:<region>:<account-
id>:agentspace/*"
                }
            }
        }
    ]
}
```

- Repeat steps 2–6 for the monitoring role in each secondary account.

Step 2: Update the operator role (IAM)

1. In the DevOps Agent console, choose the **Access** tab and find the operator role.
2. In the IAM console, remove the existing inline policy from the operator role.
3. Choose **Add permissions**, **Attach policies**, and attach the `AIDevOpsOperatorAppAccessPolicy` managed policy.
4. Choose the **Trust relationships** tab and choose **Edit trust policy**. Replace the trust policy with the following, substituting your account ID, Region, and Agent Space ID:

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "aidevops.amazonaws.com"
            },
            "Action": ["sts:AssumeRole", "sts:TagSession"],
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account-id>"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:aidevops:<region>:<account-
id>:agentspace/<agentspace-id>"
                }
            }
        }
    ]
}
```

Step 3: Update operator roles (IDC)

If you use IAM Identity Center with DevOps Agent, update each IDC operator role.

1. In the IAM console, go to **Roles** and search for `WebappIDC` to find your DevOps Agent IDC roles (for example, `DevOpsAgentRole-WebappIDC-<id>`).
2. For each IDC role:

a. Remove the existing inline policy.

b. Choose **Add permissions**, **Attach policies**, and attach the `AIDevOpsOperatorAppAccessPolicy` managed policy.

c. Choose the **Trust relationships** tab and choose **Edit trust policy**. Replace the trust policy with the following, substituting your account ID, Region, and Agent Space ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "aidevops.amazonaws.com"
            },
            "Action": ["sts:AssumeRole", "sts:TagSession"],
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account-id>"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:aidevops:<region>:<account-
id>:agentspace/<agentspace-id>"
                }
            }
        },
        {
            "Sid": "TrustedIdentityPropagation",
            "Effect": "Allow",
            "Principal": {
                "Service": "aidevops.amazonaws.com"
            },
            "Action": "sts:SetContext",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<account-id>"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:aidevops:<region>:<account-
id>:agentspace/<agentspace-id>"
                },
                "ForAllValues:ArnEquals": {
                    "sts:RequestContextProviders": [
                        "arn:aws:iam::aws:contextProvider/IdentityCenter"
                    ]
                },
                "Null": {
                    "sts:RequestContextProviders": "false"
                }
            }
        }
    ]
}
```

d. Create a new inline policy with the following permissions, substituting your account ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDevOpsAgentSSOAccess",
            "Effect": "Allow",
            "Action": [
                "sso:ListInstances",
                "sso:DescribeInstance"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowDevOpsAgentIDCUserAccess",
            "Effect": "Allow",
            "Action": "identitystore:DescribeUser",
            "Resource": [
                "arn:aws:identitystore::<account-id>:identitystore/*",
                "arn:aws:identitystore:::user/*"
            ]
        }
    ]
}
```

Reconnect IAM Identity Center (if applicable)

Agent Spaces created during public preview may have an IAM Identity Center application configured with an outdated access scope. For GA, the correct scope is **`aidevops:read_write`**. If your IAM Identity Center application has the previous scope (**`awsaidevops:read_write`**), you must disconnect and reconnect IAM Identity Center.

How to check your IAM Identity Center application scope

Run the following AWS CLI command to check the scope on your IAM Identity Center application. You can find the application ARN in the IAM Identity Center console under **Applications**.

```
aws sso-admin list-application-access-scopes \
  --application-arn arn:aws:sso::<account-id>:application/<instance-id>/<application-
id>
```

The output should show the correct scope **`aidevops:read_write`**:

```
{
    "Scopes": [
        {
            "Scope": "aidevops:read_write"
        }
    ]
}
```

If the scope shows **`awsaidevops:read_write`**, it is outdated. Follow the steps below to update it.

How to reconnect IAM Identity Center

The access scope on an AWS managed IAM Identity Center application cannot be updated directly. You must disconnect and reconnect:

1. In the AWS DevOps Agent console, go to your Agent Space and choose the **Access** tab.
2. Choose **Disconnect** next to the IAM Identity Center configuration.
3. Confirm the disconnection.
4. Choose **Connect** to set up IAM Identity Center again. The service creates a new IAM Identity Center application with the correct scope.
5. Reassign users and groups to the new application in the IAM Identity Center console.

> **Important**
>
> Disconnecting removes individual user chat and artifact history associated with IAM Identity Center user accounts. Users will need to log in again after reconnection.

Verification

After completing all steps:

1. Return to the DevOps Agent console and verify that no permission errors appear on the Agent Space **Access** tab.
2. Test the operator web app to confirm it loads and functions correctly.
3. If you use IDC, verify that users can authenticate and access the operator experience.

Troubleshooting

Permission denied errors after migration

- Verify that `AIOpsAssistantPolicy` was removed and `AIDevOpsAgentAccessPolicy` is attached to monitoring roles.
- Verify that old inline policies were removed and `AIDevOpsOperatorAppAccessPolicy` is attached to operator roles.
- Check that operator trust policies include `sts:TagSession`.
- Confirm you replaced all placeholder values (`<account-id>`, `<region>`, `<agentspace-id>`) with actual values.

Secondary accounts not working

- Each secondary account's monitoring role must be updated independently. Log into each account and repeat Step 1.

IDC authentication failures

- Verify the IDC trust policy includes both the `sts:AssumeRole/sts:TagSession` statement and the `TrustedIdentityPropagation` statement.
- Confirm the inline policy with `sso:ListInstances`, `sso:DescribeInstance`, and `identitystore:DescribeUser` was created.

On-demand chat history missing after migration

- On-demand chat histories from the public preview period are not accessible after the GA release. This is expected behavior due to enhanced security measures introduced in GA. Investigation journals and findings from public preview are not affected.

AWS EKS access setup

You can enable AWS DevOps Agent to investigate issues in your Amazon EKS clusters by running read-only `kubectl` commands against both public and private clusters. You can connect any number of EKS clusters to the same Agent Space.

Once connected, the agent can help diagnose operational issues in your clusters — describing resources, retrieving pod logs, inspecting cluster events, checking node health, and more. The agent cannot create, modify, or delete any resources in your cluster.

Prerequisites

Before setting up EKS access, ensure that your EKS cluster's authentication mode includes the EKS API. You can check this on the **Access** tab in the Amazon EKS console. If the mode doesn't include the EKS API, select a mode that does before proceeding.

Setup

These steps need to be completed from the Amazon EKS console for each cluster you wish to create an access entry for. You can find your IAM role ARN in your Agent Space (see the section called “Creating an Agent Space”) under **Capabilities > Cloud > Primary Source > Edit**.

1. Go to the **Access** tab. If the Authentication mode already says EKS API, you can add access entries. Otherwise, select a mode that includes the EKS API.
2. From the Access tab, create a new IAM access entry. Copy your primary cloud source IAM role ARN and enter it as the IAM principal for the access entry. Click **Next**.
3. Select the AWS Managed **AmazonAIOpsAssistantPolicy** access policy, and select **Cluster** for the access scope. (Alternatively, if you'd like the agent to only access certain namespaces, select the desired **Kubernetes Namespaces**). Click on **Add Policy**, and then click on **Next**.
4. Review the changes and confirm that the correct access entry policy and IAM role were chosen, and create your access entry by clicking **"Create"**.

To verify that the EKS access was configured correctly, navigate to the Operator App and start a new investigation, asking the agent a question about your cluster, such as "list all pods in the default namespace" or "show me recent events in my cluster".

Troubleshooting

If the agent can't reach your cluster, verify that the access entry is using the correct IAM role ARN shown in the setup dialog and that the **AmazonAIOpsAssistantPolicy** access policy is attached.

Connecting Azure

Azure integration enables AWS DevOps Agent to investigate resources in your Azure environment and correlate Azure DevOps pipeline deployments with operational incidents. By connecting Azure, the agent gains visibility into your Azure infrastructure and can perform root cause analysis across both AWS and Azure resources.

Azure integration consists of two independent capabilities:

- **Azure Resources** – Enables the agent to discover and investigate Azure cloud resources such as virtual machines, Azure Kubernetes Service (AKS) clusters, databases, and networking components. The agent uses Azure Resource Graph to query your resources during incident investigations.
- **Azure DevOps** – Enables the agent to access Azure DevOps repositories and pipeline execution history. The agent can correlate code changes and deployments with incidents to help identify potential root causes.

Each capability is registered at the AWS account level and can then be associated with individual Agent Spaces.

Registration methods

AWS DevOps Agent supports two methods for connecting to Azure:

- **Admin Consent** – A streamlined consent-based flow where you authorize the AWS DevOps Agent Entra application in your Azure tenant. In the console, this appears as the **Admin Consent** option. This method requires signing in with an account that has permission to perform admin consent in Microsoft Entra ID.
- **App Registration** – A self-managed approach where you create your own Entra application with federated identity credentials using Outbound Identity Federation. In the console, this appears as the **App Registration** option. This method is suitable when you need more control over the application configuration or when admin consent permissions are not available.

Both methods provide the same capabilities. You can use one or both methods within the same AWS account.

Known limitations

- **Admin Consent: one AWS account per Azure tenant** – Each Azure tenant can only have its AWS DevOps Agent Entra App associated with one AWS account at a time. To associate the same tenant with a different AWS account, you must deregister the existing registration first.
- **App Registration: unique application per registration** – Each App Registration must use a different application (client ID). You cannot register multiple configurations with the same client ID.
- **Azure DevOps: source code access** – The Azure DevOps integration provides access to pipeline execution history regardless of where the source code is hosted. However, to access the actual source code, the repository must be connected separately through a supported source provider (for example, the section called "Connecting GitHub"). Source code hosted in Bitbucket is not directly accessible through the Azure DevOps integration.

Topics

- the section called "Connecting Azure Resources"
- the section called "Connecting Azure DevOps"

Connecting Azure Resources

Azure Resources integration enables AWS DevOps Agent to discover and investigate resources in your Azure subscriptions during incident investigations. The agent uses Azure Resource Graph for resource discovery and can access metrics, logs, and configuration data across your Azure environment.

This integration follows a two-step process: register Azure at the AWS account level, then associate specific Azure subscriptions with individual Agent Spaces.

Prerequisites

Before connecting Azure Resources, ensure you have:

- Access to the AWS DevOps Agent console
- An Azure account with access to the target subscription
- For Admin Consent method: an account with permission to perform admin consent in Microsoft Entra ID

- For App Registration method: an Entra application with permissions to configure federated identity credentials, and Outbound Identity Federation enabled in your AWS account

> **Note:** You can also start registration from within an Agent Space. Navigate to **Secondary sources**, click **Add**, and select **Azure**. If Azure Cloud is not yet registered, the console guides you through registration first.

Registering Azure Resources via Admin Consent

The Admin Consent method uses a consent-based flow with the AWS DevOps Agent managed application.

Step 1: Start the registration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Go to the **Capability Providers** page
3. Locate the **Azure Cloud** section and click **Register**
4. Select the **Admin Consent** registration method

Step 2: Complete Admin Consent

1. Review the permissions being requested
2. Click to proceed — you are redirected to the Microsoft Entra admin consent page
3. Sign in with a user principal account that has permission to perform admin consent
4. Review and grant consent for the AWS DevOps Agent application

Step 3: Complete user authorization

1. After admin consent, you are prompted for user authorization to verify your identity as a member of the authorized tenant
2. Sign in with an account belonging to the same Azure tenant
3. After authorization, you are redirected back to the AWS DevOps Agent console with a success status

Step 4: Assign roles

See Assigning Azure roles below. Search for **AWS DevOps Agent** when selecting members.

Registering Azure Resources via App Registration

The App Registration method uses your own Entra application with federated identity credentials.

Step 1: Start the registration

1. In the AWS DevOps Agent console, go to the **Capability Providers** page
2. Locate the **Azure Cloud** section and click **Register**
3. Select the **App Registration** method

Step 2: Create and configure your Entra application

Follow the instructions displayed in the console to:

1. Enable Outbound Identity Federation in your AWS account (in the IAM console, go to **Account settings** → **Outbound Identity Federation**)
2. Create an Entra application in your Microsoft Entra ID, or use an existing one
3. Configure federated identity credentials on the application

Step 3: Provide registration details

Fill in the registration form with:

- **Tenant ID** – Your Azure tenant identifier
- **Tenant Name** – A display name for the tenant
- **Client ID** – The application (client) ID of the Entra application you created
- **Audience** – The audience identifier for the federated credential

Step 4: Create the IAM role

An IAM role will be automatically created when you submit the registration through the console. It permits AWS DevOps Agent to assume credentials and invoke `sts:GetWebIdentityToken`.

Step 5: Assign roles

See Assigning Azure roles below. Search for the Entra application you created when selecting members.

Step 6: Complete the registration

1. Confirm the configuration in the AWS DevOps Agent console
2. Click **Submit** to complete the registration

Assigning Azure roles

After registration, grant the application read access to your Azure subscription. This step is the same for both the Admin Consent and App Registration methods.

1. In the Azure Portal, navigate to your target subscription
2. Go to **Access Control (IAM)**
3. Click **Add** > **Add role assignment**
4. Select the **Reader** role and click **Next**
5. Click **Select members**, search for the application (either **AWS DevOps Agent** for Admin Consent, or your own Entra application for App Registration)
6. Select the application and click **Review + assign**
7. (Optional) To enable the agent to access Azure Kubernetes Service (AKS) clusters, complete the following AKS access setup.

> **Security Requirement:** The service principal must be assigned only the **Reader** role (and optionally the AKS read-only roles listed below). The Reader role serves as a security boundary that restricts the agent to read-only operations and limits the impact of indirect prompt injection attacks. Assigning roles with write or action permissions significantly increases the blast radius of prompt injection and may result in compromise of Azure resources. AWS DevOps Agent performs only read operations. The agent does not modify, create, or delete Azure resources.

AKS access setup (optional)

Step 1: Azure Resource Manager (ARM) level access

Assign **Azure Kubernetes Service Cluster User Role** to the application.

In the Azure Portal, go to **Subscriptions** → select subscription → **Access Control (IAM)** → **Add role assignment** → select **Azure Kubernetes Service Cluster User Role** → assign to the application (either **AWS DevOps Agent** for Admin Consent, or your own Entra application for App Registration).

This covers all AKS clusters in the subscription. To scope to specific clusters, assign at the resource group or individual cluster level instead.

Step 2: Kubernetes API access

Choose one option based on your cluster's authentication configuration:

Option A: Azure Role-Based Access Control (RBAC) for Kubernetes (recommended)

1. Enable Azure RBAC on the cluster if not already enabled: Azure Portal → AKS cluster → **Settings** → **Security configuration** → **Authentication and authorization** → select **Azure RBAC**
2. Assign read-only role: Azure Portal → **Subscriptions** → select subscription → **Access Control (IAM)** → **Add role assignment** → select **Azure Kubernetes Service RBAC Reader** → assign to the application

This covers all AKS clusters in the subscription.

Option B: Azure Active Directory (Azure AD) + Kubernetes RBAC

Use this if your cluster already uses the default Azure AD authentication configuration and you prefer not to enable Azure RBAC. This requires per-cluster `kubectl` setup.

1. Save the following manifest as `devops-agent-reader.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: devops-agent-reader
rules:
  - apiGroups: [""]
    resources: ["namespaces", "pods", "pods/log", "services", "events", "nodes"]
    verbs: ["get", "list"]
  - apiGroups: ["apps"]
    resources: ["deployments", "replicasets", "statefulsets", "daemonsets"]
    verbs: ["get", "list"]
```

```
  - apiGroups: ["metrics.k8s.io"]
    resources: ["pods", "nodes"]
    verbs: ["get", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: devops-agent-reader-binding
subjects:
  - kind: User
    name: "<SERVICE_PRINCIPAL_OBJECT_ID>"
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: devops-agent-reader
  apiGroup: rbac.authorization.k8s.io
```

1. Replace <SERVICE_PRINCIPAL_OBJECT_ID> with your service principal's Object ID. To find it: Azure Portal → Entra ID → Enterprise Applications → search for the application name (either **AWS DevOps Agent** for Admin Consent, or your own Entra application for App Registration).
2. Apply to each cluster:

```
az aks get-credentials --resource-group <rg> --name <cluster-name>
kubectl apply -f devops-agent-reader.yaml
```

> **Note:** Clusters using local accounts only (without Azure AD) are not supported. We recommend enabling Azure AD integration on your cluster to use this feature.

Least-privileged custom role (optional)

For tighter access control, you can create a custom Azure role scoped to only the resource providers AWS DevOps Agent uses, instead of the broad Reader role:

```
{
  "Name": "AWS DevOps Agent - Azure Reader",
  "Description": "Least-privilege read-only access for AWS DevOps Agent incident
 investigations.",
  "Actions": [
    "Microsoft.AlertsManagement/*/read",
    "Microsoft.Compute/*/read",
```

```
    "Microsoft.ContainerRegistry/*/read",
    "Microsoft.ContainerService/*/read",
    "Microsoft.ContainerService/managedClusters/commandResults/read",
    "Microsoft.DocumentDB/*/read",
    "Microsoft.Insights/*/read",
    "Microsoft.KeyVault/vaults/read",
    "Microsoft.ManagedIdentity/*/read",
    "Microsoft.Monitor/*/read",
    "Microsoft.Network/*/read",
    "Microsoft.OperationalInsights/*/read",
    "Microsoft.ResourceGraph/resources/read",
    "Microsoft.ResourceHealth/*/read",
    "Microsoft.Resources/*/read",
    "Microsoft.Sql/*/read",
    "Microsoft.Storage/*/read",
    "Microsoft.Web/*/read"
  ],
  "NotActions": [],
  "DataActions": [],
  "NotDataActions": [],
  "AssignableScopes": [
    "/subscriptions/{your-subscription-id}"
  ]
}
```

Associating a subscription with an Agent Space

After registering Azure at the account level, associate specific subscriptions with your Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Secondary sources** section, click **Add**
4. Select **Azure**
5. Provide the **Subscription ID** for the Azure subscription you want to associate
6. Click **Add** to complete the association

You can associate multiple subscriptions with the same Agent Space to give the agent visibility across your Azure environment.

Managing Azure Resources connections

- **Viewing connected subscriptions** – In the **Capabilities** tab, the **Secondary sources** section lists all connected Azure subscriptions.
- **Removing a subscription** – To disconnect a subscription from an Agent Space, select it in the **Secondary sources** list and click **Remove**. This does not affect the account-level registration.
- **Removing the registration** – To remove the Azure Cloud registration entirely, go to the **Capability Providers** page and delete the registration. All Agent Space associations must be removed first.

Connecting Azure DevOps

Azure DevOps integration enables AWS DevOps Agent to access repositories and pipeline execution history in your Azure DevOps organization. The agent can correlate code changes and deployments with operational incidents to help identify potential root causes.

> **Note:** Azure DevOps pipelines can use source code from Azure Repos, GitHub, or Bitbucket. The Azure DevOps integration provides access to pipeline execution history regardless of the source provider. However, to access the actual source code during investigations, the repository must be connected separately through a supported integration such as the section called “Connecting GitHub”. Source code in Bitbucket is not directly accessible through this integration.

This integration follows a two-step process: register Azure DevOps at the AWS account level, then associate specific projects with individual Agent Spaces.

Prerequisites

Before connecting Azure DevOps, ensure you have:

- Access to the AWS DevOps Agent console
- An Azure DevOps organization with at least one project containing a repository and pipeline history
- Permissions to add users to your Azure DevOps organization
- For Admin Consent method: an account with permission to perform admin consent in Microsoft Entra ID

- For App Registration method: an Entra application with permissions to configure federated identity credentials, and Outbound Identity Federation enabled in your AWS account

> **Note:** You can also start registration from within an Agent Space. Navigate to the **Pipelines** section, click **Add**, and select **Azure DevOps**. If Azure DevOps is not yet registered, the console guides you through registration first.

Registering Azure DevOps via Admin Consent

The Admin Consent method uses a consent-based flow with the AWS DevOps Agent managed application.

Step 1: Start the registration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Go to the **Capability Providers** page
3. Locate the **Azure DevOps** section and click **Register**
4. Enter your **Azure DevOps organization name** when prompted

Step 2: Complete Admin Consent

1. Click to proceed - you are redirected to the Microsoft Entra admin consent page
2. Sign in with a user principal account that has permission to perform admin consent
3. Review and grant consent for the AWS DevOps Agent application

Step 3: Complete user authorization

1. After admin consent, you are prompted for user authorization to verify your identity as a member of the authorized tenant
2. Sign in with an account belonging to the same Azure tenant
3. After authorization, you are redirected back to the AWS DevOps Agent console with a success status

Step 4: Grant access in Azure DevOps

See Granting access in Azure DevOps below. Search for **AWS DevOps Agent** when adding users.

Registering Azure DevOps via App Registration

App Registration is shared between Azure Resources and Azure DevOps. If you have already completed App Registration for Azure Resources, you can skip to Granting access in Azure DevOps.

Step 1: Start the ADO App Registration

1. In the AWS DevOps Agent console, go to the **Capability Providers** page
2. Locate the **Azure Cloud** section and click **Register**
3. Select the **App Registration** method

Step 2: Create and configure your Entra application

Follow the instructions displayed in the console to:

1. Enable Outbound Identity Federation in your AWS account (in the IAM console, go to **Account settings** → **Outbound Identity Federation**)
2. Create an Entra application in your Microsoft Entra ID, or use an existing one
3. Configure federated identity credentials on the application

Step 3: Provide registration details

Fill in the registration form with:

- **Tenant ID** – Your Azure tenant identifier
- **Tenant Name** – A display name for the tenant
- **Client ID** – The application (client) ID of the Entra application
- **Audience** – The audience identifier for the federated credential

Step 4: Create the IAM role

An IAM role will be automatically created when you submit the registration through the console. It permits AWS DevOps Agent to assume credentials and invoke `sts:GetWebIdentityToken`.

Step 5: Complete the registration

1. Confirm the configuration in the AWS DevOps Agent console

2. Click **Submit** to complete the registration

Step 6: Grant access in Azure DevOps

See Granting access in Azure DevOps below. Search for the Entra application you created during App Registration when adding users.

Granting access in Azure DevOps

After registration, grant the application access to your Azure DevOps organization. This step is the same for both the Admin Consent and App Registration methods.

1. In Azure DevOps, go to **Organization Settings** > **Users** > **Add Users**
2. Search for the application (either **AWS DevOps Agent** for Admin Consent, or your own Entra application for App Registration)
3. Set the access level to **Basic**
4. Under **Add to projects**, select the projects you want the agent to access
5. Under **Azure DevOps Groups**, select **Project Readers**
6. Click **Add** to complete

> **Security Requirement:** Assign only the **Project Readers** group. Read-only access serves as a security boundary that restricts the agent to read-only operations and limits the impact of indirect prompt injection attacks. Assigning groups with write or action permissions significantly increases the blast radius of prompt injection and may result in compromise of Azure DevOps resources.

Associating a project with an Agent Space

After registering Azure DevOps at the account level, associate specific projects with your Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Pipelines** section, click **Add**
4. Select **Azure DevOps** from the list of available providers
5. Select the project from the dropdown of available projects

6. Click **Add** to complete the association

Managing Azure DevOps connections

- **Viewing connected projects** – In the **Capabilities** tab, the **Pipelines** section lists all connected Azure DevOps projects.
- **Removing a project** – To disconnect a project from an Agent Space, select it in the **Pipelines** section and click **Remove**.
- **Removing the registration** – To remove the Azure DevOps registration entirely, go to the **Capability Providers** page and delete the registration. All Agent Space associations must be removed first.

Connecting to CI/CD pipelines

CI/CD pipeline integration enables AWS DevOps Agent to monitor deployments and correlate code changes with operational incidents during investigations. By connecting your CI/CD providers, the agent can track deployment events and associate them with AWS resources to help identify potential root causes during incident response.

AWS DevOps Agent supports integration with popular CI/CD platforms through a two-step process:

1. **Account-level registration** – Register your CI/CD provider once at the AWS account level
2. **Agent Space connection** – Connect specific projects or repositories to individual Agent Spaces based on your organizational needs

This approach allows you to share CI/CD provider registrations across multiple Agent Spaces while maintaining granular control over which projects are monitored by each space.

Supported CI/CD providers

AWS DevOps Agent supports the following CI/CD platforms:

- **GitHub** – Connect repositories from GitHub.com using the AWS DevOps Agent GitHub app.
- **GitLab** – Connect projects from GitLab.com, managed GitLab instances, or publicly accessible self-hosted GitLab deployments.

Topics

Connecting GitHub

GitHub integration enables AWS DevOps Agent to access code repositories and receive deployment events during incident investigations. This integration follows a two-step process: account-level registration of GitHub, followed by connecting specific repositories to individual Agent Spaces.

AWS DevOps Agent supports both GitHub.com (SaaS) and GitHub Enterprise Server (self-hosted) instances.

Prerequisites

Before connecting GitHub, ensure you have:

- Access to the AWS DevOps Agent admin console
- A GitHub user account or organization with admin permissions
- Authorization to install GitHub apps in your account or organization

For GitHub Enterprise Server, you also need:

- A GitHub Enterprise Server instance (version 3.x or later) accessible over HTTPS
- The HTTPS URL of your GitHub Enterprise Server instance (for example, `https://github.example.com`)
- (Optional) A private connection, if your GitHub Enterprise Server instance is not publicly accessible

Registering GitHub (account-level)

GitHub is registered at the AWS account level and shared among all Agent Spaces in that account. You only need to register GitHub once per AWS account.

Step 1: Navigate to pipeline providers

1. Sign in to the AWS Management Console

2. Navigate to the AWS DevOps Agent console
3. Go to the **Capabilities** tab
4. In the **Pipeline** section, click **Add**
5. Select **GitHub** from the list of available providers

If GitHub hasn't been registered yet, you'll be prompted to register it first.

Step 2: Choose connection type

On the "Register GitHub Account / Organization" screen, select whether you're connecting as a user or organization:

- **User** – Your personal GitHub account with a username and profile
- **Organization** – A shared GitHub account where multiple people can collaborate across many projects at once

If you are connecting to a GitHub Enterprise Server instance, check the **Use GitHub Enterprise Server** checkbox and enter the HTTPS URL of your instance (for example, `https://github.example.com`).

If your GitHub Enterprise Server instance is not publicly accessible, you can optionally configure a private connection to allow AWS DevOps Agent to securely reach your instance. For more information, see the section called “Connecting to privately hosted tools”.

> **Note**
>
> Do not include `/api/v3` or any trailing path in the URL — enter only the base URL.

Step 3: Set up the GitHub App

Click **Submit** to begin the app setup process. The next steps differ depending on whether you are connecting to GitHub.com or GitHub Enterprise Server.

For GitHub.com

1. You'll be redirected to GitHub to install the AWS DevOps Agent GitHub app.
2. Select which account or organization to install the app in.

3. The app allows AWS DevOps Agent to receive events from connected repositories, including deployment events.

For GitHub Enterprise Server

GitHub Enterprise Server uses a GitHub App Manifest flow, which automatically sets up a new GitHub App on your instance. This involves two redirects to your GitHub Enterprise Server instance.

1. Your browser will be redirected to your GitHub Enterprise Server instance's "Create GitHub App" page.
2. You'll see the app name pre-filled. Feel free to change the name as needed. Click **Create GitHub App**.
3. You'll be redirected back to AWS DevOps Agent, which exchanges the manifest code for app credentials.

Step 4: Select repositories and complete installation

1. You'll see the **Install & Authorize** page for the GitHub App.
2. Select which repositories to allow the app to access:
   - **All repositories** – Grant access to all current and future repositories
   - **Only select repositories** – Choose specific repositories from your account or organization
3. Click **Install & Authorize**.
4. You'll be redirected back to the AWS DevOps Agent console, where GitHub will appear as registered at the account level.

Connecting repositories to an Agent Space

After registering GitHub at the account level, you can connect specific repositories to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Pipeline** section, click **Add**
4. Select **GitHub** from the list of available providers
5. Select the subset of repositories relevant to this Agent Space

6. Click **Add** to complete the connection

You can connect different sets of repositories to different Agent Spaces based on your organizational needs.

Understanding the GitHub app

The AWS DevOps Agent GitHub app:

- Requests read-only access to your repositories
- Receives deployment events and other repository events
- Allows AWS DevOps Agent to correlate code changes with operational incidents
- Can be uninstalled at any time through your GitHub settings

For GitHub Enterprise Server, the GitHub App is automatically created on your instance during registration. You can manage the app's repository access or uninstall it through **Settings > Applications > Installed GitHub Apps**. To delete the app definition entirely, go to **Settings > Developer settings > GitHub Apps**.

Managing GitHub connections

- **Updating repository access** – To change which repositories the GitHub app can access, go to your GitHub account or organization settings (or your GitHub Enterprise Server instance settings), navigate to installed GitHub apps, and modify the AWS DevOps Agent app configuration.
- **Viewing connected repositories** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected repositories in the Pipeline section.
- **Removing GitHub connection** – To disconnect GitHub from an Agent Space, select the connection in the Pipeline section and click **Remove**. To uninstall the GitHub app completely, uninstall it from your GitHub account or organization settings. For GitHub Enterprise Server, because the GitHub App is created directly on your instance during registration, you can optionally clean up the app entirely by performing both of the following:
  - **Uninstall the app** – Go to **Settings > Applications > Installed GitHub Apps**, click **Configure** on the app, then uninstall it.
  - **Delete the app** – Go to **Settings > Developer settings > GitHub Apps**, select the app, go to the **Advanced** tab, and choose **Delete GitHub App**. **Warning:** Deleting the GitHub App

is permanent and cannot be undone. If you delete it, you will need to re-register GitHub Enterprise Server from the beginning in the AWS DevOps Agent console to create a new app.

Connecting GitLab

GitLab integration enables AWS DevOps Agent to monitor deployments from GitLab Pipelines to inform causal investigations during incident response. This integration follows a two-step process: account-level registration of GitLab, followed by connecting specific projects to individual Agent Spaces.

Registering GitLab (account-level)

GitLab is registered at the AWS account level and shared among all Agent Spaces in that account. Individual Agent Spaces can then choose which specific projects apply to their Agent Space.

Step 1: Navigate to pipeline providers

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capability Providers** page (accessible from the side navigation)
4. Find **GitLab** in the **Available** providers section under **Pipeline** and click **Register**

Step 2: Configure GitLab connection

On the GitLab registration page, configure the following:

Connection type – Select whether you're connecting as a person or a group:

- **Personal** (default) – Your individual GitLab user account with a username and profile
- **Group** – In GitLab, you use groups to manage one or more related projects at the same time

GitLab instance type – Choose which type of GitLab instance you're connecting to:

- **GitLab.com** (default) – The public GitLab service
- **Publicly accessible self-hosted GitLab** – Check the **Use GitLab self hosted endpoint** box and provide the URL to your GitLab instance

> **Note**
>
> Currently, only publicly accessible GitLab instances are supported.

Access token – Provide a GitLab personal access token:

1. In a separate browser tab, log in to your GitLab account
2. Navigate to your user settings and select **Access Tokens**
3. Create a new personal access token with the following permissions:
   - `read_repository` – Required to access repository content
   - `read_virtual_registry` – Required to access virtual registry information
   - `read_registry` – Required to access registry information
   - `api` – Required for read and write API access
   - `self_rotate` - Required for rotating tokens. This feature is currently unsupported by AWS DevOps Agent but will be supported at a later date. Adding now prevents the need to create a new token in the future.
4. Set the token expiration to a maximum of 365 days from the current date
5. Copy the generated token
6. Return to the AWS DevOps Agent console
7. Paste the token into the "Access Token" field

Step 3: Complete registration

(Optional) Tags – Add AWS tags to the GitLab registration for organizational purposes.

Click **Next** to review your configuration, then click **Submit** to complete the GitLab registration process. The system will validate your access token and establish the connection.

Connecting projects to an Agent Space

After registering GitLab at the account level, you can connect specific projects to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab

3. In the **Pipeline** section, click **Add**
4. Select **GitLab** from the list of available providers
5. Select the GitLab projects relevant to your Agent Space
6. Click **Save**

AWS DevOps Agent will monitor these projects for deployments from GitLab Pipelines to inform causal investigations.

Managing GitLab connections

- **Updating access token** – If your access token expires or needs to be updated, you can update it in the AWS DevOps Agent console by modifying the GitLab registration at the account level.
- **Viewing connected projects** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected projects in the Pipeline section.
- **Removing GitLab connection** – To disconnect GitLab projects from an Agent Space, select the connection in the Pipeline section and click **Remove**. To remove the GitLab registration completely, remove it from all Agent Spaces first, then delete the registration at the account level.

Connecting MCP Servers

Model Context Protocol (MCP) servers extend AWS DevOps Agent's investigation capabilities by providing access to data from your external observability tools, custom monitoring systems, and operational data sources. This guide explains how to connect an MCP server to AWS DevOps Agent.

Requirements

Before connecting an MCP server, ensure your server meets these requirements:

- **Streamable HTTP transport protocol** – Only MCP servers that implement the Streamable HTTP transport protocol are supported.
- **Authentication support** – Your MCP server must support OAuth 2.0 authentication flows or API key/token-based authentication.

Security considerations

When connecting MCP servers to AWS DevOps Agent, consider these security aspects:

- **Tool allowlisting –** You should allowlist only the specific tools your Agent Space needs, rather than exposing all tools from your MCP server. See Configuring MCP tools in an Agent Space for how to allow list tools per Agent Space.

Please note that the maximum tool length of any MCP tool is 64.

- **Prompt injection risks** – Custom MCP servers can introduce additional risk of prompt injection attacks. See Prompt injection protection: AWS DevOps Agent Security for more information.
- **Read-only tools and access –** Only allowlist read-only MCP tools and ensure that authentication credentials are only permitted read-only access.

See *AWS DevOps Agent Security* for more information on prompt injection and the shared responsibility model.

> **Note**
>
> If your MCP server is on a private network, see the section called "Connecting to privately hosted tools"

Registering an MCP server (account-level)

MCP servers are registered at the AWS account level and shared among all Agent Spaces in that account. Individual Agent Spaces can then choose which specific tools they need from each MCP server.

Step 1: MCP server details

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capability Providers** page (accessible from the side navigation)
4. Find **MCP Server** in the **Available** providers section and click **Register**

5. On the **MCP server details** page, enter the following information:
   - **Name** – Enter a descriptive name for your MCP server
   - **Endpoint URL** – Enter the full HTTPS URL of your MCP server endpoint
   - **Description** (optional) – Add a description to help identify the server's purpose
   - **Enable Dynamic Client Registration** – Select this checkbox if you want to allow AWS DevOps Agent to automatically register with your MCP server's authorization server
6. Click **Next**

> **Note**
>
> The MCP server endpoint URL will be displayed in AWS CloudTrail logs in your account.

Step 2: Authorization flow

Select the authentication method for your MCP server:

OAuth Client Credentials – If your MCP server uses OAuth Client Credentials flow:

1. Select **OAuth Client Credentials**
2. Click **Next**

OAuth 3LO (Three-Legged OAuth) – If your MCP server uses OAuth 3LO for authentication:

1. Select **OAuth 3LO**
2. Click **Next**

API Key – If your MCP server uses API key authentication:

1. Select **API Key**
2. Click **Next**

Step 3: Authorization configuration

Configure additional authorization parameters based on the selected authentication method:

For OAuth Client Credentials:

1. **Client ID** – Enter the client ID of the OAuth client
2. **Client Secret** – Enter the client secret of the OAuth client
3. **Exchange URL** – Enter the OAuth token exchange endpoint URL
4. **Exchange Parameters** – Enter OAuth token exchange parameters for authenticating with the service
5. **Add Scope** – Add OAuth scopes for authentication
6. Click **Next**

For OAuth 3LO:

1. **Client ID** – Enter the client ID of the OAuth client
2. **Client Secret** – Enter the client secret of the OAuth client if it's required by your OAuth client
3. **Exchange URL** – Enter the OAuth token exchange endpoint URL
4. **Authorization URL** - Enter the OAuth authorization endpoint URL
5. **Code Challenge Support** - Select this checkbox if your OAuth client supports code challenge
6. **Add Scope** – Add OAuth scopes for authentication
7. Click **Next**

For API Key:

1. Enter an API key name
2. Enter the the name of the header that will contain the API key in the request
3. Enter your API key value
4. Click **Next**

Step 4: Review and submit

1. Review all the MCP server configuration details
2. Click **Submit** to complete the registration
3. AWS DevOps Agent will validate the connection to your MCP server
4. Upon successful validation, your MCP server will be registered at the account level

Configuring MCP tools in an Agent Space

After registering an MCP server at the account level, you can configure which tools from that server are available to specific Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **MCP Servers** section, click **Add**
4. Select the registered MCP server you want to connect to this Agent Space
5. Configure which tools from this MCP server should be available to the Agent Space:
   - **Allow all tools** – Makes all tools from the MCP server available
   - **Select specific tools** – Allows you to choose which tools to allowlist
6. Click **Add** to connect the MCP server to your Agent Space

AWS DevOps Agent will now be able to use the allowlisted tools from your MCP server during investigations in this Agent Space.

Managing MCP server connections

Updating authentication credentials – If your authentication credentials need to be updated, you will need to re-register your MCP server. Navigate to the **Capability Providers** page in the AWS DevOps Agent console, locate your MCP server, remove any active associations, and click **Deregister**. Next, **register** your MCP server with the new authentication credentials and re-create any necessary associations with your Agent Space.

Viewing connected MCP servers – To see all MCP servers connected to your Agent Space, select your Agent Space, go to the **Capabilities** tab, and check the **MCP Servers** section. You can also update selected tools here.

Removing MCP server connections – To disconnect an MCP server from an Agent Space, select the server in the **MCP Servers** section and click **Remove**. To completely delete an MCP server registration, remove it from all Agent Spaces first, then delete the account-level registration.

Related topics

- Security in AWS DevOps Agent

- Setting up an Agent Space
- Prompt Injection Protection

Connecting multiple AWS Accounts

Secondary AWS accounts allow AWS DevOps Agent to investigate resources across multiple AWS accounts in your organization. When your applications span multiple accounts, adding secondary accounts ensures the agent has visibility into all relevant resources during incident investigations. Greater access to the accounts and resources composing an application ensures greater investigation accuracy.

Prerequisites

Before adding a secondary AWS account, ensure you have:

- Access to the AWS DevOps Agent console in the primary account
- Administrative access to the secondary AWS account
- IAM permissions to create roles in the secondary account

Adding a secondary AWS account

In addition to the steps below, you can use the the section called “AWS DevOps Agent CLI onboarding guide” to programmatically add secondary accounts.

Step 1: Start the secondary account configuration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Select your Agent Space
3. Go to the **Capabilities** tab
4. In the **Cloud** section, locate the **Secondary sources** subsection
5. Click **Add**

Step 2: Specify the role name

1. In the **Name your role** field, enter a name for the role you'll create in the secondary account

2. Note this name—you'll use it again when creating the role in the secondary account
3. Copy the trust policy provided in the console and save it in a scratch space

Step 3: Create the role in the secondary account

1. Open a new browser tab and sign in to the IAM console in the secondary AWS account
2. Navigate to **IAM >Roles** > **Create role**
3. Select **Custom trust policy**
4. Paste the trust policy you copied from Step 2
5. Click **Next**

Step 4: Attach the AWS managed policy

1. In the **Permissions policies** section, search for **AIOpsAssistantPolicy**
2. Select the checkbox next to the **AIOpsAssistantPolicy** managed policy
3. Click **Next**

Step 5: Name and create the role

1. In the **Role name** field, enter the same role name you provided in Step 2
2. (Optional) Add a description to help identify the role's purpose
3. Review the trust policy and attached permissions
4. Click **Create role**

Step 6: Attach the inline policy

1. In the IAM console, locate and select the role you just created
2. Go to the **Permissions** tab
3. Click **Add permissions** > **Create inline policy**
4. Switch to the **JSON** tab
5. Paste the policy you saved in Step 2
6. Paste the policy into the JSON editor in the IAM console

7. Click **Next**
8. Provide a name for the inline policy (for example, "DevOpsAgentInlinePolicy")
9. Click **Create policy**

Step 7: Complete the configuration

1. Return to the AWS DevOps Agent console in the primary account
2. Click **Next** to complete the secondary account configuration
3. Verify the connection status shows as **Active**

Understanding the required policies

AWS DevOps Agent requires three policy components to access resources in a secondary account:

- **Trust policy** – Allows AWS DevOps Agent in the primary account to assume the role in the secondary account. This establishes the trust relationship between accounts.
- **AIOpsAssistantPolicy (AWS managed policy)** – Provides the core read-only permissions AWS DevOps Agent needs to investigate resources in the secondary account. This policy is maintained by AWS and updated as new capabilities are added.
- **Inline policy** – Provides additional permissions specific to your Agent Space configuration. This policy is generated based on your Agent Space settings and may include permissions for specific integrations or features.

In the primary account, the AWS DevOps Agent IAM Role must be able to assume the role created in the secondary account.

Managing secondary accounts

- **Viewing connected accounts** – In the **Capabilities** tab, the **Secondary sources** subsection lists all connected secondary accounts with their connection status.
- **Updating the IAM role** – If you need to modify permissions, update the inline policy attached to the role in the secondary account. Changes take effect immediately.
- **Removing a secondary account** – To disconnect a secondary account, select it in the **Secondary sources** list and click **Remove**. This does not delete the IAM role in the secondary account.

Connecting telemetry sources

AWS DevOps Agent provides three ways to connect to your telemetry sources.

Built-in, 2-way integration

Currently, AWS DevOps Agent supports Dynatrace users with a built-in, 2-way integration enabling the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it via a AWS DevOps Agent-hosted Dynatrace MCP server.
- **Automated Investigation triggering** - Dynatrace Workflows can be configured to trigger incident resolution Investigations from Dynatrace Problems.
- **Telemetry introspection** - AWS DevOps Agent can introspect Dynatrace telemetry as it investigates an issue via the AWS DevOps Agent-hosted Dynatrace MCP server.
- **Status updates** - AWS DevOps Agent will publish key investigation findings, root cause analyses, and generated mitigation plans to the Dynatrace user interface.

To learn about 2-way integrations, see

- the section called “Connecting Dynatrace”

Built-in, 1-way integration

Currently, AWS DevOps Agent supports AWS CloudWatch, Datadog, Grafana, New Relic, and Splunk users with built-in, 1 way integrations.

Security best practice: When configuring credentials for built-in 1-way integrations, we recommend scoping API keys and tokens to read-only access. AWS DevOps Agent uses these credentials for telemetry introspection only and does not require write access to your telemetry provider.

The AWS CloudWatch built-in, 1-way integration requires no additional setup and enables the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it via your configured primary and secondary AWS cloud accounts.
- **Telemetry introspection** - AWS DevOps Agent can introspect AWS CloudWatch telemetry as it investigates an issue via the IAM role(s) provided during primary and secondary AWS cloud account configuration.

The Datadog, Grafana, New Relic, and Splunk built-in, 1 way integrations require setup and enable the following:

- **Automated Investigation triggering** - Datadog, Grafana, New Relic, and Splunk events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect Datadog, Grafana, New Relic, and Splunk telemetry as it investigates an issue via each provider's remote MCP server.

To learn about 1-way integrations, see the following:

- the section called “Connecting DataDog”
- the section called “Connecting Grafana”
- the section called “Connecting New Relic”
- the section called “Connecting Splunk”

Bring-your-own telemetry sources

For any other telemetry source, including Prometheus metrics, you can leverage AWS DevOps Agent’s support for both webhook and MCP server integration.

To learn about bring-your-own integrations, see the following

- the section called “Invoking DevOps Agent through Webhook”
- the section called “Connecting MCP Servers”

Connecting Dynatrace

Built-in, 2-way integration

Currently, AWS DevOps Agent supports Dynatrace users with a built-in, 2-way integration enabling the following:

- **Topology resource mapping** - AWS DevOps Agent will augment your DevOps Agent Space Topology with entities and relationships available to it from your Dynatrace environment.
- **Automated Investigation triggering** - Dynatrace Workflows can be configured to trigger incident resolution Investigations from Dynatrace Problems.
- **Telemetry introspection** - AWS DevOps Agent can introspect Dynatrace telemetry as it investigates an issue via the AWS DevOps Agent-hosted Dynatrace MCP server.
- **Status updates** - AWS DevOps Agent will publish key investigation findings, root cause analyses, and generated mitigation plans to the Dynatrace user interface.

Onboarding

Onboarding Process

Onboarding your Dynatrace observability system involves three stages:

1. **Connect** - Establish connection to Dynatrace by configuring account access credentials, with all the environments you may need
2. **Enable** - Activate Dynatrace in specific Agent spaces with specific Dynatrace environments
3. **Configure your Dynatrace environment** - download the workflows and dashboard and import into Dynatrace, making a note of the webhook details to trigger investigations in designated Agent spaces

Step 1: Connect

Establish connection to your Dynatrace environment

Configuration

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Find **Dynatrace** in the **Available** providers section under **Telemetry** and click **Register**
3. **Create OAuth client in Dynatrace, with the detailed permissions.**

a. See Dynatrace documentation
b. When ready press next
c. You can connect multiple Dynatrace environments and later scope to specific ones for each DevOps Agent Space you may have.

4. Enter your Dynatrace details from the OAuth client setup:
   - **Client Name**
   - **Client ID**
   - **Client Secret**
   - **Account URN**
5. Click Next
6. Review and add

Step 2: Enable

Activate Dynatrace in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Locate the Telemetry section, Press Add
4. You will notice Dynatrace with 'Registered' status. Click on add to add this to your agent space
5. Dynatrace Environment ID - Provide the Dynatrace environment ID you would like to associate with this DevOps agent space.
6. Enter one or more Dynatrace Entity IDs - these help DevOps agent discover your most important resources, examples might be services or applications. **If you are unsure you can press remove.**
7. Review and press Save
8. Copy the Webhook URL and Webhook Secret. See Dynatrace documentation to add these credentials to Dynatrace.

Step 3: Configure your Dynatrace environment

To complete your Dynatrace set up you will need to perform certain setup steps in your Dynatrace environment. Follow the instructions in the Dynatrace documentation.

Supported Event Schemas

AWS DevOps Agent supports two types of events from Dynatrace using webhooks. The supported event schemas are documented below:

Incident Event

Incident events are used to trigger an investigation. The event schema is:

```
{
    "event.id": string;
    "event.status": "ACTIVE" | "CLOSED";
    "event.status_transition": string;
    "event.description": string;
    "event.name": string;
    "event.category": "AVAILABILITY" | "ERROR" | "SLOWDOWN" | "RESOURCE_CONTENTION" |
 "CUSTOM_ALERT" | "MONITORING_UNAVAILABLE" | "INFO";
    "event.start"?: string;
    "affected_entity_ids"?: string[];
}
```

Mitigation Event

Mitigation events are used to trigger generating a mitigation report for the investigation on next steps. The event schema is:

```
{
    "task_id": string;
    "task_version": number;
    "event.type": "mitigation_request";
}
```

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agent spaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details

2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Dynatrace
5. Press remove

Step 2: Deregister from account

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Dynatrace

Connecting DataDog

Built-in, 1 way integration

Currently, AWS DevOps Agent supports Datadog users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering** - Datadog events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect Datadog telemetry as it investigates an issue via each provider's remote MCP server.

Onboarding

Step 1: Connect

Establish connection to your Datadog remote MCP endpoint with account access credentials

Configuration

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Find **Datadog** in the **Available** providers section under **Telemetry** and click **Register**
3. Enter your Datadog MCP server details:

- **Server Name** - Unique identifier (e.g., my-datadog-server)
- **Endpoint URL** - Your Datadog MCP server endpoint. The endpoint URL varies depending on your Datadog site. See the Datadog site endpoint table below.
- **Description** - Optional server description

4. Click Next
5. Review and submit

Datadog site endpoints

The MCP endpoint URL varies depending on your Datadog site. To identify your site, check the URL in your browser when logged into Datadog, or see Access the Datadog site.

Datadog Site	Site Domain	MCP Endpoint URL
US1 (default)	`datadoghq.com`	`https://mcp.datadoghq.com/api/unstable/mcp-server/mcp`
US3	`us3.datadoghq.com`	`https://mcp.us3.datadoghq.com/api/unstable/mcp-server/mcp`
US5	`us5.datadoghq.com`	`https://mcp.us5.datadoghq.com/api/unstable/mcp-server/mcp`
EU1	`datadoghq.eu`	`https://mcp.datadoghq.eu/api/unstable/mcp-server/mcp`
AP1	`ap1.datadoghq.com`	`https://mcp.ap1.datadoghq.com/api/unstable/mcp-server/mcp`

Datadog Site	Site Domain	MCP Endpoint URL
AP2	`ap2.datadoghq.com`	`https://mcp.ap2.datadoghq.com/api/unstable/mcp-server/mcp`

Authorization

Complete OAuth authorization by:

- Authorizing as your user on the Datadog OAuth page
- If not logged in, click Allow, login, then authorize

Once configured, Datadog becomes available across all Agent spaces.

Step 2: Enable

Activate DataDog in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select Datadog
6. Next
7. Review and press Save
8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure Datadog to send events to trigger an investigation, for example from an alarm.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
}
```

Send webhooks with Datadog https://docs.datadoghq.com/integrations/webhooks/ (note select no authorization and instead use the custom header option).

Learn more: Datadog Remote MCP Server

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agent spaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details

2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Datadog
5. Press remove

Step 2: Deregister from account

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Datadog

Connecting Grafana

Grafana integration enables AWS DevOps Agent to query metrics, dashboards, and alerting data from your Grafana instance during incident investigations. This integration follows a two-step process: account-level registration of Grafana, followed by connecting it to individual Agent Spaces.

To improve security, the Grafana integration only enables read-only tools. Write tools are disabled and cannot be enabled. This means the agent can query and read data from your Grafana instance but cannot create, modify, or delete any Grafana resources such as dashboards, alerts, or annotations. For more information, see Security in AWS DevOps Agent.

Grafana requirements

Before connecting Grafana, ensure you have:

- Grafana version 9.0 or later. Some features, particularly datasource-related operations, may not work correctly with earlier versions due to missing API endpoints.
- A Grafana instance accessible over HTTPS. Both public and private network endpoints are supported. With private network connectivity, your Grafana instance can be hosted inside a VPC with no public internet access. For details, see the section called “Connecting to privately hosted tools”.
- A Grafana service account with an access token that has appropriate read permissions

Registering Grafana (account-level)

Grafana is registered at the AWS account level and shared among all Agent Spaces in that account.

Step 1: Configure Grafana

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capability Providers** page (accessible from the side navigation)
4. Find **Grafana** in the **Available** providers section under **Telemetry** and click **Register**
5. On the **Configure Grafana** page, enter the following information:
   - **Service Name** (required) – Enter a descriptive name for your Grafana server using alphanumeric characters, hyphens, and underscores only. For example, `my-grafana-server`.
   - **Grafana URL** (required) – Enter the full HTTPS URL of your Grafana instance. For example, `https://myinstance.grafana.net`.
   - **Service Account Access Token** (required) – Enter a Grafana service account access token. Tokens typically start with `glsa_`. To create a service account token, navigate to your Grafana instance, go to **Administration > Service accounts**, create a service account with Viewer role, and generate a token.
   - **Description** (optional) – Add a description to help identify the server's purpose. For example, `Production Grafana server for monitoring`.
6. (Optional) Add AWS tags to the registration for organizational purposes.
7. Click **Next**

Step 2: Review and submit Grafana registration

1. Review all the Grafana configuration details
2. Click **Submit** to complete the registration
3. Upon successful registration, Grafana appears in the **Currently registered** section of the Capability Providers page

Adding Grafana to an Agent Space

After registering Grafana at the account level, you can connect it to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Telemetry** section, click **Add**
4. Select **Grafana** from the list of available providers
5. Click **Save**

Configuring Grafana alert webhooks

You can configure Grafana to automatically trigger AWS DevOps Agent investigations when alerts fire by sending webhooks through Grafana contact points. For details on webhook authentication methods and credential management, see the section called “Invoking DevOps Agent through Webhook”.

Step 1: Create a custom notification template

In your Grafana instance, navigate to **Alerting > Contact points > Notification templates** and create a new template with the following content:

```
{{ define "devops-agent-payload" }}
{
  "eventType": "incident",
  "incidentId": "{{ (index .Alerts 0).Labels.alertname }}-{{ (index .Alerts
 0).Fingerprint }}",
  "action": "{{ if eq .Status "resolved" }}resolved{{ else }}created{{ end }}",
  "priority": "{{ if eq .Status "resolved" }}MEDIUM{{ else }}HIGH{{ end }}",
  "title": "{{ (index .Alerts 0).Labels.alertname }}",
  "description": "{{ (index .Alerts 0).Annotations.summary }}",
  "service": "{{ if (index .Alerts 0).Labels.job }}{{ (index .Alerts 0).Labels.job }}
{{ else }}grafana{{ end }}",
  "timestamp": "{{ (index .Alerts 0).StartsAt }}",
  "data": {
    "metadata": {
      {{ range $k, $v := (index .Alerts 0).Labels }}
      "{{ $k }}": "{{ $v }}",
      {{ end }}
      "_source": "grafana"
    }
  }
}
{{ end }}
```

This template formats Grafana alerts into the webhook payload structure expected by AWS DevOps Agent. It maps alert labels, annotations, and status into the appropriate fields, and includes all alert labels as metadata.

> **Note:** This template processes only the first alert in a group. Grafana groups multiple firing alerts into a single notification by default. To ensure each alert is sent individually, configure your notification policies to group by `alertname`. Additionally, this template does not escape special JSON characters in label values or annotations. Ensure that alert labels and the `summary` annotation do not contain characters such as double quotes or newlines, which would produce invalid JSON.

Step 2: Create a webhook contact point

1. In Grafana, navigate to **Alerting > Contact points** and click **Add contact point**
2. Select **Webhook** as the integration type
3. Set the **URL** to your AWS DevOps Agent webhook endpoint
4. Under **Optional Webhook settings**, configure the authentication headers based on your webhook type. See Webhook authentication methods for details.
5. Set the **Message** field to use your custom template: `{{ template "devops-agent-payload" . }}`
6. Click **Save contact point**

Step 3: Assign the contact point to a notification policy

1. Navigate to **Alerting > Notification policies**
2. Edit an existing policy or create a new one
3. Set the contact point to the webhook contact point you created
4. Click **Save policy**

When a matching alert fires, Grafana will send the formatted payload to AWS DevOps Agent, which will start an investigation automatically.

Limitations

- **ClickHouse data source tools** – ClickHouse data source tools are not currently supported.

- **Proactive incident prevention** – the section called “Proactive incident prevention” does not currently use Grafana tools. Support is planned for a future release.

Amazon Managed Grafana considerations

If you are using Amazon Managed Grafana (AMG), be aware of the following limitations:

- **Webhook contact points are not supported** – AMG does not currently support webhook contact points in its alerting configuration. You cannot use AMG to send alert webhooks directly to AWS DevOps Agent. For details, see Alerting contact points in Amazon Managed Grafana.
- **Service account token expiration** – AMG service account tokens have a maximum expiration of 30 days. You will need to rotate tokens and update your Grafana registration in AWS DevOps Agent before they expire. See Managing Grafana connections for how to update credentials. For details on AMG token limits, see Service accounts in Amazon Managed Grafana.

Managing Grafana connections

- **Updating credentials** – If your service account token expires or needs to be updated, deregister Grafana from the Capability Providers page and re-register with the new token.
- **Viewing connected instances** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected telemetry sources.
- **Removing Grafana** – To disconnect Grafana from an Agent Space, select it in the Telemetry section and click **Remove**. To completely remove the registration, remove it from all Agent Spaces first, then deregister from the Capability Providers page.

Connecting New Relic

Built-in, 1 way integration

Currently, AWS DevOps Agent supports New Relic users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering** - New Relic events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect New Relic telemetry as it investigates an issue via each provider's remote MCP server.

Onboarding

Step 1: Connect

Establish connection to your New Relic remote MCP endpoint with account access credentials

Please use a Full Platform User (not Basic/Core) in New relic to enable New Relic MCP tools.

Configuration

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Find **New Relic** in the **Available** providers section under **Telemetry** and click **Register**
3. Follow the instructions to obtain your New Relic API Key
4. Enter your New Relic MCP server API Key details:
   - **Account ID:** Enter your New Relic account ID obtained above
   - **API Key:** Enter the API Key obtained above
   - **Select US or EU region** based on where your New Relic account is.
5. Click Add

Step 2: Enable

Activate New Relic in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select New Relic
6. Next
7. Review and press Save
8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure New Relic to send events to trigger an investigation, for example from an alarm. For more details on setting up webhooks, see Change tracking webhooks.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
}
```

Send webhooks with New Relic https://newrelic.com/instant-observability/webhook-notifications. You can either select Bearer token for the authorization type, or select no authorization and add the `Authorization: Bearer <Token>` as a custom header instead.

Learn more: https://docs.newrelic.com/docs/agentic-ai/mcp/overview/

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove from all agent spaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select New Relic
5. Press remove

Step 2: Deregister from account

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to New Relic

Connecting Splunk

Built-in, 1 way integration

Currently, AWS DevOps Agent supports Splunk users with built-in, 1 way integration, enabling the following:

- **Automated Investigation triggering** - Splunk events can be configured to trigger AWS DevOps Agent incident resolution Investigations via AWS DevOps Agent webhooks.
- **Telemetry introspection** - AWS DevOps Agent can introspect Splunk telemetry as it investigates an issue via each provider's remote MCP server.

Prerequisites

Getting a Splunk API token

You will need an MCP URL and token to connect Splunk.

Splunk Administrator steps

Your Splunk Administrator needs to perform the following steps:

- enable REST API access
- enable token authentication on the deployment.
- create a new role 'mcp_user', the new role does not need to have any capabilities.
- assign the role 'mcp_user' to any users on the deployment who are authorized to use the MCP server.
- create the token for the authorized users with audience as 'mcp' and set the appropriate expiration, if the user does not have the permission to create tokens themselves.

Splunk User steps

A Splunk user needs to perform the following steps:

- Get an appropriate token from the Splunk Administrator or create one themselves, if they have the permission. The audience for the token must be 'mcp'.

Onboarding

Step 1: Connect

Establish connection to your Splunk remote MCP endpoint with account access credentials

Configuration

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Find **Splunk** in the **Available** providers section under **Telemetry** and click **Register**
3. Enter your Splunk MCP server details:
   - **Server Name** - Unique identifier (e.g., my-splunk-server)
   - **Endpoint URL** - Your Splunk MCP server endpoint:

`https://<YOUR_SPLUNK_DEPLOYMENT_NAME>.api.scs.splunk.com/<YOUR_SPLUNK_DEPLOYMENT_NAME>/mcp/v1/`

- **Description** - Optional server description
- **Token Name** - The name of the bearer token for authentication: `my-splunk-token`
- **Token Value** The bearer token value for authentication

Step 2: Enable

Activate Splunk in a specific Agent space and configure appropriate scoping

Configuration

1. From the agent spaces page, select an agent space and press view details (if you have not yet created an agent space see the section called “Creating an Agent Space”)
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Press Add
5. Select Splunk
6. Next
7. Review and press Save
8. Copy the Webhook URL and API Key

Step 3: Configure webhooks

Using the Webhook URL and API Key you can configure Splunk to send events to trigger an investigation, for example from an alarm.

To ensure that events sent can be used by the DevOps Agent, make sure that the data transmitted to the webhook matches the data schema specified below. Events that do not match this schema may be ignored by DevOps Agent.

Set the method and the headers

```
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": "Bearer <Token>",
```

```
    },
```

Send the body as a JSON string.

```
{
    eventType: 'incident';
    incidentId: string;
    action: 'created' | 'updated' | 'closed' | 'resolved';
    priority: "CRITICAL" | "HIGH" | "MEDIUM" | "LOW" | "MINIMAL";
    title: string;
    description?: string;
    timestamp?: string;
    service?: string;
    // The original event generated by service is attached here.
    data?: object;
}
```

Send webhooks with Splunk https://help.splunk.com/en/splunk-enterprise/alert-and-respond/alerting-manual/9.4/configure-alert-actions/use-a-webhook-alert-action (note select no authorization and instead use the custom header option)

Learn more:

- Splunk's MCP Server Documentation: https://help.splunk.com/en/splunk-cloud-platform/mcp-server-for-splunk-platform/about-mcp-server-for-splunk-platform
- Access requirements and limitations for the Splunk Cloud Platform REST API: https://docs.splunk.com/Documentation/SplunkCloud/latest/RESTTUT/RESTandCloud
- Manage authentication tokens in Splunk Cloud Platform: https://help.splunk.com/en/splunk-cloud-platform/administer/manage-users-and-security/9.3.2411/authenticate-into-the-splunk-platform-with-tokens/manage-or-delete-authentication-tokens
- Create and manage roles with Splunk Web: https://docs.splunk.com/Documentation/SplunkCloud/latest/Security/Addandeditroles

Removal

The telemetry source is connected at two levels at the agent space level and at account level. To completely remove it you must first remove it from all agent spaces where it is used and then it can be unregistered.

Step 1: Remove from agent space

1. From the agent spaces page, select an agent space and press view details
2. Select the Capabilities tab
3. Scroll down to the Telemetry section
4. Select Splunk
5. Press remove

Step 2: Deregister from account

1. Go to the **Capability Providers** page (accessible from the side navigation)
2. Scroll to the **Currently registered** section.
3. Check the agent space count is zero (if not repeat Step 1 above in your other agent spaces)
4. Press Deregister next to Splunk

Connecting to ticketing and chat

AWS DevOps Agent is designed to act as a member of your team by participating in your team's existing communication channels. You can connect DevOps Agent to your ticketing and alarming systems, like ServiceNow and PagerDuty, to automatically launch investigations from incident tickets, accelerating incident response within your existing workflows to reduce meant time to recover (MTTR). You can also connect your DevOps Agent to your team collaboration systems like Slack to receive activity summaries from your DevOps Agent in a chat channel.

To learn about connecting ticketing and chat integrations, see the following:

- the section called "Connecting PagerDuty"
- the section called "Connecting ServiceNow"
- the section called "Connecting Slack"

Connecting PagerDuty

PagerDuty integration enables AWS DevOps Agent to access and update incident data, on-call schedules, and service information from your PagerDuty account during incident investigations and automated response. This integration uses OAuth 2.0 for secure authentication.

> **Important**
>
> AWS DevOps Agent only supports the newer PagerDuty OAuth 2.0 (Scoped OAuth). Legacy PagerDuty OAuth with redirect uri is not supported.

PagerDuty requirements

Before connecting PagerDuty, ensure you have:

- A PagerDuty account with your OAuth client ID and client secret
- Your PagerDuty account subdomain (for example, if your PagerDuty URL is `https://your-company.pagerduty.com`, the subdomain is `your-company`)

Registering PagerDuty

PagerDuty is registered at the AWS account level and shared among all Agent Spaces in that account.

Step 1: Configure access in PagerDuty

1. Sign in to the AWS Management Console
2. Navigate to the AWS DevOps Agent console
3. Go to the **Capability Providers** page (accessible from the side navigation)
4. Find **PagerDuty** in the **Available** providers section under **Communication** and click **Register**
5. Follow the guided setup on the **Configure access in PagerDuty** page:

Check your service region and subdomain:

- **Account scope** – Select your PagerDuty region (**US** or **EU**) and enter your PagerDuty subdomain. For example, if your PagerDuty URL is `https://your-company.pagerduty.com`, enter `your-company`.

Create a new app in PagerDuty:

- In a separate browser tab, log in to PagerDuty and navigate to **Integrations > App Registration**
- Create a new app using **OAuth 2.0 Scoped OAuth**

- Under **Permissions**, grant the following minimum required scopes: `incidents.read`, `incidents.write`, and `services.read`
- Enable **Events Integration** to allow bi-directional communication between AWS DevOps Agent and PagerDuty

Configure OAuth credentials:

- **Permission scope** – The minimum required scopes are: `incidents.read`, `incidents.write`, `services.read`
- **Client name** – Enter a descriptive name for your OAuth client
- **Client ID** – Enter the OAuth client ID from your PagerDuty app registration
- **Client secret** – Enter the OAuth client secret from your PagerDuty app registration

Step 2: Review and submit PagerDuty registration

1. Review all the PagerDuty configuration details
2. Click **Submit** to complete the registration
3. Upon successful registration, PagerDuty appears in the **Currently registered** section of the Capability Providers page

Adding PagerDuty to an Agent Space

After registering PagerDuty at the account level, you can connect it to individual Agent Spaces:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Capabilities** tab
3. In the **Communications** section, click **Add**
4. Select **PagerDuty** from the list of available providers
5. Click **Save**

Managing PagerDuty connections

- **Updating credentials** – If your OAuth credentials need to be updated, deregister PagerDuty from the Capability Providers page and re-register with the new credentials.

- **Viewing connections** – In the AWS DevOps Agent console, select your Agent Space and go to the Capabilities tab to view connected communication integrations.
- **Removing PagerDuty** – To disconnect PagerDuty from an Agent Space, select it in the Communications section and click **Remove**. To completely remove the registration, remove it from all Agent Spaces first, then deregister from the Capability Providers page.

Webhook support

AWS DevOps Agent only supports PagerDuty V3 webhooks. Earlier webhook versions are not supported.

For more information about PagerDuty V3 webhook subscriptions, see Webhooks Overview in the PagerDuty developer documentation.

Connecting ServiceNow

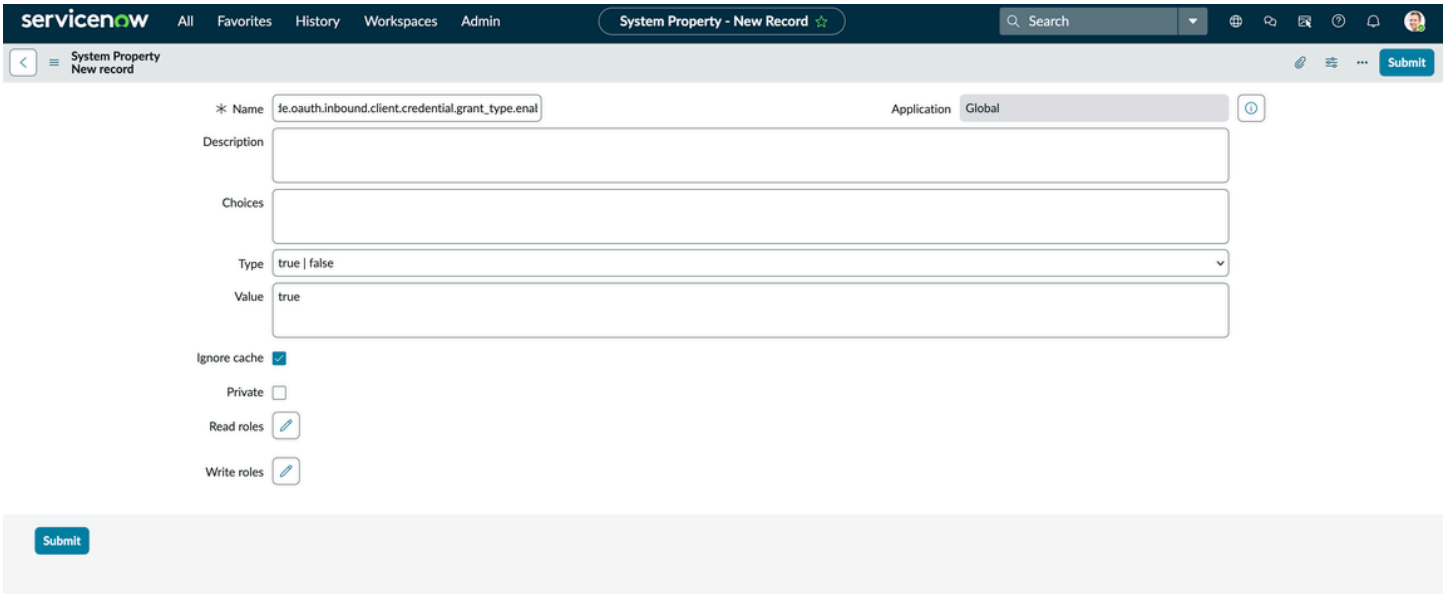
This tutorial walks you through connecting a ServiceNow instance to AWS DevOps Agent to enable it to automatically initiate incident response investigations when a ticket is created and post its key findings into the originating ticket. It also contains examples for how to configure your ServiceNow instance to send only specific tickets to a DevOps Agent Space and how to orchestrate ticket routing across multiple DevOps Agent Spaces.

Initial Setup

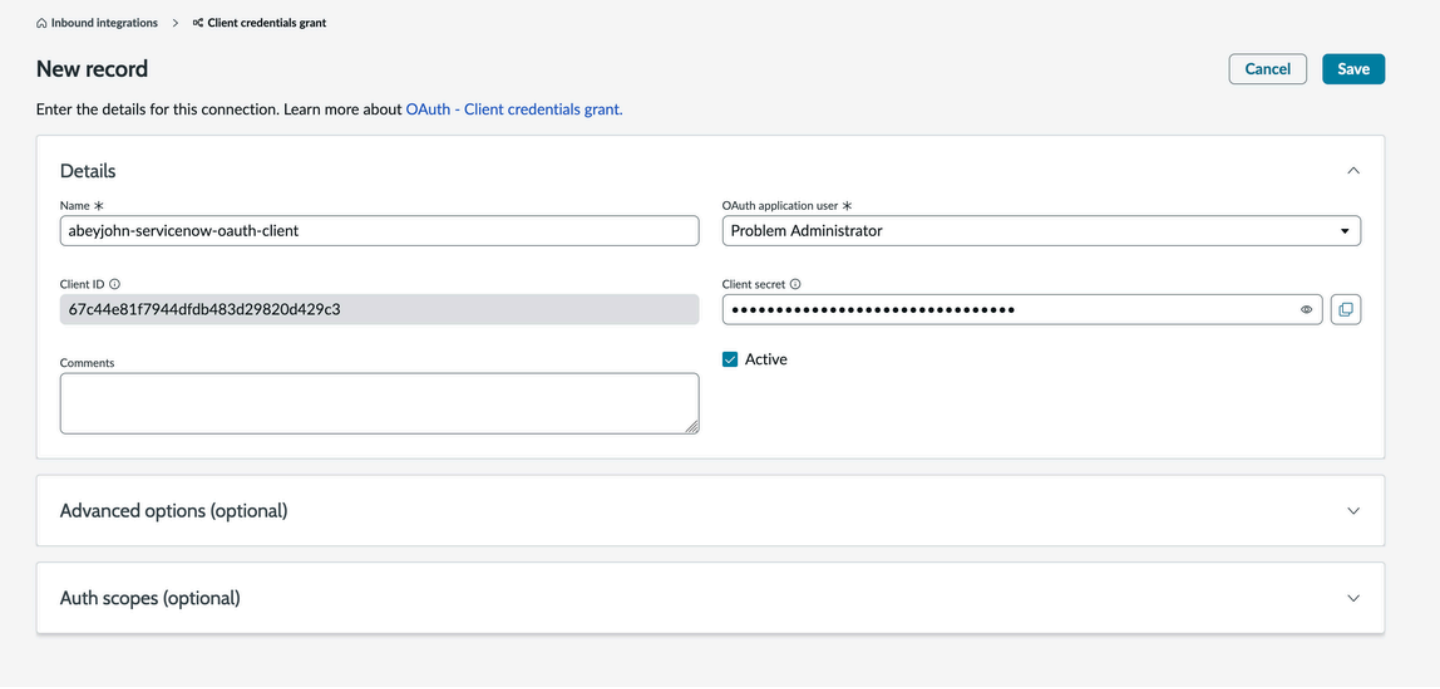
The first step is to create in ServiceNow an OAuth application client that AWS DevOps can use to access your ServiceNow instance.

Create a ServiceNow OAuth application client

1. Enable your instance's client credential system property
   a. Search `sys_properties.list` in the filter search box and then hit enter (it will not show the option but hitting enter works)
   b. Choose New
   c. Add the name as `glide.oauth.inbound.client.credential.grant_type.enabled` and the value to true with type as true | false



1. Navigate to System OAuth > Application Registry from the filter search box
2. Choose “New” > “New Inbound Integration Experience” > “New Integration” > “OAuth - Client Credentials Grant”
3. Pick a name and set the OAuth application user to “Problem Administrator”, click “Save“

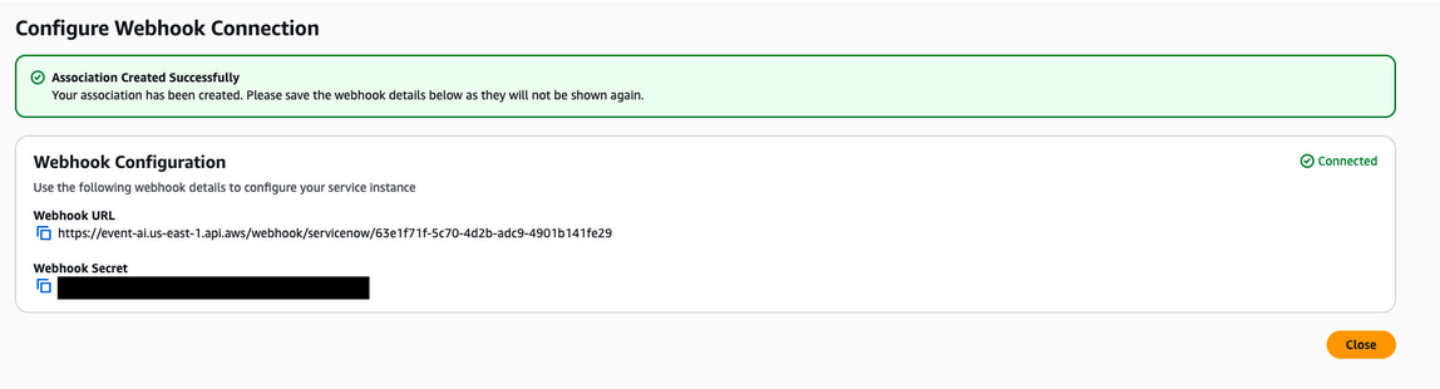


Connect your ServiceNow OAuth client to AWS DevOps Agent

1. You can start this process in two places. First, by navigating to the **Capability Providers** page and finding **ServiceNow** under **Communication**, then clicking **Register**. Alternatively you can select any DevOps Agent Space you may have created and navigate to Capabilities → Communications → Add → ServiceNow and click Register.
2. Next, authorize DevOps Agent to access your ServiceNow instance using the OAuth application client you just created.

- Follow the next steps, and save the resulting information about the webhook

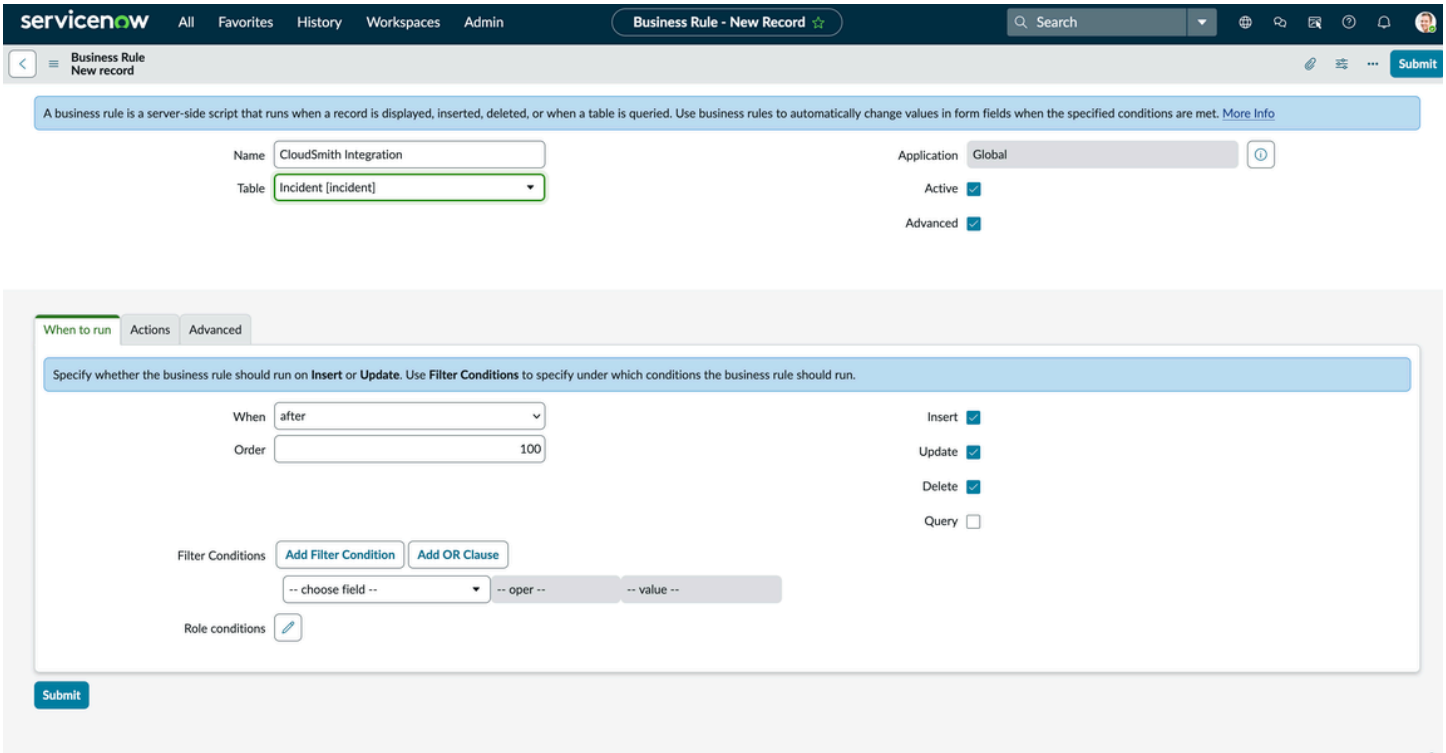
> **Important**
>
> You will not see this information again



Configure your ServiceNow Business Rule

Once you have established connectivity, you'll need to configure a business rule in ServiceNow to send tickets to your DevOps Agent Space(s).

1. Navigate to Activity Subscriptions → Administration → Business Rules, and click New.
2. Set the "Table" field to "Incident [incident]", check the "Advanced" box, and set the rule to run after Insert, Update, and Delete.



1. Navigate to the "Advanced" tab and add the following webhook script, inserting your webhook secret and URL where indicated, and click Submit.

```
(function executeRule(current, previous /*null when async*/ ) {

    var WEBHOOK_CONFIG = {
        webhookSecret: GlideStringUtil.base64Encode('<<< INSERT WEBHOOK SECRET HERE
 >>>'),
        webhookUrl: '<<< INSERT WEBHOOK URL HERE >>>'
    };

    function generateHMACSignature(payloadString, secret) {
```

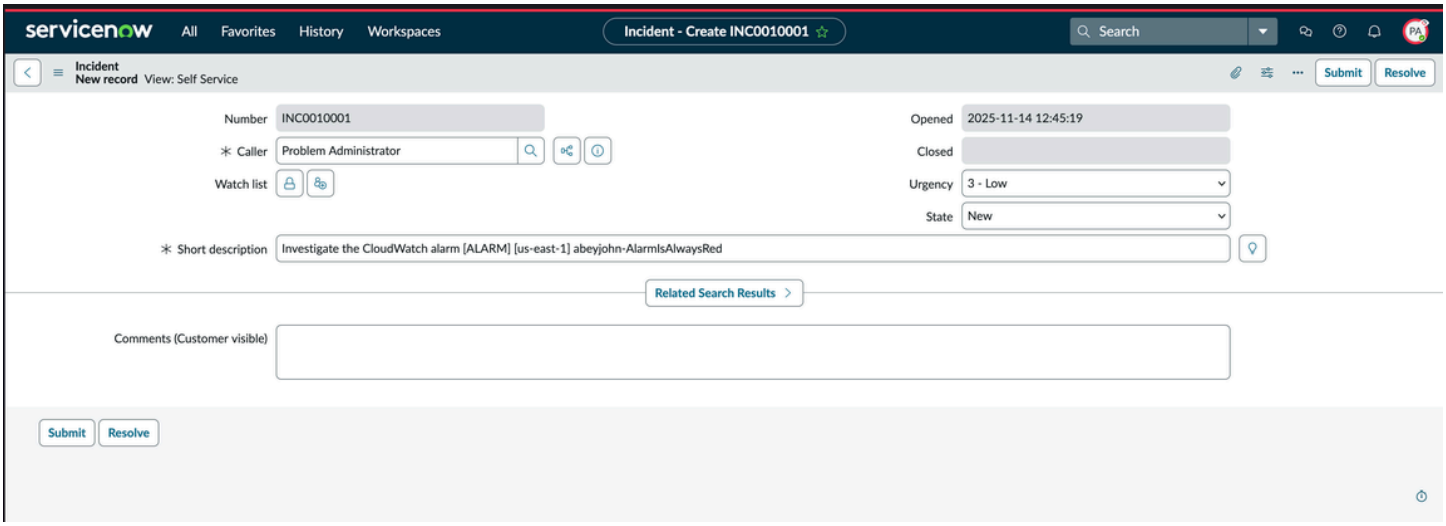
```
        try {
            var mac = new GlideCertificateEncryption();
            var signature = mac.generateMac(secret, "HmacSHA256", payloadString);
            return signature;
        } catch (e) {
            gs.error('HMAC generation failed: ' + e);
            return null;
        }
    }

    function callWebhook(payload, config) {
        try {
            var timestamp = new Date().toISOString();
            var payloadString = JSON.stringify(payload);
            var payloadWithTimestamp =`${timestamp}:${payloadString}`;

            var signature = generateHMACSignature(payloadWithTimestamp,
config.webhookSecret);

            if (!signature) {
                gs.error('Failed to generate signature');
                return false;
            }

            gs.info('Generated signature: ' + signature);

            var request = new sn_ws.RESTMessageV2();
            request.setEndpoint(config.webhookUrl);
            request.setHttpMethod('POST');

            request.setRequestHeader('Content-Type', 'application/json');
            request.setRequestHeader('x-amzn-event-signature', signature);
            request.setRequestHeader('x-amzn-event-timestamp', timestamp);

            request.setRequestBody(payloadString);

            var response = request.execute();
            var httpStatus = response.getStatusCode();
            var responseBody = response.getBody();

            if (httpStatus >= 200 && httpStatus < 300) {
                gs.info('Webhook sent successfully. Status: ' + httpStatus);
                return true;
            } else {
```

```
                gs.error('Webhook failed. Status: ' + httpStatus + ', Response: ' +
responseBody);
                return false;
            }

        } catch (ex) {
            gs.error('Error sending webhook: ' + ex.getMessage());
            return false;
        }
    }

    function createReference(field) {
        if (!field || field.nil()) {
            return null;
        }

        return {
            link: field.getLink(true),
            value: field.toString()
        };
    }

    function getStringValue(field) {
        if (!field || field.nil()) {
            return null;
        }
        return field.toString();
    }

    function getIntValue(field) {
        if (!field || field.nil()) {
            return null;
        }
        var val = parseInt(field.toString());
        return isNaN(val) ? null : val;
    }

    var eventType = (current.operation() == 'insert') ? "create" : "update";

    var incidentEvent = {
        eventType: eventType.toString(),
        sysId: current.sys_id.toString(),
        priority: getStringValue(current.priority),
        impact: getStringValue(current.impact),
```

```
        active: getStringValue(current.active),
        urgency: getStringValue(current.urgency),
        description: getStringValue(current.description),
        shortDescription: getStringValue(current.short_description),
        parent: getStringValue(current.parent),
        incidentState: getStringValue(current.incident_state),
        severity: getStringValue(current.severity),
        problem: createReference(current.problem),
        additionalContext: {}
    };

    incidentEvent.additionalContext = {
        number: current.number.toString(),
        opened_at: getStringValue(current.opened_at),
        opened_by: current.opened_by.nil() ? null :
 current.opened_by.getDisplayValue(),
        assigned_to: current.assigned_to.nil() ? null :
 current.assigned_to.getDisplayValue(),
        category: getStringValue(current.category),
        subcategory: getStringValue(current.subcategory),
        knowledge: getStringValue(current.knowledge),
        made_sla: getStringValue(current.made_sla),
        major_incident: getStringValue(current.major_incident)
    };

    for (var key in incidentEvent.additionalContext) {
        if (incidentEvent.additionalContext[key] === null) {
            delete incidentEvent.additionalContext[key];
        }
    }

    gs.info(JSON.stringify(incidentEvent, null, 2)); // Pretty print for logging only

    if (WEBHOOK_CONFIG.webhookUrl && WEBHOOK_CONFIG.webhookSecret) {
        callWebhook(incidentEvent, WEBHOOK_CONFIG);
    } else {
        gs.info('Webhook not configured.');
    }

})(current, previous);
```

If you chose to register your ServiceNow connection from the **Capability Providers** page, you now need to navigate to the DevOps Agent Space you want to investigate ServiceNow incident tickets,

select Capabilities → Communications and then register the ServiceNow instance you registered on the Capability Providers page. Now, everything should be set up, and all incidents where the caller is set to “Problem Administrator” (to mimic the permissions you gave the AWS DevOps OAuth client) will trigger a incident response investigation in the configured DevOps Agent Space. You can test this by creating a new incident in ServiceNow and setting the Caller field of the incident as “Problem Administrator.”



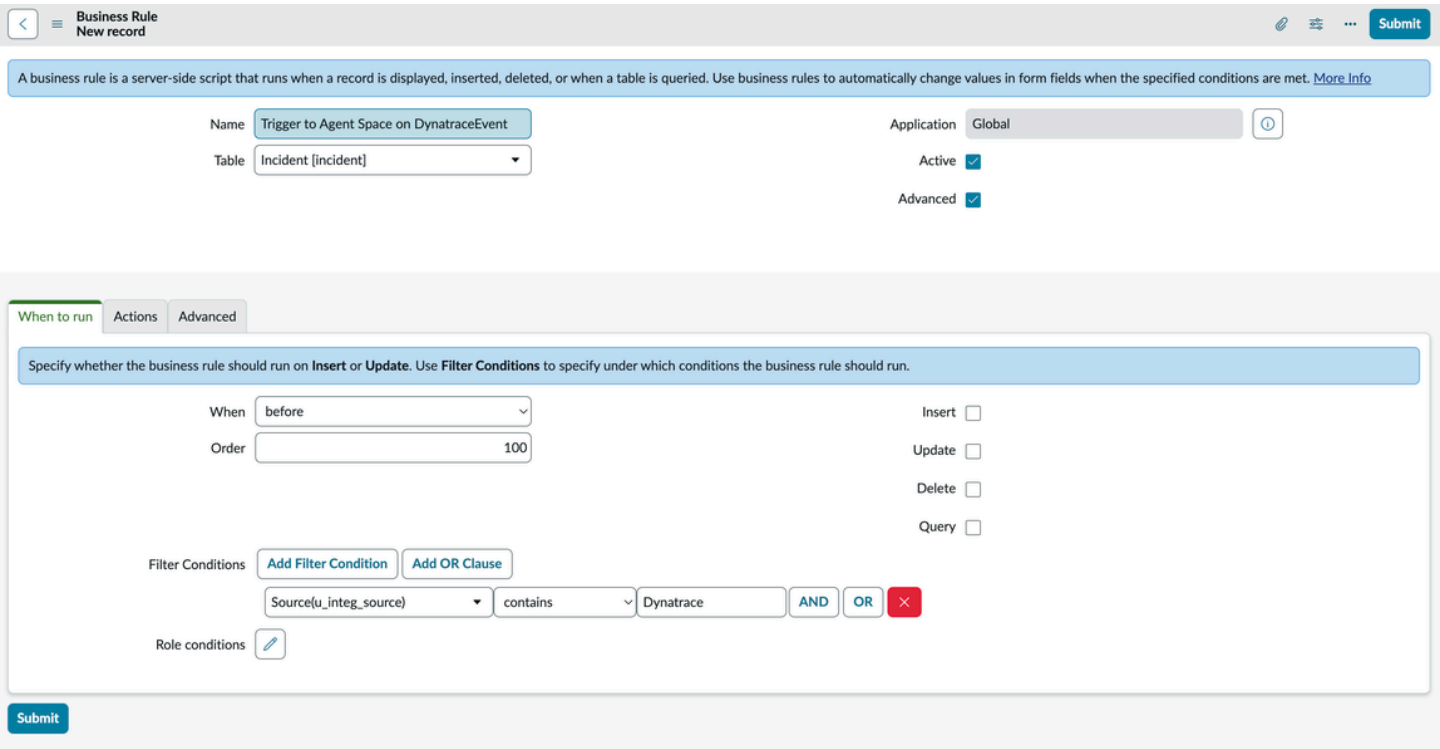
ServiceNow ticket updates

During all triggered incident response Investigations, your DevOps Agent will provide updates of its key findings, root cause analyses, and mitigation plans into the originating ticket. The agent findings are posted to the comments of an incident, and we'll currently only post agent records of type `finding`, `cause`, `investigation_summary`, `mitigation_summary` , and investigation status updates (e.g `AWS DevOps Agent started/finished its investigation`).

Ticket routing and orchestration examples

Scenario: Filtering which incidents are sent to a DevOps Agent Space

This is a simple scenario but needs some configuration in ServiceNow to create a field in ServiceNow to track incident source. For the purpose of this example, create a new Source (u_source) field using the SNOW form builder. This will enable tracking the incident source and use it to route requests from a particular source to a DevOps Agent Space. Routing is accomplished by creating a Service Now Business Rule and in the When to run tab setting “When” triggers and “Filter Conditions.” In this example the filter conditions are set as follows:

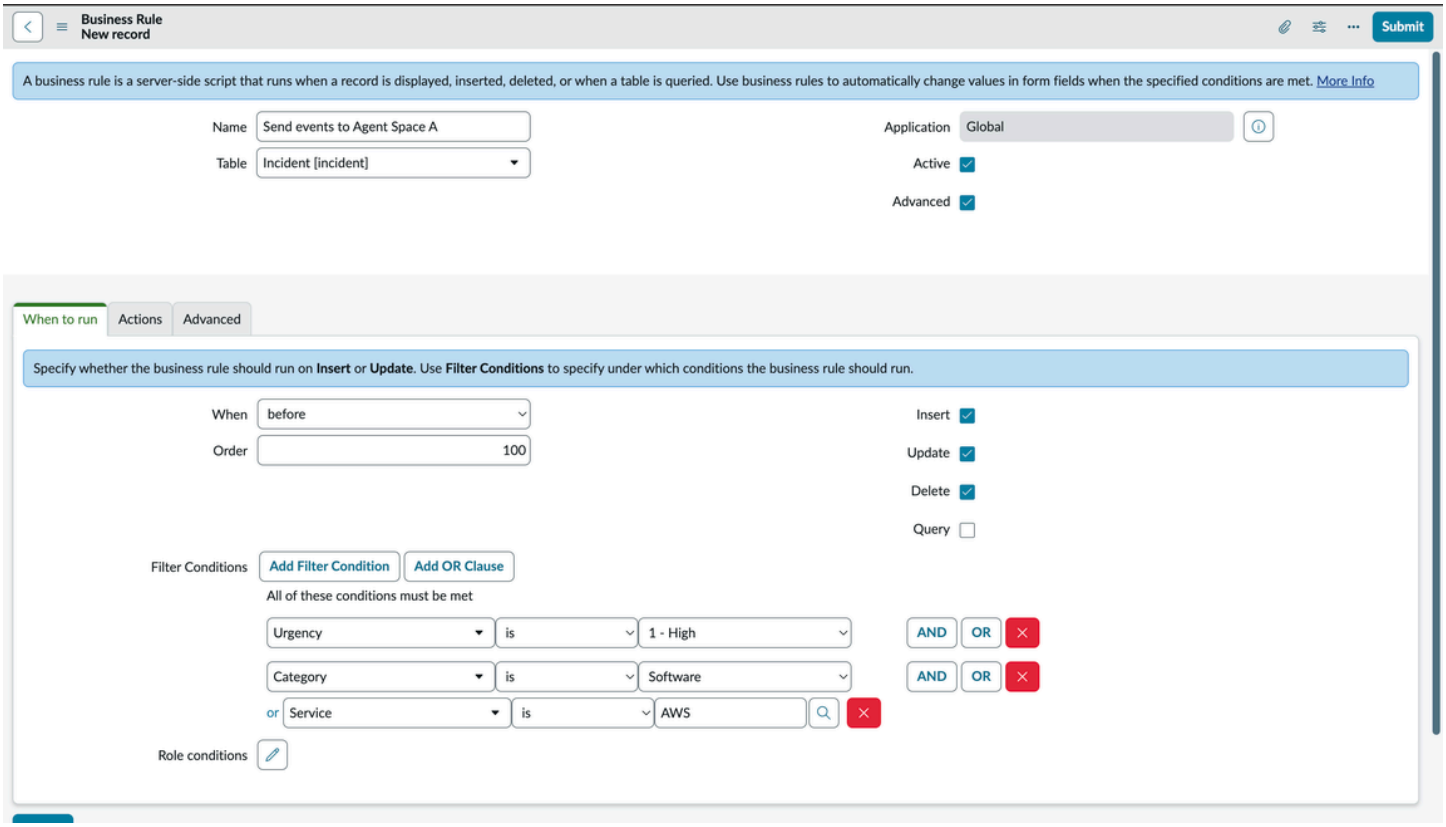


Scenario: Routing incidents across multiple DevOps Agent Spaces

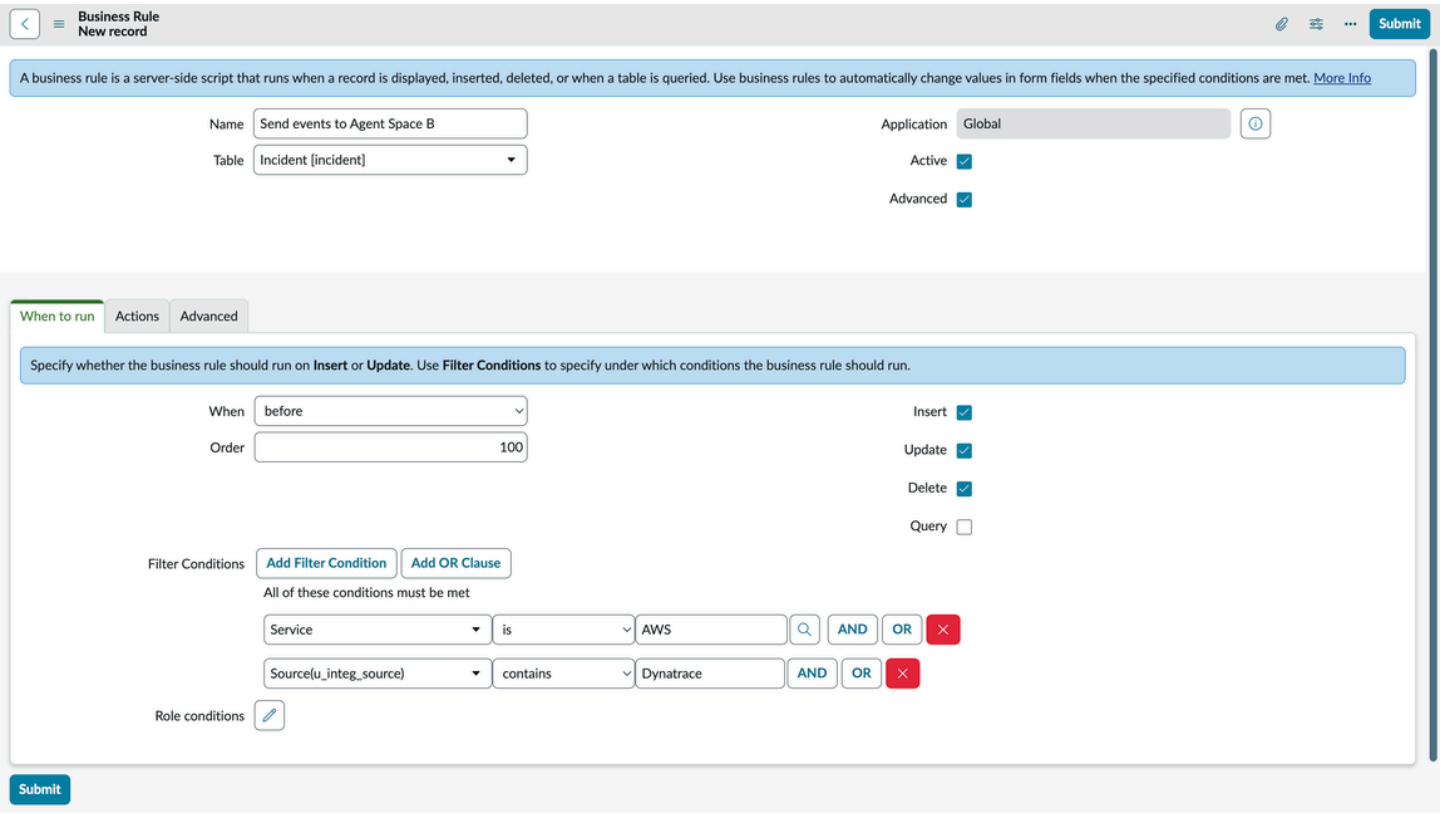
This example shows how to trigger an Investigation in DevOps Agent Space B when the urgency is 1, category is `Software` , or Service is `AWS`, and trigger an Investigation in DevOps Agent Space A when the service is `AWS`, and source is `Dynatrace`.

This scenario can be accomplished in two ways. The webhook script itself can be updated to include this business logic. In this scenario we will show how to accomplish it with a ServiceNow Business Rule, for transparency and simplify debugging. Routing is accomplished by creating two Service Now Business Rules.

- Create a Business Rule in ServiceNow for DevOps Agent Space A and create a condition using the condition builder to only send the events based on our specified condition.



- Next, create another Business Rule in ServiceNow for AgentSpace B for which the business rule will only trigger when Service is AWS and source is Dynatrace.



Now, when you create a new Incident that matches the condition specified, it will either trigger an investigation on DevOps Agent Space A or DevOps Agent Space B, providing you with fine grained control over incident routing.

Connecting Slack

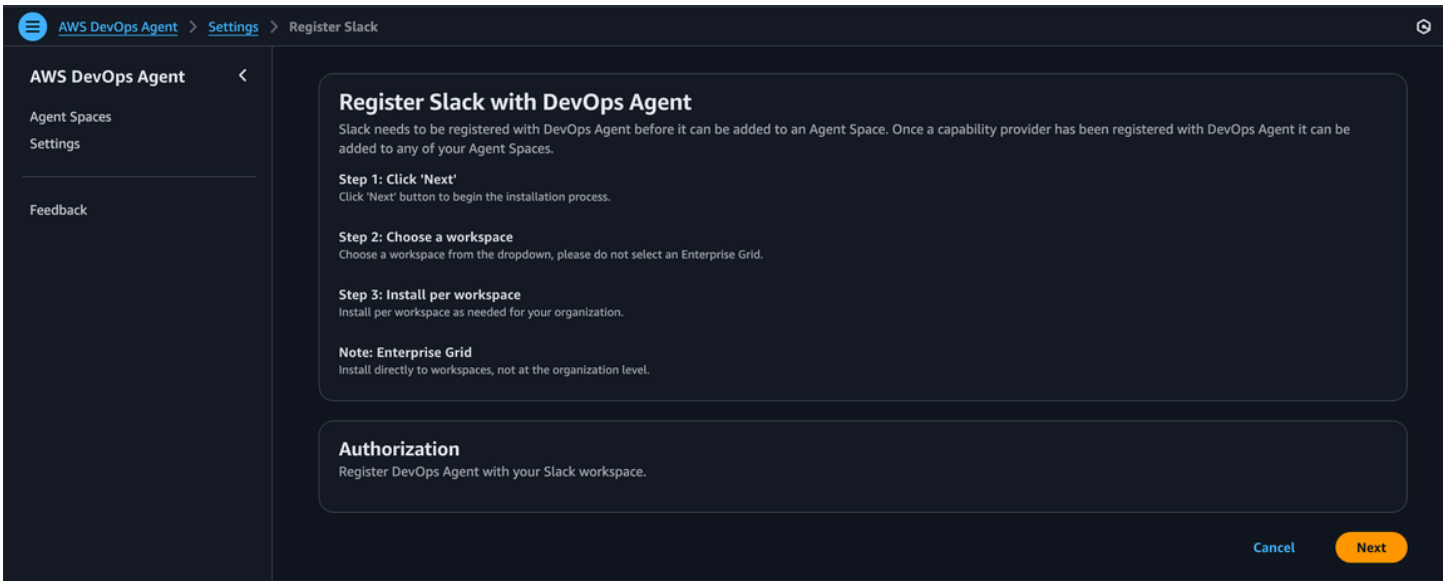
You can configure AWS DevOps Agent to update a Slack channel you select with incident response investigation key findings, root cause analyses, and generated mitigation plans.

Before you begin

Slack needs to be registered with DevOps Agent before it can be added to an Agent Space. To integrate AWS DevOps Agent with Slack you must meet these requirements:

- Have access to a Slack workspace with the ability to install and authorize third-party applications
- Have identified the Slack channels where you want AWS DevOps Agent to send notifications

Register Slack integration with AWS DevOps Agent



1. From the **Capability Providers** page in the AWS DevOps Agent console, find **Slack** in the **Available** providers section under **Communication** and click **Register**.
2. Choose the **Register** button.
3. You will be redirected to Slack to authorize the AWS DevOps Agent application for your workspace.
4. On the Slack authorization page, install directly to workspaces, not at the organization level.
5. Choose a workspace from the dropdown. Do not select an Enterprise Grid.
6. Install per workspace as needed for your organization.
7. Review the requested scopes and click **Allow** to authorize the integration.
8. After authorization, you'll return to the AWS DevOps Agent console.

Associate Slack with your DevOps Agent Space(s)

After registering Slack in your DevOps Agent Space, you can associate it with your DevOps Agent Space(s):

1. From the **Capabilities** tab within your configured AgentSpace, navigate to **Communications** > **Slack**.
2. Select **Add Slack**
3. Enter the Channel ID

4. Choose **Create** to complete the Slack configuration.

> **Note**
>
> The agent's bot user must be added to private channels before it can post messages.

> **Important**
>
> Uninstalling the Slack app may result in the Slack app not being able to be reinstalled. Please avoid uninstalling the Slack app.

Invoking DevOps Agent through Webhook

Webhooks allow external systems to automatically trigger AWS DevOps Agent investigations. This enables integration with ticketing systems, monitoring tools, and other platforms that can send HTTP requests when incidents occur.

Prerequisites

Before configuring webhook access, ensure you have:

- An Agent Space configured in AWS DevOps Agent
- Access to the AWS DevOps Agent console
- The external system that will send webhook requests

Webhook types

AWS DevOps Agent supports the following types of webhooks:

- **Integration-specific webhooks** – Automatically generated when you configure third-party integrations like Dynatrace, Splunk, Datadog, New Relic, ServiceNow, or Slack. These webhooks are associated with the specific integration and use authentication methods determined by the integration type

- **Generic webhooks** – Can be manually created for triggering investigations from any source not covered by a specific integration. Generic webhooks currently use **HMAC** authentication (bearer token not currently available).
- **Grafana alert webhooks** – Grafana can send alert notifications directly to AWS DevOps Agent through webhook contact points. For setup instructions including a custom notification template, see Connecting Grafana.

Webhook authentication methods

The authentication method for your webhook depends on which integration it's associated with:

HMAC authentication – Used by:

- Dynatrace integration webhooks
- Generic webhooks (not linked to a specific third-party integration)

Bearer token authentication – Used by:

- Splunk integration webhooks
- Datadog integration webhooks
- New Relic integration webhooks
- ServiceNow integration webhooks
- Slack integration webhooks

Configuring webhook access

Step 1: Navigate to the webhook configuration

1. Sign in to the AWS Management Console and navigate to the AWS DevOps Agent console
2. Select your Agent Space
3. Go to the **Capabilities** tab
4. In the **Webhook** section, click **Configure**

Step 2: Generate webhook credentials

For integration-specific webhooks:

Webhooks are automatically generated when you complete the configuration of a third-party integration. The webhook endpoint URL and credentials are provided at the end of the integration setup process.

For generic webhooks:

1. Click **Generate webhook**
2. The system will generate an HMAC key pair
3. Securely store the generated key and secret—you won't be able to retrieve them again
4. Copy the webhook endpoint URL provided

Step 3: Configure your external system

Use the webhook endpoint URL and credentials to configure your external system to send requests to AWS DevOps Agent. The specific configuration steps depend on your external system.

Managing webhook credentials

Removing credentials – To delete webhook credentials, go to the webhook configuration section and click **Remove**. After removing credentials, the webhook endpoint will no longer accept requests until you generate new credentials.

Regenerating credentials – To generate new credentials, remove the existing credentials first, then generate a new key pair or token.

Using the webhook

Webhook request format

To trigger an investigation, your external system should send an HTTP POST request to the webhook endpoint URL.

For Version 1 (HMAC authentication):

Headers:

- `Content-Type: application/json`

- `x-amzn-event-signature: <HMAC signature>`
- `x-amzn-event-timestamp: <+%Y-%m-%dT%H:%M:%S.000Z>`

The HMAC signature is generated by signing the request body with your secret key using SHA-256.

For Version 2 (Bearer token authentication):

Headers:

- `Content-Type: application/json`
- `Authorization: Bearer <your-token>`

Request body:

The request body should include information about the incident:

```
json

{
  "title": "Incident title",
  "severity": "high",
  "affectedResources": ["resource-id-1", "resource-id-2"],
  "timestamp": "2025-11-23T18:00:00Z",
  "description": "Detailed incident description",
  "data": {
    "metadata": {
        "region": "us-east-1",
        "environment": "production"
    }
  }
}
```

Example code

Version 1 (HMAC authentication) - JavaScript:

```
const crypto = require('crypto');

// Webhook configuration
const webhookUrl = 'https://your-webhook-endpoint.amazonaws.com/invoke';
const webhookSecret = 'your-webhook-secret-key';
```

```
// Incident data
const incidentData = {
    eventType: 'incident',
    incidentId: 'incident-123',
    action: 'created',
    priority: "HIGH",
    title: 'High CPU usage on production server',
    description: 'High CPU usage on production server host ABC in AWS account 1234
 region us-east-1',
    timestamp: new Date().toISOString(),
    service: 'MyTestService',
    data: {
      metadata: {
        region: 'us-east-1',
        environment: 'production'
      }
    }
};

// Convert data to JSON string
const payload = JSON.stringify(incidentData);
const timestamp = new Date().toISOString();
const hmac = crypto.createHmac("sha256", webhookSecret);
hmac.update(`${timestamp}:${payload}`, "utf8");
const signature = hmac.digest("base64");

// Send the request
fetch(webhookUrl, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'x-amzn-event-timestamp': timestamp,
    'x-amzn-event-signature': signature
  },
  body: payload
})
.then(res => {
  console.log(`Status Code: ${res.status}`);
  return res.text();
})
.then(data => {
  console.log('Response:', data);
})
```

```
.catch(error => {
  console.error('Error:', error);
});
```

Version 1 (HMAC authentication) - cURL:

```
#!/bin/bash

# Configuration
WEBHOOK_URL="https://event-ai.us-east-1.api.aws/webhook/generic/YOUR_WEBHOOK_ID"
SECRET="YOUR_WEBHOOK_SECRET"

# Create payload
TIMESTAMP=$(date -u +%Y-%m-%dT%H:%M:%S.000Z)
INCIDENT_ID="test-alert-$(date +%s)"

PAYLOAD=$(cat <<EOF
{
"eventType": "incident",
"incidentId": "$INCIDENT_ID",
"action": "created",
"priority": "HIGH",
"title": "Test Alert",
"description": "Test alert description",
"service": "TestService",
"timestamp": "$TIMESTAMP"
}
EOF
)

# Generate HMAC signature
SIGNATURE=$(echo -n "${TIMESTAMP}:${PAYLOAD}" | openssl dgst -sha256 -hmac "$SECRET" -
binary | base64)

# Send webhook
curl -X POST "$WEBHOOK_URL" \
-H "Content-Type: application/json" \
-H "x-amzn-event-timestamp: $TIMESTAMP" \
-H "x-amzn-event-signature: $SIGNATURE" \
-d "$PAYLOAD"
```

Version 2 (Bearer token authentication) - JavaScript:

```
function sendEventToWebhook(webhookUrl, secret) {
  const timestamp = new Date().toISOString();

  const payload = {
    eventType: 'incident',
    incidentId: 'incident-123',
    action: 'created',
    priority: "HIGH",
    title: 'Test Alert',
    description: 'Test description',
    timestamp: timestamp,
    service: 'TestService',
    data: {}
  };

  fetch(webhookUrl, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "x-amzn-event-timestamp": timestamp,
      "Authorization": `Bearer ${secret}`,  // Fixed: template literal
    },
    body: JSON.stringify(payload),
  });
}
```

Version 2 (Bearer token authentication) - cURL:

```
#!/bin/bash

# Configuration
WEBHOOK_URL="https://event-ai.us-east-1.api.aws/webhook/generic/YOUR_WEBHOOK_ID"
SECRET="YOUR_WEBHOOK_SECRET"

# Create payload
TIMESTAMP=$(date -u +%Y-%m-%dT%H:%M:%S.000Z)
INCIDENT_ID="test-alert-$(date +%s)"

PAYLOAD=$(cat <<EOF
{
"eventType": "incident",
"incidentId": "$INCIDENT_ID",
"action": "created",
```

```
"priority": "HIGH",
"title": "Test Alert",
"description": "Test alert description",
"service": "TestService",
"timestamp": "$TIMESTAMP"
}
EOF
)

# Send webhook
curl -X POST "$WEBHOOK_URL" \
-H "Content-Type: application/json" \
-H "x-amzn-event-timestamp: $TIMESTAMP" \
-H "Authorization: Bearer $SECRET" \
-d "$PAYLOAD"
```

Troubleshooting webhooks

If you do not receive a 200

A 200 and a message like webhook received indicate the authentication passed and the message has been queued for the system to verify and process. If you do not get a 200 but a 4xx most likely there is something wrong with the authentication or headers. Try sending manually using the curl options to help debug the authentication.

If you receive a 200 but an investigation does not start

Likely cause is a misformated payload.

1. Check both timestamp and incident id are updated and unique. Duplicate messages are deduplicated.
2. Check the message is valid JSON
3. Check the format is correct

If you receive a 200 and investigation is immediately cancelled

Most likely you have hit the limit for the month. Please talk to your AWS contact to ask for a rate limit change if appropriate.

Related topics

- the section called “Creating an Agent Space”
- the section called “What is a DevOps Agent Web App?”
- the section called “DevOps Agent IAM permissions”

Integrating AWS DevOps Agent with Amazon EventBridge

You can integrate AWS DevOps Agent with your event-driven applications by using events that occur during investigation and mitigation lifecycles. AWS DevOps Agent sends events to Amazon EventBridge when the state of an investigation or mitigation changes. You can then create EventBridge rules that take action based on these events.

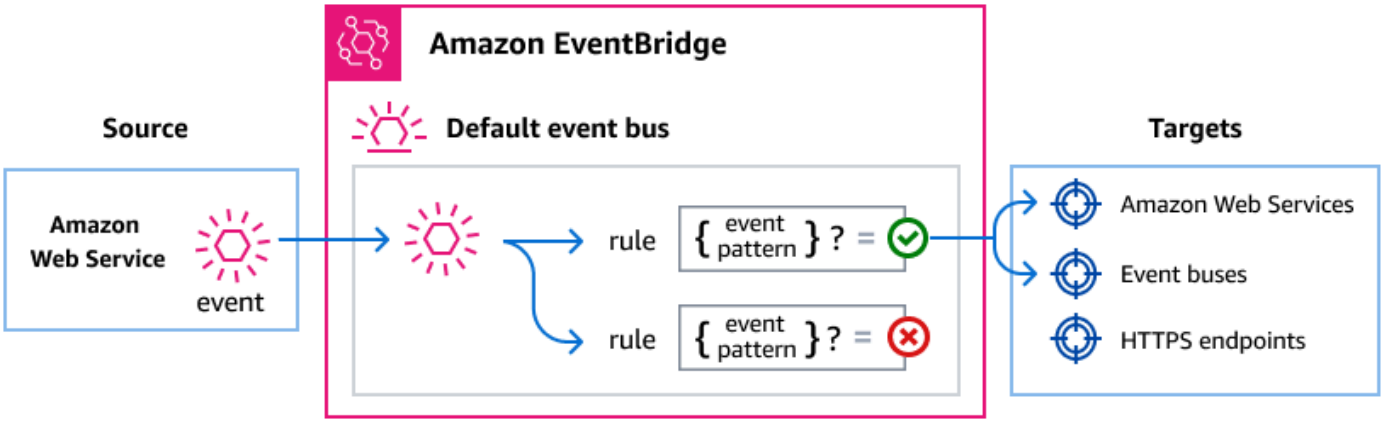
For example, you can create rules that perform the following actions:

- Invoke an AWS Lambda function to process investigation results when an investigation completes.
- Send an Amazon SNS notification when an investigation fails or times out.
- Update a ticketing system when a new investigation is created.
- Start an AWS Step Functions workflow when a mitigation action completes.

How EventBridge routes AWS DevOps Agent events

AWS DevOps Agent sends events to the EventBridge default event bus. EventBridge then evaluates the events against the rules that you create. When an event matches a rule's event pattern, EventBridge sends the event to the specified targets.

The following diagram shows how EventBridge routes AWS DevOps Agent events.



1. AWS DevOps Agent sends an event to the EventBridge default event bus when an investigation or mitigation lifecycle state changes.
2. EventBridge evaluates the event against the rules that you created.
3. If the event matches a rule's event pattern, EventBridge sends the event to the targets specified in the rule.

AWS DevOps Agent events

AWS DevOps Agent sends the following events to EventBridge. All events use the source `aws.aidevops`.

Supported investigation events

detail-type	Description
`Investigation Created`	An investigation was created in the agent space.
`Investigation Priority Updated`	The priority of an investigation was changed.
`Investigation In Progress`	An investigation started active analysis.
`Investigation Completed`	An investigation finished successfully with findings.

detail-type	Description
`Investigation Failed`	An investigation encountered an error and could not complete.
`Investigation Timed Out`	An investigation exceeded the maximum allowed duration.
`Investigation Cancelled`	An investigation was canceled before completion.
`Investigation Pending Triage`	An investigation is awaiting triage before active analysis begins.
`Investigation Linked`	An investigation was linked to a related incident or ticket.

Supported mitigation events

detail-type	Description
`Mitigation In Progress`	A mitigation action started.
`Mitigation Completed`	A mitigation action finished successfully.
`Mitigation Failed`	A mitigation action encountered an error and could not complete.
`Mitigation Timed Out`	A mitigation action exceeded the maximum allowed duration.
`Mitigation Cancelled`	A mitigation action was canceled before completion.

For detailed field descriptions and example events, see the section called “AWS DevOps Agent events detail reference”.

Creating event patterns that match AWS DevOps Agent events

EventBridge rules use event patterns to select events and route them to targets. An event pattern matches the structure of the events that it handles. You create event patterns to filter AWS DevOps Agent events based on the event fields.

The following examples show event patterns for common use cases.

Match all AWS DevOps Agent events

The following event pattern matches all events from AWS DevOps Agent.

```
{
  "source": ["aws.aidevops"]
}
```

Match only investigation events

The following event pattern uses a prefix match to select only investigation lifecycle events.

```
{
  "source": ["aws.aidevops"],
  "detail-type": [{"prefix": "Investigation"}]
}
```

Match only completion and failure events

The following event pattern matches events for completed or failed investigations and mitigations.

```
{
  "source": ["aws.aidevops"],
  "detail-type": [
    "Investigation Completed",
    "Investigation Failed",
    "Mitigation Completed",
    "Mitigation Failed"
  ]
}
```

Match events for a specific agent space

The following event pattern matches events from a specific agent space.

```
{
  "source": ["aws.aidevops"],
  "detail": {
    "metadata": {
      "agent_space_id": ["your-agent-space-id"]
    }
  }
}
```

For more information about event patterns, see Amazon EventBridge event patterns in the *Amazon EventBridge User Guide*.

Amazon EventBridge permissions

AWS DevOps Agent doesn't require additional permissions to deliver events to EventBridge. The events are sent to the default event bus automatically.

Depending on the targets that you configure for your EventBridge rules, you might need to add specific permissions. For more information about the permissions required for targets, see Using resource-based policies for Amazon EventBridge in the *Amazon EventBridge User Guide*.

Additional EventBridge resources

For more information about EventBridge concepts and configuration, see the following topics in the *Amazon EventBridge User Guide*:

- EventBridge event buses
- EventBridge events
- EventBridge event patterns
- EventBridge rules
- EventBridge targets

AWS DevOps Agent events detail reference

Events from AWS services have common metadata fields, including `source`, `detail-type`, `account`, `region`, and `time`. These events also contain a `detail` field with data specific to the

service. For AWS DevOps Agent events, the `source` is always `aws.aidevops` and the `detail-type` identifies the specific event.

Investigation events

The following `detail-type` values identify investigation events:

- `Investigation Created`
- `Investigation Priority Updated`
- `Investigation In Progress`
- `Investigation Completed`
- `Investigation Failed`
- `Investigation Timed Out`
- `Investigation Cancelled`
- `Investigation Pending Triage`
- `Investigation Linked`

The `source` and `detail-type` fields are included below because they contain specific values for AWS DevOps Agent events. For definitions of the other metadata fields that are included in all events, see Event structure in the *Amazon EventBridge Events Reference*.

The following is the JSON structure for investigation events.

```
{
  . . .,
  "detail-type" : "string",
  "source" : "aws.aidevops",
  . . .,
  "detail" : {
    "version" : "string",
    "metadata" : {
      "agent_space_id" : "string",
      "task_id" : "string",
      "execution_id" : "string"
    },
    "data" : {
      "task_type" : "string",
```

```
      "priority" : "string",
      "status" : "string",
      "created_at" : "string",
      "updated_at" : "string",
      "summary_record_id" : "string"
    }
  }
}
```

`detail-type` Identifies the type of event. For investigation events, this is one of the event names listed previously.

`source` Identifies the service that generated the event. For AWS DevOps Agent events, this value is `aws.aidevops`.

`detail` A JSON object that contains event-specific data. The `detail` object includes the following fields:

- `version` (string) – The schema version of the event detail. Currently `1.0.0`.
- `metadata.agent_space_id` (string) – The unique identifier of the agent space where the event originated.
- `metadata.task_id` (string) – The unique identifier of the task.
- `metadata.execution_id` (string) – The unique identifier of the execution run. Present when an execution has been assigned to the investigation.
- `data.task_type` (string) – The type of task. Value: `INVESTIGATION`.
- `data.priority` (string) – The priority level. Values: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `MINIMAL`.
- `data.status` (string) – The current status. Values: `PENDING_START`, `IN_PROGRESS`, `COMPLETED`, `FAILED`, `TIMED_OUT`, `CANCELLED`, `PENDING_TRIAGE`, `LINKED`.
- `data.created_at` (string) – ISO 8601 timestamp when the task was created.
- `data.updated_at` (string) – ISO 8601 timestamp when the task was last updated.
- `data.summary_record_id` (string) – The identifier of the summary record containing investigation findings. Included when a summary is generated for the completed investigation. You can retrieve the summary content through the AWS DevOps Agent API by using this identifier to look up the journal record with a record type of `investigation_summary_md`.

Example: Investigation Completed event

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789015",
  "detail-type": "Investigation Completed",
  "source": "aws.aidevops",
  "account": "123456789012",
  "time": "2026-03-12T18:10:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:aidevops:us-
east-1:123456789012:agentspace/8f6187a7-0388-4926-8217-3a0fe32f757c"
  ],
  "detail": {
    "version": "1.0.0",
    "metadata": {
      "agent_space_id": "8f6187a7-0388-4926-8217-3a0fe32f757c",
      "task_id": "a1b2c3d4-5678-90ab-cdef-example11111",
      "execution_id": "b2c3d4e5-6789-01ab-cdef-example22222"
    },
    "data": {
      "task_type": "INVESTIGATION",
      "priority": "CRITICAL",
      "status": "COMPLETED",
      "created_at": "2026-03-12T18:00:00Z",
      "updated_at": "2026-03-12T18:10:00Z",
      "summary_record_id": "d4e5f6g7-6789-01ab-cdef-example44444"
    }
  }
}
```


Example: Investigation Failed event


```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-123456789016",
  "detail-type": "Investigation Failed",
  "source": "aws.aidevops",
  "account": "123456789012",
  "time": "2026-03-12T18:10:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:aidevops:us-
east-1:123456789012:agentspace/8f6187a7-0388-4926-8217-3a0fe32f757c"
```

```
  ],
  "detail": {
    "version": "1.0.0",
    "metadata": {
      "agent_space_id": "8f6187a7-0388-4926-8217-3a0fe32f757c",
      "task_id": "a1b2c3d4-5678-90ab-cdef-example11111",
      "execution_id": "b2c3d4e5-6789-01ab-cdef-example22222"
    },
    "data": {
      "task_type": "INVESTIGATION",
      "priority": "CRITICAL",
      "status": "FAILED",
      "created_at": "2026-03-12T18:00:00Z",
      "updated_at": "2026-03-12T18:10:00Z"
    }
  }
}
```

Mitigation events

The following `detail-type` values identify mitigation events:

- `Mitigation In Progress`
- `Mitigation Completed`
- `Mitigation Failed`
- `Mitigation Timed Out`
- `Mitigation Cancelled`

The `source` and `detail-type` fields are included below because they contain specific values for AWS DevOps Agent events. For definitions of the other metadata fields that are included in all events, see Event structure in the *Amazon EventBridge Events Reference*.

The following is the JSON structure for mitigation events.

```
{
  . . .,
  "detail-type" : "string",
  "source" : "aws.aidevops",
  . . .,
  "detail" : {
```

```
    "version" : "string",
    "metadata" : {
      "agent_space_id" : "string",
      "task_id" : "string",
      "execution_id" : "string"
    },
    "data" : {
      "task_type" : "string",
      "priority" : "string",
      "status" : "string",
      "created_at" : "string",
      "updated_at" : "string",
      "summary_record_id" : "string"
    }
  }
}
```

`detail-type` Identifies the type of event. For mitigation events, this is one of the event names listed previously.

`source` Identifies the service that generated the event. For AWS DevOps Agent events, this value is `aws.aidevops`.

`detail` A JSON object that contains event-specific data. The `detail` object includes the following fields:

- `version` (string) – The schema version of the event detail. Currently `1.0.0`.
- `metadata.agent_space_id` (string) – The unique identifier of the agent space where the event originated.
- `metadata.task_id` (string) – The unique identifier of the task.
- `metadata.execution_id` (string) – The unique identifier of the execution run. Present when an execution has been assigned to the mitigation.
- `data.task_type` (string) – The type of task. Value: `INVESTIGATION`.
- `data.priority` (string) – The priority level. Values: `CRITICAL`, `HIGH`, `MEDIUM`, `LOW`, `MINIMAL`.
- `data.status` (string) – The current status. Values: `IN_PROGRESS`, `COMPLETED`, `FAILED`, `TIMED_OUT`, `CANCELLED`.
- `data.created_at` (string) – ISO 8601 timestamp when the task was created.
- `data.updated_at` (string) – ISO 8601 timestamp when the task was last updated.

- `data.summary_record_id` (string) – The identifier of the summary record containing mitigation findings. Included when a summary is generated for the completed mitigation. You can retrieve the summary content through the AWS DevOps Agent API by using this identifier to look up the journal record with a record type of `mitigation_summary_md`.

Example: Mitigation Completed event

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-12345678901c",
  "detail-type": "Mitigation Completed",
  "source": "aws.aidevops",
  "account": "123456789012",
  "time": "2026-03-12T18:20:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:aidevops:us-
east-1:123456789012:agentspace/8f6187a7-0388-4926-8217-3a0fe32f757c"
  ],
  "detail": {
    "version": "1.0.0",
    "metadata": {
      "agent_space_id": "8f6187a7-0388-4926-8217-3a0fe32f757c",
      "task_id": "a1b2c3d4-5678-90ab-cdef-example11111",
      "execution_id": "c3d4e5f6-7890-12ab-cdef-example33333"
    },
    "data": {
      "task_type": "INVESTIGATION",
      "priority": "CRITICAL",
      "status": "COMPLETED",
      "created_at": "2026-03-12T18:00:00Z",
      "updated_at": "2026-03-12T18:20:00Z",
      "summary_record_id": "e5f6g7h8-7890-12ab-cdef-example55555"
    }
  }
}
```

Example: Mitigation Failed event

```
{
  "version": "0",
  "id": "12345678-1234-1234-1234-12345678901d",
```

```
  "detail-type": "Mitigation Failed",
  "source": "aws.aidevops",
  "account": "123456789012",
  "time": "2026-03-12T18:20:00Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:aidevops:us-
east-1:123456789012:agentspace/8f6187a7-0388-4926-8217-3a0fe32f757c"
  ],
  "detail": {
    "version": "1.0.0",
    "metadata": {
      "agent_space_id": "8f6187a7-0388-4926-8217-3a0fe32f757c",
      "task_id": "a1b2c3d4-5678-90ab-cdef-example11111",
      "execution_id": "c3d4e5f6-7890-12ab-cdef-example33333"
    },
    "data": {
      "task_type": "INVESTIGATION",
      "priority": "CRITICAL",
      "status": "FAILED",
      "created_at": "2026-03-12T18:00:00Z",
      "updated_at": "2026-03-12T18:20:00Z"
    }
  }
}
```


Vended Logs and Metrics

You can monitor your agent spaces and service operations by using vended Amazon CloudWatch metrics and logs. This topic describes the CloudWatch metrics that the AWS DevOps Agent automatically publishes to your account and the vended logs that you can configure for delivery to your preferred destinations.

Vended CloudWatch metrics

AWS DevOps Agent automatically publishes metrics to Amazon CloudWatch in your account. These metrics are available without any configuration. You can use them to monitor usage, track operational activity, and create alarms.

Service-Linked Role

To have Amazon CloudWatch metrics published in your account for this service, AWS DevOps Agent will automatically create the service-linked role **AWSServiceRoleForAIDevOps** Service-Linked Role for you. If the IAM role invoking the API do not have appropriate permission the resource creation will fail with an InvalidParameterException.

> **Important**
>
> Customers who created their AgentSpace before March 13, 2026 will need to manually create the **AWSServiceRoleForAIDevOps** Service Linked Role to have CloudWatch metrics for AWS DevOps Agent published in their account.

Manually Create Service-Linked Role (For existing customers)

Do one of the following:

- In the IAM console, create the **AWSServiceRoleForAIDevOps** role under the **AWS DevOps Agent** service.
- From the AWS CLI, run the following command:

```
aws iam create-service-linked-role --aws-service-name aidevops.amazonaws.com
```

Namespace

All metrics are published under the `AWS/AIDevOps` namespace.

Dimensions

All metrics include the following dimension.

Dimension	Description
`AgentSpaceUUID`	The unique identifier of the agent space. To aggregate metrics across all agent spaces in your account, use CloudWatch math expressio ns or omit the dimension filter.

Metrics reference

Metric name	Description	Unit	Publishing frequency	Useful statistics
ConsumedChatRequests	The number of chat requests that an agent space consumed. To get the total count for your account, use the SUM statistic across all AgentSpaceUUID dimensions.	Count	Every 5 minutes	Sum, Average
ConsumedInvestigationTime	The time spent running investigations in an agent space.	Seconds	Every 5 minutes	Sum, Average, Maximum
ConsumedEvaluationTime	The time spent running evaluations in an agent space.	Seconds	Every 5 minutes	Sum, Average, Maximum
TopologyCompletionCount	The number of topology processing completions. AWS DevOps Agent emits this metric when a topology finishes	Count	Event-driven (emitted on each completion)	Sum, SampleCount

Metric name	Description	Unit	Publishing frequency	Useful statistics
	processin g, whether from initial creation during onboarding, a manual update, or a scheduled daily refresh.			

Viewing metrics in the CloudWatch console

1. Open the CloudWatch console.
2. In the navigation pane, choose **Metrics**, and then choose **All metrics**.
3. Choose the **AWS/AIDevOps** namespace.
4. Choose **By AgentSpace** to view metrics for your agent spaces.

> **Note**
>
> You can create CloudWatch alarms on these metrics to receive notifications when usage exceeds a threshold. For example, create an alarm on `ConsumedChatRequests` to monitor chat request consumption.

Prerequisites

Before you configure log delivery, make sure that you have the following:

- An active AWS account with access to the AWS DevOps Agent console
- An IAM principal with permissions for CloudWatch Logs delivery APIs
- (Optional) An Amazon S3 bucket or Amazon Data Firehose delivery stream, if you plan to use those as log destinations

Vended logs

AWS DevOps Agent supports vended logs that provide visibility into events that your agent spaces and service registrations process. Vended logs use the Amazon CloudWatch Logs infrastructure to deliver logs to your preferred destination.

To use vended logs, you must configure a delivery destination. The following destinations are supported:

- **Amazon CloudWatch Logs** – A log group in your account
- **Amazon S3** – An S3 bucket in your account
- **Amazon Data Firehose** – A Firehose delivery stream in your account

Supported log types

A single log type is supported: `APPLICATION_LOGS`. This log type covers all operational events that the service emits.

Log event types

The following table summarizes the events that AWS DevOps Agent logs.

Event	Description	Log level
Agent inbound event received	An agent is triggered by an integrated source and receives an inbound event (for example, a PagerDuty incident event).	INFO
Agent inbound event dropped	An inbound event was dropped before the agent processed it. The log includes the reason (for example, malformed data).	TBD
Agent outbound communica tion failure	An outbound communica tion to a third-party integrati	TBD

Event	Description	Log level
	on failed. The log includes the task ID and destination identifier (for example, an authentication error).	
Topology creation queued	A topology creation job was queued for processing.	INFO
Topology creation started	A topology creation job began processing.	INFO
Topology creation finished	A topology creation job completed processing. This event applies to initial creation, updates, and daily refreshes.	INFO
Resource discovery failed	Resource discovery during topology creation encounter ed a failure.	ERROR
Service registration failed	Service registration encounter s an unrecoverable failure	ERROR
Webhook Validation fails	When webhook received by Devops agent doesn't match the expected schema	ERROR
Association Validation status updates	When a Agent space associati on(typical primary/secondary account), validation status changes from valid to invalid and vice versa(for example, caused by malformed role, that is not assumable by the service).	ERROR/INFO

Permissions

AWS DevOps Agent uses CloudWatch vended logs (V2 permissions) to deliver logs. To set up log delivery, the IAM role that configures the delivery must have the following permissions:

- `aidevops:AllowVendedLogDeliveryForResource` – Required to allow log delivery for the agent space resource.
- Permissions for the CloudWatch Logs delivery APIs (`logs:PutDeliverySource`, `logs:PutDeliveryDestination`, `logs:CreateDelivery`, and related operations).
- Permissions specific to your chosen delivery destination.

For the full IAM policy that is required for each destination type, see the following topics in the *Amazon CloudWatch Logs User Guide*:

- Logs sent to CloudWatch Logs
- Logs sent to Amazon S3
- Logs sent to Firehose

Configure log delivery (console)

AWS DevOps Agent provides two locations in the AWS Management Console to configure log delivery:

- **Service Registration settings page** – Configure log delivery for service-level events. These logs use the service ARN (`arn:aws:aidevops:<region>:<account-id>:service/<account-id>`) as the resource.
- **Agent Space page** – Configure log delivery for events that are specific to an individual agent space. These logs use the agent space ARN (`arn:aws:aidevops:<region>:<account-id>:agentspace/<agent-space-id>`) as the resource.

To configure log delivery for a service registration

1. Open the AWS DevOps Agent console in the AWS Management Console.
2. In the navigation pane, choose **Settings**.
3. In the **Capability Providers > Logs** tab, choose **Configure**.
4. For **Destination type**, choose one of the following:

5. **CloudWatch Logs** – Select or create a log group.
6. **Amazon S3** – Enter the S3 bucket ARN.
7. **Amazon Data Firehose** – Select or create a Firehose delivery stream.
8. For **Additional settings** – *optional*, you can specify the following options:
   a. For **Field selection**, select the log field names that you want to deliver to your destination. You can select access log fields and a subset of real-time access log fields.
   b. (Amazon S3 only) For **Partitioning**, specify the path to partition your log file data.
   c. (Amazon S3 only) For **Hive-compatible file format**, you can select the checkbox to use Hive-compatible S3 paths. This helps simplify loading new data into your Hive-compatible tools.
   d. For **Output format**, specify your preferred format.
   e. For **Field delimiter**, specify how to separate log fields.
9. Choose **Save**.
10. Verify that the delivery status shows **Active**.

To configure log delivery for an agent space

1. Open the AWS DevOps Agent console in the AWS Management Console.
2. Choose the agent space that you want to configure.
3. In the **Configuration** tab, choose **Configure**.
4. For **Destination type**, choose one of the following:
5. **CloudWatch Logs** – Select or create a log group.
6. **Amazon S3** – Enter the S3 bucket ARN.
7. **Amazon Data Firehose** – Select or create a Firehose delivery stream.
8. For **Additional settings – *optional** *, you can specify the following options:
   a. For **Field selection**, select the log field names that you want to deliver to your destination. You can select access log fields and a subset of real-time access log fields.
   b. (Amazon S3 only) For **Partitioning**, specify the path to partition your log file data.
   c. (Amazon S3 only) For **Hive-compatible file format**, you can select the checkbox to use Hive-compatible S3 paths. This helps simplify loading new data into your Hive-compatible tools.
   d. For **Output format**, specify your preferred format.
   e. For **Field delimiter**, specify how to separate log fields.
9. Choose **Save**.

10. Verify that the delivery status shows **Active**.

Configure log delivery (CloudWatch API)

You can also use the CloudWatch Logs API to configure log delivery programmatically. A working log delivery consists of three elements:

- A **DeliverySource** – Represents the AWS DevOps Agent space resource that generates the logs.
- A **DeliveryDestination** – Represents the destination where logs are written.
- A **Delivery** – Connects a delivery source to a delivery destination.

Step 1: Create a delivery source

Use the PutDeliverySource operation to create a delivery source. Pass the ARN of your AWS DevOps Agent space resource and specify `APPLICATION_LOGS` as the log type.

The following example creates a delivery source for an agent space:

```
{
    "name": "my-agent-space-delivery-source",
    "resourceArn": "arn:aws:aidevops:us-east-1:123456789012:agentspace/my-agent-space-
id",
    "logType": "APPLICATION_LOGS"
}
```

The following example creates a delivery source for the service:

```
{
    "name": "my-service-delivery-source",
    "resourceArn": "arn:aws:aidevops:us-east-1:123456789012:service",
    "logType": "APPLICATION_LOGS"
}
```

Step 2: Create a delivery destination

Use the PutDeliveryDestination operation to configure where logs are stored. You can choose Amazon CloudWatch Logs, Amazon S3, or Amazon Data Firehose.

The following example creates a CloudWatch Logs destination:

```
{
    "name": "my-cwl-destination",
    "deliveryDestinationConfiguration": {
        "destinationResourceArn": "arn:aws:logs:us-east-1:123456789012:log-group:/aws/
aidevops/my-agent-space"
    },
    "outputFormat": "json"
}
```

The following example creates an Amazon S3 destination:

```
{
    "name": "my-s3-destination",
    "deliveryDestinationConfiguration": {
        "destinationResourceArn": "arn:aws:s3:::my-aidevops-logs-bucket"
    },
    "outputFormat": "json"
}
```

The following example creates an Amazon Data Firehose destination:

```
{
    "name": "my-firehose-destination",
    "deliveryDestinationConfiguration": {
        "destinationResourceArn": "arn:aws:firehose:us-
east-1:123456789012:deliverystream/my-aidevops-log-stream"
    },
    "outputFormat": "json"
}
```

Note

If you deliver logs cross-account, you must use PutDeliveryDestinationPolicy in the destination account to authorize the delivery.

If you want to use CloudFormation, you can use the following:

- Delivery
- DeliveryDestination

- DeliverySource

The `ResourceArn` is the `AgentSpaceArn`, and `LogType` must be `APPLICATION_LOGS` as the supported log type.

Step 3: Create a delivery

Use the CreateDelivery operation to link the delivery source to the delivery destination.

```
{
    "deliverySourceName": "my-agent-space-delivery-source",
    "deliveryDestinationArn": "arn:aws:logs:us-east-1:123456789012:delivery-
destination:my-cwl-destination"
}
```

AWS CloudFormation

You can also configure log delivery by using AWS CloudFormation with the following resources:

- AWS::Logs::DeliverySource
- AWS::Logs::DeliveryDestination
- AWS::Logs::Delivery

Set `ResourceArn` to the AWS DevOps Agent agent space or service ARN, and set `LogType` to `APPLICATION_LOGS`.

Log schema reference

AWS DevOps Agent uses a shared log schema across all event types. Not every log event uses every field.

The following table describes the fields in the log schema.

Field	Type	Description
event_timestamp	Long	Unix timestamp of when the event occurred

Field	Type	Description
resource_arn	String	ARN of the resource that generated the event
optional_account_id	String	AWS account ID associated with the log.
optional_level	String	Log level: `INFO`, `WARN`, `ERROR`
optional_agent_space_id	String	Identifier of the agent space.
optional_association_id	String	Association identifier for the log.
optional_status	String	Status of the topology operation.
optional_webhook_id	String	Webhook identifier.
optional_mcp_endpoint_url	String	MCP server endpoint URL
optional_service_type	String	Type of the Service: `DYNATRACE` , `DATADOG`, `GITHUB`, `SLACK`, `SERVICENOW` .
optional_service_endpoint_url	String	Endpoint URL for third-party integrations.
optional_service_id	String	Identifier of the source.
request_id	String	Request identifier for correlating with AWS CloudTrail or support tickets.
optional_operation	String	Name of the operation that was performed.

Field	Type	Description
optional_task_type	String	Agent backlog task type: `INVESTIGATION` or `EVALUATION`
optional_task_id	String	Agent Backlog Task IDAgent backlog task identifier.
optional_reference	String	Reference from an agent task (for example, a Jira ticket).
optional_error_type	String	Error type
optional_error_message	String	Error description when an operation fails.
optional_details	String (JSON)	Service-specific event payload that contains operation parameters and results.

Manage and disable log delivery

You can modify or remove log delivery at any time from the AWS DevOps Agent console in the AWS Management Console or by using the CloudWatch Logs API.

Manage log delivery (console)

1. Open the AWS DevOps Agent console in the AWS Management Console.
2. Navigate to the **Settings** page (for service-level logs) or the specific **Agent Space** page (for Agent Space-level logs).
3. In the **Configuration** tab (for Agent Space-level logs) or **Capability Providers > Logs** tab (for service-level logs), choose the delivery to modify.
4. Update the configuration as needed and choose **Save**.

> **Note:** You can't change the destination type of an existing delivery. To change the destination type, delete the current delivery and create a new one.

Disable log delivery (console)

1. Open the AWS DevOps Agent console in the AWS Management Console.
2. Navigate to the **Settings** page (for service-level logs) or the specific **Agent Space** page (for Agent Space-level logs).
3. In the **Configuration** tab (for Agent Space-level logs) or **Capability Providers > Logs** tab (for service-level logs),, select the delivery to remove.
4. Choose **Delete** and confirm.

Disable log delivery (API)

To remove a log delivery by using the API, delete the resources in the following order:

1. Delete the delivery by using DeleteDelivery.
2. Delete the delivery source by using DeleteDeliverySource.
3. (Optional) If the delivery destination is no longer needed, delete it by using DeleteDeliveryDestination.

> **Important**
>
> You are responsible for removing log delivery resources after you delete the agent space resource that generates the logs (for example, after you delete an agent space). If you don't remove these resources, orphaned delivery configurations might remain.

Pricing

The AWS DevOps Agent does not charge for enabling vended logs. However, you can incur charges for the delivery, ingestion, storage or access, depending on the log delivery destination that you select. For pricing details, see **Vended Logs** on the **Logs** tab at Amazon CloudWatch Pricing.

For destination-specific pricing, see the following:

- Amazon CloudWatch Logs Pricing
- Amazon S3 Pricing
- Amazon Data Firehose Pricing

Connecting to privately hosted tools

Private connections overview

AWS DevOps Agent can be extended with custom Model Context Protocol (MCP) tools and other integrations that give the agent access to internal systems such as private package registries, self-hosted observability platforms, internal documentation APIs, and source control instances (see: *Configuring capabilities for AWS DevOps Agent*). These services often run inside an Amazon Virtual Private Cloud (Amazon VPC) with restricted or no public internet access, which means AWS DevOps Agent can't reach them by default.

Private connections for AWS DevOps Agent let you securely connect your Agent Space to services running in your VPC without exposing them to the public internet. Private connections work with any integration that needs to reach a private endpoint, including MCP servers, self-hosted Grafana or Splunk instances, and source control systems like GitHub Enterprise Server and GitLab Self-Managed.

> **Note**
>
> If your privately hosted tools make outbound requests to the AWS DevOps Agent from within your VPC, this traffic can also be secured by using a VPC Endpoint so it stays within the AWS network. For example, this can be used with tools that trigger the DevOps Agent via webhook events (see: the section called “Invoking DevOps Agent through Webhook”). For more information, see the section called “VPC Endpoints (AWS PrivateLink)”.

How private connections work

A private connection creates a secure network path between AWS DevOps Agent and a target resource in your VPC. Under the hood, AWS DevOps Agent uses Amazon VPC Lattice to establish this secure private connectivity path. VPC Lattice is an application networking service that lets you connect, secure, and monitor communication between applications across VPCs, accounts, and compute types, without managing the underlying network infrastructure.

When you create a private connection, the following occurs:

- You provide the VPC, subnets, and (optionally) security groups that have network connectivity to your target service.

- AWS DevOps Agent creates a service-managed resource gateway and provisions its elastic network interfaces (ENIs) in the subnets you specified.
- The agent uses the resource gateway to route traffic to your target service's IP address or DNS name over the private network path.

The resource gateway is fully managed by AWS DevOps Agent and appears as a read-only resource in your account (named `aidevops-{your-private-connection-name}`). You don't need to configure or maintain it. The only resources created in your VPC are ENIs in the subnets you specify. These ENIs serve as the entry point for private traffic and are managed entirely by the service. They don't accept inbound connections from the internet, and you retain full control over their traffic through your own security groups.

Security

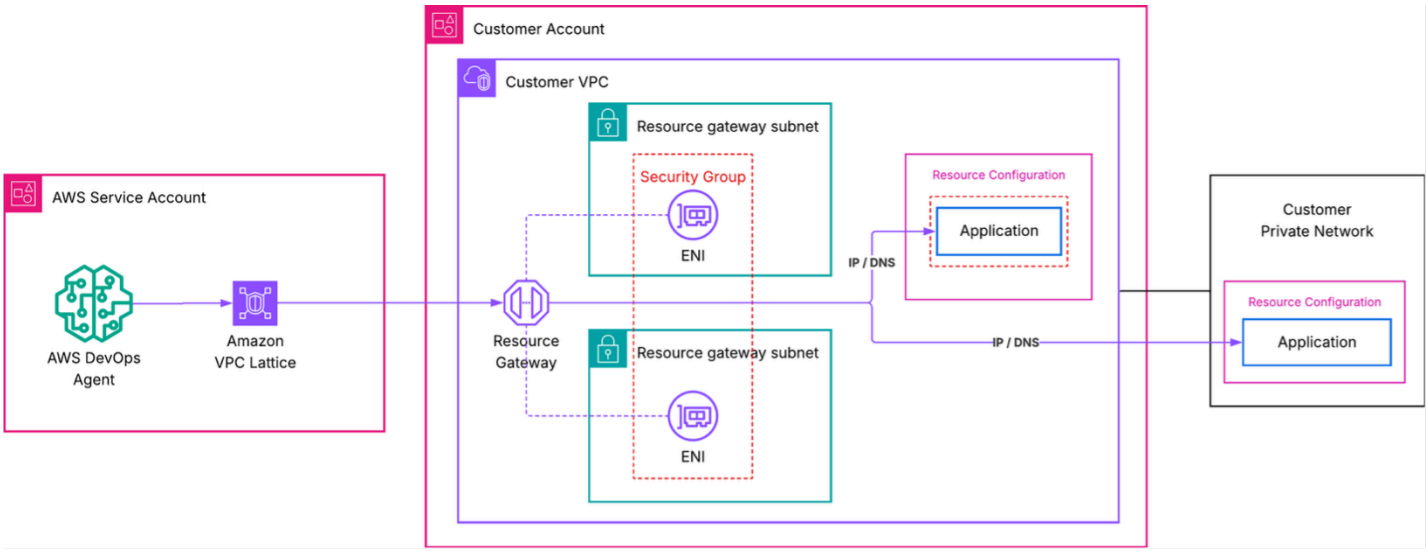
Private connections are designed with multiple layers of security:

- **No public internet exposure** – All traffic between AWS DevOps Agent and your target service stays on the AWS network. Your service never needs a public IP address or internet gateway.
- **Service-controlled resource gateway** – The service-managed resource gateway is read-only in your account. It can only be used by AWS DevOps Agent, and no other service or principal can route traffic through it. You can verify this in AWS CloudTrail logs, which record all VPC Lattice API calls.
- **Your security groups, your rules** – You control inbound and outbound traffic to the ENIs through security groups that you own and manage. If you don't specify security groups, AWS DevOps Agent creates a default security group scoped to the ports you define.
- **Service-linked roles with least privilege** – AWS DevOps Agent uses a service-linked role to create only the necessary VPC Lattice and Amazon EC2 resources. This role is scoped to resources tagged with `AWSAIDevOpsManaged` and cannot access any other resources in your account.

> **Note**
>
> If your organization has service control policies (SCPs) that restrict VPC Lattice API actions, the service-managed resource gateway is created through a service-linked role. Ensure your SCPs permit the necessary actions for the service-linked role.

Architecture

The following diagram shows the network path for a private connection.



In this architecture:

- AWS DevOps Agent initiates a request to your target service.
- Amazon VPC Lattice routes the request through the service-managed resource gateway in your VPC. For advanced setups using your own VPC Lattice resources, see Advanced setup using existing VPC Lattice resources.
- An ENI in your VPC receives the traffic and forwards it to your target service's IP address or DNS name.
- Your security groups govern which traffic is allowed through the ENIs.
- From your target service's perspective, the request originates from private IP addresses of ENIs within your VPC.

Create a private connection

You can create a private connection using the AWS Management Console or the AWS CLI.

> **Note**
>
> The following Availability Zones aren't supported by VPC Lattice: `use1-az3`, `usw1-az2`, `apne1-az3`, `apne2-az2`, `euc1-az2`, `euw1-az4`, `cac1-az3`, `ilc1-az2`.

Prerequisites

Before creating a private connection, verify that you have the following:

- **An active Agent Space** – You need an existing Agent Space in your account. If you don't have one, see *Getting started with AWS DevOps Agent*.
- **A privately reachable target service** – Your MCP server, observability platform, or other service must be reachable at a known private IP address or DNS name from the VPC where the resource gateway is deployed. The service can run in the same VPC, a peered VPC, or on-premises, as long as it's routable from the resource gateway subnets. The service must serve HTTPS traffic with a minimum TLS version of 1.2 on a port that you specify when creating the connection.
- **Subnets in your VPC** – Identify 1–20 subnets where the ENIs will be created. We recommend selecting subnets in multiple Availability Zones for high availability. These subnets must have network connectivity to your target service. One subnet per Availability Zone can be used by VPC Lattice.
- **(Optional) Security groups** – If you want to control traffic with specific rules, prepare up to five security group IDs to attach to the ENIs. If you omit security groups, AWS DevOps Agent creates a default security group.

Private connections are account-level resources. After you create a private connection, you can reuse it across multiple integrations and Agent Spaces that need to reach the same host.

Create a private connection using the console

1. Open the AWS DevOps Agent console.
2. In the navigation pane, choose **Capability providers**, then choose **Private connections**.
3. Choose **Create a new connection**.
4. For **Name**, enter a descriptive name for the connection, such as `my-mcp-tool-connection`.
5. For **VPC**, select the VPC where the resource gateway ENIs will be deployed.
6. For **Subnets**, select one or more subnets (up to 20). We recommend choosing subnets in at least two Availability Zones.
7. For **IP address type**, select the type of IP address of your target service (`IPv4`, `IPv6`, or `DualStack`).

8. (Optional) For **Number of IPv4 addresses**, if you selected IPv4 or Dualstack for the IP address type, you can enter the number of IPv4 addresses per ENI for your resource gateway. The default is 16 IPv4 addresses per ENI.
9. (Optional) For **Security groups**, select existing security groups (up to 5) to restrict what traffic is allowed to reach your target service. If you don't select any, a default security group is created.
10. (Optional) For **Port ranges**, specify the TCP ports your target application listens on (for example, `443` or `8080-8090`). You can specify up to 11 port ranges.
11. For **Host address**, enter the IP address or DNS name of your target service (for example, `mcp.internal.example.com` or `10.0.1.50`). The service must be reachable from the selected VPC. If you choose a DNS name, it must be resolvable from the selected VPC.
12. (Optional) For **Certificate public key**, if the host address you specified uses TLS certificates issued by a private certificate authority, enter the PEM-encoded public key of the certificate. This allows AWS DevOps Agent to trust the TLS connection to your target service.
13. Choose **Create connection**.

The connection status changes to **Create in progress**. This process can take up to 10 minutes. When the status changes to **Active**, the network path is ready.

If the status changes to **Create failed**, verify the following:

- The subnets you specified have available IP addresses.
- Your account has not reached VPC Lattice service quotas.
- No restrictive IAM policies are preventing the service-linked role from creating resources.

> **Note**
>
> These steps can also be performed by selecting `Create a new private connection` during the registration of a capability provider. For more information, see Use a private connection with a capability provider.

Create a private connection using the AWS CLI

Run the following command to create a private connection. Replace the placeholder values with your own.

```
aws devops-agent create-private-connection \
    --name my-mcp-tool-connection \
    --mode '{
        "serviceManaged": {
            "hostAddress": "mcp.internal.example.com",
            "vpcId": "vpc-0123456789abcdef0",
            "subnetIds": [
                "subnet-0123456789abcdef0",
                "subnet-0123456789abcdef1"
            ],
            "securityGroupIds": [
                "sg-0123456789abcdef0"
            ],
            "portRanges": ["443"]
        }
    }'
```

The response includes the connection name and a status of `CREATE_IN_PROGRESS`:

```
{
    "name": "my-mcp-tool-connection",
    "status": "CREATE_IN_PROGRESS",
    "resourceGatewayId": "rgw-0123456789abcdef0",
    "hostAddress": "mcp.internal.example.com",
    "vpcId": "vpc-0123456789abcdef0"
}
```

To check the connection status, use the `describe-private-connection` command:

```
aws devops-agent describe-private-connection \
    --name my-mcp-tool-connection
```

When the status is `ACTIVE`, your private connection is ready to use.

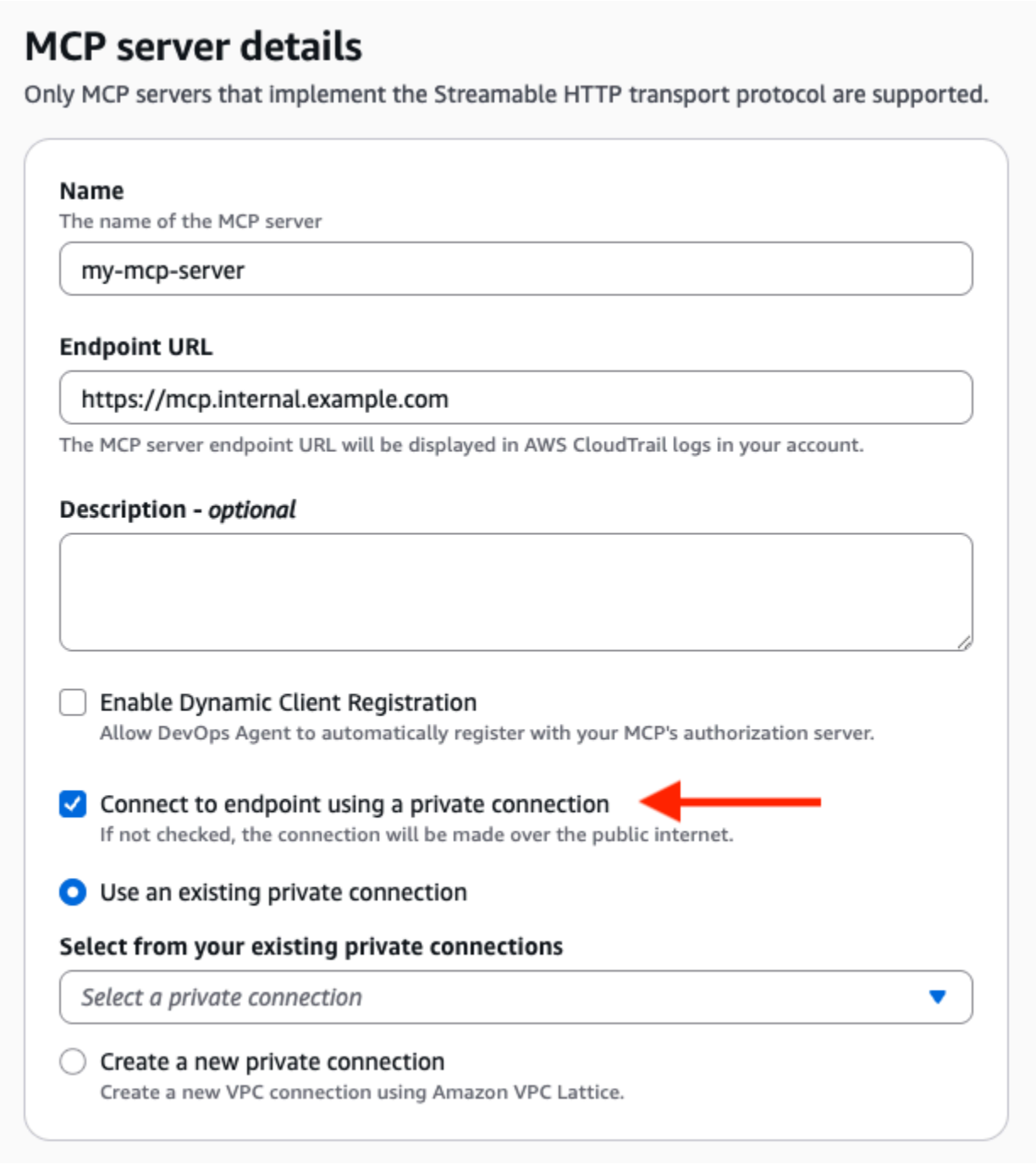
Use a private connection with a capability provider

To use a private connection, you can link to it during the registration of a capability provider. Supported capabilities that can be used with private connections include: `GitHub`, `GitLab`, `MCP Server`, and `Grafana`. You can perform this step using the AWS Management Console or the AWS CLI.

> **Note**
>
> When registering a capability provider, AWS DevOps Agent validates that the endpoint is reachable and responding. Ensure your target service is running and accepting connections before completing registration.

Use a private connection with a capability provider using the console

In the AWS DevOps Agent console, private connections can be linked to a capability during registration by selecting the "Connect to endpoint using a private connection" option.



1. Open the AWS DevOps Agent console and navigate to your Agent Space.
2. In the **Capability Providers** section, choose **Registration**.
3. Select **Register** for the capability type you want to use with the private connection.
4. On the registration details view, enter the Endpoint URL you want to connect to using the private connection (for example, `https://mcp.internal.example.com`).
5. Select **Connect to endpoint using a private connection**.

6. Either select an existing private connection that corresponds to the Endpoint URL you want to connect to, or select **Create a new private connection** to create one.
7. Complete the registration process for the capability provider.

Use a private connection with a capability provider using the AWS CLI

You can register capabilities with a private connection by including the `private-connection-name` argument. Below is an example of registering an MCP Server with API Key authorization using the `my-mcp-tool-connection` private connection. Replace the placeholder values with your own.

```
aws devops-agent register-service \
    --service mcpserver \
    --private-connection-name my-mcp-tool-connection \
    --service-details '{
        "mcpserver": {
            "name": "my-mcp-tool",
            "endpoint": "https://mcp.internal.example.com",
            "authorizationConfig": {
                "apiKey": {
                    "apiKeyName": "api-key",
                    "apiKeyValue": "secret-value",
                    "apiKeyHeader": "x-api-key"
                }
            }
        }
    }' \
    --region us-east-1
```

Verify a private connection

After the private connection reaches the **Active** state and has been utilized by a capability provider, verify that AWS DevOps Agent can reach your target service:

1. Open the AWS DevOps Agent console and navigate to your Agent Space.
2. Start a new chat session.
3. Invoke a command that uses the integration backed by your private connection. For example, if your MCP tool provides access to an internal knowledge base, ask the agent a question that requires that knowledge base.

4. Confirm that the agent returns results from the private service.

If the connection fails, check the following:

- **VPC Lattice limits** - Verify that you have not reached any resource gateway or other VPC Lattice quota limits
- **Security group rules** – Verify that the security groups attached to the ENIs allow outbound traffic on the port your service listens on. Also verify that your service's security group allows inbound traffic on the target port. Traffic arrives from VPC Lattice data plane IPs within your VPC CIDR range. You can use security group referencing (allowing the ENI security group as a source) or allow inbound from the VPC CIDR.
- **Subnet connectivity** – Verify that the subnets you selected can route traffic to your service. If the service runs in a different subnet, confirm that the route tables allow traffic between them.
- **Service availability** – Confirm that your service is running and accepting connections on the expected port.
- **Unsupported Availability Zone** - Verify your subnets are in supported Availability Zones. Run `aws ec2 describe-subnets --subnet-ids <your-subnet-ids> --query 'Subnets[*].[SubnetId,AvailabilityZoneId]'` and check against the unsupported Availability Zones listed above.

Delete a private connection

You can delete unused private connections using the AWS Management Console or the AWS CLI.

Delete a private connection using the console

1. Open the AWS DevOps Agent console.
2. In the navigation pane, choose **Capability providers**, then choose **Private connections**.
3. Select the **Actions** menu for the private connection you want to delete, and select **Remove**.

The private connection will be displayed with a status of "Removing connection" while AWS DevOps Agent removes the managed resource gateway and ENIs from your VPC. After deletion completes, the connection no longer appears in your list of private connections.

Delete a private connection using the AWS CLI

```
aws devops-agent delete-private-connection \
    --name my-mcp-tool-connection
```

The response returns a status of DELETE_IN_PROGRESS. AWS DevOps Agent removes the managed resource gateway and ENIs from your VPC. After deletion completes, the connection no longer appears in your list of private connections.

Advanced setup using existing VPC Lattice resources

If your organization already uses Amazon VPC Lattice and manages your own resource configurations, you can create a private connection in self-managed mode. Instead of having AWS DevOps Agent create a resource gateway for you, you provide the Amazon Resource Name (ARN) of an existing resource configuration that points to your target service.

This approach is useful when you:

- Want full control over the resource gateway and resource configuration lifecycle.
- Need to share resource configurations across multiple AWS accounts or services.
- Require VPC Lattice access logs for detailed traffic monitoring.
- Run a hub-and-spoke network architecture.

To create a self-managed private connection with the AWS CLI:

```
aws devops-agent create-private-connection \
    --name my-advanced-connection \
    --mode '{
        "selfManaged": {
            "resourceConfigurationId": "arn:aws:vpc-lattice:us-
east-1:123456789012:resourceconfiguration/rcfg-0123456789abcdef0"
        }
    }'
```

For more details on setting up VPC Lattice resource gateways and resource configurations, see the Amazon VPC Lattice User Guide.

Related topics

- the section called “VPC Endpoints (AWS PrivateLink)”
- the section called “Connecting MCP Servers”
- *Configuring capabilities for AWS DevOps Agent*
- *AWS DevOps Agent Security*
- the section called “DevOps Agent IAM permissions”

AWS DevOps Agent Security

This document provides information about security considerations, data protection, access controls, and compliance capabilities for AWS DevOps Agent. Use this information to understand how AWS DevOps Agent is designed to meet your security and compliance requirements.

Multi-layered security

AWS DevOps Agent implements security at multiple layers. Even if broader permissions are granted to the agent's IAM role, the agent enforces its own internal access controls to limit the scope of its actions. For example, if a customer adds a full Amazon S3 access IAM policy to the agent's IAM role, AWS DevOps Agent will ensure that only logs after the `AWSLogs` prefix are read for troubleshooting purposes.

We recommend following the principle of least privilege when configuring IAM permissions for AWS DevOps Agent, and implementing security at multiple layers. Defense in depth ensures that no single misconfiguration can compromise the security of your environment.

Agent Spaces

Agent Spaces serve as the primary security boundary in AWS DevOps Agent. Each Agent Space:

- Operates independently with its own configurations and permissions
- Defines which AWS accounts and resources the agent can access
- Establishes connections to third-party platforms

Agent Spaces maintain strict isolation to ensure security and prevent unintended access across different environments or teams.

Regional processing and data flow

AWS DevOps Agent operates globally with regional processing capabilities. The agent retrieves operational data from AWS regions across all AWS accounts granted access within the configured Agent Space. This multi-region cross-account data collection ensures comprehensive incident analysis while respecting geographical boundaries for inference processing.

Amazon Bedrock usage and cross-region inference

AWS DevOps Agent will automatically select the optimal region within your geography to process your inference requests. This maximizes available compute resources, model availability, and delivers the best customer experience. Your data will remain stored only in the region where your Agent Space is created, however, input prompts and output results may be processed outside that region as described in the following list. All data will be transmitted encrypted across Amazon's secure network.

AWS DevOps Agent will securely route your inference requests to available compute resources within the geographic area where the request originated, as follows:

- Inference requests originating in the European Union will be processed within the European Union.
- Inference requests originating in the United States will be processed within the United States.
- Inference requests originating in Australia will be processed within Australia.
- Inference requests originating within Japan will be processed within Japan.
- If an inference request originates in an area not listed, it will be processed by default within the United States.
- DevOps Agent and Bedrock are not impacted by customer policies in Service Control Policies (SCPs) or Control Tower that restrict customer content to specific regions
- Bedrock may use regions other than the originating region within your geography to perform stateless inference to optimize performance and availability

Identity and access management

Authentication methods

AWS DevOps Agent provides two authentication methods to log into the AWS DevOps Agent Space web app:

- **AWS Identity Center integration** – The primary authentication method uses OAuth 2.0 with session-based authentication using HTTP-only cookies. AWS Identity Center can federate with external identity providers through standard OIDC and SAML protocols, including providers like Okta, Ping Identity, and Microsoft Entra ID. This method supports multi-factor authentication

through your identity provider. AWS Identity Center defaults to session durations of up to 12 hours and can be configured to a desired duration.

- **IAM authentication link** – An alternative method provides direct access to the web app from the AWS Management Console using JWT-based tokens derived from an existing AWS Management Console session. This option is useful for evaluating AWS DevOps Agent before implementing full Identity Center integration as well as gaining administrative access if the AWS DevOps Agent web app becomes inaccessible through Identity Center based authentication. Sessions are limited to 10 minutes.

IAM roles

AWS DevOps Agent uses IAM roles to define access permissions:

- **Primary account role** – Grants the agent access to resources in the AWS account where you create the Agent Space as well as access to secondary account roles.
- **Secondary account roles** – Grants the agent access to resources in additional AWS accounts connected to the Agent Space.
- **Web app role** – Grants users access to AWS DevOps Agent investigation data and findings in the web app.

These roles should be configured following the principle of least privilege, granting only the necessary read-only permissions required for investigations.

Data protection

Data encryption

AWS DevOps Agent encrypts all customer data:

- **Encryption at rest** – All data is encrypted with AWS-managed keys.
- **Encryption in transit** – All retrieved logs, metrics, knowledge items, ticket metadata, and other data are encrypted in transit inside the agent's private network and to outside networks.

Data storage and retention

Data is stored in the region where your Agent Space is created, while inference processing may occur within your geography as described in the Amazon Bedrock usage section above.

Personal identifiable information (PII)

AWS DevOps Agent does not filter PII information when summarizing data gathered during investigations, recommendation evaluations, or chat responses. It is recommended that PII data be redacted before storing in observability logs.

Agent journal and audit logging

Agent journal

Both the Incident Investigation and Prevention capabilities maintain detailed journals that:

- Log every reasoning step and action taken
- Create complete transparency into agent decision-making processes
- Cannot be modified by the agents once recorded, minimizing attacks such as prompt injection from hiding important actions
- Include all chat messages from the Investigation page

AWS CloudTrail integration

All AWS DevOps Agent API calls are automatically captured by AWS CloudTrail within the hosting AWS account. Using the information collected by CloudTrail, you can determine:

- The request that was made to the agent
- The IP address from which the request was made
- Who made the request
- When it was made

Prompt injection protection

A prompt injection attack occurs when an attacker embeds malicious instructions into external data, such as a webpage or document, that a generative AI system will later process. AWS DevOps Agent natively consumes many data sources as part of its normal operations, including logs, resource tags, and other operational data. AWS DevOps Agent protects against prompt injection attacks through the safeguards below, but it is important to ensure all connected data sources and user access to those data sources are trusted. See Shared responsibility model section for more.

Prompt injection safeguards:

- **Limited write capabilities** – The tools available to the agent are not able to mutate resources, with the exception of opening tickets and support cases. This prevents malicious instructions from modifying your infrastructure or applications.
- **Account boundary enforcement** – AWS DevOps Agent only operates within the boundary permitted by the roles assigned to the agent in the primary and connected secondary AWS accounts. The agent cannot access or modify resources outside of its configured scope.
- **AI safety protections** – AWS DevOps Agent uses models with AI Safety Level 3 (ASL-3) protections. These protections include classifiers that detect and prevent prompt injection attacks before they can affect agent behavior.
- **Immutable audit trail** – The agent journal logs every reasoning step and action taken. Journal entries cannot be modified by the agent once recorded, preventing prompt injection attacks from hiding malicious actions.

While AWS DevOps Agent provides multiple layers of protection against prompt injection attacks, certain configurations can increase risk:

- **Custom MCP server tools** – The bring-your-own MCP feature allows you to introduce custom tools to the agent, which can present additional opportunities for prompt injection. Custom tools may not have the same security controls as native AWS DevOps Agent tools, and malicious instructions could potentially leverage these tools in unintended ways. See Shared responsibility model section for more.
- **Authorized user attacks** – Users who are authorized to operate within the AWS account boundary or connected tools have a higher chance of attempting an attack against the agent. These users may have the ability to modify data sources that the agent consumes, such as logs or resource tags, making it easier to embed malicious instructions that the agent will process.

To mitigate these risks:

1. Carefully review and test custom MCP servers before deploying them in Agent Spaces.
   a. Ensure they are only permitted to perform read-only actions
   b. Verify that users of external tools accessed by MCP servers are trusted entities, as AWS DevOps Agents interfacing with MCP rely on the implicit trust relationship established between these tool users and the AWS DevOps Agent
2. Apply the principle of least privilege when granting users access to systems that provide data to the agent
3. Regularly audit which MCP servers are connected to your Agent Spaces
4. Since any content retrieved from allowlisted URLs could attempt to manipulate the agent's behavior, only include trusted sources in your allowlist.

Integration security

AWS DevOps Agent supports several integration types, each with its own security model:

- **Native bidirectional integrations** – Built-in integrations that can send data to the agent and receive updates from the agent. This uses the vendor’s authentication methods
- **MCP servers** – Remote Model Context Protocol servers that utilize OAuth 2.0 authentication flows and API keys to securely communicate with external systems.
- **Webhook triggers** – Investigation triggers from remote services such as tickets or observability systems. Webhooks use Hash-based Message Authentication Code (HMAC) for security.
- **Outbound communication** – Integrations like Slack and ticketing systems receive updates from the agent but do not yet support bidirectional communication.

Registration providers

Some external tools are authenticated at the account level and shared among all Agent Spaces in the account. When you register these tools, you authenticate once at the account level, and then each Agent Space can connect to specific resources within that registered connection.

The following tools use account-level registration:

- **GitHub** – Uses OAuth flow for authentication. After registering GitHub at the account level, each Agent Space can connect to specific repositories within your GitHub organization.

- **Dynatrace** – Uses OAuth token authentication. After registering Dynatrace at the account level, each Agent Space can connect to specific Dynatrace environments or monitoring configurations.
- **Slack** – Uses OAuth token authentication. After registering Slack at the account level, each Agent Space can connect to specific Slack channels channels.
- **Datadog** – Uses MCP with OAuth flow for authentication. After registering Datadog at the account level, each Agent Space can connect to specific Datadog monitoring resources.
- **New Relic** – Uses API key authentication. After registering New Relic at the account level, each Agent Space can connect to specific New Relic monitoring configurations.
- **Splunk** – Uses bearer token authentication. After registering Splunk at the account level, each Agent Space can connect to specific Splunk data sources.
- **GitLab** – Uses access token authentication. After registering GitLab at the account level, each Agent Space can connect to specific GitLab repositories.
- **ServiceNow** – Uses OAuth client key/token authentication. After registering ServiceNow at the account level, each Agent Space can connect to specific ServiceNow instances or ticket queues.
- **General public accessible remote MCP servers** – Use OAuth flow for authentication. After registering a remote MCP server at the account level, each Agent Space can connect to specific resources exposed by that server.

Network connectivity

AWS DevOps Agent connects to your third-party systems and remote MCP servers to perform investigations and other operations.

Inbound traffic from AWS DevOps Agent to your systems

AWS DevOps Agent initiates outbound connections to your third-party systems and remote MCP servers, which arrive as inbound traffic to your infrastructure. How you secure this traffic depends on how your tools are hosted:

- **Privately hosted tools** – If your tools are reachable from within an AWS VPC, you can use AWS DevOps Agent *private connections* to keep traffic isolated to AWS networks, and off of the public internet. For more information, see the section called “Connecting to privately hosted tools”.
- **Publicly hosted tools** – If your tools are reachable over the public internet and use IP allowlisting or firewall rules, you must allow inbound traffic from the following AWS DevOps Agent source IP addresses:

- Asia Pacific (Sydney) (ap-southeast-2)
  - `13.237.95.197`
  - `13.238.84.102`
- Asia Pacific (Tokyo) (ap-northeast-1)
  - `13.192.12.233`
  - `35.74.181.230`
  - `57.183.50.158`
- Europe (Frankfurt) (eu-central-1)
  - `18.158.110.140`
  - `52.57.96.160`
  - `52.59.55.56`
- Europe (Ireland) (eu-west-1)
  - `34.251.85.24`
  - `52.30.157.157`
  - `52.51.192.222`
- US East (N. Virginia) (us-east-1)
  - `34.228.181.128`
  - `44.219.176.187`
  - `54.226.244.221`
- US West (Oregon) (us-west-2)
  - `34.212.16.133`
  - `52.89.67.212`
  - `54.187.135.61`

Outbound traffic from your VPC to AWS DevOps Agent

For outbound traffic from your AWS VPC to AWS DevOps Agent (for example, using the section called “Invoking DevOps Agent through Webhook”), you can use VPC Endpoints to keep this network traffic isolated to AWS networks. For more information, see the section called “VPC Endpoints (AWS PrivateLink)”.

Shared responsibility model

AWS responsibilities

AWS is responsible for:

- Maintaining the security of data retrieved by the agent
- Securing native tools available for use by the agent
- Protecting the infrastructure that runs AWS DevOps Agent

Customer responsibilities

Customers are responsible for:

- Managing user access to the agent space
- Limiting access to trusted users of external systems that provide inputs to the agent, such as services and resources that produce logs, CloudTrail events, tickets, and more – that may be used to attempt malicious prompt injection.
- Ensure all connected data sources have trusted data that is unlikely to be used to attempt prompt injection attacks
- Ensuring bring-your-own MCP server integrations operate securely
- Ensuring IAM roles assigned to the agent are properly scoped
- Redacting PII data before storing in observability logs and other agent data sources
- Following the recommended practice of granting only read-only permissions to connected data sources, including bring-your-own MCP servers

Data usage

AWS does not use agent data, chat messages, or data from integrated data sources to train models or improve the product. The AWS DevOps Agent Space uses customer in-product feedback to improve the agent's responses and investigations, but AWS does not use it to improve the service itself.

Compliance

At preview, AWS DevOps Agent is not compliant with standards including SOC 2, PCI-DSS, ISO 27001, or FedRAMP. AWS will announce which compliance certifications will be available at a later time.

DevOps Agent IAM permissions

AWS DevOps Agent uses service-specific AWS Identity and Access Management (IAM) actions to control access to its features and capabilities. These actions determine what users can do within the AWS DevOps Agent console and Operator Web App. This is separate from the AWS service API permissions that the agent itself uses to investigate your resources.

For more information about limiting agent access, see Limiting Agent Access in an AWS Account.

Agent Space management actions

These actions control access to Agent Space configuration and management:

- **aidevops:GetAgentSpace** – Allows users to view details about an Agent Space, including its configuration, status, and associated accounts. Users need this permission to access an Agent Space in the AWS Management Console.
- **aidevops:GetAssociation** – Allows users to view details about a specific account association, including the IAM role configuration and connection status.
- **aidevops:ListAssociations** – Allows users to list all AWS account associations configured for an Agent Space, including both primary and secondary accounts.

Investigation and execution actions

These actions control access to incident investigation features:

- **aidevops:ListExecutions** – Allows users to view execution metadata—including ID, status, and more—for investigations, mitigations, evaluations, and chat conversations associated with a task.
- **aidevops:ListJournalRecords** – Allows users to access detailed logs that show the agent's reasoning steps, actions taken, and data sources consulted during an investigation, mitigation, evaluation, and chat conversation. This is useful for understanding how the agent reached its conclusions.

Chat management actions

Chat requires the following IAM permissions to function:

- **aidevops:ListChats** – Allows users to list and access chat conversation history.
- **aidevops:CreateChat** – Allows users to create new chat conversations.
- **aidevops:SendMessage** – Allows users to submit queries and receive streaming responses.

Topology and discovery actions

These actions control access to application resource mapping features:

- **aidevops:DiscoverTopology** – Allows users to trigger topology discovery and mapping for an Agent Space. This action initiates the process of scanning AWS accounts and building the application resource topology.

Prevention and recommendation actions

These actions control access to the Prevention feature:

- **aidevops:ListGoals** – Allows users to view prevention goals and objectives that the agent is working toward based on recent incident patterns.
- **aidevops:ListRecommendations** – Allows users to view all recommendations generated by the Prevention feature, including their priority and category.
- **aidevops:GetRecommendation** – Allows users to view detailed information about a specific recommendation, including the incidents it would have prevented and implementation guidance.

Backlog task management actions

These actions control the ability to manage recommendations as backlog tasks:

- **aidevops:CreateBacklogTask** – Allows users to create an incident investigation or prevention evaluation task.
- **aidevops:UpdateBacklogTask** – Allows users to approve a mitigation plan or cancel an active investigation or evaluation.
- **aidevops:GetBacklogTask** – Allows users to retrieve details about a specific task.

- **aidevops:ListBacklogTasks** – Allows users to list tasks for an Agent Space, filtered by task type, status, priority, or creation time.

Knowledge management actions

These actions control the ability to add and manage custom knowledge that the agent can use during investigations:

- **aidevops:CreateKnowledgeItem** – Allows users to add custom knowledge items, such as skills, troubleshooting guides, or application-specific information that the agent should reference.
- **aidevops:ListKnowledgeItems** – Allows users to view all knowledge items configured for an Agent Space.
- **aidevops:GetKnowledgeItem** – Allows users to retrieve the details of a specific knowledge item.
- **aidevops:UpdateKnowledgeItem** – Allows users to modify existing knowledge items to keep information current.
- **aidevops:DeleteKnowledgeItem** – Allows users to remove knowledge items that are no longer relevant.

AWS Support integration actions

These actions control integration with AWS Support cases:

- **aidevops:InitiateChatForCase** – Allows users to start a chat session with AWS Support directly from an investigation, automatically providing context about the incident.
- **aidevops:EndChatForCase** – Allows users to end an active AWS Support case chat session.
- **aidevops:DescribeSupportLevel** – Allows users to check the AWS Support plan level for the account to determine available support options.

Usage and monitoring actions

These actions control access to usage information:

- **aidevops:GetAccountUsage** – Allows users to view the AWS DevOps Agent monthly quota for investigation hours, prevention evaluation hours, and chat requests, as well as the current month's usage.

Common IAM policy examples

Administrator policy

This policy grants full access to all AWS DevOps Agent features:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "aidevops:*",
      "Resource": "*"
    }
  ]
}
```

Operator policy

This policy grants access to investigation and prevention features without administrative capabilities:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "aidevops:GetAgentSpace",
        "aidevops:InvokeAgent",
        "aidevops:ListExecutions",
        "aidevops:ListJournalRecords",
        "aidevops:ListAssociations",
        "aidevops:GetAssociation",
        "aidevops:DiscoverTopology",
        "aidevops:ListRecommendations",
        "aidevops:GetRecommendation",
        "aidevops:CreateBacklogTask",
        "aidevops:UpdateBacklogTask",
        "aidevops:GetBacklogTask",
        "aidevops:ListBacklogTasks",
        "aidevops:ListKnowledgeItems",
        "aidevops:GetKnowledgeItem",
```

```
        "aidevops:InitiateChatForCase",
        "aidevops:EndChatForCase",
        "aidevops:ListChats",
        "aidevops:CreateChat",
        "aidevops:SendMessage",
        "aidevops:ListGoals",
        "aidevops:CreateKnowledgeItem",
        "aidevops:UpdateKnowledgeItem",
        "aidevops:DescribeSupportLevel",
        "aidevops:ListPendingMessages"
      ],
      "Resource": "*"
    }
  ]
}
```

Read-only policy

This policy grants view-only access to investigations and recommendations:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "aidevops:GetAgentSpace",
        "aidevops:ListAssociations",
        "aidevops:GetAssociation",
        "aidevops:ListExecutions",
        "aidevops:ListJournalRecords",
        "aidevops:ListRecommendations",
        "aidevops:GetRecommendation",
        "aidevops:ListBacklogTasks",
        "aidevops:GetBacklogTask",
        "aidevops:ListKnowledgeItems",
        "aidevops:GetKnowledgeItem",
        "aidevops:GetAccountUsage"
      ],
      "Resource": "*"
    }
  ]
}
```

Using service-linked roles for AWS DevOps Agent

AWS DevOps Agent 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 AWS DevOps Agent. Service-linked roles are predefined by AWS DevOps Agent and include all the permissions that the service requires to call other AWS services on your behalf.

Service-linked role permissions

The `AWSServiceRoleForAIDevOps` service-linked role trusts the `aidevops.amazonaws.com` service principal to assume the role.

The role uses the managed policy `AWSServiceRoleForAIDevOpsPolicy` with the following permissions:

- `cloudwatch:PutMetricData` – Publish usage metrics to the `AWS/AIDevOps` CloudWatch namespace. Scoped by a `cloudwatch:namespace` condition to only allow the `AWS/AIDevOps` namespace.
- `vpc-lattice:CreateResourceGateway` – Create VPC Lattice resource gateways for private connections. Scoped by an `aws:RequestTag/AWSAIDevOpsManaged` condition so the service can only create resource gateways that carry the `AWSAIDevOpsManaged` tag.
- `vpc-lattice:TagResource` – Tag VPC Lattice resource gateways. Scoped by an `aws:RequestTag/AWSAIDevOpsManaged` condition.
- `vpc-lattice:DeleteResourceGateway` – Delete VPC Lattice resource gateways. Scoped by an `aws:ResourceTag/AWSAIDevOpsManaged` condition so the service can only delete resource gateways it created.
- `vpc-lattice:GetResourceGateway` – Retrieve information about VPC Lattice resource gateways. Scoped by an `aws:ResourceTag/AWSAIDevOpsManaged` condition so the service can only read resource gateways it created.
- `ec2:DescribeVpcs`, `ec2:DescribeSubnets`, `ec2:DescribeSecurityGroups` – Retrieve information about VPC networking resources required to configure resource gateways. These read-only actions apply to all VPC resources because the EC2 API does not support resource-level permissions for Describe calls.
- `iam:CreateServiceLinkedRole` – Create the VPC Lattice service-linked role required for resource gateway operations. This permission is scoped to the `vpc-lattice.amazonaws.com` service principal only and cannot be used to create service-linked roles for any other service.

Creating the service-linked role

You don't need to manually create the `AWSServiceRoleForAIDevOps` service-linked role. When you start using AWS DevOps Agent, the service creates the service-linked role for you.

To allow the service to create the role on your behalf, you must have the `iam:CreateServiceLinkedRole` permission. We recommend scoping this permission with an `iam:AWSServiceName` condition for `aidevops.amazonaws.com` to follow the principle of least privilege. For more information, see Service-linked role permissions.

Editing the service-linked role

You cannot edit the `AWSServiceRoleForAIDevOps` service-linked role. After the role is created, you cannot change the name of the role because various entities might reference the role by name. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role.

Deleting the service-linked role

If you no longer need to use AWS DevOps Agent, we recommend that you delete the `AWSServiceRoleForAIDevOps` service-linked role. Before you can delete the role, you must first remove any private connections configured in your Agent Space. Deleting the service-linked role does not automatically remove VPC Lattice resource gateways tagged with `AWSAIDevOpsManaged` that were previously created by the service. You should delete these resource gateways manually if they are no longer needed. For more information, see Deleting a service-linked role.

AWS Managed policies for AWS DevOps Agent

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 AWS DevOps Agent.

AIDevOpsAgentReadOnlyAccess

Provides read only access to Amazon DevOps Agent via the AWS Management Console

```
{
```

```
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AIDevOpsAgentReadOnlyAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:Get*",
    "aidevops:List*",
    "aidevops:SearchServiceAccessibleResource"
   ],
   "Resource": "*"
  }
 ]
}
```

AIDevOpsAgentFullAccess

Provides full access to Amazon DevOps Agent via the AWS Management Console

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AIDevOpsAgentSpaceAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:CreateAgentSpace",
    "aidevops:DeleteAgentSpace",
    "aidevops:GetAgentSpace",
    "aidevops:ListAgentSpaces",
    "aidevops:UpdateAgentSpace"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AIDevOpsServiceAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:DeregisterService",
    "aidevops:GetService",
    "aidevops:ListServices",
    "aidevops:RegisterService",
    "aidevops:SearchServiceAccessibleResource"
```

```
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsAssociationAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:AssociateService",
     "aidevops:DisassociateService",
     "aidevops:GetAssociation",
     "aidevops:ListAssociations",
     "aidevops:UpdateAssociation",
     "aidevops:ValidateAwsAssociations"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsWebhookAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:ListWebhooks"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsOperatorAppAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:DisableOperatorApp",
     "aidevops:EnableOperatorApp",
     "aidevops:GetOperatorApp",
     "aidevops:UpdateOperatorAppIdpConfig"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsKnowledgeAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:CreateKnowledgeItem",
     "aidevops:DeleteKnowledgeItem",
     "aidevops:GetKnowledgeItem",
     "aidevops:ListKnowledgeItems",
     "aidevops:ListKnowledgeItemVersions",
```

```
    "aidevops:UpdateKnowledgeItem"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AIDevOpsBacklogAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:CreateBacklogTask",
    "aidevops:GetBacklogTask",
    "aidevops:ListBacklogTasks",
    "aidevops:ListGoals",
    "aidevops:UpdateBacklogTask",
    "aidevops:UpdateGoal"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AIDevOpsRecommendationAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:GetRecommendation",
    "aidevops:ListRecommendations",
    "aidevops:UpdateRecommendation"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AIDevOpsAgentChatAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:CreateChat",
    "aidevops:ListChats",
    "aidevops:ListPendingMessages",
    "aidevops:SendMessage"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AIDevOpsJournalAccess",
   "Effect": "Allow",
   "Action": [
    "aidevops:ListExecutions",
    "aidevops:ListJournalRecords"
```

```
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsTopologyAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:DiscoverTopology"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsSupportAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:DescribeSupportLevel",
     "aidevops:EndChatForCase",
     "aidevops:InitiateChatForCase"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsUsageAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:GetAccountUsage"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsTaggingAccess",
    "Effect": "Allow",
    "Action": [
     "aidevops:ListTagsForResource",
     "aidevops:TagResource",
     "aidevops:UntagResource"
    ],
    "Resource": "*"
   },
   {
    "Sid": "AIDevOpsVendedLogs",
    "Effect": "Allow",
    "Action": [
     "aidevops:AllowVendedLogDeliveryForResource"
```

```
    ],
    "Resource": "*"
   }
  ]
}
```

AIDevOpsOperatorAppAccessPolicy

Provides access to use the AWS DevOps operator web app for an Agent Space.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowOperatorAgentSpaceActions",
   "Effect": "Allow",
   "Action": [
    "aidevops:GetAgentSpace",
    "aidevops:GetAssociation",
    "aidevops:ListAssociations",
    "aidevops:CreateBacklogTask",
    "aidevops:GetBacklogTask",
    "aidevops:UpdateBacklogTask",
    "aidevops:ListBacklogTasks",
    "aidevops:ListJournalRecords",
    "aidevops:DiscoverTopology",
    "aidevops:ListGoals",
    "aidevops:ListRecommendations",
    "aidevops:ListExecutions",
    "aidevops:GetRecommendation",
    "aidevops:UpdateRecommendation",
    "aidevops:CreateKnowledgeItem",
    "aidevops:ListKnowledgeItems",
    "aidevops:ListKnowledgeItemVersions",
    "aidevops:GetKnowledgeItem",
    "aidevops:UpdateKnowledgeItem",
    "aidevops:DeleteKnowledgeItem",
    "aidevops:ListPendingMessages",
    "aidevops:InitiateChatForCase",
    "aidevops:EndChatForCase",
    "aidevops:DescribeSupportLevel",
    "aidevops:ListChats",
    "aidevops:CreateChat",
```

```
     "aidevops:SendMessage"
    ],
    "Resource": "arn:aws:aidevops:*:*:agentspace/${aws:PrincipalTag/AgentSpaceId}",
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   },
   {
    "Sid": "AllowOperatorAccountActions",
    "Effect": "Allow",
    "Action": [
     "aidevops:GetAccountUsage"
    ],
    "Resource": "*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   },
   {
    "Sid": "AllowSupportOperatorActions",
    "Effect": "Allow",
    "Action": [
     "support:DescribeCases",
     "support:InitiateChatForCase",
     "support:DescribeSupportLevel"
    ],
    "Resource": "*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   }
  ]
}
```

AIDevOpsAgentAccessPolicy

Provides permissions required by the AWS DevOps Agent to conduct investigations and perform analysis on customer AWS resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AIOPSServiceAccess",
            "Effect": "Allow",
            "Action": [
                "access-analyzer:GetAnalyzer",
                "access-analyzer:List*",
                "acm-pca:Describe*",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:GetCertificateAuthorityCsr",
                "acm-pca:List*",
                "acm:DescribeCertificate",
                "acm:GetAccountConfiguration",
                "aidevops:GetKnowledgeItem",
                "aidevops:ListKnowledgeItems",
                "airflow:List*",
                "amplify:GetApp",
                "amplify:GetBranch",
                "amplify:GetDomainAssociation",
                "amplify:List*",
                "aoss:BatchGetCollection",
                "aoss:BatchGetLifecyclePolicy",
                "aoss:BatchGetVpcEndpoint",
                "aoss:GetAccessPolicy",
                "aoss:GetSecurityConfig",
                "aoss:GetSecurityPolicy",
                "aoss:List*",
                "appconfig:GetApplication",
                "appconfig:GetConfigurationProfile",
                "appconfig:GetEnvironment",
                "appconfig:GetHostedConfigurationVersion",
                "appconfig:List*",
                "appflow:Describe*",
                "appflow:List*",
                "application-autoscaling:Describe*",
```

```
                "application-signals:BatchGetServiceLevelObjectiveBudgetReport",
                "application-signals:GetService",
                "application-signals:GetServiceLevelObjective",
                "application-signals:List*",
                "applicationinsights:Describe*",
                "applicationinsights:List*",
                "apprunner:Describe*",
                "apprunner:List*",
                "appstream:Describe*",
                "appstream:List*",
                "appsync:GetApiAssociation",
                "appsync:GetDataSource",
                "appsync:GetDomainName",
                "appsync:GetFunction",
                "appsync:GetGraphqlApi",
                "appsync:GetGraphqlApiEnvironmentVariables",
                "appsync:GetIntrospectionSchema",
                "appsync:GetResolver",
                "appsync:GetSourceApiAssociation",
                "appsync:List*",
                "aps:Describe*",
                "aps:List*",
                "arc-zonal-shift:GetManagedResource",
                "arc-zonal-shift:List*",
                "athena:GetCapacityAssignmentConfiguration",
                "athena:GetCapacityReservation",
                "athena:GetDataCatalog",
                "athena:GetNamedQuery",
                "athena:GetPreparedStatement",
                "athena:GetWorkGroup",
                "athena:List*",
                "auditmanager:GetAssessment",
                "auditmanager:List*",
                "autoscaling:Describe*",
                "backup-gateway:GetHypervisor",
                "backup-gateway:List*",
                "backup:Describe*",
                "backup:GetBackupPlan",
                "backup:GetBackupSelection",
                "backup:GetBackupVaultAccessPolicy",
                "backup:GetBackupVaultNotifications",
                "backup:GetRestoreTestingPlan",
                "backup:GetRestoreTestingSelection",
                "backup:List*",
```

```
                "batch:DescribeComputeEnvironments",
                "batch:DescribeJobQueues",
                "batch:DescribeSchedulingPolicies",
                "batch:List*",
                "bedrock:GetAgent",
                "bedrock:GetAgentActionGroup",
                "bedrock:GetAgentAlias",
                "bedrock:GetAgentKnowledgeBase",
                "bedrock:GetDataSource",
                "bedrock:GetGuardrail",
                "bedrock:GetKnowledgeBase",
                "bedrock:List*",
                "budgets:Describe*",
                "budgets:List*",
                "ce:Describe*",
                "ce:GetAnomalyMonitors",
                "ce:GetAnomalySubscriptions",
                "ce:List*",
                "chatbot:Describe*",
                "chatbot:GetMicrosoftTeamsChannelConfiguration",
                "chatbot:List*",
                "cleanrooms-ml:GetTrainingDataset",
                "cleanrooms-ml:List*",
                "cleanrooms:GetAnalysisTemplate",
                "cleanrooms:GetCollaboration",
                "cleanrooms:GetConfiguredTable",
                "cleanrooms:GetConfiguredTableAnalysisRule",
                "cleanrooms:GetConfiguredTableAssociation",
                "cleanrooms:GetMembership",
                "cleanrooms:List*",
                "cloudformation:Describe*",
                "cloudformation:GetResource",
                "cloudformation:GetStackPolicy",
                "cloudformation:GetTemplate",
                "cloudformation:List*",
                "cloudfront:Describe*",
                "cloudfront:GetCachePolicy",
                "cloudfront:GetCloudFrontOriginAccessIdentity",
                "cloudfront:GetContinuousDeploymentPolicy",
                "cloudfront:GetDistribution",
                "cloudfront:GetDistributionConfig",
                "cloudfront:GetFunction",
                "cloudfront:GetKeyGroup",
                "cloudfront:GetMonitoringSubscription",
```

```
                "cloudfront:GetOriginAccessControl",
                "cloudfront:GetOriginRequestPolicy",
                "cloudfront:GetPublicKey",
                "cloudfront:GetRealtimeLogConfig",
                "cloudfront:GetResponseHeadersPolicy",
                "cloudfront:List*",
                "cloudtrail:Describe*",
                "cloudtrail:GetChannel",
                "cloudtrail:GetEventConfiguration",
                "cloudtrail:GetEventDataStore",
                "cloudtrail:GetEventSelectors",
                "cloudtrail:GetInsightSelectors",
                "cloudtrail:GetQueryResults",
                "cloudtrail:GetResourcePolicy",
                "cloudtrail:GetTrail",
                "cloudtrail:GetTrailStatus",
                "cloudtrail:List*",
                "cloudtrail:LookupEvents",
                "cloudtrail:StartQuery",
                "cloudwatch:Describe*",
                "cloudwatch:GenerateQuery",
                "cloudwatch:GetDashboard",
                "cloudwatch:GetInsightRuleReport",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:GetMetricStream",
                "cloudwatch:GetService",
                "cloudwatch:GetServiceLevelObjective",
                "cloudwatch:List*",
                "codeartifact:Describe*",
                "codeartifact:GetDomainPermissionsPolicy",
                "codeartifact:GetRepositoryPermissionsPolicy",
                "codeartifact:List*",
                "codebuild:BatchGetFleets",
                "codebuild:List*",
                "codecommit:GetRepository",
                "codecommit:GetRepositoryTriggers",
                "codedeploy:BatchGetDeployments",
                "codedeploy:BatchGetDeploymentTargets",
                "codedeploy:GetApplication",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:GetDeploymentTarget",
                "codedeploy:List*",
                "codeguru-profiler:Describe*",
```

```
                "codeguru-profiler:GetNotificationConfiguration",
                "codeguru-profiler:GetPolicy",
                "codeguru-profiler:List*",
                "codeguru-reviewer:Describe*",
                "codeguru-reviewer:List*",
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:List*",
                "codestar-connections:GetConnection",
                "codestar-connections:GetRepositoryLink",
                "codestar-connections:GetSyncConfiguration",
                "codestar-connections:List*",
                "codestar-notifications:Describe*",
                "codestar-notifications:List*",
                "cognito-identity:DescribeIdentityPool",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:ListIdentityPools",
                "cognito-identity:ListTagsForResource",
                "cognito-idp:AdminListGroupsForUser",
                "cognito-idp:DescribeIdentityProvider",
                "cognito-idp:DescribeResourceServer",
                "cognito-idp:DescribeRiskConfiguration",
                "cognito-idp:DescribeUserImportJob",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolDomain",
                "cognito-idp:GetGroup",
                "cognito-idp:GetLogDeliveryConfiguration",
                "cognito-idp:GetUICustomization",
                "cognito-idp:GetUserPoolMfaConfig",
                "cognito-idp:GetWebACLForResource",
                "cognito-idp:ListGroups",
                "cognito-idp:ListIdentityProviders",
                "cognito-idp:ListResourceServers",
                "cognito-idp:ListUserPoolClients",
                "cognito-idp:ListUserPools",
                "cognito-idp:ListTagsForResource",
                "comprehend:Describe*",
                "comprehend:List*",
                "config:Describe*",
                "config:GetStoredQuery",
                "config:List*",
                "connect:Describe*",
                "connect:GetTaskTemplate",
                "connect:List*",
```

```
                "databrew:Describe*",
                "databrew:List*",
                "datapipeline:Describe*",
                "datapipeline:GetPipelineDefinition",
                "datapipeline:List*",
                "datasync:Describe*",
                "datasync:List*",
                "deadline:GetFarm",
                "deadline:GetFleet",
                "deadline:GetLicenseEndpoint",
                "deadline:GetMonitor",
                "deadline:GetQueue",
                "deadline:GetQueueEnvironment",
                "deadline:GetQueueFleetAssociation",
                "deadline:GetStorageProfile",
                "deadline:List*",
                "detective:GetMembers",
                "detective:List*",
                "devicefarm:GetDevicePool",
                "devicefarm:GetInstanceProfile",
                "devicefarm:GetNetworkProfile",
                "devicefarm:GetProject",
                "devicefarm:GetTestGridProject",
                "devicefarm:GetVPCEConfiguration",
                "devicefarm:List*",
                "devops-guru:Describe*",
                "devops-guru:GetResourceCollection",
                "devops-guru:List*",
                "dms:Describe*",
                "dms:List*",
                "ds:Describe*",
                "dynamodb:Describe*",
                "dynamodb:GetResourcePolicy",
                "dynamodb:List*",
                "ec2:Describe*",
                "ec2:GetAssociatedEnclaveCertificateIamRoles",
                "ec2:GetIpamPoolAllocations",
                "ec2:GetIpamPoolCidrs",
                "ec2:GetManagedPrefixListEntries",
                "ec2:GetNetworkInsightsAccessScopeContent",
                "ec2:GetSnapshotBlockPublicAccessState",
                "ec2:GetTransitGatewayMulticastDomainAssociations",
                "ec2:GetTransitGatewayRouteTableAssociations",
                "ec2:GetTransitGatewayRouteTablePropagations",
```

```
                "ec2:GetVerifiedAccessEndpointPolicy",
                "ec2:GetVerifiedAccessGroupPolicy",
                "ec2:GetVerifiedAccessInstanceWebAcl",
                "ec2:SearchLocalGatewayRoutes",
                "ec2:SearchTransitGatewayRoutes",
                "ecr:Describe*",
                "ecr:GetLifecyclePolicy",
                "ecr:GetRegistryPolicy",
                "ecr:GetRepositoryPolicy",
                "ecr:List*",
                "ecs:Describe*",
                "ecs:List*",
                "eks:AccessKubernetesApi",
                "eks:Describe*",
                "eks:List*",
                "elasticache:Describe*",
                "elasticache:List*",
                "elasticbeanstalk:Describe*",
                "elasticbeanstalk:List*",
                "elasticfilesystem:Describe*",
                "elasticloadbalancing:GetResourcePolicy",
                "elasticloadbalancing:GetTrustStoreCaCertificatesBundle",
                "elasticloadbalancing:GetTrustStoreRevocationContent",
                "elasticloadbalancing:Describe*",
                "elasticmapreduce:Describe*",
                "elasticmapreduce:List*",
                "emr-containers:Describe*",
                "emr-containers:List*",
                "emr-serverless:GetApplication",
                "emr-serverless:List*",
                "es:Describe*",
                "es:List*",
                "events:Describe*",
                "events:List*",
                "evidently:GetExperiment",
                "evidently:GetFeature",
                "evidently:GetLaunch",
                "evidently:GetProject",
                "evidently:GetSegment",
                "evidently:List*",
                "firehose:Describe*",
                "firehose:List*",
                "fis:GetExperimentTemplate",
                "fis:GetTargetAccountConfiguration",
```

```
                "fis:List*",
                "fms:GetNotificationChannel",
                "fms:GetPolicy",
                "fms:List*",
                "forecast:Describe*",
                "forecast:List*",
                "frauddetector:BatchGetVariable",
                "frauddetector:Describe*",
                "frauddetector:GetDetectors",
                "frauddetector:GetDetectorVersion",
                "frauddetector:GetEntityTypes",
                "frauddetector:GetEventTypes",
                "frauddetector:GetExternalModels",
                "frauddetector:GetLabels",
                "frauddetector:GetListElements",
                "frauddetector:GetListsMetadata",
                "frauddetector:GetModelVersion",
                "frauddetector:GetOutcomes",
                "frauddetector:GetRules",
                "frauddetector:GetVariables",
                "frauddetector:List*",
                "fsx:Describe*",
                "gamelift:Describe*",
                "gamelift:List*",
                "globalaccelerator:Describe*",
                "globalaccelerator:List*",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetJob",
                "glue:GetRegistry",
                "glue:GetSchema",
                "glue:GetSchemaVersion",
                "glue:GetTable",
                "glue:GetTags",
                "glue:GetTrigger",
                "glue:List*",
                "glue:querySchemaVersionMetadata",
                "grafana:Describe*",
                "grafana:List*",
                "greengrass:Describe*",
                "greengrass:GetDeployment",
                "greengrass:List*",
                "groundstation:GetConfig",
                "groundstation:GetDataflowEndpointGroup",
```

```
                "groundstation:GetMissionProfile",
                "groundstation:List*",
                "guardduty:GetDetector",
                "guardduty:GetFilter",
                "guardduty:GetIPSet",
                "guardduty:GetMalwareProtectionPlan",
                "guardduty:GetMasterAccount",
                "guardduty:GetMembers",
                "guardduty:GetThreatIntelSet",
                "guardduty:List*",
                "health:DescribeEvents",
                "health:DescribeEventDetails",
                "healthlake:Describe*",
                "healthlake:List*",
                "iam:GetGroup",
                "iam:GetGroupPolicy",
                "iam:GetInstanceProfile",
                "iam:GetLoginProfile",
                "iam:GetOpenIDConnectProvider",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:GetSAMLProvider",
                "iam:GetServerCertificate",
                "iam:GetServiceLinkedRoleDeletionStatus",
                "iam:GetUser",
                "iam:GetUserPolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListOpenIDConnectProviders",
                "iam:ListRolePolicies",
                "iam:ListRoles",
                "iam:ListServerCertificates",
                "iam:ListVirtualMFADevices",
                "identitystore:DescribeGroup",
                "identitystore:DescribeGroupMembership",
                "identitystore:ListGroupMemberships",
                "identitystore:ListGroups",
                "imagebuilder:GetComponent",
                "imagebuilder:GetContainerRecipe",
                "imagebuilder:GetDistributionConfiguration",
                "imagebuilder:GetImage",
                "imagebuilder:GetImagePipeline",
                "imagebuilder:GetImageRecipe",
```

```
                "imagebuilder:GetInfrastructureConfiguration",
                "imagebuilder:GetLifecyclePolicy",
                "imagebuilder:GetWorkflow",
                "imagebuilder:List*",
                "inspector2:List*",
                "inspector:Describe*",
                "inspector:List*",
                "internetmonitor:GetMonitor",
                "internetmonitor:List*",
                "iot:Describe*",
                "iot:GetPackage",
                "iot:GetPackageVersion",
                "iot:GetPolicy",
                "iot:GetThingShadow",
                "iot:GetTopicRule",
                "iot:GetTopicRuleDestination",
                "iot:GetV2LoggingOptions",
                "iot:List*",
                "iotanalytics:Describe*",
                "iotanalytics:List*",
                "iotevents:Describe*",
                "iotevents:List*",
                "iotsitewise:Describe*",
                "iotsitewise:List*",
                "iotwireless:GetDestination",
                "iotwireless:GetDeviceProfile",
                "iotwireless:GetFuotaTask",
                "iotwireless:GetMulticastGroup",
                "iotwireless:GetNetworkAnalyzerConfiguration",
                "iotwireless:GetServiceProfile",
                "iotwireless:GetWirelessDevice",
                "iotwireless:GetWirelessGateway",
                "iotwireless:GetWirelessGatewayTaskDefinition",
                "iotwireless:List*",
                "ivs:GetChannel",
                "ivs:GetEncoderConfiguration",
                "ivs:GetPlaybackRestrictionPolicy",
                "ivs:GetRecordingConfiguration",
                "ivs:GetStage",
                "ivs:List*",
                "ivschat:GetLoggingConfiguration",
                "ivschat:GetRoom",
                "ivschat:List*",
                "kafka:Describe*",
```

```
                "kafka:GetClusterPolicy",
                "kafka:List*",
                "kafkaconnect:Describe*",
                "kafkaconnect:List*",
                "kendra:Describe*",
                "kendra:List*",
                "kinesis:Describe*",
                "kinesis:GetResourcePolicy",
                "kinesis:List*",
                "kinesisanalytics:Describe*",
                "kinesisanalytics:List*",
                "kinesisvideo:Describe*",
                "kms:DescribeKey",
                "kms:ListResourceTags",
                "kms:ListKeys",
                "kms:GetKeyPolicy",
                "kms:GetKeyRotationStatus",
                "kms:ListAliases",
                "kms:ListKeyRotations",
                "lakeformation:Describe*",
                "lakeformation:GetLFTag",
                "lakeformation:GetResourceLFTags",
                "lakeformation:List*",
                "lambda:GetAlias",
                "lambda:GetCodeSigningConfig",
                "lambda:GetEventSourceMapping",
                "lambda:GetFunctionCodeSigningConfig",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunctionEventInvokeConfig",
                "lambda:GetFunctionRecursionConfig",
                "lambda:GetFunctionUrlConfig",
                "lambda:GetLayerVersion",
                "lambda:GetLayerVersionPolicy",
                "lambda:GetPolicy",
                "lambda:GetProvisionedConcurrencyConfig",
                "lambda:GetRuntimeManagementConfig",
                "lambda:List*",
                "launchwizard:GetDeployment",
                "launchwizard:List*",
                "license-manager:GetLicense",
                "license-manager:List*",
                "lightsail:GetAlarms",
                "lightsail:GetBuckets",
                "lightsail:GetCertificates",
```

```
                "lightsail:GetContainerServices",
                "lightsail:GetDisk",
                "lightsail:GetDisks",
                "lightsail:GetInstance",
                "lightsail:GetInstances",
                "lightsail:GetLoadBalancer",
                "lightsail:GetLoadBalancers",
                "lightsail:GetLoadBalancerTlsCertificates",
                "lightsail:GetStaticIp",
                "lightsail:GetStaticIps",
                "logs:Describe*",
                "logs:FilterLogEvents",
                "logs:GetDataProtectionPolicy",
                "logs:GetDelivery",
                "logs:GetDeliveryDestination",
                "logs:GetDeliveryDestinationPolicy",
                "logs:GetDeliverySource",
                "logs:GetLogAnomalyDetector",
                "logs:GetLogDelivery",
                "logs:GetLogGroupFields",
                "logs:GetQueryResults",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopLiveTail",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "m2:GetApplication",
                "m2:GetEnvironment",
                "m2:List*",
                "macie2:GetAllowList",
                "macie2:GetCustomDataIdentifier",
                "macie2:GetFindingsFilter",
                "macie2:GetMacieSession",
                "macie2:List*",
                "mediaconnect:Describe*",
                "mediaconnect:List*",
                "medialive:Describe*",
                "medialive:GetCloudWatchAlarmTemplate",
                "medialive:GetCloudWatchAlarmTemplateGroup",
                "medialive:GetEventBridgeRuleTemplate",
                "medialive:GetEventBridgeRuleTemplateGroup",
                "medialive:GetSignalMap",
                "medialive:List*",
                "mediapackage-vod:Describe*",
```

```
                "mediapackage-vod:List*",
                "mediapackage:Describe*",
                "mediapackage:List*",
                "mediapackagev2:GetChannel",
                "mediapackagev2:GetChannelGroup",
                "mediapackagev2:GetChannelPolicy",
                "mediapackagev2:GetOriginEndpoint",
                "mediapackagev2:GetOriginEndpointPolicy",
                "mediapackagev2:List*",
                "memorydb:Describe*",
                "memorydb:List*",
                "mobiletargeting:GetInAppTemplate",
                "mobiletargeting:List*",
                "mq:Describe*",
                "mq:List*",
                "network-firewall:Describe*",
                "network-firewall:List*",
                "networkmanager:Describe*",
                "networkmanager:GetConnectAttachment",
                "networkmanager:GetConnectPeer",
                "networkmanager:GetCoreNetwork",
                "networkmanager:GetCoreNetworkPolicy",
                "networkmanager:GetCustomerGatewayAssociations",
                "networkmanager:GetDevices",
                "networkmanager:GetLinkAssociations",
                "networkmanager:GetLinks",
                "networkmanager:GetSites",
                "networkmanager:GetSiteToSiteVpnAttachment",
                "networkmanager:GetTransitGatewayPeering",
                "networkmanager:GetTransitGatewayRegistrations",
                "networkmanager:GetTransitGatewayRouteTableAttachment",
                "networkmanager:GetVpcAttachment",
                "networkmanager:List*",
                "oam:GetLink",
                "oam:GetSink",
                "oam:GetSinkPolicy",
                "oam:List*",
                "omics:GetAnnotationStore",
                "omics:GetReferenceStore",
                "omics:GetRunGroup",
                "omics:GetSequenceStore",
                "omics:GetVariantStore",
                "omics:GetWorkflow",
                "omics:List*",
```

```
                "organizations:Describe*",
                "organizations:List*",
                "osis:GetPipeline",
                "osis:List*",
                "payment-cryptography:GetAlias",
                "payment-cryptography:GetKey",
                "payment-cryptography:List*",
                "pca-connector-ad:GetConnector",
                "pca-connector-ad:GetDirectoryRegistration",
                "pca-connector-ad:GetServicePrincipalName",
                "pca-connector-ad:GetTemplate",
                "pca-connector-ad:GetTemplateGroupAccessControlEntry",
                "pca-connector-ad:List*",
                "pca-connector-scep:GetChallengeMetadata",
                "pca-connector-scep:GetConnector",
                "pca-connector-scep:List*",
                "personalize:Describe*",
                "personalize:List*",
                "pi:DescribeDimensionKeys",
                "pi:GetResourceMetadata",
                "pi:GetResourceMetrics",
                "pi:ListAvailableResourceDimensions",
                "pi:ListAvailableResourceMetrics",
                "pipes:Describe*",
                "pipes:List*",
                "proton:GetEnvironmentTemplate",
                "proton:GetServiceTemplate",
                "proton:List*",
                "qbusiness:GetApplication",
                "qbusiness:GetDataSource",
                "qbusiness:GetIndex",
                "qbusiness:GetPlugin",
                "qbusiness:GetRetriever",
                "qbusiness:GetWebExperience",
                "qbusiness:List*",
                "ram:GetPermission",
                "ram:GetResourceShares",
                "ram:List*",
                "rds:Describe*",
                "rds:List*",
                "redshift-serverless:GetNamespace",
                "redshift-serverless:GetWorkgroup",
                "redshift-serverless:List*",
                "redshift:Describe*",
```

```
                "refactor-spaces:GetApplication",
                "refactor-spaces:GetEnvironment",
                "refactor-spaces:GetRoute",
                "refactor-spaces:List*",
                "rekognition:Describe*",
                "rekognition:List*",
                "resiliencehub:Describe*",
                "resiliencehub:List*",
                "resource-explorer-2:GetDefaultView",
                "resource-explorer-2:GetIndex",
                "resource-explorer-2:GetView",
                "resource-explorer-2:List*",
                "resource-explorer-2:Search",
                "resource-groups:GetGroup",
                "resource-groups:GetGroupConfiguration",
                "resource-groups:GetGroupQuery",
                "resource-groups:GetTags",
                "resource-groups:List*",
                "route53-recovery-control-config:Describe*",
                "route53-recovery-control-config:List*",
                "route53-recovery-readiness:GetCell",
                "route53-recovery-readiness:GetReadinessCheck",
                "route53-recovery-readiness:GetRecoveryGroup",
                "route53-recovery-readiness:GetResourceSet",
                "route53-recovery-readiness:List*",
                "route53:GetDNSSEC",
                "route53:GetHealthCheck",
                "route53:GetHealthCheckStatus",
                "route53:GetHostedZone",
                "route53:List*",
                "route53profiles:GetProfile",
                "route53profiles:GetProfileAssociation",
                "route53profiles:GetProfileResourceAssociation",
                "route53profiles:List*",
                "route53resolver:GetFirewallDomainList",
                "route53resolver:GetFirewallRuleGroup",
                "route53resolver:GetFirewallRuleGroupAssociation",
                "route53resolver:GetOutpostResolver",
                "route53resolver:GetResolverConfig",
                "route53resolver:GetResolverQueryLogConfig",
                "route53resolver:GetResolverQueryLogConfigAssociation",
                "route53resolver:GetResolverRule",
                "route53resolver:GetResolverRuleAssociation",
                "route53resolver:List*",
```

```
                "rum:GetAppMonitor",
                "rum:List*",
                "s3-outposts:ListEndpoints",
                "s3-outposts:ListOutpostsWithS3",
                "s3:GetAccessGrant",
                "s3:GetAccessGrantsInstance",
                "s3:GetAccessGrantsLocation",
                "s3:GetAccessPoint",
                "s3:GetAccessPointConfigurationForObjectLambda",
                "s3:GetAccessPointForObjectLambda",
                "s3:GetAccessPointPolicy",
                "s3:GetAccessPointPolicyForObjectLambda",
                "s3:GetAccessPointPolicyStatusForObjectLambda",
                "s3:GetBucketAbac",
                "s3:GetBucketAcl",
                "s3:GetBucketCORS",
                "s3:GetBucketLocation",
                "s3:GetBucketLogging",
                "s3:GetBucketMetadataTableConfiguration",
                "s3:GetBucketNotification",
                "s3:GetBucketObjectLockConfiguration",
                "s3:GetBucketOwnershipControls",
                "s3:GetBucketPolicy",
                "s3:GetBucketPublicAccessBlock",
                "s3:GetBucketTagging",
                "s3:GetBucketVersioning",
                "s3:GetEncryptionConfiguration",
                "s3:GetLifecycleConfiguration",
                "s3:GetMultiRegionAccessPoint",
                "s3:GetMultiRegionAccessPointPolicy",
                "s3:GetMultiRegionAccessPointPolicyStatus",
                "s3:GetReplicationConfiguration",
                "s3:GetStorageLensConfiguration",
                "s3:GetStorageLensConfigurationTagging",
                "s3:GetStorageLensGroup",
                "s3:ListAllMyBuckets",
                "sagemaker:Describe*",
                "sagemaker:List*",
                "scheduler:GetSchedule",
                "scheduler:GetScheduleGroup",
                "scheduler:List*",
                "schemas:Describe*",
                "schemas:GetResourcePolicy",
                "schemas:List*",
```

```
                "secretsmanager:Describe*",
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:List*",
                "securityhub:BatchGetAutomationRules",
                "securityhub:BatchGetSecurityControls",
                "securityhub:Describe*",
                "securityhub:GetConfigurationPolicy",
                "securityhub:GetConfigurationPolicyAssociation",
                "securityhub:GetEnabledStandards",
                "securityhub:GetFindingAggregator",
                "securityhub:GetInsights",
                "securityhub:List*",
                "securitylake:GetSubscriber",
                "securitylake:List*",
                "servicecatalog:Describe*",
                "servicecatalog:GetApplication",
                "servicecatalog:GetAttributeGroup",
                "servicecatalog:List*",
                "servicequotas:GetServiceQuota",
                "ses:Describe*",
                "ses:GetAccount",
                "ses:GetAddonInstance",
                "ses:GetAddonSubscription",
                "ses:GetArchive",
                "ses:GetConfigurationSet",
                "ses:GetConfigurationSetEventDestinations",
                "ses:GetContactList",
                "ses:GetDedicatedIpPool",
                "ses:GetDedicatedIps",
                "ses:GetEmailIdentity",
                "ses:GetEmailTemplate",
                "ses:GetIngressPoint",
                "ses:GetRelay",
                "ses:GetRuleSet",
                "ses:GetTemplate",
                "ses:GetTrafficPolicy",
                "ses:List*",
                "shield:Describe*",
                "shield:List*",
                "signer:GetSigningProfile",
                "signer:List*",
                "sns:GetDataProtectionPolicy",
                "sns:GetSubscriptionAttributes",
                "sns:GetTopicAttributes",
```

```
                "sns:List*",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sqs:List*",
                "ssm-contacts:GetContact",
                "ssm-contacts:GetContactChannel",
                "ssm-contacts:List*",
                "ssm-incidents:GetReplicationSet",
                "ssm-incidents:GetResponsePlan",
                "ssm-incidents:List*",
                "ssm-sap:GetApplication",
                "ssm-sap:List*",
                "ssm:Describe*",
                "ssm:GetDefaultPatchBaseline",
                "ssm:GetDocument",
                "ssm:GetParameters",
                "ssm:GetPatchBaseline",
                "ssm:GetResourcePolicies",
                "ssm:List*",
                "sso:GetInlinePolicyForPermissionSet",
                "sso:GetManagedApplicationInstance",
                "sso:GetPermissionsBoundaryForPermissionSet",
                "sso:GetSharedSsoConfiguration",
                "sso:ListAccountAssignments",
                "sso:ListApplicationAssignments",
                "sso:ListApplications",
                "sso:ListCustomerManagedPolicyReferencesInPermissionSet",
                "sso:ListInstances",
                "sso:ListManagedPoliciesInPermissionSet",
                "sso:ListTagsForResource",
                "states:GetExecutionHistory",
                "states:Describe*",
                "states:List*",
                "support:CreateCase",
                "support:DescribeCases",
                "synthetics:Describe*",
                "synthetics:GetCanary",
                "synthetics:GetCanaryRuns",
                "synthetics:GetGroup",
                "synthetics:List*",
                "tag:GetResources",
                "timestream:Describe*",
                "timestream:List*",
                "transfer:Describe*",
```

```
                "transfer:List*",
                "verifiedpermissions:GetIdentitySource",
                "verifiedpermissions:GetPolicy",
                "verifiedpermissions:GetPolicyStore",
                "verifiedpermissions:GetPolicyTemplate",
                "verifiedpermissions:GetSchema",
                "verifiedpermissions:List*",
                "vpc-lattice:GetAccessLogSubscription",
                "vpc-lattice:GetAuthPolicy",
                "vpc-lattice:GetListener",
                "vpc-lattice:GetResourcePolicy",
                "vpc-lattice:GetRule",
                "vpc-lattice:GetService",
                "vpc-lattice:GetServiceNetwork",
                "vpc-lattice:GetServiceNetworkServiceAssociation",
                "vpc-lattice:GetServiceNetworkVpcAssociation",
                "vpc-lattice:GetTargetGroup",
                "vpc-lattice:List*",
                "wafv2:GetIPSet",
                "wafv2:GetLoggingConfiguration",
                "wafv2:GetRegexPatternSet",
                "wafv2:GetRuleGroup",
                "wafv2:GetWebACL",
                "wafv2:GetWebACLForResource",
                "wafv2:List*",
                "workspaces-web:GetBrowserSettings",
                "workspaces-web:GetIdentityProvider",
                "workspaces-web:GetNetworkSettings",
                "workspaces-web:GetPortal",
                "workspaces-web:GetPortalServiceProviderMetadata",
                "workspaces-web:GetTrustStore",
                "workspaces-web:GetUserAccessLoggingSettings",
                "workspaces-web:GetUserSettings",
                "workspaces-web:List*",
                "workspaces:Describe*",
                "xray:BatchGetTraces",
                "xray:GetGroup",
                "xray:GetGroups",
                "xray:GetSamplingRules",
                "xray:GetServiceGraph",
                "xray:GetTraceSummaries",
                "xray:List*"
            ],
            "Resource": "*"
```

```
        },
        {
            "Sid": "AIOPSAPIGatewayAccess",
            "Effect": "Allow",
            "Action": [
                "apigateway:GET"
            ],
            "Resource": [
                "arn:aws:apigateway:*::/restapis",
                "arn:aws:apigateway:*::/restapis/*",
                "arn:aws:apigateway:*::/restapis/*/deployments",
                "arn:aws:apigateway:*::/restapis/*/deployments/*",
                "arn:aws:apigateway:*::/restapis/*/resources/*/methods/*/integrations",
                "arn:aws:apigateway:*::/restapis/*/resources/*/methods/*/integrations/
*",
                "arn:aws:apigateway:*::/restapis/*/stages",
                "arn:aws:apigateway:*::/restapis/*/stages/*",
                "arn:aws:apigateway:*::/apis",
                "arn:aws:apigateway:*::/apis/*",
                "arn:aws:apigateway:*::/apis/*/deployments",
                "arn:aws:apigateway:*::/apis/*/deployments/*",
                "arn:aws:apigateway:*::/apis/*/integrations",
                "arn:aws:apigateway:*::/apis/*/integrations/*",
                "arn:aws:apigateway:*::/apis/*/stages",
                "arn:aws:apigateway:*::/apis/*/stages/*",
                "arn:aws:apigateway:*::/domainnames/*"
            ]
        }
    ]
}
```

Limiting Agent Access in an AWS Account

AWS DevOps Agent uses IAM roles to discover and describe AWS resources during incident investigations and preventative evaluations. You can control the level of access the agent has by configuring IAM policies attached to these roles. The application topology doesn't show everything the agent has access to—IAM policies are the only way to truly limit what AWS service APIs and resources the agent can access.

Understanding IAM roles for AWS DevOps Agent

AWS DevOps Agent uses IAM roles to access resources in two types of accounts:

- **Primary account role** – Grants the agent access to resources in the AWS account where you create the Agent Space.
- **Secondary account roles** – Grants the agent access to resources in additional AWS accounts that you connect to the Agent Space.

For either type of account, you can restrict which AWS services the agent can access, limit access to specific resources within those services, and control which regions the agent can operate in.

Choosing your resource boundaries

When limiting resource access, you need to include enough permissions for the agent to successfully investigate application incidents. This includes:

- All resources for in-scope applications that the agent should monitor and investigate
- All supporting infrastructure that those applications depend on

Supporting infrastructure may include:

- Networking components (VPCs, subnets, load balancers, API gateways)
- Data stores (databases, caches, object storage)
- Compute resources (EC2 instances, Lambda functions, containers)
- Monitoring and logging services (CloudWatch, CloudTrail)
- Identity and access management resources needed to understand permissions

If you restrict access too narrowly, the agent may not be able to identify root causes that originate in supporting infrastructure outside your defined boundaries.

Restricting service access

You can limit which AWS services the agent can access by modifying the IAM policies attached to the agent's roles. When creating custom policies, follow these best practices:

- **Grant only read-only permissions** – The agent needs to read resource configurations, metrics, and logs during investigations. Avoid granting permissions that allow the agent to modify or delete resources.

- **Limit to necessary services** – Include only the AWS services that contain resources relevant to your applications. For example, if your application doesn't use Amazon RDS, don't include RDS permissions in the policy.
- **Use specific actions instead of wildcards** – Instead of granting `service:*` permissions, specify individual actions like `cloudwatch:GetMetricData` or `ec2:DescribeInstances`.

Example policy restricting to specific services:

```
json

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:DescribeAlarms",
        "logs:GetLogEvents",
        "logs:FilterLogEvents",
        "ec2:DescribeInstances",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": "*"
    }
  ]
}
```

Restricting resource access

To limit the agent to specific resources within a service, use resource-level permissions in your IAM policies. This allows you to grant access only to resources that match specific patterns.

Using resource ARN patterns:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": "arn:aws:lambda:*:*:function:production-*"
    }
  ]
}
```

This example limits the agent to accessing only Lambda functions with names that begin with "production-".

Using tag-based restrictions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Environment": "production"
        }
      }
    }
  ]
}
```

This example limits the agent to accessing only EC2 instances tagged with `Environment=production`.

Restricting regional access

To limit which AWS regions the agent can access, use the `aws:RequestedRegion` condition key in your IAM policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:Describe*",
        "lambda:Get*",
        "cloudwatch:Get*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-west-2"
          ]
        }
      }
    }
  ]
}
```

This example limits the agent to accessing resources only in the us-east-1 and us-west-2 regions.

Creating custom IAM policies

When you create an Agent Space or add secondary accounts, you have the option to create a custom IAM role using a policy template. This allows you to implement the principle of least privilege.

When creating an Agent Space

From the DevOps Agent console in the AWS Management Console...

- Choose **Create a new DevOps Agent role using a policy document** and follow the instructions

When editing an Agent Space

From the DevOps Agent console in the AWS Management Console...

- Select the **Capabilities** tab

- Select the secondary account you want to edit from the **Cloud** section and click Edit
- Chose **Create a new DevOps Agent policy using a template** and follow the instructions

Custom policy best practices

- **Grant only read-only permissions** – Avoid permissions that allow resource modification or deletion
- **Use resource-level permissions when possible** – Restrict access to specific resources using ARN patterns or tags
- **Regularly review and audit permissions** – Periodically review the agent's IAM policies to ensure they still align with your security requirements

Setting Up IAM Identity Center Authentication

IAM Identity Center authentication provides a centralized way to manage user access to the AWS DevOps Agent Space web application. This guide explains how to configure IAM Identity Center authentication and manage users.

Prerequisites

Before setting up IAM Identity Center authentication, ensure you have:

- IAM Identity Center enabled in your organization or account
- Administrator permissions in AWS DevOps Agent
- An Agent Space configured or ready to create

Authentication options

AWS DevOps Agent offers two authentication methods for accessing the Agent Space web app:

IAM Identity Center authentication – Recommended for production environments. Provides centralized user management, integration with external identity providers, and sessions up to 12 hours.

Admin access (IAM authentication) – Provides quick access for administrators during initial setup and configuration. Sessions are limited to 30 minutes.

Configuring IAM Identity Center during Agent Space creation

When you create an Agent Space, you can configure IAM Identity Center authentication on the **Access** tab:

Step 1: Navigate to the Web app configuration

1. After configuring your Agent Space details and AWS account access, proceed to the **Access** tab
2. You'll see two sections: "Connect IAM Identity Center" and "Admin access"

Step 2: Configure IAM Identity Center integration

In the **Connect [Agent Space] to IAM Identity Center** section:

1. **Verify the IAM Identity Center instance** – The console displays which Identity Center instance will manage Web App user access (for example, `ssoins-7223a9580931edbe`). Your closest IAM Identity Center instance will automatically be pre-populated.
2. **Select the IAM Identity Center Application Role Name option** – Choose one of three options:

Auto-create a new DevOps Agent role (recommended):

- The system automatically creates a new service role with appropriate permissions
- This is the simplest option and works for most use cases

Assign an existing role:

- Use an existing IAM role that you've already created
- The system will verify the role has the required permissions
- Choose this option if your organization has pre-created roles for AWS DevOps Agent

Create a new DevOps Agent role using a policy template:

- Use the provided policy details to create your own custom role in the IAM Console
- Choose this option if you need to customize the role permissions

After clicking Connect, the system automatically:

- Creates or configures the specified IAM role
- Sets up an IAM Identity Center application for your Agent Space
- Establishes trust relationships between IAM Identity Center and the Agent Space web app
- Configures OAuth 2.0 authentication flows for secure user access

Alternative: Using admin access

If you want to access the Agent Space web app immediately without setting up IAM Identity Center:

1. In the **Admin access** section, note the IAM Role ARN that provides administrator access (for example, `arn:aws:iam::440491339484:role/service-role/DevOpsAgentRole-WebappAdmin-15ppoc42`)
2. Click the blue **Admin access** button to launch the Agent Space web app with IAM authentication
3. Sessions using this method are limited to 30 minutes

> **Note**
>
> Admin access is intended for initial setup and configuration. For production use and ongoing operations, configure IAM Identity Center authentication.

Adding users and groups

After configuring IAM Identity Center authentication, you need to grant specific users and groups access to the Agent Space web app:

Step 1: Access user management

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Access** tab
3. Under **User Access**, click **Manage Users and Groups**

Step 2: Add users or groups

1. Choose **Add Users or Groups**

2. Search for users or groups in your IAM Identity Center directory
3. Select the checkboxes next to the users or groups you want to add
4. Click **Add** to grant them access

The selected users can now access the Agent Space web app using their IAM Identity Center credentials.

Working with external identity providers

If you're using an external identity provider (such as Okta, Microsoft Entra ID, or Ping Identity) with IAM Identity Center:

- Users and groups are synchronized from your external identity provider to IAM Identity Center
- When you add users and groups to the Agent Space web app, you're selecting from the synchronized directory
- User attributes and group memberships are maintained by your external identity provider
- Changes in your identity provider are automatically reflected in IAM Identity Center after synchronization

How users access the Agent Space web app

After you've added users to your Agent Space:

1. Share the Agent Space web app URL with authorized users
2. When users navigate to the URL, they're redirected to the IAM Identity Center login page
3. After entering their credentials (and completing MFA if configured), they're redirected back to the Agent Space web app
4. Their session is valid for 8 hours by default (configurable by the Identity Center administrator)

Managing user access

You can update user access at any time:

Adding more users or groups:

- Follow the same steps described above to add additional users or groups

Removing access:

1. In the **User Access** section, find the user or group to remove
2. Click the **Remove** button next to their name
3. Confirm the removal

Removed users will lose access immediately, but active sessions may continue until they expire.

Session management

IAM Identity Center sessions for the Agent Space web app have the following characteristics:

- **Default session duration** – 8 hours
- **Session security** – HTTP-only cookies for enhanced protection
- **Multi-factor authentication** – Supported when configured in IAM Identity Center
- **API credentials** – Short-duration (15-minute) SigV4 credentials are issued for API calls and renewed automatically

To configure session duration:

1. Navigate to the IAM Identity Center console
2. Go to **Settings** > **Authentication**
3. Under **Session duration**, configure your preferred duration (from 1 hour to 12 hours)
4. Choose **Save changes**

Disconnecting Identity Center

1. In your Agent Space's console, click **Actions** in the top-right and select **Disconnect from IAM Identity Center**
2. Confirm in confirmation dialog

Setting Up External Identity Provider (IdP) Authentication

External identity provider (IdP) authentication allows your organization to use an existing OIDC-compatible identity provider, such as Okta or Microsoft Entra ID, to manage user access to the

AWS DevOps Agent Space web application. Users sign in with their corporate credentials directly through your IdP, without requiring AWS IAM Identity Center.

Prerequisites

Before setting up external IdP authentication, ensure you have:

- An OIDC-compatible identity provider (Okta or Microsoft Entra ID)
- Administrator access to your identity provider
- Administrator permissions to access AWS DevOps Agent console
- An Agent Space configured or ready to create

How it works

When you configure external IdP authentication:

- Users navigate to the Agent Space web app URL
- They are redirected to your identity provider's login page
- After authenticating with their corporate credentials, they are redirected back to the web app
- The web app exchanges the authentication token for short-lived AWS credentials scoped to the Agent Space

Sessions are valid for up to 8 hours. Credentials are automatically refreshed using OIDC refresh tokens without requiring users to re-authenticate.

Configuring external IdP authentication

Step 1: Register an application in your identity provider

Choose your identity provider and follow the corresponding setup instructions.

Option A: Okta

1. In the Okta Admin Console, navigate to **Applications** > **Applications** and choose **Create App Integration**
2. Select **OIDC - OpenID Connect** as the sign-in method and **Web Application** as the application type. Choose **Next**

3. Set a descriptive name for the application (for example, `AWS DevOps Agent`)
4. Under **Grant type**, ensure the following are checked:
   - **Authorization Code** (default)
   - **Refresh Token** — This is required for session refresh. If not enabled, users will be unable to maintain sessions.

> **Note**
>
> Okta does not enable the Refresh Token grant type by default. You must explicitly enable it.

1. Leave the **Sign-in redirect URIs** as the default value for now — you will update it after configuring the Agent Space
2. Under **Assignments**, assign the users or groups that should have access
3. Choose **Save**
4. On the application's **General** tab, note the following values:
   - **Client ID**
   - **Client secret** — Choose **Copy** to save this value securely
5. Note your **Okta domain** — this is your Issuer URL (for example, `https://dev-12345678.okta.com`).

> **Note**
>
> **On the** Sign On **tab, verify the** Issuer **is set to** Okta URL (not Dynamic). This ensures a stable issuer URL.

> **Note**
>
> **Do not add a** groups claim to the ID token in your authorization server's Claims tab. AWS DevOps Agent does not use group membership from your IdP.

Option B: Microsoft Entra ID

1. In the Azure portal, navigate to **Microsoft Entra ID** > **App registrations** > **New registration**
2. Set a descriptive name (for example, `AWS DevOps Agent`)
3. Under **Supported account types**, select the option appropriate for your organization (typically **Accounts in this organizational directory only**)
4. Leave the **Redirect URI** blank for now. Choose **Register**
5. On the application **Overview** page, note the following values:
   - **Application (client) ID** — used as the Client ID when configuring the Agent Space
   - **Directory (tenant) ID** — used to construct the Issuer URL
6. Navigate to **Certificates & secrets** > **New client secret**
   - Set a description and expiration period
   - Choose **Add** and copy the secret **Value** immediately — it will not be shown again
7. The Issuer URL for Entra ID follows this format. Replace `{tenant-id}` with your Directory (tenant) ID from step 5:
   - `https://login.microsoftonline.com/{tenant-id}/v2.0`

> **Note**
>
> **Do not enable the** groups **optional claim in** Token configuration . AWS DevOps Agent does not use group membership from your IdP.

Step 2: Enable the Operator App with IdP authentication

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Access** tab
3. Under **User access**, choose **External identity provider**
4. In the configuration form, configure the following:
   - **Identity Provider** — Select your identity provider (Okta or Microsoft Entra ID)
   - **Issuer URL** — The OIDC issuer URL from your identity provider
   - **Client ID** — The client ID from the OIDC application you created
   - **Client Secret** — The client secret from your OIDC application

5. Under **Identity Provider Application Role Name**, choose one of three options:
   - **Auto-create a new DevOps Agent role** (recommended) — Creates a new service role with appropriate permissions
   - **Assign an existing role** — Use an existing IAM role that you've already created
   - **Create a new DevOps Agent role using a policy template** — Use the provided details to create your own role in the IAM Console
6. Review the **Callback URL** warning alert displayed at the bottom of the form. Copy this URL — you will need to add it to your identity provider's allowed redirect URIs before users can sign in.
7. Choose **Connect**

After choosing **Connect**, the console displays the **External Identity Provider Configuration** with the following details:

- **Provider** — The identity provider you selected
- **Issuer URL** — The configured OIDC issuer URL
- **Client ID** — The configured client ID
- **IAM Role ARN** — The IAM role used for user access
- **Callback URL** — Configure this URL in your identity provider as an allowed redirect URI
- **Login URL** — Use this URL to access the web app through your identity provider

Step 3: Add the callback URL to your identity provider

Okta

1. In the Okta Admin Console, navigate to your application's **General** tab
2. Under **Login**, choose **Edit**
3. Add the callback URL as a **Sign-in redirect URI**:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/idp/callback`
4. (Optional) Set the **Initiate login URI** to enable IdP-initiated login from the Okta dashboard:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/idp/login`
5. (Recommended) Add a **Sign-out redirect URI** to redirect users back to the web app after logout. Without this, users may see an error page when logging out:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/welcome`

6. Choose **Save**

Microsoft Entra ID

1. In the Azure portal, navigate to your application's **Authentication** page
2. Under **Platform configurations**, choose **Add a platform** > **Web**
3. Enter the callback URL as the **Redirect URI**:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/idp/callback`
4. (Optional) Add a sign-out redirect URI to redirect users back to the web app after logout:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/welcome`
5. Choose **Configure**

Step 4: Verify the configuration

1. Navigate to the **Login URL** shown in the console:
   - `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/idp/login`
2. You should be redirected to your identity provider's login page
3. Sign in with your corporate credentials
4. After successful authentication, you are redirected back to the Agent Space web app

Updating IdP configuration

You can rotate the client secret without disconnecting:

1. In the AWS DevOps Agent console, select your Agent Space
2. Go to the **Access** tab
3. Under **External Identity Provider Configuration**, choose **Rotate client secret**
4. Enter the new **Client Secret**
5. Choose **Save**

To change any other IdP configuration field (such as Issuer URL, Client ID, or identity provider), you must disconnect the existing IdP and configure a new one.

How users access the Agent Space web app

After configuring external IdP authentication:

- Share the Agent Space web app URL with authorized users
- When users navigate to the URL, they are redirected to your identity provider's login page
- After entering their credentials (and completing MFA if configured by your IdP), they are redirected back to the Agent Space web app
- Sessions refresh automatically — see Session management for details

Session management

External IdP sessions for the Agent Space web app have the following characteristics:

- **Session duration** — Browser sessions last up to 8 hours. This is not configurable in AWS DevOps Agent. If your IdP's session lifetime exceeds 8 hours, users may be re-authenticated automatically on their next visit without entering credentials. Configure your IdP's session and token lifetimes according to your organization's security requirements.
- **Credential refresh** — Sessions are automatically refreshed using OIDC refresh tokens without requiring users to re-authenticate
- **Multi-factor authentication** — Supported when configured in your identity provider. The IdP handles MFA during login — no additional configuration is needed in AWS DevOps Agent

Logout behavior

When a user clicks **Logout** in the web app:

1. All session cookies are cleared immediately
2. The user is redirected to the identity provider's OIDC logout endpoint to terminate the SSO session
3. If a sign-out redirect URI is configured, the user is redirected back to the web app welcome page

Revoking user access

To immediately revoke a user's access, you can revoke their sessions directly in your identity provider's admin portal:

- **Okta** — In the Okta Admin Console, navigate to **Directory** > **People**, select the user, choose **More Actions** > **Clear User Sessions**
- **Microsoft Entra ID** — In the Azure portal, navigate to **Users**, select the user, and choose **Revoke sessions**

Security considerations

Client secret storage — The client secret you provide during setup is encrypted using your customer-managed KMS key if you provided one when creating the Agent Space, or a service-owned key otherwise. It is never returned in API responses or displayed in the console after initial configuration.

Client secret rotation — Entra client secrets have a configurable expiration. Set a reminder to rotate the secret before it expires using the **Rotate client secret** option in the AWS DevOps Agent console. If the secret expires, users will be unable to log in until it is rotated.

Token lifetime management — The lifetime of tokens (access tokens, refresh tokens) issued by your identity provider is controlled by your IdP's configuration. We recommend configuring appropriate token lifetimes in your IdP:

- **Okta** — Configure token lifetimes under **Security** > **API** > **Authorization Servers** > **Access Policies**
- **Microsoft Entra ID** — Configure token lifetimes using token lifetime policies

Groups claim — Do not enable the groups claim in your identity provider's token configuration. AWS DevOps Agent does not currently use group membership from your IdP.

User identifier — AWS DevOps Agent uses a provider-specific claim to uniquely identify users:

- **Okta** — Uses the `sub` claim from the ID token
- **Microsoft Entra ID** — Uses the `oid` (object identifier) claim from the ID token

These identifiers are immutable and appear in CloudTrail logs for audit purposes.

Disconnecting external IdP

1. In the AWS DevOps Agent console, select your Agent Space

2. Go to the **Access** tab
3. Under **User access**, choose **Disconnect**
4. Review the impacts listed in the confirmation dialog and confirm

Disconnecting will:

- Remove the IdP configuration from the Agent Space
- Prevent users from logging in through the external identity provider
- Remove individual chat and artifact history associated with IdP user accounts

Active user sessions will continue until they expire or the next credential refresh fails.

Troubleshooting

- **Redirect to IdP fails** — Verify the Issuer URL matches your IdP's OIDC discovery endpoint. For Okta, ensure the **Issuer** is set to **Okta URL** (not **Dynamic**) on the **Sign On** tab. For Entra, use the format `https://login.microsoftonline.com/{tenant-id}/v2.0`.
- **Access denied or policy error (Okta)** — Verify the user or their group is assigned to the application under **Assignments**. Check **Sign On** > **Sign On Policy** rules.
- **IdP configuration error after login** — Your identity provider did not return a refresh token. Ensure the `offline_access` scope and refresh token grant type are enabled:
  - **Okta** — Go to your application's **General** tab and enable the **Refresh Token** checkbox under **Grant type**
  - **Entra** — Go to **API permissions** and ensure `offline_access` is listed under delegated permissions
- **Authentication succeeds but web app shows error** — Verify the redirect URI in your IdP exactly matches the **Callback URL** shown in the AWS DevOps Agent console.
- **Authentication failures** — If the **groups** optional claim is enabled in your IdP, disable it. AWS DevOps Agent does not use group claims.
- **Login fails after IdP authentication** — For Entra, verify `requestedAccessTokenVersion` is not set to `null` in the application **Manifest**. For Okta, verify the **Issuer URL** is correct.
- **Error page after clicking Logout (Okta)** — If you see a `post_logout_redirect_uri` error after logging out, add `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/welcome` as a **Sign-out redirect URI** in your Okta application's **General** tab.

- **Users stay on identity provider page after logout (Entra)** — To redirect users back to the web app after logout, add `https://{agentSpaceId}.aidevops.global.app.aws/authorizer/welcome` as a **Redirect URI** in your Entra application's **Authentication** page.

Encryption at rest for AWS DevOps Agent

AWS DevOps Agent encrypts all customer data at rest. By default, AWS DevOps Agent uses AWS owned keys to automatically encrypt your data at no additional charge. You cannot view, manage, or audit the use of AWS owned keys. However, you do not need to take any action to protect these keys. Your data is automatically secured.

You can choose to encrypt your data using a symmetric customer managed key that you create, own, and manage in AWS Key Management Service (AWS KMS). Because you have full control of this layer of encryption, you can perform tasks such as the following:

- Establishing and maintaining key policies
- Enabling and disabling key policies
- Rotating key cryptographic material
- Adding tags
- Creating key aliases
- Scheduling keys for deletion

For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> AWS DevOps Agent automatically enables encryption at rest using AWS owned keys to protect customer data at no charge. Standard AWS KMS charges apply when you use a customer managed key. For more information about pricing, see AWS Key Management Service pricing.

Customer managed keys

Customer managed keys are KMS keys in your AWS account that you create, own, and manage. You have full control over these KMS keys, including establishing and maintaining their key policies.

When you configure a customer managed key, AWS DevOps Agent uses it to protect sensitive resource data. AWS DevOps Agent uses envelope encryption with the AWS Encryption SDK hierarchical keyring. Your KMS key is used to generate branch keys, which in turn protect your data.

You can specify a customer managed key when you create the following resources:

- **Agent Space** — Encrypts Agent Space details and content created from the DevOps Agent Web App related to investigations, skills, and chat.
- **Service** — Encrypts third-party service credentials at rest.

To configure a customer managed key in AWS DevOps Agent, follow these steps.

Step 1: Create a customer managed key

You can create a symmetric customer managed key by using the AWS KMS console or the AWS KMS API. The key must meet the following requirements:

Property	Requirement
Key type	Symmetric
Key spec	`SYMMETRIC_DEFAULT`
Key usage	`ENCRYPT_DECRYPT`

> **Note**
>
> AWS DevOps Agent only supports symmetric encryption KMS keys with the `SYMMETRIC_DEFAULT` key spec and the `ENCRYPT_DECRYPT` key usage. Multi-Region keys and asymmetric keys are not currently supported.

For more information, see Creating a symmetric customer managed key in the *AWS Key Management Service Developer Guide*.

Step 2: Set the key policy

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it.

Your key policy must grant permissions to both the calling principal (your IAM identity) and the AWS DevOps Agent service. AWS DevOps Agent accesses your key using two sets of credentials:

1. **Your caller credentials** — Used for all synchronous operations, including key validation, encryption at resource creation time, and any API call that returns a direct response to the caller.
2. **AWS DevOps Agent service principal** — Used for asynchronous operations that run in the background, such as operational investigations, incident analysis, event correlation, and root cause analysis generation.

The following table lists the required KMS actions:

KMS action	Description
`kms:DescribeKey`	Validate key configuration at resource creation time
`kms:GenerateDataKey`	Generate data encryption keys for envelope encryption
`kms:Decrypt`	Decrypt data
`kms:Encrypt`	Encrypt data
`kms:ReEncrypt`	Re-encrypt data under the same or different key

AWS DevOps Agent validates all of these permissions at configuration time using dry-run operations. If any permission is missing, the request fails with an exception.

The following is an example key policy. Replace the placeholder values with your own.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCallerAccessViaService",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/DevOpsAgentUserRole"
      },
      "Action": [
        "kms:DescribeKey",
        "kms:GenerateDataKey*",
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "aidevops.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowDevOpsAgentServiceDescribeKeyAccess",
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDevOpsAgentAccessForAgentSpace",
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey*",
        "kms:Decrypt",
```

```
        "kms:Encrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:aidevops:us-east-1:111122223333:agentspace/*"
        },
        "StringLike": {
          "kms:EncryptionContext:aws-crypto-ec:aws:aidevops:arn": "arn:aws:aidevops:us-
east-1:111122223333:agentspace/*"
        }
      }
    },
    {
      "Sid": "AllowDevOpsAgentAccessForService",
      "Effect": "Allow",
      "Principal": {
        "Service": "aidevops.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey*",
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:aidevops:us-east-1:111122223333:service/*"
        },
        "StringLike": {
          "kms:EncryptionContext:aws-crypto-ec:aws:aidevops:arn": "arn:aws:aidevops:us-
east-1:111122223333:service/*"
        }
      }
    }
  ]
}
```

The policy contains the following statements:

- **AllowKeyAdministration** — Grants the account root full administrative access to the key. Replace `111122223333` with your AWS account ID.
- **AllowCallerAccessViaService** — Grants your IAM principals the KMS permissions required for all synchronous AWS DevOps Agent operations. This includes key validation at resource creation time, as well as encrypt and decrypt operations for any API call that returns a direct response to the caller. The `kms:ViaService` condition ensures that you can use the key only through the AWS DevOps Agent service. Replace `111122223333` with your AWS account ID and `us-east-1` with your AWS Region.
- **AllowDevOpsAgentServiceAccessForAgentSpace / AllowDevOpsAgentServiceAccessForService** — Grants the `aidevops.amazonaws.com` service principal the KMS permissions required for asynchronous operations. AWS DevOps Agent uses this service principal to encrypt and decrypt your data when performing background operations such as operational investigations, analyzing incidents, correlating events across services, and generating root cause analyses. Without this access, AWS DevOps Agent cannot read the encrypted data needed to carry out investigations on your behalf. The `aws:SourceArn` condition restricts access to requests originating from your AWS DevOps Agent resources, and the `kms:EncryptionContext` condition ensures that the encryption context matches your resource ARNs. Replace `111122223333` with your AWS account ID and `us-east-1` with your AWS Region.

For more information about key policies, see Key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Step 3: Specify the key when creating a resource

After you create your key and configure the key policy, you can specify the key when creating AWS DevOps Agent resources.

Console

To configure a customer managed key when creating an Agent Space in the console:

1. Open the AWS DevOps Agent console.
2. Choose **Create Agent Space** or **Register Service**.
3. Enter the agent space details (name, description, and IAM role).
4. Expand the **Advanced Configuration** section.

5. Under **Encryption key type**, select **Customer managed key**.
6. Choose a KMS key from the dropdown list, or enter a KMS key ARN.
7. Review the key policy displayed in the **Key policy** expandable section. Ensure that you have attached this policy to your KMS key. You can use the copy button to copy the policy.
8. Complete the remaining configuration and choose **Create**.

> **Note**
>
> If you do not see your KMS key in the dropdown list, verify that the key meets the requirements in Step 1 and that you have `kms:ListKeys` and `kms:DescribeKey` permissions.

API

Creating an Agent Space with a customer managed key

Specify the `kmsKeyArn` parameter when creating an agent space. The value must be the full KMS key ARN.

```
{
  "name": "my-agent-space",
  "description": "An encrypted agent space",
  "kmsKeyArn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Registering a Service with a customer managed key

Specify the `kmsKeyArn` parameter when registering a service. The value must be the full KMS key ARN. This parameter is supported across all service types, including Dynatrace, ServiceNow, PagerDuty, GitLab, GitHub, and MCP Servers.

```
{
  "service": "dynatrace",
  "kmsKeyArn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "serviceDetails": { ... }
```

```
}
```

> **Note**
>
> You must specify the customer managed key at resource creation time. You cannot add or change the customer managed key for an existing resource.

AWS DevOps Agent encryption context

An encryption context is a set of non-secret key-value pairs that contain additional contextual information about the data. AWS KMS uses the encryption context as additional authenticated data to support authenticated encryption. When you include an encryption context in a request to encrypt data, AWS KMS binds the encryption context to the encrypted data. To decrypt data, you must include the same encryption context in the request.

AWS DevOps Agent uses the following encryption context on all cryptographic operations:

```
{
  "aws-crypto-ec:aws:aidevops:arn": "arn:aws:aidevops:{region}:{accountId}:
{resourceType}/{resourceId}"
}
```

The encryption context value is the ARN of the AWS DevOps Agent resource being encrypted. You can use this encryption context in your key policy conditions and in AWS CloudTrail logs to audit how your key is being used.

Key management

If you disable or schedule deletion of your KMS key, AWS DevOps Agent cannot decrypt your data. This results in `AccessDeniedException` errors on operations that read encrypted data.

> **Important**
>
> If you choose to use a customer managed key, you are responsible for managing the key and its permissions. If the key is disabled or deleted, or if AWS DevOps Agent loses permission to use the key, you lose access to the encrypted data.

The following table describes common failure scenarios:

Action	Impact
Key policy permissions revoked	`AccessDeniedException` on encrypt and decrypt operations
KMS key is disabled	`DisabledException` on encrypt and decrypt operations
KMS key is scheduled for deletion	`KMSInvalidStateException` on encrypt and decrypt operations
KMS key is deleted	Permanent data loss — encrypted data cannot be recovered

Before disabling or deleting a key:

1. Verify that no active AWS DevOps Agent resources depend on the key.
2. Consider disabling the key first to test the impact before scheduling deletion.
3. AWS KMS enforces a minimum waiting period before key deletion, giving you time to cancel if needed.

> **Note:**: AWS DevOps Agent does not automatically re-encrypt data under a new key. If you need to rotate to a new customer managed key, you must create a new resource with the new key.

Monitoring your encryption keys

When you use a customer managed key with AWS DevOps Agent, you can use AWS CloudTrail to track requests that AWS DevOps Agent sends to AWS KMS.

You can filter CloudTrail events by:

- **Event source** — `kms.amazonaws.com`
- **Encryption context key** — `aws-crypto-ec:aws:aidevops:arn`
- **Key ARN** — Your customer managed key ARN in the request parameters

For more information, see Logging AWS KMS API calls with AWS CloudTrail in the *AWS Key Management Service Developer Guide*.

VPC Endpoints (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS DevOps Agent. You can access AWS DevOps Agent as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS DevOps Agent.

You establish this private connection by creating an interface endpoint, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for AWS DevOps Agent.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for AWS DevOps Agent VPC endpoints

Before you set up an interface endpoint for AWS DevOps Agent, review Considerations in the *AWS PrivateLink Guide*.

AWS DevOps Agent supports making API calls through the following VPC endpoints.

Category	Endpoint suffix
AWS DevOps Agent Control Plane API Actions	`aidevops`
AWS DevOps Agent Runtime Operations	`aidevops-dataplane`
AWS DevOps Agent Webhook Events	`event-ai`

Create an interface endpoint for AWS DevOps Agent

You can create an interface endpoint for AWS DevOps Agent using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for AWS DevOps Agent using the following service names:

- com.amazonaws.{region}.aidevops
- com.amazonaws.{region}.aidevops-dataplane
- com.amazonaws.{region}.event-ai

After you create the endpoint, you have the option to enable a private DNS hostname. Enable this setting by selecting **Enable Private DNS Name** in the VPC console when you create the VPC endpoint.

If you enable private DNS for the interface endpoint, you can make API requests to AWS DevOps Agent using its default Regional DNS name. The following example shows the format of the default Regional DNS name.

- aidevops.{region}.api.aws
- aidevops-dataplane.{region}.amazonaws.com
- event-ai.{region}.api.aws

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to AWS DevOps Agent through the interface endpoint. To control the access allowed to AWS DevOps Agent from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Quotas

AWS DevOps Agent quotas include number of agent spaces, concurrent investigations and more. You can request increases for some quotas, but not all quotas can be increased. These increases are not granted immediately, so it may take a couple of hours to days for your increase to become effective. Unless otherwise noted, each quota is Region-specific.

The following table describes the quotas for AWS DevOps Agent.

Name	Default	Adjustable	Description
Agent spaces per account per Region	10	Yes	The maximum number of agent spaces that you can create per account in each AWS Region.
Concurrent investigations per agent space	3	Yes	The maximum number of incident resolution investigations that can run concurrently in a single agent space.
Concurrent evaluations per agent space	1	No	The maximum number of incident prevention evaluations that can run concurrently in a single agent space.
Concurrent on-demand invocations per agent space	10	Yes	The maximum number of on-demand DevOps invocations that can run concurrently in a single agent space.

Requesting a quota increase

You can request a quota increase by using one of the following options:

- **From the AWS Management Console** – Open the Service Quotas console. In the navigation pane, choose **AWS services**. Select **DevOps Agent**, select a quota, and follow the directions to request a quota increase. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.
- **From the AWS CLI** – Use the request-service-quota-increase AWS CLI command. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.