User Guide

AWS AppConfig

AWS AppConfig: User Guide

Table of Contents

What is AWS AppConfig?

AWS AppConfig feature flags and dynamic configurations help software builders quickly and securely adjust application behavior in production environments without full code deployments. AWS AppConfig speeds up software release frequency, improves application resiliency, and helps you address emergent issues more quickly. With feature flags, you can gradually release new capabilities to users and measure the impact of those changes before fully deploying the new capabilities to all users. With operational flags and dynamic configurations, you can update block lists, allow lists, throttling limits, logging verbosity, and perform other operational tuning to quickly respond to issues in production environments.

Note

AWS AppConfig is a capability of AWS Systems Manager.

Improve efficiency and release changes faster

Using feature flags with new capabilities speeds up the process of releasing changes to production environments. Instead of relying on long-lived development branches that require complicated merges before a release, feature flags enable you to write software using trunk-based development. Feature flags enable you to safely roll out pre-release code in a CI/CD pipeline that is hidden from users. When you are ready to release the changes, you can update the feature flag without deploying new code. After the launch is complete, the flag can still function as a block switch to disable a new feature or capability without the need to roll back the code deployment.

Avoid unintended changes or failures with built-in safety features

AWS AppConfig offers the following safety features to help you avoid enabling feature flags or updating configuration data that could cause application failures.

- **Validators**: A validator ensures that your configuration data is syntactically and semantically correct before deploying the changes to production environments.
- **Deployment strategies**: A deployment strategy enables you to slowly release changes to production environments over minutes or hours.
- **Monitoring and automatic rollback**: AWS AppConfig integrates with Amazon CloudWatch to monitor changes to your applications. If your application becomes unhealthy because of a

bad configuration change and that change triggers an alarm in CloudWatch, AWS AppConfig automatically rolls back the change to minimize impact on your application users.

Secure and scalable feature flag deployments

AWS AppConfig integrates with AWS Identity and Access Management (IAM) to provide fine-grain, role-based access to the service. AWS AppConfig also integrates with AWS Key Management Service (AWS KMS) for encryption and AWS CloudTrail for auditing. Before being released to external customers, all AWS AppConfig safety controls were initially developed with and validated by internal customers that use the service at scale.

AWS AppConfig use cases

Despite the fact that application configuration content can vary greatly from application to application, AWS AppConfig supports the following use cases, which cover a broad spectrum of customer needs:

- **Feature flags and toggles –** Safely release new capabilities to your customers in a controlled environment. Instantly roll back changes if you experience a problem.
- **Application tuning –** Carefully introduce application changes while testing the impact of those changes with users in production environments.
- **Allow list or block list –** Control access to premium features or instantly block specific users without deploying new code.
- **Centralized configuration storage –** Keep your configuration data organized and consistent across all of your workloads. You can use AWS AppConfig to deploy configuration data stored in the AWS AppConfig hosted configuration store, AWS Secrets Manager, Systems Manager Parameter Store, or Amazon S3.

Benefits of using AWS AppConfig

AWS AppConfig offers the following benefits for your organization:

- **Reduce unexpected down time for your customers**

  AWS AppConfig reduces application downtime by enabling you to create rules to validate your configuration. Configurations that aren't valid can't be deployed. AWS AppConfig provides the following two options for validating configurations:

  - For syntactic validation, you can use a JSON schema. AWS AppConfig validates your configuration by using the JSON schema to ensure that configuration changes adhere to the application requirements.
  - For semantic validation, AWS AppConfig can call an AWS Lambda function that you own to validate the data within your configuration.
- **Quickly deploy changes across a set of targets**

  AWS AppConfig simplifies the administration of applications at scale by deploying configuration changes from a central location. AWS AppConfig supports configurations stored in the AWS AppConfig hosted configuration store, Systems Manager Parameter Store, Systems Manager (SSM) documents, and Amazon S3. You can use AWS AppConfig with applications hosted on EC2 instances, AWS Lambda, containers, mobile applications, or IoT devices.

  Targets don't need to be configured with the Systems Manager SSM Agent or the IAM instance profile required by other Systems Manager capabilities. This means that AWS AppConfig works with unmanaged instances.
- **Update applications without interruptions**

  AWS AppConfig deploys configuration changes to your targets at runtime without a heavy-weight build process or taking your targets out of service.
- **Control deployment of changes across your application**

  When deploying configuration changes to your targets, AWS AppConfig enables you to minimize risk by using a deployment strategy. Deployment strategies allow you to slowly roll out configuration changes to your fleet. If you experience a problem during the deployment, you can roll back the configuration change before it reaches the majority of yours hosts.

How AWS AppConfig works

This section provides a high-level description of how AWS AppConfig works and how you get started.

1. Identify configuration values in code you want to manage in the cloud

Before you start creating AWS AppConfig artifacts, we recommend you identify configuration data in your code that you want to dynamically manage using AWS AppConfig. Good examples include feature flags or toggles, allow and block lists, logging verbosity, service limits, and throttling rules, to name a few.

If your configuration data already exists in the cloud, you can take advantage of AWS AppConfig validation, deployment, and extension features to further streamline configuration data management.

2. Create an application namespace

To create a namespace, you create an AWS AppConfig artifact called an application. An application is simply an organizational construct like a folder.

3. Create environments

For each AWS AppConfig application, you define one or more environments. An environment is a logical grouping of targets, such as applications in a `Beta` or `Production` environment, AWS Lambda functions, or containers. You can also define environments for application subcomponents, such as the `Web`, `Mobile`, and `Back-end`.

You can configure Amazon CloudWatch alarms for each environment. The system monitors alarms during a configuration deployment. If an alarm is triggered, the system rolls back the configuration.

4. Create a configuration profile

A configuration profile includes, among other things, a URI that enables AWS AppConfig to locate your configuration data in its stored location and a profile type. AWS AppConfig supports two configuration profile types: feature flags and freeform configurations. Feature flag configuration profiles store their data in the AWS AppConfig hosted configuration store and the URI is simply `hosted`. For freeform configuration profiles, you can store your data in the AWS AppConfig hosted configuration store or any AWS service that integrates with AWS AppConfig, as described in Creating a free form configuration profile in AWS AppConfig.

A configuration profile can also include optional validators to ensure your configuration data is syntactically and semantically correct. AWS AppConfig performs a check using the validators when you start a deployment. If any errors are detected, the deployment rolls back to the previous configuration data.

5. Deploy configuration data

When you create a new deployment, you specify the following:

- An application ID
- A configuration profile ID
- A configuration version
- An environment ID where you want to deploy the configuration data
- A deployment strategy ID that defines how fast you want the changes to take effect

When you call the StartDeployment API action, AWS AppConfig performs the following tasks:

1. Retrieves the configuration data from the underlying data store by using the location URI in the configuration profile.
2. Verifies the configuration data is syntactically and semantically correct by using the validators you specified when you created your configuration profile.
3. Caches a copy of the data so it is ready to be retrieved by your application. This cached copy is called the *deployed data*.

6. Retrieve the configuration

You can configure AWS AppConfig Agent as a local host and have the agent poll AWS AppConfig for configuration updates. The agent calls the StartConfigurationSession and GetLatestConfiguration API actions and caches your configuration data locally. To retrieve the data, your application makes an HTTP call to the localhost server. AWS AppConfig Agent supports several use cases, as described in Simplified retrieval methods.

If AWS AppConfig Agent isn't supported for your use case, you can configure your application to poll AWS AppConfig for configuration updates by directly calling the StartConfigurationSession and GetLatestConfiguration API actions.

Get started with AWS AppConfig

The following resources can help you work directly with AWS AppConfig.

View more AWS videos on the Amazon Web Services YouTube Channel.

The following blogs can help you learn more about AWS AppConfig and its capabilities:

- Using AWS AppConfig feature flags
- Best Practices for validating AWS AppConfig Feature Flags and Configuration Data

SDKs

For information about AWS AppConfig language-specific SDKs, see the following resources:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Pricing for AWS AppConfig

Pricing for AWS AppConfig is pay-as-you-go based on configuration data and feature flag retrieval. We recommend using the AWS AppConfig Agent to help optimize costs. For more information, see AWS Systems Manager Pricing.

AWS AppConfig quotas

Information about AWS AppConfig endpoints and service quotas along with other Systems Manager quotas is in the Amazon Web Services General Reference.

Note

For information about quotas for services that store AWS AppConfig configurations, see About configuration store quotas and limitations.

Setting up AWS AppConfig

If you haven't already done so, sign up for an AWS account and create an administrative user.

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentia ls with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM

Which user needs programmatic access?	To	By
		user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Configure permissions for automatic rollback

You can configure AWS AppConfig to roll back to a previous version of a configuration in response to one or more Amazon CloudWatch alarms. When you configure a deployment to respond to CloudWatch alarms, you specify an AWS Identity and Access Management (IAM) role. AWS AppConfig requires this role so that it can monitor CloudWatch alarms. This procedure is optional, but highly recommended.

Note

The IAM role must belong to the current account. By default, AWS AppConfig can only monitor alarms owned by the current account. If you want to configure AWS AppConfig to roll back deployments in response to metrics from a different account, you must configure cross account alarms. For more information, see Cross-account cross-Region CloudWatch console in the *Amazon CloudWatch User Guide*.

Use the following procedures to create an IAM role that enables AWS AppConfig to rollback based on CloudWatch alarms. This section includes the following procedures.

1. Step 1: Create the permission policy for rollback based on CloudWatch alarms

2. Step 2: Create the IAM role for rollback based on CloudWatch alarms
3. Step 3: Add a trust relationship

Step 1: Create the permission policy for rollback based on CloudWatch alarms

Use the following procedure to create an IAM policy that gives AWS AppConfig permission to call the `DescribeAlarms` API action.

To create an IAM permission policy for rollback based on CloudWatch alarms

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **Create policy** page, choose the **JSON** tab.
4. Replace the default content on the JSON tab with the following permission policy, and then choose **Next: Tags**.

   Note

   To return information about CloudWatch composite alarms, the DescribeAlarms API operation must be assigned * permissions, as shown here. You can't return information about composite alarms if `DescribeAlarms` has a narrower scope.

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "cloudwatch:DescribeAlarms"
                ],
                "Resource": "*"
            }
        ]
    }
```

5. Enter tags for this role, and then choose **Next: Review**.

Step 1: Create the permission policy for rollback based on CloudWatch alarms 11

6. On the **Review** page, enter **`SSMCloudWatchAlarmDiscoveryPolicy`** in the **Name** field.
7. Choose **Create policy**. The system returns you to the **Policies** page.

Step 2: Create the IAM role for rollback based on CloudWatch alarms

Use the following procedure to create an IAM role and assign the policy you created in the previous procedure to it.

To create an IAM role for rollback based on CloudWatch alarms

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Immediately under **Choose the service that will use this role**, choose **EC2: Allows EC2 instances to call AWS services on your behalf**, and then choose **Next: Permissions**.
5. On the **Attached permissions policy** page, search for **SSMCloudWatchAlarmDiscoveryPolicy**.
6. Choose this policy and then choose **Next: Tags**.
7. Enter tags for this role, and then choose **Next: Review**.
8. On the **Create role** page, enter **`SSMCloudWatchAlarmDiscoveryRole`** in the **Role name** field, and then choose **Create role**.
9. On the **Roles** page, choose the role you just created. The **Summary** page opens.

Step 3: Add a trust relationship

Use the following procedure to configure the role you just created to trust AWS AppConfig.

To add a trust relationship for AWS AppConfig

1. In the **Summary** page for the role you just created, choose the **Trust Relationships** tab, and then choose **Edit Trust Relationship**.
2. Edit the policy to include only "`appconfig.amazonaws.com`", as shown in the following example:

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

Step 2: Create the IAM role for rollback based on CloudWatch alarms 12

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. Choose **Update Trust Policy**.

Creating feature flags and free form configuration data in AWS AppConfig

The topics in this section help you complete the following tasks in AWS AppConfig. These tasks create important artifacts for deploying configuration data.

1. **Create an application namespace**

   To create an application namespace, you create an AWS AppConfig artifact called an application. An application is simply an organizational construct like a folder.

2. **Create environments**

   For each AWS AppConfig application, you define one or more environments. An environment is a logical deployment group of AWS AppConfig targets, such as applications in a `Beta` or `Production` environment. You can also define environments for application subcomponents, such as `AWS Lambda functions`, `Containers`, `Web`, `Mobile`, and `Back-end`.

   You can configure Amazon CloudWatch alarms for each environment to automatically rollback problematic configuration changes. The system monitors alarms during a configuration deployment. If an alarm is triggered, the system rolls back the configuration.

3. **Create a configuration profile**

   Configuration data is a collection of settings that influence the behavior of your application. A *configuration profile* includes, among other things, a URI that enables AWS AppConfig to locate your configuration data in its stored location and a profile type. AWS AppConfig supports two configuration profile types: feature flags and freeform configurations. Feature flag configuration profiles store their data in the AWS AppConfig hosted configuration store and the URI is simply `hosted`. For freeform configuration profiles, you can store your data in the AWS AppConfig hosted configuration store or another Systems Manager capability or AWS service that integrates with AWS AppConfig, as described in Creating a free form configuration profile in AWS AppConfig.

   A configuration profile can also include optional validators to ensure your configuration data is syntactically and semantically correct. AWS AppConfig performs a check using the validators when you start a deployment. If any errors are detected, the deployment stops before making any changes to the targets of the configuration.

Note

Unless you have specific needs for storing secrets in AWS Secrets Manager or managing data in Amazon Simple Storage Service (Amazon S3), we recommend hosting your configuration data in the AWS AppConfig hosted configuration store as it offers the most features and enhancements.

The following section includes feature flag and freeform configuration data samples.

Feature flag configuration data

The following feature flag configuration data enables or disables mobile payments and default payments on a per-region basis.

JSON

```
{
  "allow_mobile_payments": {
    "enabled": false
  },
  "default_payments_per_region": {
    "enabled": true
  }
}
```

YAML

```
---
allow_mobile_payments:
  enabled: false
default_payments_per_region:
  enabled: true
```

Operational configuration data

The following freeform configuration data enforces limits on how an application processes requests.

JSON

```
{
  "throttle-limits": {
    "enabled": "true",
    "throttles": [
      {
        "simultaneous_connections": 12
      },
      {
        "tps_maximum": 5000
      }
    ],
    "limit-background-tasks": [
      true
    ]
  }
}
```

YAML

```
---
throttle-limits:
  enabled: 'true'
  throttles:
  - simultaneous_connections: 12
  - tps_maximum: 5000
  limit-background-tasks:
  - true
```

Access control list configuration data

The following access control list freeform configuration data specifies which users or groups can access an application.

JSON

```
{
  "allow-list": {
    "enabled": "true",
    "cohorts": [
      {
        "internal_employees": true
      },
      {
        "beta_group": false
      },
      {
        "recent_new_customers": false
      },
      {
        "user_name": "Jane_Doe"
      },
      {
        "user_name": "John_Doe"
      }
    ]
  }
}
```

YAML

```
---
allow-list:
  enabled: 'true'
  cohorts:
  - internal_employees: true
  - beta_group: false
  - recent_new_customers: false
  - user_name: Jane_Doe
  - user_name: Ashok_Kumar
```

Topics

About the configuration profile IAM role

You can create the IAM role that provides access to the configuration data by using AWS AppConfig. Or you can create the IAM role yourself. If you create the role by using AWS AppConfig, the system creates the role and specifies one of the following permissions policies, depending on which type of configuration source you choose.

Configuration source is a Secrets Manager secret

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
             ],
            "Resource": [
                "arn:aws:secretsmanager:AWS Region:account_ID:secret:secret_name-
a1b2c3"
            ]
        }
    ]
}
```

Configuration source is a Parameter Store parameter

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": [
                "arn:aws:ssm:AWS Region:account_ID:parameter/parameter_name"
            ]
        }
    ]
    }
```

Configuration source is an SSM document

```
{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetDocument"
            ],
            "Resource": [
                "arn:aws:ssm:AWS Region:account_ID:document/document_name"
            ]
        }
    ]
}
```

If you create the role by using AWS AppConfig, the system also creates the following trust relationship for the role.

```
{

  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating a namespace for your application in AWS AppConfig

The procedures in this section help you create an AWS AppConfig artifact called an application. An application is simply an organizational construct like a folder that identifies the namespace of your application. This organizational construct has a relationship with some unit of executable code. For example, you could create an application called MyMobileApp to organize and manage configuration data for a mobile application installed by your users. You must create these artifacts before you can use AWS AppConfig to deploy and retrieve feature flags or free form configuration data.

Note

You can use AWS CloudFormation to create AWS AppConfig artifacts, including applications, environments, configuration profiles, deployments, deployment strategies, and hosted configuration versions. For more information, see AWS AppConfig resource type reference in the *AWS CloudFormation User Guide*.

Topics

Creating an AWS AppConfig application (console)

Use the following procedure to create an AWS AppConfig application by using the AWS Systems Manager console.

To create an application

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Applications**, and then choose **Create application**.
3. For **Name**, enter a name for the application.
4. For **Description**, enter information about the application.
5. (Optional) In the **Extensions** section, choose an extension from the list. For more information, see About AWS AppConfig extensions.
6. (Optional) In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
7. Choose **Create application**.

AWS AppConfig creates the application and then displays the **Environments** tab. Proceed to Creating environments for your application in AWS AppConfig.

Creating an AWS AppConfig application (command line)

The following procedure describes how to use the AWS CLI (on Linux or Windows) or AWS Tools for PowerShell to create an AWS AppConfig application.

To create an application step by step

1. Open the AWS CLI.
2. Run the following command to create an application.

   Linux

   ```
   aws appconfig create-application \
     --name A_name_for_the_application \
     --description A_description_of_the_application \
     --tags User_defined_key_value_pair_metadata_for_the_application
   ```

   Windows

   ```
   aws appconfig create-application ^
     --name A_name_for_the_application ^
     --description A_description_of_the_application ^
     --tags User_defined_key_value_pair_metadata_for_the_application
   ```

   PowerShell

   ```
   New-APPCApplication `
     -Name Name_for_the_application `
     -Description Description_of_the_application `
     -Tag Hashtable_type_user_defined_key_value_pair_metadata_for_the_application
   ```

   The system returns information like the following.

   Linux

   ```
   {
      "Id": "Application ID",
      "Name": "Application name",
      "Description": "Description of the application"
   ```

```
}
```

Windows

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

PowerShell

```
ContentLength    : Runtime of the command
Description      : Description of the application
HttpStatusCode   : HTTP Status of the runtime
Id               : Application ID
Name             : Application name
ResponseMetadata : Runtime Metadata
```

Creating environments for your application in AWS AppConfig

For each AWS AppConfig application, you define one or more environments. An environment is a logical deployment group of AppConfig targets, such as applications in a `Beta` or `Production` environment, AWS Lambda functions, or containers. You can also define environments for application subcomponents, such as the `Web`, `Mobile`, and `Back-end`. You can configure Amazon CloudWatch alarms for each environment. The system monitors alarms during a configuration deployment. If an alarm is triggered, the system rolls back the configuration.

Before You Begin

If you want to enable AWS AppConfig to roll back a configuration in response to a CloudWatch alarm, then you must configure an AWS Identity and Access Management (IAM) role with permissions to enable AWS AppConfig to respond to CloudWatch alarms. You choose this role in the following procedure. For more information, see Configure permissions for automatic rollback.

Topics

Creating an AWS AppConfig environment (console)

Use the following procedure to create an AWS AppConfig environment by using the AWS Systems Manager console.

To create an environment

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Applications**, and then choose the name of an application to open the details page.
3. Choose the **Environments** tab, and then choose **Create environment**.
4. For **Name**, enter a name for the environment.
5. For **Description**, enter information about the environment.
6. (Optional) In the **Monitors** section, choose the **IAM role** field, and then choose an IAM role with permission to roll back a configuration if an alarm is triggered.
7. In the **CloudWatch alarms** list, choose one or more alarms to monitor. AWS AppConfig rolls back your configuration deployment if one of these alarms goes into an alarm state.
8. (Optional) In the **Associate extensions** section, choose an extension from the list. For more information, see About AWS AppConfig extensions.
9. (Optional) In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
10. Choose **Create environment**.

AWS AppConfig creates the environment and then displays the **Environment details** page. Proceed to Creating a configuration profile in AWS AppConfig.

Creating an AWS AppConfig environment (command line)

The following procedure describes how to use the AWS CLI (on Linux or Windows) or AWS Tools for PowerShell to create an AWS AppConfig environment.

To create an environment step by step

1. Open the AWS CLI.
2. Run the following command to create an environment.

Linux

```
aws appconfig create-environment \
  --application-id The_application_ID \
  --name A_name_for_the_environment \
  --description A_description_of_the_environment \
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" \
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

Windows

```
aws appconfig create-environment ^
  --application-id The_application_ID ^
  --name A_name_for_the_environment ^
  --description A_description_of_the_environment ^
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" ^
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

PowerShell

```
New-APPCEnvironment `
  -Name Name_for_the_environment `
  -ApplicationId The_application_ID
  -Description Description_of_the_environment `
  -Monitors
 @{"AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn"} `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_environment
```

The system returns information like the following.

Linux

```
{
   "ApplicationId": "The application ID",
   "Id": "The_environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment",
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

Windows

```
{
   "ApplicationId": "The application ID",
   "Id": "The environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment"
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

PowerShell

```
ApplicationId     : The application ID
ContentLength     : Runtime of the command
Description       : Description of the environment
HttpStatusCode    : HTTP Status of the runtime
Id                : The environment ID
Monitors          : {ARN of the Amazon CloudWatch alarm, ARN of the IAM role for
 AppConfig to monitor AlarmArn}
Name              : Name of the environment
Response Metadata : Runtime Metadata
State             : State of the environment
```

Proceed to Creating a configuration profile in AWS AppConfig.

Creating a configuration profile in AWS AppConfig

Configuration data is a collection of settings that influence the behavior of your application. A *configuration profile* includes, among other things, a URI that enables AWS AppConfig to locate your configuration data in its stored location and a configure type. AWS AppConfig supports two types of configuration profiles: feature flags and freeform configurations. A feature flag configuration stores data in the AWS AppConfig hosted configuration store and the URI is simply `hosted`. A freeform configuration can store data in the AWS AppConfig hosted configuration store, various Systems Manager capabilities, or an AWS service that integrates with AWS AppConfig. For more information, see Creating a free form configuration profile in AWS AppConfig.

A configuration profile can also include optional validators to ensure your configuration data is syntactically and semantically correct. AWS AppConfig performs a check using the validators when you start a deployment. If any errors are detected, the deployment stops before making any changes to the targets of the configuration.

Note

If possible, we recommend hosting your configuration data in the AWS AppConfig hosted configuration store as it offers the most features and enhancements.

Topics

About validators

When you create a configuration profile, you have the option to specify up to two validators. A validator ensures that your configuration data is syntactically and semantically correct. If you plan to use a validator, you must create it before you create the configuration profile. AWS AppConfig supports the following types of validators:

- **AWS Lambda functions**: Supported for feature flags and free form configurations.

- **JSON Schema**: Supported for free form configurations. (AWS AppConfig automatically validates feature flags against a JSON Schema.)

Topics

- AWS Lambda function validators
- JSON Schema validators

AWS Lambda function validators

Lambda function validators must be configured with the following event schema. AWS AppConfig uses this schema to invoke the Lambda function. The content is a base64-encoded string, and the URI is a string.

```
{
    "applicationId": "The application ID of the configuration profile being 
 validated",
    "configurationProfileId": "The ID of the configuration profile being validated",
    "configurationVersion": "The version of the configuration profile being validated",
    "content": "Base64EncodedByteString",
    "uri": "The configuration uri"
}
```

AWS AppConfig verifies that the Lambda `X-Amz-Function-Error` header is set in the response. Lambda sets this header if the function throws an exception. For more information about `X-Amz-Function-Error`, see Error Handling and Automatic Retries in AWS Lambda in the *AWS Lambda Developer Guide*.

Here is a simple example of a Lambda response code for a successful validation.

```
import json

def handler(event, context):
     #Add your validation logic here
     print("We passed!")
```

Here is a simple example of a Lambda response code for an unsuccessful validation.

```
def handler(event, context):
     #Add your validation logic here
```

```
        raise Exception("Failure!")
```

Here is another example that validates only if the configuration parameter is a prime number.

```
function isPrime(value) {
    if (value < 2) {
        return false;
    }

    for (i = 2; i < value; i++) {
        if (value % i === 0) {
            return false;
        }
    }

    return true;
}

exports.handler = async function(event, context) {
    console.log('EVENT: ' + JSON.stringify(event, null, 2));
    const input = parseInt(Buffer.from(event.content, 'base64').toString('ascii'));
    const prime = isPrime(input);
    console.log('RESULT: ' + input + (prime ? ' is' : ' is not') + ' prime');
    if (!prime) {
        throw input + "is not prime";
    }
}
```

AWS AppConfig calls your validation Lambda when calling the `StartDeployment` and `ValidateConfigurationActivity` API operations. You must provide `appconfig.amazonaws.com` permissions to invoke your Lambda. For more information, see Granting Function Access to AWS Services. AWS AppConfig limits the validation Lambda run time to 15 seconds, including start-up latency.

JSON Schema validators

If you create a configuration in an SSM document, then you must specify or create a JSON Schema for that configuration. A JSON Schema defines the allowable properties for each application configuration setting. The JSON Schema functions like a set of rules to ensure that new or updated configuration settings conform to the best practices required by your application. Here is an example.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "$id$",
  "description": "BasicFeatureToggle-1",
  "type": "object",
  "additionalProperties": false,
  "patternProperties": {
      "[^\\s]+$": {
          "type": "boolean"
      }
  },
  "minProperties": 1
}
```

When you create a configuration from an SSM document, the system automatically verifies that the configuration conforms to the schema requirements. If it doesn't, AWS AppConfig returns a validation error.

Important

Note the following important information about JSON Schema validators:

- Configuration data stored in SSM documents must validate against an associated JSON Schema before you can add the configuration to the system. SSM parameters do not require a validation method, but we recommend that you create a validation check for new or updated SSM parameter configurations by using AWS Lambda.
- A configuration in an SSM document uses the `ApplicationConfiguration` document type. The corresponding JSON Schema, uses the `ApplicationConfigurationSchema` document type.
- AWS AppConfig supports JSON Schema version 4.X for inline schema. If your application configuration requires a different version of JSON Schema, then you must create a Lambda validator.

Creating a feature flag configuration profile in AWS AppConfig

You can use feature flags to enable or disable features within your applications or to configure different characteristics of your application features using flag attributes. AWS AppConfig stores feature flag configurations in the AWS AppConfig hosted configuration store in a feature flag format that contains data and metadata about your flags and the flag attributes. For more information about the AWS AppConfig hosted configuration store, see About the AWS AppConfig hosted configuration store section.

Topics

Before you begin

In the following procedure, in the optional **Encryption** section, you can choose an AWS Key Management Service (AWS KMS) key. This customer managed key enables you to encrypt new configuration data versions in the AWS AppConfig hosted configuration store. For more information about this key, see **AWS AppConfig supports customer manager keys** in Security in AWS AppConfig.

The following procedure also gives you the option to associate an extension with a feature flag configuration profile. An extension augments your ability to inject logic or behavior at different points during the AWS AppConfig workflow of creating or deploying a configuration. For more information, see About AWS AppConfig extensions.

Lastly, in the **Feature flag attributes** section, when you enter the attribute details of a new feature flag, you can specify constraints. Constraints ensure that any unexpected attribute values are not deployed to your application. AWS AppConfig supports the following types of flag attributes and their corresponding constraints.

Type	Constraint	Description
String	Regular Expression	Regex pattern for the string
	Enum	List of acceptable values for the string
Number	Minimum	Minimum numeric value for the attribute

Type	Constraint	Description
	Maximum	Maximum numeric value for the attribute
Boolean	None	None
String array	Regular Expression	Regex pattern for the elements of the array
	Enum	List of acceptable values for the elements of the array
Number array	Minimum	Minimum numeric value for the elements of the array
	Maximum	Maximum numeric value for the elements of the array

Creating a feature flag configuration profile (console)

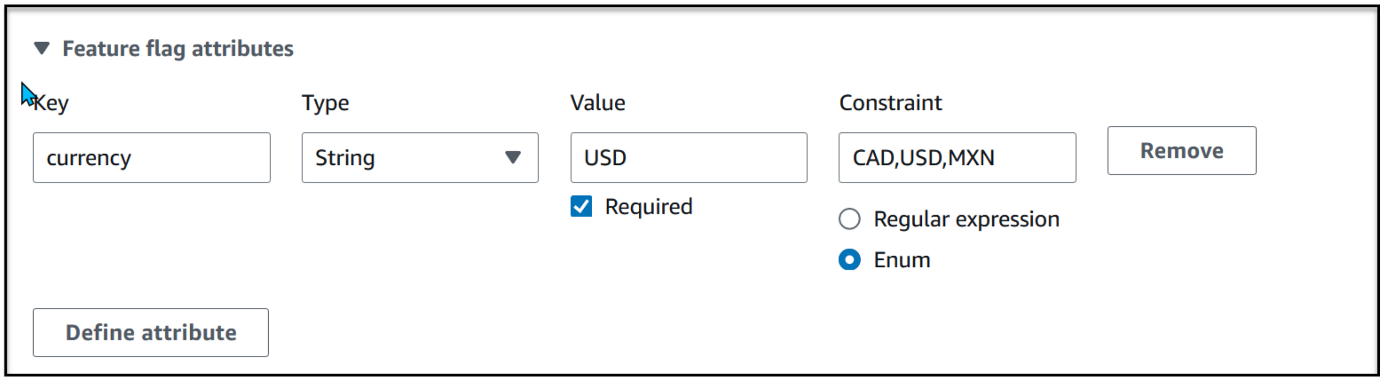
Use the following procedure to create an AWS AppConfig feature flag configuration profile by using the AWS AppConfig console.

To create a configuration profile

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Applications**, and then choose an application you created in Creating a namespace for your application in AWS AppConfig.
3. Choose the **Configuration profiles and feature flags** tab, and then choose **Create configuration**.
4. In the **Configuration options** section, choose **Feature flag**.
5. Scroll down. In the **Configuration profile** section, for **Configuration profile name**, enter a name.
6. (Optional) Expand **Description** and enter a description.
7. (Optional) Expand **Additional options** and complete the following, as necessary.

   a. In the **Encryption** list, choose an AWS Key Management Service (AWS KMS) key from the list.
   b. In the **Associate extensions** section, choose an extension from the list.
   c. In the **Tags** section, choose **Add new tag**, and then specify a key and optional value.

8. Choose **Next**.
9. In the **Feature flag definition** section, for **Flag name**, enter a name.
10. For **Flag key** enter a flag identifier to distinguish flags within the same configuration profile. Flags within the same configuration profile can't have the same key. After the flag is created, you can edit the flag name, but not the flag key.
11. (Optional) Expand **Description** and enter information about this flag.
12. Select **This is a short-term flag** and optionally choose a date for when the flag should be disabled or deleted. Note that AWS AppConfig does *not* disable the flag.
13. In the **Flag attributes** section, choose **Define attribute**. Attributes enable you to provide additional values within your flag.
14. For **Key**, specify a flag key and choose its type from the **Type** list. You can optionally validate attribute values against specified constraints. The following image shows an example.



Key	Type	Value	Constraint
currency	String	USD	CAD,USD,MXN
		✓ Required	Regular expression [✓ Enum]

Choose **Define attribute** to add additional attributes.

> **Note**
>
> Note the following information.
>
> - For attribute names, the word "enabled" is reserved. You can't create a feature flag attribute called "enabled". There are no other reserved words.
> - The attributes of a feature flag are only included in the `GetLatestConfiguration` response if that flag is enabled.

- Flag attribute keys for a given flag must be unique.
- Select **Required value** to specify whether an attribute value is required.

15. In the **Feature flag value** section, choose **Enabled** to enable the flag. Use this same toggle to disable a flag when it reaches a specified deprecration date, if applicable.
16. Choose **Next**.
17. On the **Review and save** page, verify the details of the flag and then **Save and continue to deploy**.

Proceed to Deploying feature flags and configuration data in AWS AppConfig.

Creating a feature flag and a feature flag configuration profile (command line)

The following procedure describes how to use the AWS Command Line Interface (on Linux or Windows) or Tools for Windows PowerShell to create an AWS AppConfig feature flag configuration profile. If you prefer, you can use AWS CloudShell to run the commands listed below. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.

To create a feature flags configuration step by step

1. Open the AWS CLI.
2. Create a feature flag configuration profile specifying its **Type** as `AWS.AppConfig.FeatureFlags`. The configuration profile must use `hosted` for the location URI.

   Linux

   ```
   aws appconfig create-configuration-profile \
     --application-id The_application_ID \
     --name A_name_for_the_configuration_profile \
     --location-uri hosted \
     --type AWS.AppConfig.FeatureFlags
   ```

   Windows

   ```
   aws appconfig create-configuration-profile ^
     --application-id The_application_ID ^
     --name A_name_for_the_configuration_profile ^
   ```

```
  --location-uri hosted ^
  --type AWS.AppConfig.FeatureFlags
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -LocationUri hosted `
  -Type AWS.AppConfig.FeatureFlags
```

3. Create your feature flag configuration data. Your data must be in a JSON format and conform to the `AWS.AppConfig.FeatureFlags` JSON schema. For more information about the schema, see Type reference for AWS.AppConfig.FeatureFlags.
4. Use the `CreateHostedConfigurationVersion` API to save your feature flag configuration data to AWS AppConfig.

   Linux

```
aws appconfig create-hosted-configuration-version \
  --application-id The_application_ID \
  --configuration-profile-id The_configuration_profile_id \
  --content-type "application/json" \
  --content file://path/to/feature_flag_configuration_data \
  file_name_for_system_to_store_configuration_data
```

   Windows

```
aws appconfig create-hosted-configuration-version ^
  --application-id The_application_ID ^
  --configuration-profile-id The_configuration_profile_id ^
  --content-type "application/json" ^
  --content file://path/to/feature_flag_configuration_data ^
  file_name_for_system_to_store_configuration_data
```

   PowerShell

```
New-APPCHostedConfigurationVersion `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_id `
  -ContentType "application/json" `
  -Content file://path/to/feature_flag_configuration_data `
  file_name_for_system_to_store_configuration_data
```

Here's a Linux sample command.

```
aws appconfig create-hosted-configuration-version \
  --application-id 1a2b3cTestApp \
  --configuration-profile-id 4d5e6fTestConfigProfile \
  --content-type "application/json" \
  --content Base64Content
```

The `content` parameter uses the following `base64` encoded data.

```
{
  "flags": {
    "flagkey": {
      "name": "WinterSpecialBanner"
    }
  },
  "values": {
    "flagkey": {
      "enabled": true
    }
  },
  "version": "1"
}
```

The system returns information like the following.

Linux

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

Windows

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

PowerShell

```
ApplicationId          : 1a2b3cTestApp
ConfigurationProfileId : 4d5e6fTestConfigProfile
VersionNumber          : 1
ContentType            : application/json
```

The `service_returned_content_file` contains your configuration data that includes some AWS AppConfig generated metadata.

Note

When you create the hosted configuration version, AWS AppConfig verifies that your data conforms to the `AWS.AppConfig.FeatureFlags` JSON schema. AWS AppConfig additionally validates that each feature flag attribute in your data satisfies the constraints you defined for those attributes.

Type reference for AWS.AppConfig.FeatureFlags

Use the `AWS.AppConfig.FeatureFlags` JSON schema as a reference to create your feature flag configuration data.

```
{
      "$schema": "http://json-schema.org/draft-07/schema#",
      "definitions": {
        "flagSetDefinition": {
          "type": "object",
          "properties": {
            "version": {
```

```
            "$ref": "#/definitions/flagSchemaVersions"
          },
          "flags": {
            "$ref": "#/definitions/flagDefinitions"
          },
          "values": {
            "$ref": "#/definitions/flagValues"
          }
        },
        "required": ["version", "flags"],
        "additionalProperties": false
      },
      "flagDefinitions": {
        "type": "object",
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-]{0,63}$": {
            "$ref": "#/definitions/flagDefinition"
          }
        },
        "maxProperties": 100,
        "additionalProperties": false
      },
      "flagDefinition": {
        "type": "object",
        "properties": {
          "name": {
            "$ref": "#/definitions/customerDefinedName"
          },
          "description": {
            "$ref": "#/definitions/customerDefinedDescription"
          },
          "_createdAt": {
            "type": "string"
          },
          "_updatedAt": {
            "type": "string"
          },
          "_deprecation": {
            "type": "object",
            "properties": {
              "status": {
                "type": "string",
                "enum": ["planned"]
              }
            },
           "additionalProperties": false
          },
          "attributes": {
            "$ref": "#/definitions/attributeDefinitions"
          }
        },
        "additionalProperties": false
      },
      "attributeDefinitions": {
        "type": "object",
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-_]{0,63}$": {
            "$ref": "#/definitions/attributeDefinition"
          }
        },
        "maxProperties": 25,
        "additionalProperties": false
      },
      "attributeDefinition": {
        "type": "object",
        "properties": {
          "description": {
            "$ref": "#/definitions/customerDefinedDescription"
          },
          "constraints": {
            "oneOf": [
              { "$ref": "#/definitions/numberConstraints" },
              { "$ref": "#/definitions/stringConstraints" },
              { "$ref": "#/definitions/arrayConstraints" },
              { "$ref": "#/definitions/boolConstraints" }
            ]
          }
        },
        "additionalProperties": false
      },
      "flagValues": {
        "type": "object",
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-_]{0,63}$": {
            "$ref": "#/definitions/flagValue"
          }
        },
        "maxProperties": 100,
        "additionalProperties": false
      },
      "flagValue": {
        "type": "object",
        "properties": {
          "enabled": {
            "type": "boolean"
          },
          "_createdAt": {
            "type": "string"
          },
          "_updatedAt": {
            "type": "string"
          }
        },
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-_]{0,63}$": {
            "$ref": "#/definitions/attributeValue",
            "maxProperties": 25
          }
        },
        "required": ["enabled"],
        "additionalProperties": false
      },
      "attributeValue": {
        "oneOf": [
          { "type": "string", "maxLength": 1024 },
          { "type": "number" },
          { "type": "boolean" },
          {
            "type": "array",
            "oneOf": [
              {
                "items": {
                  "type": "string",
                  "maxLength": 1024
                }
              },
              {
                "items": {
                  "type": "number"
                }
              }
            ]
          }
        ],
        "additionalProperties": false
      },
      "stringConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["string"]
    },
          "required": {
            "type": "boolean"
          },
          "pattern": {
            "type": "string",
            "maxLength": 1024
          },
          "enum": {
            "type": "array",
            "maxLength": 100,
            "items": {
              "oneOf": [
                {
                  "type": "string",
                  "maxLength": 1024
                },
                {
                  "type": "integer"
                }
              ]
            }
          }
        },
        "required": ["type"],
        "not": {
          "required": ["pattern", "enum"]
        },
        "additionalProperties": false
      },
      "numberConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["number"]
    },
          "required": {
            "type": "boolean"
          },
          "minimum": {
            "type": "integer"
          },
          "maximum": {
            "type": "integer"
          }
        },
        "required": ["type"],
        "additionalProperties": false
      },
      "arrayConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["array"]
    },
          "required": {
            "type": "boolean"
          },
          "elements": {
            "$ref": "#/definitions/elementConstraints"
    }
        },
        "required": ["type"],
        "additionalProperties": false
      },
      "boolConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["boolean"]
    },
          "required": {
            "type": "boolean"
          }
        },
        "required": ["type"],
        "additionalProperties": false
      },
      "elementConstraints": {
        "oneOf": [
          { "$ref": "#/definitions/numberConstraints" },
          { "$ref": "#/definitions/stringConstraints" }
        ]
      },
      "customerDefinedName": {
        "type": "string",
        "pattern": "^[^\\n]{1,64}$"
      },
      "customerDefinedDescription": {
        "type": "string",
        "maxLength": 1024
      },
      "flagSchemaVersions": {
        "type": "string",
        "enum": ["1"]
      }
    },
    "type": "object",
    "$ref": "#/definitions/flagSetDefinition",
    "additionalProperties": false
  }
```

Important

To retrieve feature flag configuration data, your application must call the `GetLatestConfiguration` API. You can't retrieve feature flag configuration data by calling `GetConfiguration`, which is deprecated. For more information, see GetLatestConfiguration in the *AWS AppConfig API Reference*.

When your application calls GetLatestConfiguration and receives a newly deployed configuration, the information that defines your feature flags and attributes is removed. The simplified JSON contains a map of keys that match each of the flag keys you specified. The simplified JSON also contains mapped values of `true` or `false` for the `enabled` attribute. If a flag sets `enabled` to `true`, any attributes of the flag will be present as well. The following JSON schema describes the format of the JSON output.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "patternProperties": {
    "^[a-z][a-zA-Z\\d-_]{0,63}$": {
      "$ref": "#/definitions/attributeValuesMap"
    }
  },
  "maxProperties": 100,
  "additionalProperties": false,
  "definitions": {
    "attributeValuesMap": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        }
      },
      "required": ["enabled"],
      "patternProperties": {
        "^[a-z][a-zA-Z\\d-_]{0,63}$": {
          "$ref": "#/definitions/attributeValue"
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
    },
    "attributeValue": {
      "oneOf": [
        { "type": "string","maxLength": 1024 },
        { "type": "number" },
        { "type": "boolean" },
        {
          "type": "array",
          "oneOf": [
            {
              "items": {
                "oneOf": [
                  {
                    "type": "string",
                    "maxLength": 1024
                  }
                ]
              }
            },
            {
              "items": {
                "oneOf": [
                  {
                    "type": "number"
                  }
                ]
              }
            }
          ]
        }
      ],
      "additionalProperties": false
    }
  }
}
```

Create a feature flag configuration profile that uses variants

Feature flag variants enable you to list a set of possible flag values for a single feature flag. When requesting a flag, your application can provide additional context that is evaluated against a set of user-defined rules. Depending on the rules and context specified in the flag request, different flag values (variant data) is returned to the application. Flag variants are useful for high-cardinality segmentation. Flag variants are often used in application experimentation and rollouts based on user IDs.

Creating a free form configuration profile in AWS AppConfig

A *configuration profile* includes, among other things, a URI that enables AWS AppConfig to locate your configuration data in its stored location and a profile type. AWS AppConfig supports two configuration profile types: feature flags and freeform configurations. Feature flag configuration profiles store their data in the AWS AppConfig hosted configuration store and the URI is simply `hosted`. For freeform configuration profiles, you can store your data in the AWS AppConfig hosted configuration store or any of the following AWS services and Systems Manager capabilities:

Location	Supported file types
AWS AppConfig hosted configuration store	YAML, JSON, and text if added using the AWS Management Console. Any file type if added

Location	Supported file types
	using the AWS AppConfig CreateHostedConfigurationVersion API action.
Amazon Simple Storage Service (Amazon S3)	Any
AWS CodePipeline	Pipeline (as defined by the service)
AWS Secrets Manager	Secret (as defined by the service)
AWS Systems Manager Parameter Store	Standard and secure string parameters (as defined by Parameter Store)
AWS Systems Manager document store (SSM documents)	YAML, JSON, text

A configuration profile can also include optional validators to ensure your configuration data is syntactically and semantically correct. AWS AppConfig performs a check using the validators when you start a deployment. If any errors are detected, the deployment stops before making any changes to the targets of the configuration.

Note

If possible, we recommend hosting your configuration data in the AWS AppConfig hosted configuration store as it offers the most features and enhancements.

For freeform configurations stored in the AWS AppConfig hosted configuration store or SSM documents, you can create the freeform configuration by using the Systems Manager console at the time you create a configuration profile. The process is described later in this topic.

For freeform configurations stored in Parameter Store, Secrets Manager, or Amazon S3, you must create the parameter, secret, or object first and store it in the relevant configuration store. After you store the configuration data, use the procedure in this topic to create the configuration profile.

Topics

- About configurations stored in Amazon S3
- Creating a freeform configuration and configuration profile

About configuration store quotas and limitations

Configuration stores supported by AWS AppConfig have the following quotas and limitations.

	AWS AppConfig hosted configuration store	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager Document store	AWS CodePipeline
Configuration size limit	2 MB default, 4 MB maximum	2 MB Enforced by AWS AppConfig, not S3	4 KB (free tier) / 8 KB (advanced parameters)	64 KB	64 KB	2 MB Enforced by AWS AppConfig, not CodePipeline
Resource storage limit	1 GB	Unlimited	10,000 parameters (free tier) / 100,000 parameters (advanced parameters)	500,000	500 documents	Limited by the number of configuration profiles per application (100 profiles per application)

	AWS AppConfig hosted configuration store	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager Document store	AWS CodePipeline
Server-side encryption	Yes	SSE-S3, SSE-KMS	Yes	Yes	No	Yes
AWS CloudFormation support	Yes	Not for creating or updating data	Yes	Yes	No	Yes
Pricing	Free	See Amazon S3 pricing	See AWS Systems Manager pricing	See AWS Secrets Manager pricing	Free	See AWS CodePipeline pricing

About the AWS AppConfig hosted configuration store

AWS AppConfig includes an internal or hosted configuration store. Configurations must be 2 MB or smaller. The AWS AppConfig hosted configuration store provides the following benefits over other configuration store options.

- You don't need to set up and configure other services such as Amazon Simple Storage Service (Amazon S3) or Parameter Store.
- You don't need to configure AWS Identity and Access Management (IAM) permissions to use the configuration store.
- You can store configurations in YAML, JSON, or as text documents.
- There is no cost to use the store.
- You can create a configuration and add it to the store when you create a configuration profile.

About configurations stored in Amazon S3

You can store configurations in an Amazon Simple Storage Service (Amazon S3) bucket. When you create the configuration profile, you specify the URI to a single S3 object in a bucket. You also specify the Amazon Resource Name (ARN) of an AWS Identity and Access Management (IAM) role that gives AWS AppConfig permission to get the object. Before you create a configuration profile for an Amazon S3 object, be aware of the following restrictions.

Restriction	Details
Size	Configurations stored as S3 objects can be a maximum of 1 MB in size.
Object encryption	A configuration profile can target SSE-S3 and SSE-KMS encrypted objects.
Storage classes	AWS AppConfig supports the following S3 storage classes: `STANDARD`, `INTELLIGENT_TIERING`, `REDUCED_REDUNDANCY`, `STANDARD_IA`, and `ONEZONE_IA`. The following classes are not supported: All S3 Glacier classes (`GLACIER` and `DEEP_ARCHIVE`).
Versioning	AWS AppConfig requires that the S3 object use versioning.

Configuring permissions for a configuration stored as an Amazon S3 object

When you create a configuration profile for a configuration stored as an S3 object, you must specify an ARN for an IAM role that gives AWS AppConfig permission to get the object. The role must include the following permissions.

Permissions to access the S3 object

- s3:GetObject
- s3:GetObjectVersion

Permissions to list S3 buckets

s3:ListAllMyBuckets

Permissions to access the S3 bucket where the object is stored

- s3:GetBucketLocation
- s3:GetBucketVersioning
- s3:ListBucket
- s3:ListBucketVersions

Complete the following procedure to create a role that enables AWS AppConfig to get a configuration stored in an S3 object.

Creating the IAM Policy for Accessing an S3 Object

Use the following procedure to create an IAM policy that enables AWS AppConfig to get a configuration stored in an S3 object.

To create an IAM policy for accessing an S3 object

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **Create policy** page, choose the **JSON** tab.
4. Update the following sample policy with information about your S3 bucket and configuration object. Then paste the policy into the text field on the **JSON** tab. Replace the *placeholder values* with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/my-configurations/my-
configuration.json"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning",
        "s3:ListBucketVersions",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

5. Choose **Review policy**.
6. On the **Review policy** page, type a name in the **Name** box, and then type a description.
7. Choose **Create policy**. The system returns you to the **Roles** page.

Creating the IAM Role for Accessing an S3 Object

Use the following procedure to create an IAM role that enables AWS AppConfig to get a configuration stored in an S3 object.

To create an IAM role for accessing an Amazon S3 object

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. On the **Select type of trusted entity** section, choose **AWS service**.
4. In the **Choose a use case** section, under **Common use cases**, choose **EC2**, and then choose **Next: Permissions**.
5. On the **Attach permissions policy** page, in the search box, enter the name of the policy you created in the previous procedure.
6. Choose the policy and then choose **Next: Tags**.

7. On the **Add tags (optional)** page, enter a key and an optional value, and then choose **Next: Review**.
8. On the **Review** page, type a name in the **Role name** field, and then type a description.
9. Choose **Create role**. The system returns you to the **Roles** page.
10. On the **Roles** page, choose the role you just created to open the **Summary** page. Note the **Role Name** and **Role ARN**. You will specify the role ARN when you create the configuration profile later in this topic.

Creating a Trust Relationship

Use the following procedure to configure the role you just created to trust AWS AppConfig.

To add a trust relationship

1. In the **Summary** page for the role you just created, choose the **Trust Relationships** tab, and then choose **Edit Trust Relationship**.
2. Delete `"ec2.amazonaws.com"` and add `"appconfig.amazonaws.com"`, as shown in the following example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. Choose **Update Trust Policy**.

Creating a freeform configuration and configuration profile

This section describes how to create a freeform configuration and configuration profile. Before you begin, note the following information.

- The following procedure requires you to specify an IAM service role so that AWS AppConfig can access your configuration data in the configuration store you choose. This role is not required if you use the AWS AppConfig hosted configuration store. If you choose S3, Parameter Store, or the Systems Manager document store, then you must either choose an existing IAM role or choose the option to have the system automatically create the role for you. For more information, about this role, see About the configuration profile IAM role.
- The following procedure also gives you the option to associate an extension with a feature flag configuration profile. An extension augments your ability to inject logic or behavior at different points during the AWS AppConfig workflow of creating or deploying a configuration. For more information, see About AWS AppConfig extensions.
- If you want to create a configuration profile for configurations stored in S3, you must configure permissions. For more information about permissions and other requirements for using S3 as a configuration store, see About configurations stored in Amazon S3.
- If you want to use validators, review the details and requirements for using them. For more information, see About validators.

Topics

Creating an AWS AppConfig freeform configuration profile (console)

Use the following procedure to create an AWS AppConfig freeform configuration profile and (optionally) a freeform-configuration by using the AWS Systems Manager console.

To create a freeform configuration profile

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Applications**, and then choose an application you created in Creating a namespace for your application in AWS AppConfig.
3. Choose the **Configuration profiles and feature flags** tab, and then choose **Create configuration**.
4. In the **Configuration options** section, choose **Freeform configuration**.
5. For **Configuration profile name**, enter a name for the configuration profile.

6. (Optional) Expand **Description** and enter a description.
7. (Optional) Expand **Additional options** and complete the following, as necessary.

   a. In the **Associate extensions** section, choose an extension from the list.

   b. In the **Tags** section, choose **Add new tag**, and then specify a key and optional value.
8. Choose **Next**.
9. On the **Specify configuration data** page, in the **Configuration defition** section, choose an option.
10. Complete the fields for the option you selected, as described in the following table.

Option selected	Details
AWS AppConfig hosted configuration	Choose either **Text**, **JSON**, or **YAML**, and enter your configuration in the field. Go to Step 12 in this procedure.
Amazon S3 object	Enter the object URI in the **S3 object source** field and go to Step 11 in this procedure.
AWS CodePipeline	Choose **Next** and go to Step 12 in this procedure.
Secrets Manager secret	Choose the secret from the list go to Step 11 in this procedure.
AWS Systems Manager parameter	Choose the parameter from the list and go to Step 11 in this procedure.
AWS Systems Manager document	1. Choose a document from the list or choose **Create new document**. 2. If you choose **Create new document**, for **Document name**, enter a name. Optionall y, expand **Version name** and enter a name for the document version. 3. For **Application configuration schema**, either choose the JSON schema from the list or choose **Create schema**. If you

Option selected	Details
	choose **Create schema**, Systems Manager opens the **Create schema** page. Enter the schema details, and then choose **Create aplication configuration schema**. 4. In the **Content** section, choose either **YAML** or **JSON** and then enter the configuration data in the field.

11. In the **Service role** section, choose **New service role** to have AWS AppConfig create the IAM role that provides access to the configuration data. AWS AppConfig automatically populates the **Role name** field based on the name you entered earlier. Or, choose **Existing service role**. Choose the role by using the **Role ARN** list.
12. Optionally, on the **Add validators** page, choose either **JSON Schema** or **AWS Lambda**. If you choose **JSON Schema**, enter the JSON Schema in the field. If you choose **AWS Lambda**, choose the function Amazon Resource Name (ARN) and the version from the list.

    Important

    Configuration data stored in SSM documents must validate against an associated JSON Schema before you can add the configuration to the system. SSM parameters do not require a validation method, but we recommend that you create a validation check for new or updated SSM parameter configurations by using AWS Lambda.

13. Choose **Next**.
14. On the **Review and save** page, choose **Save and continue to deploy**.

Important

If you created a configuration profile for AWS CodePipeline, then you must create a pipeline in CodePipeline that specifies AWS AppConfig as the *deploy provider*. You don't need to perform Deploying feature flags and configuration data in AWS AppConfig. However, you must configure a client to receive application configuration updates as described in Retrieving configurations by directly calling APIs. For information about creating a pipeline that specifies AWS AppConfig as the deploy provider, see Tutorial:

Create a Pipeline that Uses AWS AppConfig as a Deployment Provider in the *AWS CodePipeline User Guide*.

Proceed to Deploying feature flags and configuration data in AWS AppConfig.

Creating an AWS AppConfig freeform configuration profile (command line)

The following procedure describes how to use the AWS CLI (on Linux or Windows) or AWS Tools for PowerShell to create an AWS AppConfig freeform configuration profile. If you prefer, you can use AWS CloudShell to run the commands listed below. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.

Note

For freeform configurations hosted in the AWS AppConfig hosted configuration store, you specify `hosted` for the location URI.

To create a configuration profile by using the AWS CLI

1. Open the AWS CLI.
2. Run the following command to create a freeform configuration profile.

   Linux

```
aws appconfig create-configuration-profile \
  --application-id The_application_ID \
  --name A_name_for_the_configuration_profile \
  --description A_description_of_the_configuration_profile \
  --location-uri A_URI_to_locate_the_configuration or hosted \
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 \
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile \
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id The_application_ID ^
  --name A_name_for_the_configuration_profile ^
  --description A_description_of_the_configuration_profile ^
  --location-uri A_URI_to_locate_the_configuration or hosted  ^
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 ^
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile ^
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -Description Description_of_the_configuration_profile `
  -LocationUri A_URI_to_locate_the_configuration or hosted `
  -
RetrievalRoleArn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_
 `

  -
Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_configuration_profile
 `

  -Validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Important

Note the following important information.

- If you created a configuration profile for AWS CodePipeline, then you must create a pipeline in CodePipeline that specifies AWS AppConfig as the *deploy provider*. You don't need to perform Deploying feature flags and configuration data in AWS AppConfig. However, you must configure a client to receive application configuration updates as described in Retrieving configurations by directly calling APIs. For information about creating a pipeline that specifies AWS AppConfig as the deploy provider, see Tutorial: Create a Pipeline that Uses AWS AppConfig as a Deployment Provider in the *AWS CodePipeline User Guide*.

- If you created a configuration in the AWS AppConfig hosted configuration store, you can create new versions of the configuration by using the CreateHostedConfigurationVersion API operations. To view AWS CLI details and sample commands for this API operation, see create-hosted-configuration-version in the *AWS CLI Command Reference*.

Proceed to Deploying feature flags and configuration data in AWS AppConfig.

Other sources of configuration data

This topic includes information about other AWS services that integrate with AWS AppConfig.

AWS AppConfig integration with AWS Secrets Manager

Secrets Manager helps you to securely encrypt, store, and retrieve credentials for your databases and other services. Instead of hardcoding credentials in your apps, you can make calls to Secrets Manager to retrieve your credentials whenever needed. Secrets Manager helps you protect access to your IT resources and data by enabling you to rotate and manage access to your secrets.

When you create a freeform configuration profile, you can choose Secrets Manager as the source of your configuration data. You must onboard with Secrets Manager and create a secret before you create the configuration profile. For more information about Secrets Manager, see What is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*. For information about creating a configuration profile that uses Secrets Manager, see Creating feature flags and free form configuration data in AWS AppConfig.

Deploying feature flags and configuration data in AWS AppConfig

After you create required artifacts for working with feature flags and freeform configuration data, you can create a new deployment. When you create a new deployment, you specify the following information:

- An application ID
- A configuration profile ID
- A configuration version
- An environment ID where you want to deploy the configuration data
- A deployment strategy ID that defines how fast you want the changes to take effect
- An AWS Key Management Service (AWS KMS) key ID to encrypt the data using a customer managed key.

When you call the StartDeployment API action, AWS AppConfig performs the following tasks:

1. Retrieves the configuration data from the underlying data store by using the location URI in the configuration profile.
2. Verifies the configuration data is syntactically and semantically correct by using the validators you specified when you created your configuration profile.
3. Caches a copy of the data so it is ready to be retrieved by your application. This cached copy is called the *deployed data*.

AWS AppConfig integrates with Amazon CloudWatch to monitor deployments. If a deployment initiates an alarm in CloudWatch, AWS AppConfig automatically rolls back the deployment to minimize impact on your application users.

Topics

Working with deployment strategies

A deployment strategy enables you to slowly release changes to production environments over minutes or hours. An AWS AppConfig deployment strategy defines the following important aspects of a configuration deployment.

Setting	Description
Deployment type	Deployment type defines how the configura tion deploys or *rolls out*. AWS AppConfig supports **Linear** and **Exponential** deployment types. • **Linear**: For this type, AWS AppConfig processes the deployment by increments of the growth factor evenly distributed over the deployment. Here's an example timeline for a 10 hour deployment that uses 20% linear growth: Elapsed time → Deployment progress: 0 hour → 0% 2 hour → 20% 4 hour → 40% 6 hour → 60% 8 hour → 80% 10 hour → 100% • **Exponential**: For this type, AWS AppConfig processes the deployment exponentially using the following formula: `G*(2^N)`. In this formula, G is the step percentage

Elapsed time	Deployment progress
0 hour	0%
2 hour	20%
4 hour	40%
6 hour	60%
8 hour	80%
10 hour	100%

Setting	Description

	specified by the user and N is the number of steps until the configuration is deployed to all targets. For example, if you specify a growth factor of 2, then the system rolls out the configuration as follows:
	`2*(2^0)`
	`2*(2^1)`
	`2*(2^2)`
	Expressed numerically, the deployment rolls out as follows: 2% of the targets, 4% of the targets, 8% of the targets, and continues until the configuration has been deployed to all targets.

Step percentage (growth factor)	This setting specifies the percentage of callers to target during each step of the deployment. **Note** In the SDK and the AWS AppConfig API Reference, `step percentage` is called `growth factor`.
Deployment time	This setting specifies an amount of time during which AWS AppConfig deploys to hosts. This is not a timeout value. It is a window of time during which the deployment is processed in intervals.

Setting	Description
Bake time	This setting specifies the amount of time AWS AppConfig monitors for Amazon CloudWatc h alarms after the configuration has been deployed to 100% of its targets, before considering the deployment to be complete. If an alarm is triggered during this time, AWS AppConfig rolls back the deploymen t. You must configure permissions for AWS AppConfig to roll back based on CloudWatch alarms. For more information, see Configure permissions for automatic rollback.

You can choose a predefined strategy included with AWS AppConfig or create your own.

Topics

- Predefined deployment strategies
- Create a deployment strategy

Predefined deployment strategies

AWS AppConfig includes predefined deployment strategies to help you quickly deploy a configuration. Instead of creating your own strategies, you can choose one of the following when you deploy a configuration.

Deployment strategy	Description
AppConfig.Linear20PercentEvery6Minutes	**AWS recommended**: This strategy deploys the configuration to 20% of all targets every six minutes for a 30 minute deployment. The system monitors for Amazon CloudWatch alarms for 30 minutes. If no alarms are received in this time, the deployment is complete. If an alarm is

Deployment strategy	Description
	triggered during this time, AWS AppConfig rolls back the deployment. We recommend using this strategy for production deployments because it aligns with AWS best practices and includes additiona l emphasis on deployment safety due to its long duration and bake time.
AppConfig.Canary10Percent20Minutes	**AWS recommended**: This strategy processes the deployment exponentially using a 10% growth factor over 20 minutes. The system monitors for CloudWatch alarms for 10 minutes. If no alarms are received in this time, the deployment is complete. If an alarm is triggered during this time, AWS AppConfig rolls back the deployment. We recommend using this strategy for production deployments because it aligns with AWS best practices for configuration deployments.
AppConfig.AllAtOnce	**Quick**: This strategy deploys the configuration to all targets immediately. The system monitors for CloudWatch alarms for 10 minutes. If no alarms are received in this time, the deployment is complete. If an alarm is triggered during this time, AWS AppConfig rolls back the deployment.

Deployment strategy	Description
AppConfig.Linear50PercentEvery30Seconds	**Testing/demonstration**: This strategy deploys the configuration to half of all targets every 30 seconds for a one-minute deployment. The system monitors for Amazon CloudWatch alarms for 1 minute. If no alarms are received in this time, the deployment is complete. If an alarm is triggered during this time, AWS AppConfig rolls back the deployment. We recommend using this strategy only for testing or demonstration purposes because it has a short duration and bake time.

Create a deployment strategy

If you don't want to use one of the predefined deployment strategies, you can create your own. You can create a maximum of 20 deployment strategies. When you deploy a configuration, you can choose the deployment strategy that works best for the application and the environment.

Creating an AWS AppConfig deployment strategy (console)

Use the following procedure to create an AWS AppConfig deployment strategy by using the AWS Systems Manager console.

To create a deployment strategy

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Deployment strategies**, and then choose **Create deployment strategy**.
3. For **Name**, enter a name for the deployment strategy.
4. For **Description**, enter information about the deployment strategy.
5. For **Deployment type**, choose a type.

6. For **Step percentage**, choose the percentage of callers to target during each step of the deployment.
7. For **Deployment time**, enter the total duration for the deployment in minutes or hours.
8. For **Bake time**, enter the total time, in minutes or hours, to monitor for Amazon CloudWatch alarms before proceeding to the next step of a deployment or before considering the deployment to be complete.
9. In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
10. Choose **Create deployment strategy**.

Important

If you created a configuration profile for AWS CodePipeline, then you must create a pipeline in CodePipeline that specifies AWS AppConfig as the *deploy provider*. You don't need to perform Deploying a configuration. However, you must configure a client to receive application configuration updates as described in Retrieving configurations by directly calling APIs. For information about creating a pipeline that specifies AWS AppConfig as the deploy provider, see Tutorial: Create a Pipeline that Uses AWS AppConfig as a Deployment Provider in the *AWS CodePipeline User Guide*.

Proceed to Deploying a configuration.

Creating an AWS AppConfig deployment strategy (command line)

The following procedure describes how to use the AWS CLI (on Linux or Windows) or AWS Tools for PowerShell to create an AWS AppConfig deployment strategy.

To create a deployment strategy step by step

1. Open the AWS CLI.
2. Run the following command to create a deployment strategy.

   Linux

```
aws appconfig create-deployment-strategy \
  --name A_name_for_the_deployment_strategy \
  --description A_description_of_the_deployment_strategy \
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 \
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 \
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 \
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 \
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document \
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

Windows

```
aws appconfig create-deployment-strategy ^
  --name A_name_for_the_deployment_strategy ^
  --description A_description_of_the_deployment_strategy ^
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 ^
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 ^
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 ^
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 ^
  --name A_name_for_the_deployment_strategy ^
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document ^
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

PowerShell

```
New-APPCDeploymentStrategy `
  --Name A_name_for_the_deployment_strategy `
  --Description A_description_of_the_deployment_strategy `
  --DeploymentDurationInMinutes Total_amount_of_time_for_a_deployment_to_last `
  --FinalBakeTimeInMinutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 `

  --
GrowthFactor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_i
 `

  --
GrowthType The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_
 `

  --
ReplicateTo To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document
 `

  --
Tag Hashtable_type_User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

The system returns information like the following.

Linux

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
   "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
   "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

Windows

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
  "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
  "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
  "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
  "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

PowerShell

```
ContentLength                : Runtime of the command
DeploymentDurationInMinutes : Total amount of time the deployment lasted
Description                  : Description of the deployment strategy
FinalBakeTimeInMinutes       : The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete
GrowthFactor                 : The percentage of targets that received a deployed
 configuration during each interval
GrowthType                   : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode               : HTTP Status of the runtime
Id                           : The deployment strategy ID
Name                         : Name of the deployment strategy
ReplicateTo                  : The Systems Manager (SSM) document where the
 deployment strategy is saved
ResponseMetadata             : Runtime Metadata
```

Deploying a configuration

After you create required artifacts for working with feature flags and freeform configuration data, you can create a new deployment by using the AWS Management Console, the AWS CLI, or the SDK. Starting a deployment in AWS AppConfig calls the StartDeployment API operation. This call includes the IDs of the AWS AppConfig application, the environment, the configuration profile, and (optionally) the configuration data version to deploy. The call also includes the ID of the deployment strategy to use, which determines how the configuration data is deployed.

If you deploy secrets stored in AWS Secrets Manager, Amazon Simple Storage Service (Amazon S3) objects encrypted with a customer managed key, or secure string parameters stored in AWS Systems Manager Parameter Store encrypted with a customer managed key, you must specify a value for the `KmsKeyIdentifier` parameter. If your configuration is not encrypted or is encrypted with an AWS managed key, specifying a value for the `KmsKeyIdentifier` parameter is not required.

Note

The value you specify for `KmsKeyIdentifier` must be a customer managed key. This doesn't have to be the same key you used to encrypt your configuration.
When you start a deployment with a `KmsKeyIdentifier`, the permission policy attached to your AWS Identity and Access Management (IAM) principal must allow the `kms:GenerateDataKey` operation.

AWS AppConfig monitors the distribution to all hosts and reports status. If a distribution fails, then AWS AppConfig rolls back the configuration.

Note

You can only deploy one configuration at a time to an environment. However, you can deploy one configuration each to different environments at the same time.

Deploy a configuration (console)

Use the following procedure to deploy an AWS AppConfig configuration by using the AWS Systems Manager console.

To deploy a configuration by using the console

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **Applications**, and then choose an application you created in Creating a namespace for your application in AWS AppConfig.
3. On the **Environments** tab, fill the radio button for an environment, and then choose **View details**.
4. Choose **Start deployment**.
5. For **Configuration**, choose a configuration from the list.

6. Depending on the source of your configuration, use the version list to choose the version you want to deploy.
7. For **Deployment strategy**, choose a strategy from the list.
8. (Optional) For **Deployment description**, enter a description.
9. For **Additional encryption options**, choose a AWS Key Management Service key from the list.
10. (Optional) In the **Tags** section, choose **Add new tag** and enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
11. Choose **Start deployment**.

Deploy a configuration (commandline)

The following procedure describes how to use the AWS CLI (on Linux or Windows) or AWS Tools for PowerShell to deploy an AWS AppConfig configuration.

To deploy a configuration step by step

1. Open the AWS CLI.
2. Run the following command to deploy a configuration.

   Linux

   ```
   aws appconfig start-deployment \
     --application-id The_application_ID \
     --environment-id The_environment_ID \
     --deployment-strategy-id The_deployment_strategy_ID \
     --configuration-profile-id The_configuration_profile_ID \
     --configuration-version The_configuration_version_to_deploy \
     --description A_description_of_the_deployment \
     --tags User_defined_key_value_pair_metadata_of_the_deployment
   ```

   Windows

   ```
   aws appconfig start-deployment ^
     --application-id The_application_ID ^
     --environment-id The_environment_ID ^
     --deployment-strategy-id The_deployment_strategy_ID ^
     --configuration-profile-id The_configuration_profile_ID ^
     --configuration-version The_configuration_version_to_deploy ^
     --description A_description_of_the_deployment ^
   ```

```
  --tags User_defined_key_value_pair_metadata_of_the_deployment
```

PowerShell

```
Start-APPCDeployment `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_ID `
  -ConfigurationVersion The_configuration_version_to_deploy `
  -DeploymentStrategyId The_deployment_strategy_ID `
  -Description A_description_of_the_deployment `
  -EnvironmentId The_environment_ID `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_deployment
```

The system returns information like the following.

Linux

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
   "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
   "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
   "State": "The state of the deployment",

   "EventLog": [
      {
         "Description": "A description of the deployment event",
         "EventType": "The type of deployment event",
         "OccurredAt": The date and time the event occurred,
         "TriggeredBy": "The entity that triggered the deployment event"
      }
   ],

   "PercentageComplete": The percentage of targets for which the deployment is
 available,
   "StartedAt": The time the deployment started,
   "CompletedAt": The time the deployment completed
}
```

Windows

```
{
      "ApplicationId": "The ID of the application that was deployed",
      "EnvironmentId" : "The ID of the environment",
      "DeploymentStrategyId": "The ID of the deployment strategy that was
    deployed",
      "ConfigurationProfileId": "The ID of the configuration profile that was
    deployed",
      "DeploymentNumber": The sequence number of the deployment,
      "ConfigurationName": "The name of the configuration",
      "ConfigurationLocationUri": "Information about the source location of the
    configuration",
      "ConfigurationVersion": "The configuration version that was deployed",
      "Description": "The description of the deployment",
      "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
      "GrowthType": "The linear or exponential algorithm used to define how
    percentage grew over time",
      "GrowthFactor": The percentage of targets to receive a deployed configuration
    during each interval,
      "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
    considering the deployment to be complete,
      "State": "The state of the deployment",

      "EventLog": [
         {
            "Description": "A description of the deployment event",
            "EventType": "The type of deployment event",
            "OccurredAt": The date and time the event occurred,
            "TriggeredBy": "The entity that triggered the deployment event"
         }
```

```
   ],

   "PercentageComplete": The percentage of targets for which the deployment is
 available,
   "StartedAt": The time the deployment started,
   "CompletedAt": The time the deployment completed
}
```

PowerShell

```
ApplicationId                : The ID of the application that was deployed
CompletedAt                  : The time the deployment completed
ConfigurationLocationUri     : Information about the source location of the
 configuration
ConfigurationName            : The name of the configuration
ConfigurationProfileId       : The ID of the configuration profile that was
 deployed
ConfigurationVersion         : The configuration version that was deployed
ContentLength                : Runtime of the deployment
DeploymentDurationInMinutes : Total amount of time the deployment lasted
DeploymentNumber             : The sequence number of the deployment
DeploymentStrategyId         : The ID of the deployment strategy that was
 deployed
Description                  : The description of the deployment
EnvironmentId                : The ID of the environment that was deployed
EventLog                     : {Description : A description of the deployment
 event, EventType : The type of deployment event, OccurredAt : The date and time
 the event occurred,
         TriggeredBy : The entity that triggered the deployment event}
FinalBakeTimeInMinutes       : Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete
GrowthFactor                 : The percentage of targets to receive a deployed
 configuration during each interval
GrowthType                   : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode               : HTTP Status of the runtime
PercentageComplete           : The percentage of targets for which the deployment
 is available
ResponseMetadata             : Runtime Metadata
StartedAt                    : The time the deployment started
State                        : The state of the deployment
```

AWS AppConfig deployment integration with CodePipeline

AWS AppConfig is an integrated deploy action for AWS CodePipeline (CodePipeline). CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. CodePipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. For more information, see What is AWS CodePipeline?

The integration of AWS AppConfig with CodePipeline offers the following benefits:

- Customers who use CodePipeline to manage orchestration now have a lightweight means of deploying configuration changes to their applications without having to deploy their entire code base.
- Customers who want to use AWS AppConfig to manage configuration deployments but are limited because AWS AppConfig does not support their current code or configuration store, now have additional options. CodePipeline supports AWS CodeCommit, GitHub, and BitBucket (to name a few).

Note

AWS AppConfig integration with CodePipeline is only supported in AWS Regions where CodePipeline is available.

How integration works

You start by setting up and configuring CodePipeline. This includes adding your configuration to a CodePipeline-supported code store. Next, you set up your AWS AppConfig environment by performing the following tasks:

- Create a namespace and a configuration profile
- Choose a predefined deployment strategy or create your own

After you complete these tasks, you create a pipeline in CodePipeline that specifies AWS AppConfig as the *deploy provider*. You can then make a change to your configuration and upload it to your CodePipeline code store. Uploading the new configuration automatically starts a new deployment in CodePipeline. After the deployment completes, you can verify your changes. For information about creating a pipeline that specifies AWS AppConfig as the deploy provider, see Tutorial: Create a Pipeline That Uses AWS AppConfig as a Deployment Provider in the *AWS CodePipeline User Guide*.

Retrieving feature flags and configuration data in AWS AppConfig

Your application retrieves feature flags and free form configuration data by establishing a configuration session using the AWS AppConfig Data service. If you use one of the simplified retrieval methods described in this section, either the AWS AppConfig Agent Lambda extension or AWS AppConfig Agent manages a series of API calls and session tokens on your behalf. You configure AWS AppConfig Agent as a local host and have the agent poll AWS AppConfig for configuration updates. The agent calls the StartConfigurationSession and GetLatestConfiguration API actions and caches your configuration data locally. To retrieve the data, your application makes an HTTP call to the localhost server. AWS AppConfig Agent supports several use cases, as described in Simplified retrieval methods.

If you prefer, you can manually call these API actions to retrieve a configuration. The API process works as follows:

Your application establishes a configuration session using the `StartConfigurationSession` API action. Your session's client then makes periodic calls to `GetLatestConfiguration` to check for and retrieve the latest data available.

When calling `StartConfigurationSession`, your code sends identifiers (ID or name) of an AWS AppConfig application, environment, and configuration profile that the session tracks.

In response, AWS AppConfig provides an `InitialConfigurationToken` to be given to the session's client and used the first time it calls `GetLatestConfiguration` for that session.

When calling `GetLatestConfiguration`, your client code sends the most recent `ConfigurationToken` value it has and receives in response:

- `NextPollConfigurationToken`: the `ConfigurationToken` value to use on the next call to `GetLatestConfiguration`.
- The configuration: the latest data intended for the session. This may be empty if the client already has the latest version of the configuration.

This section includes the following information.

Contents

- About the AWS AppConfig data plane service
- Simplified retrieval methods
- Retrieving configurations by directly calling APIs

About the AWS AppConfig data plane service

On Nov 18, 2021, AWS AppConfig released a new data plane service. This service replaces the previous process of retrieving configuration data by using the `GetConfiguration` API action. The data plane service uses two new API actions, StartConfigurationSession and GetLatestConfiguration. The data plane service also uses new endpoints.

If you started using AWS AppConfig before January 28, 2022, the service might be calling the `GetConfiguration` API action directly or it might be using a client provided by AWS, such as the AWS AppConfig Agent Lambda extension, to call this API action. If you call the `GetConfiguration` API action directly, take steps to use the `StartConfigurationSession` and `GetLatestConfiguration` API actions. If you are using the AWS AppConfig Agent Lambda extension, see the section titled *How this change impacts the AWS AppConfig Agent Lambda extension* later in this topic.

The new data plane API actions offer the following benefits over the `GetConfiguration` API action, which is now deprecated.

1. You don't need to manage a `ClientID` parameter. With the data plane service, `ClientID` is managed internally by the session token created by `StartConfigurationSession`.
2. You no longer need to include `ClientConfigurationVersion` to indicate the cached version of your configuration data. With the data plane service, `ClientConfigurationVersion` is managed internally by the session token created by `StartConfigurationSession`.
3. The new dedicated endpoint for data plane API calls improves code structure by separating control plane and data plane calls.
4. The new data plane service improves future extensibility for data plane operations. By utilizing a configuration session that manages configuration data retrieval, the AWS AppConfig team can create more powerful enhancements in the future.

Migrating from `GetConfiguration` to `GetLatestConfiguration`

To start using the new data plane service, you need to update your code that calls the `GetConfiguration` API action. Start a configuration session by using the `StartConfigurationSession` API action, and then call the `GetLatestConfiguration` API action to retrieve configuration data. To improve performance, we recommend you cache your configuration data locally. For more information, see Retrieving configurations by directly calling APIs.

How this change impacts the AWS AppConfig Agent Lambda extension

This change has no direct impact on how the AWS AppConfig Agent Lambda extension works. Older versions of the AWS AppConfig Agent Lambda extension called the `GetConfiguration` API action on your behalf. Newer versions call the data plane API actions. If you are using the AWS AppConfig Lambda extension, we recommend you update your extension to the most recent Amazon Resource Name (ARN) and update permissions for the new API calls. For more information, see Retrieving configuration data using the AWS AppConfig Agent Lambda extension.

Simplified retrieval methods

AWS AppConfig offers several simplified methods for retrieving configuration data. If you use AWS AppConfig feature flags or free form configuration data in an AWS Lambda function, you can use the AWS AppConfig Agent Lambda extension to retrieve configurations. If you have applications running on Amazon EC2 instances, you can use AWS AppConfig Agent to retrieve configurations. AWS AppConfig Agent also supports applications running on Amazon Elastic Kubernetes Service (Amazon EKS) or Amazon Elastic Container Service (Amazon ECS) container images.

After you complete the initial set up, these methods of retrieving configuration data are simpler than directly calling AWS AppConfig APIs. They automatically implement best practices and may lower your cost of using AWS AppConfig as a result of fewer API calls to retrieve configurations.

Topics

Retrieving configuration data using the AWS AppConfig Agent Lambda extension

An AWS Lambda extension is a companion process that augments the capabilities of a Lambda function. An extension can start before a function is invoked, run in parallel with a function, and continue to run after a function invocation is processed. In essence, a Lambda extension is like a client that runs in parallel to a Lambda invocation. This parallel client can interface with your function at any point during its lifecycle.

If you use AWS AppConfig feature flags or other dynamic configuration data in a Lambda function, then we recommend that you add the AWS AppConfig Agent Lambda extension as a layer to your Lambda function. This makes calling feature flags simpler, and the extension itself includes best practices that simplify using AWS AppConfig while reducing costs. Reduced costs result from fewer API calls to the AWS AppConfig service and shorter Lambda function processing times. For more information about Lambda extensions, see Lambda extensions in the *AWS Lambda Developer Guide*.

Note

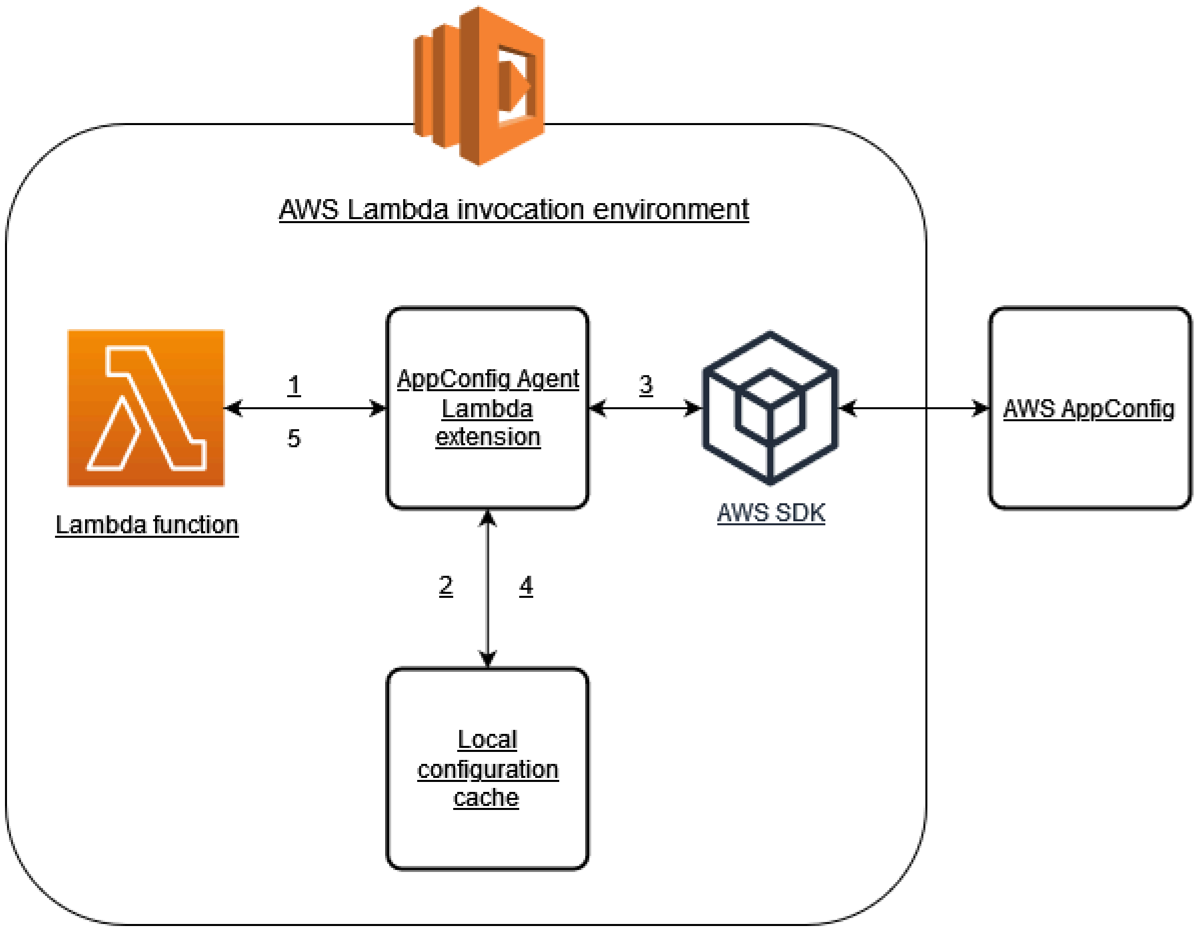
AWS AppConfig pricing is based on the number of times a configuration is called and received. Your costs increase if your Lambda performs multiple cold starts and retrieves new configuration data frequently.

This topic includes information about the AWS AppConfig Agent Lambda extension and the procedure for how to configure the extension to work with your Lambda function.

How it works

If you use AWS AppConfig to manage configurations for a Lambda function *without* Lambda extensions, then you must configure your Lambda function to receive configuration updates by integrating with the StartConfigurationSession and GetLatestConfiguration API actions.

Integrating the AWS AppConfig Agent Lambda extension with your Lambda function simplifies this process. The extension takes care of calling the AWS AppConfig service, managing a local cache of retrieved data, tracking the configuration tokens needed for the next service calls, and periodically checking for configuration updates in the background. The following diagram shows how it works.


AWS Lambda invocation environment
Lambda function
1
5
AppConfig Agent Lambda extension
3
AWS SDK
AWS AppConfig
2
4
Local configuration cache

1. You configure the AWS AppConfig Agent Lambda extension as a layer of your Lambda function.
2. To access its configuration data, your function calls the AWS AppConfig extension at an HTTP endpoint running on `localhost:2772`.
3. The extension maintains a local cache of the configuration data. If the data isn't in the cache, the extension calls AWS AppConfig to get the configuration data.
4. Upon receiving the configuration from the service, the extension stores it in the local cache and passes it to the Lambda function.
5. AWS AppConfig Agent Lambda extension periodically checks for updates to your configuration data in the background. Each time your Lambda function is invoked, the extension checks the elapsed time since it retrieved a configuration. If the elapsed time is greater than the configured poll interval, the extension calls AWS AppConfig to check for newly deployed data, updates the local cache if there has been a change, and resets the elapsed time.

Note

- Lambda instantiates separate instances corresponding to the concurrency level that your function requires. Each instance is isolated and maintains its own local cache of your configuration data. For more information about Lambda instances and concurrency, see Managing concurrency for a Lambda function.
- The amount of time it takes for a configuration change to appear in a Lambda function, after you deploy an updated configuration from AWS AppConfig, depends on the deployment strategy you used for the deployment and the polling interval you configured for the extension.

Before you begin

Before you enable the AWS AppConfig Agent Lambda extension, do the following:

- Organize the configurations in your Lambda function so that you can externalize them into AWS AppConfig.
- Create AWS AppConfig artifacts and configuration data, including feature flags or freeform configuration data. For more information, see Creating feature flags and free form configuration data in AWS AppConfig.
- Add `appconfig:StartConfigurationSession` and `appconfig:GetLatestConfiguration` to the AWS Identity and Access Management (IAM) policy used by the Lambda function execution role. For more information, see AWS Lambda execution role in the *AWS Lambda Developer Guide*. For more information about AWS AppConfig permissions, see Actions, resources, and condition keys for AWS AppConfig in the *Service Authorization Reference*.

Adding the AWS AppConfig Agent Lambda extension

To use the AWS AppConfig Agent Lambda extension, you need to add the extension to your Lambda. This can be done by adding the AWS AppConfig Agent Lambda extension to your Lambda function as a layer or by enabling the extension on a Lambda function as a container image.

Note

The AWS AppConfig extension is runtime agnostic and supports all runtimes.

Adding the AWS AppConfig Agent Lambda extension by using a layer and an ARN

To use the AWS AppConfig Agent Lambda extension, you add the extension to your Lambda function as a layer. For information about how to add a layer to your function, see Configuring extensions in the *AWS Lambda Developer Guide*. The name of the extension in the AWS Lambda console is **AWS-AppConfig-Extension**. Also note that when you add the extension as a layer to your Lambda, you must specify an Amazon Resource Name (ARN). Choose an ARN from one of the following lists that corresponds with the platform and AWS Region where you created the Lambda.

- x86-64 platform
- ARM64 platform

If you want to test the extension before you add it to your function, you can verify that it works by using the following code example.

```
import urllib.request


def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name'
    config = urllib.request.urlopen(url).read()
    return config
```

To test it, create a new Lambda function for Python, add the extension, and then run the Lambda function. After you run the Lambda function, the AWS AppConfig Lambda function returns the configuration you specified for the http://localhost:2772 path. For information about creating a Lambda function, see Create a Lambda function with the console in the *AWS Lambda Developer Guide*.

To add the AWS AppConfig Agent Lambda extension as a container image, see Using a container image to add the AWS AppConfig Agent Lambda extension.

Configuring the AWS AppConfig Agent Lambda extension

You can configure the extension by changing the following AWS Lambda environment variables. For more information, see Using AWS Lambda environment variables in the *AWS Lambda Developer Guide*.

Prefetching configuration data

The environment variable `AWS_APPCONFIG_EXTENSION_PREFETCH_LIST` can improve the start-up time of your function. When the AWS AppConfig Agent Lambda extension is initialized, it retrieves the specified configuration from AWS AppConfig before Lambda starts to initialize your function and invoke your handler. In some cases, the configuration data is already available in the local cache before your function requests it.

To use the prefetch capability, set the value of the environment variable to the path corresponding to your configuration data. For example, if your configuration corresponds to an application, environment, and configuration profile respectively named "my_application", "my_environment", and "my_configuration_data", the path would be `/applications/my_application/environments/my_environment/configurations/my_configuration_data`. You can specify multiple configuration items by listing them as a comma-separated list (If you have a resource name that includes a comma, use the resource's ID value instead of its name).

Accessing configuration data from another account

The AWS AppConfig Agent Lambda extension can retrieve configuration data from another account by specifying an IAM role that grants permissions to the data. To set this up, follow these steps:

1. In the account where AWS AppConfig is used to manage the configuration data, create a role with a trust policy that grants the account running the Lambda function access to the `appconfig:StartConfigurationSession` and `appconfig:GetLatestConfiguration` actions, along with the partial or complete ARNs corresponding to the AWS AppConfig configuration resources.
2. In the account running the Lambda function, add the `AWS_APPCONFIG_EXTENSION_ROLE_ARN` environment variable to the Lambda function with the ARN of the role created in step 1.
3. (Optional) If needed, an external ID can be specified using the `AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID` environment variable. Similarly, a session name can be configured using the `AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME` environment variable.

Note

Note the following information.

- The AWS AppConfig Agent Lambda extension can only retrieve data from one account. If you specify an IAM role, the extension will not be able to retrieve configuration data from the account in which the Lambda function is running.
- AWS Lambda logs information about the AWS AppConfig Agent Lambda extension and the Lambda function by using Amazon CloudWatch Logs.

Environment variable	Details	Default value
`AWS_APPCONFIG_EXTENSION_HTTP_PORT`	This environment variable specifies the port on which the local HTTP server that hosts the extension runs.	2772
`AWS_APPCONFIG_EXTENSION_LOG_LEVEL`	This environment variable specifies which AWS AppConfig extension-specific logs are sent to Amazon CloudWatch Logs for a function. Valid, case-insensitive values are: `debug`, `info`, `warn`, `error`, and `none`. Debug includes detailed information, including timing information, about the extension.	`info`
`AWS_APPCONFIG_EXTENSION_MAX_CONNECTIONS`	This environment variable configures the maximum number of connections the extension uses to retrieve configurations from AWS AppConfig.	3

Environment variable	Details	Default value
`AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS`	This environment variable controls how often the extension polls AWS AppConfig for an updated configuration in seconds.	45
`AWS_APPCONFIG_EXTENSION_POLL_TIMEOUT_MILLIS`	This environment variable controls the maximum amount of time, in milliseconds, the extension waits for a response from AWS AppConfig when refreshing data in the cache. If AWS AppConfig does not respond in the specified amount of time, the extension skips this poll interval and returns the previously updated cached data.	3000
`AWS_APPCONFIG_EXTENSION_PREFETCH_LIST`	This environment variable specifies the configuration data that the extension starts to retrieve before the function initializes and the handler runs. It can reduce the function's cold start time significantly.	None

Environment variable	Details	Default value
`AWS_APPCONFIG_EXTENSION_PROXY_HEADERS`	This environment variable specifies headers required by the proxy referenced in the `AWS_APPCONFIG_EXTENSION_PROXY_URL` environment variable. The value is a comma-separated list of headers. Each header uses the following form: `"header: value"`	None
`AWS_APPCONFIG_EXTENSION_PROXY_URL`	This environment variable specifies the proxy URL to use for connections from the AWS AppConfig extension to AWS services. HTTPS and HTTP URLs are supported.	None
`AWS_APPCONFIG_EXTENSION_ROLE_ARN`	This environment variable specifies the IAM role ARN corresponding to a role that should be assumed by the AWS AppConfig extension to retrieve configuration.	None
`AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID`	This environment variable specifies the external id to use in conjunction with the assumed role ARN.	None

Environment variable	Details	Default value
`AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME`	This environment variable specifies the session name to be associated with the credentials for the assumed IAM role.	None
`AWS_APPCONFIG_EXTENSION_SERVICE_REGION`	This environment variable specifies an alternative Region the extension should use to call the AWS AppConfig service. When undefined, the extension uses the endpoint in the current Region.	None
`AWS_APPCONFIG_EXTENSION_MANIFEST`	This environment variable configures AWS AppConfig Agent to take advantage of additional per-configuration features like *multi-account retrievals* and *save configuration to disk*. You can enter one of the following values: • `"app:env:manifest-config"` • `"file:/fully/qualified/path/to/manifest.json"` For more information about these features, see Additional retrieval features.	true

Environment variable	Details	Default value
AWS_APPCONFIG_EXTENSION_WAIT_ON_MANIFEST	This environment variable configures AWS AppConfig Agent to wait until the manifest is processed before completing startup.	true

Retrieving one or more flags from a feature flag configuration

For feature flag configurations (configurations of type `AWS.AppConfig.FeatureFlags`), the Lambda extension enables you to retrieve a single flag or a subset of flags in a configuration. Retrieving one or two flags is useful if your Lambda only needs to use a few flags from the configuration profile. The following examples use Python.

Note

The ability to call a single feature flag or a subset of flags in a configuration is only available in the AWS AppConfig Agent Lambda extension version 2.0.45 and higher.

You can retrieve AWS AppConfig configuration data from a local HTTP endpoint. To access a specific flag or a list of flags, use the `?flag=`*`flag_name`* query parameter for an AWS AppConfig configuration profile.

To access a single flag and its attributes

```
import urllib.request

def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name'
    config = urllib.request.urlopen(url).read()
    return config
```

To access multiple flags and their attributes

```
import urllib.request
```

```
def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two'
    config = urllib.request.urlopen(url).read()
    return config
```

Viewing AWS AppConfig Agent Lambda extension logs

You can view log data for the AWS AppConfig Agent Lambda extension in the AWS Lambda logs. Log entries are prefaced with `appconfig agent`. Here's an example.

```
[appconfig agent] 2024/05/07 04:19:01 ERROR retrieve failure for
 'SourceEventConfig:SourceEventConfigEnvironment:SourceEventConfigProfile':
 StartConfigurationSession: api error AccessDenied: User:
 arn:aws:sts::0123456789:assumed-role/us-east-1-LambdaRole/extension1 is not authorized
 to perform: sts:AssumeRole on resource: arn:aws:iam::0123456789:role/test1 (retry in
 60s)
```

Available versions of the AWS AppConfig Agent Lambda extension

This topic includes information about AWS AppConfig Agent Lambda extension versions. The AWS AppConfig Agent Lambda extension supports Lambda functions developed for the x86-64 and ARM64 (Graviton2) platforms. To work properly, your Lambda function must be configured to use the specific Amazon Resource Name (ARN) for the AWS Region where it is currently hosted. You can view AWS Region and ARN details later in this section.

Important

Note the following important details about the AWS AppConfig Agent Lambda extension.

- The `GetConfiguration` API action was deprecated on January 28, 2022. Calls to receive configuration data should use the `StartConfigurationSession` and `GetLatestConfiguration` APIs instead. If you are using a version of the AWS AppConfig Agent Lambda extension created before January 28, 2022, you might have to configure permission to the new APIs. For more information, see About the AWS AppConfig data plane service.

- AWS AppConfig supports all of the versions listed in Older extension versions. We recommend that you periodically update to the latest version to take advantage of extension enhancements.

Topics

AWS AppConfig Agent Lambda Extension release notes

The following table describes changes made to recent versions of the AWS AppConfig Lambda extension.

Version	Launch date	Notes
2.0.358	12/01/2023	Added support for the following retrieval features: • *Multi-account retrieval*: Use AWS AppConfig Agent from a primary or *retrieval* AWS account to retrieve configuration data from multiple vendor accounts. • *Write configuration copy to disk*: Use AWS AppConfig Agent to write configura tion data to disk. This feature enables customers with applications that read configuration data from

Version	Launch date	Notes
		disk to integrate with AWS AppConfig.
2.0.181	08/14/2023	Added support for the Israel (Tel Aviv) il-central-1 AWS Region.
2.0.165	02/21/2023	Minor bug fixes. No longer restricting extension use to specific runtime versions via the AWS Lambda console. Added support for the following AWS Regions: • Middle East (UAE), me-central-1 • Asia Pacific (Hyderabad), ap-south-2 • Asia Pacific (Melbourne), ap-southeast-4 • Europe (Spain), eu-south-2 • Europe (Zurich), eu-central-2

Version	Launch date	Notes
2.0.122	08/23/2022	Added support for a tunneling proxy, which can be configured with the `AWS_APPCONFIG_EXTENSION_PROXY_URL` and `AWS_APPCONFIG_EXTENSION_PROXY_HEADERS` environment variables. Added .NET 6 as a runtime. For more information about environment variables, see Configuring the AWS AppConfig Agent Lambda extension.
2.0.58	05/03/2022	Improved support for Graviton2 (ARM64) processors in Lambda.
2.0.45	03/15/2022	Added support for calling a single feature flag. Previously, customers called feature flags grouped into a configuration profile and had to parse the response client-side. With this release, customers can use a `flag=<flag-name>` parameter when calling the HTTP localhost endpoint to get the value of a single flag. Also added initial support for Graviton2 (ARM64) processors.

Finding your Lambda extension version number

Use the following procedure to locate the version number of your currently configured AWS AppConfig Agent Lambda extension. To work properly, your Lambda function must be configured to use the specific Amazon Resource Name (ARN) for the AWS Region where it is currently hosted.

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose the Lambda function where you want to add the `AWS-AppConfig-Extension` layer.
3. In the **Layers** section, choose **Add a layer**.
4. In the **Choose a layer** section, choose **AWS-AppConfig-Extension** from the **AWS layers** list.
5. Use the **Version** list to choose a version number.
6. Choose **Add**.
7. Use the **Test** tab to test the function.
8. After the test completes, view the log output. Locate the AWS AppConfig Agent Lambda extension version in the **Details of the Execution** section. This version must match the required URLs for that version.

x86-64 platform

When you add the extension as a layer to your Lambda, you must specify an ARN. Choose an ARN from the following table that corresponds with the AWS Region where you created the Lambda. These ARNs are for Lambda functions developed for the x86-64 platform.

Version 2.0.358

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:93`

Region	ARN
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:141`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:161`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:93`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:106`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:47`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:125`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:93`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:98`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:159`

Region	ARN
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:83`
Europe (Spain)	`arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:44`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:76`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:76`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:83`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:98`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:108`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:101`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:106`

Region	ARN
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:106`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:79`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:20`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:107`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:47`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:128`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:83`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:22`
Middle East (UAE)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:49`

Region	ARN
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:85`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:54`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:54`

ARM64 platform

When you add the extension as a layer to your Lambda, you must specify an ARN. Choose an ARN from the following table that corresponds with the AWS Region where you created the Lambda. These ARNs are for Lambda functions developed for the ARM64 platform.

Version 2.0.358

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:61`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:45`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:18`

Region	ARN
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:63`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:13`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:49`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:5`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:63`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:45`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:17`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:18`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:11`

Region	ARN
Europe (Spain)	`arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:5`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:11`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:51`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:16`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:16`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:58`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:49`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:16`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:5`

Region	ARN
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:49`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:5`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:16`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:11`
Middle East (UAE)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:5`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:13`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:5`

Older extension versions

This section lists the ARNs and AWS Regions for older versions of the AWS AppConfig Lambda extension. This list doesn't contain information for all previous versions of the AWS AppConfig Agent Lambda extension, but it will be updated when new versions are released.

Older extension versions (x86-64 platform)

The following tables list ARNs and the AWS Regions for older versions of the AWS AppConfig Agent Lambda extension developed for the x86-64 platform.

Date replaced by newer extension: 12/01/2023

Version 2.0.181

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:113`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:81`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:124`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:146`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:81`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:93`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:32`

Region	ARN
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:110`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:81`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:82`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:142`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:73`
Europe (Spain)	`arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:29`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:68`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:68`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:73`

Region	ARN
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:84`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:93`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:86`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:91`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:93`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:64`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:5`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:94`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:32`

Region	ARN
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:113`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:73`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:7`
Middle East (UAE)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:34`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:73`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:46`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:46`

Date replaced by newer extension: 08/14/2023

Version 2.0.165

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:110`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:79`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:121`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:143`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:79`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:91`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:29`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:108`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:79`

Region	ARN
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:80`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:139`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:71`
Europe (Spain)	`arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:26`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:66`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:66`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:71`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:82`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:91`

Region	ARN
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:84`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:89`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:91`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:60`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:2`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:92`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:29`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:110`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:71`

Region	ARN
Middle East (UAE)	`arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:31`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:71`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:44`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:44`

Date replaced by newer extension: 02/21/2023

Version 2.0.122

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:82`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:59`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:93`

Region	ARN
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:114`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:59`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:70`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:82`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:59`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:60`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:111`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:54`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:52`

Region	ARN
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:52`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:54`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:62`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:70`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:59`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:64`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:70`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:37`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:71`

Region	ARN
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:82`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:54`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:54`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:29`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:29`

Date replaced by newer extension: 08/23/2022

Version 2.0.58

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:69`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:50`

Region	ARN
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:78`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:101`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:50`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:59`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:69`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:50`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:51`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:98`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:47`

Region	ARN
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:46`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:46`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:47`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:49`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:59`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:46`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:51`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:59`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:24`

Region	ARN
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:60`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:69`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:47`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:47`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:23`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:23`

Date replaced by newer extension: 04/21/2022

Version 2.0.45

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:68`

Region	ARN
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:49`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:77`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:100`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:49`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:58`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:68`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:49`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:50`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:97`

Region	ARN
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:46`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:45`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:45`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:46`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:48`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:58`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:45`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:50`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:58`

Region	ARN
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:23`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:59`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:68`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:46`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:46`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:22`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:22`

Date replaced by newer extension: 03/15/2022

Version 2.0.30

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:61`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:47`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:61`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:89`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:47`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:54`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:59`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:47`
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:48`

Region	ARN
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:86`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:44`
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:43`
China (Ningxia)	`arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:43`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:44`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:45`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:42`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:54`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:45`

Region	ARN
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:54`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:13`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:55`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:61`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:44`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:44`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:20`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:20`

Older extension versions (ARM64 platform)

The following tables list ARNs and the AWS Regions for older versions of the AWS AppConfig Agent Lambda extension developed for the ARM64 platform.

Date replaced by newer extension: 12/01/2023

Version 2.0.181

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:46`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:33`
US West (N. California)	`arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:1`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:48`
Canada (Central)	`arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:1`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:36`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:48`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:33`

Region	ARN
Europe (Paris)	`arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:1`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:1`
Europe (Milan)	`arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:37`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:43`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:36`

Region	ARN
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:36`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:1`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:1`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:1`

Date replaced by newer extension: 03/30/2023

Version 2.0.165

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:43`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:31`

Region	ARN
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:45`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:34`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:46`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:31`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:35`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:41`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:34`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:34`

Date replaced by newer extension: 02/21/2023

Version 2.0.122

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:15`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:11`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:16`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:13`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:20`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:11`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:15`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:16`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:13`

Region	ARN
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:13`

Date replaced by newer extension: 08/23/2022

Version 2.0.58

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:2`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:2`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:3`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:2`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:7`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2`

Region	ARN
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:2`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:3`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:2`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:2`

Date replaced by newer extension: 04/21/2022

Version 2.0.45

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:1`
US East (Ohio)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:1`
US West (Oregon)	`arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:2`

Region	ARN
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:1`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:6`
Europe (London)	`arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:2`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:1`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:1`

Using a container image to add the AWS AppConfig Agent Lambda extension

You can package your AWS AppConfig Agent Lambda extension as a container image to upload it to your container registry hosted on Amazon Elastic Container Registry (Amazon ECR).

To add the AWS AppConfig Agent Lambda extension as a Lambda container image

1. Enter the AWS Region and the Amazon Resource Name (ARN) in the AWS Command Line Interface (AWS CLI) as shown below. Replace the Region and ARN value with your Region and the matching ARN to retrieve a copy of the Lambda layer. AWS AppConfig provides ARNs for x86-64 and ARM64 platforms.

```
aws lambda get-layer-version-by-arn \
  --region AWS Region \
  --arn extension ARN
```

   Here's an example.

```
aws lambda get-layer-version-by-arn \
  --region us-east-1 \
  --arn arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128
```

   The response looks like the following:

```
{
  "LayerVersionArn": "arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-
Extension:128",
  "Description": "AWS AppConfig extension: Use dynamic configurations deployed via
 AWS AppConfig for your AWS Lambda functions",
  "CreatedDate": "2021-04-01T02:37:55.339+0000",
  "LayerArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension",

  "Content": {
    "CodeSize": 5228073,
    "CodeSha256": "8otOgbLQbexpUm3rKlMhvcE6Q5TvwcLCKrc4Oe+vmMY=",
    "Location" : "S3-Bucket-Location-URL"
  },

  "Version": 30,
  "CompatibleRuntimes": [
    "python3.8",
    "python3.7",
    "nodejs12.x",
    "ruby2.7"
  ],
}
```

2. In the above response, the value returned for `Location` is the URL of the Amazon Simple Storage Service (Amazon S3) bucket that contains the Lambda extension. Paste the URL into your web browser to download the Lambda extension .zip file.

   > **Note**
   >
   > The Amazon S3 bucket URL is available for only 10 minutes.

   (Optional) Alternatively, you can also use the following `curl` command to download the Lambda extension.

   ```
   curl -o extension.zip "S3-Bucket-Location-URL"
   ```

   (Optional) Alternatively, you can combine **Step 1** and **Step 2** to retrieve the ARN and download the .zip extension file all at once.

   ```
   aws lambda get-layer-version-by-arn \
     --arn extension ARN \
     | jq -r '.Content.Location' \
     | xargs curl -o extension.zip
   ```

3. Add the following lines in your `Dockerfile` to add the extension to your container image.

   ```
   COPY extension.zip extension.zip
   RUN yum install -y unzip \
     && unzip extension.zip /opt \
     && rm -f extension.zip
   ```

4. Ensure that the Lambda function execution role has the appconfig:GetConfiguration permission set.

Example

This section includes an example for enabling the AWS AppConfig Agent Lambda extension on a container image-based Python Lambda function.

1. Create a `Dockerfile` that is similar to the following.

```
FROM public.ecr.aws/lambda/python:3.8 AS builder
COPY extension.zip extension.zip
RUN yum install -y unzip \
  && unzip extension.zip -d /opt \
  && rm -f extension.zip

FROM public.ecr.aws/lambda/python:3.8
COPY --from=builder /opt /opt
COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Download the extension layer to the same directory as the `Dockerfile`.

```
aws lambda get-layer-version-by-arn \
  --arn extension ARN \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. Create a Python file named `index.py` in the same directory as the `Dockerfile`.

```
import urllib.request

def handler(event, context):
    return {
       # replace parameters here with your application, environment, and
 configuration names
       'configuration': get_configuration('myApp', 'myEnv', 'myConfig'),
    }

def get_configuration(app, env, config):
    url = f'http://localhost:2772/applications/{app}/environments/{env}/
configurations/{config}'
    return urllib.request.urlopen(url).read()
```

4. Run the following steps to build the `docker` image and upload it to Amazon ECR.

```
// set environment variables
export ACCOUNT_ID = <YOUR_ACCOUNT_ID>
export REGION = <AWS_REGION>

// create an ECR repository
aws ecr create-repository --repository-name test-repository
```

```
// build the docker image
docker build -t test-image .

// sign in to AWS
aws ecr get-login-password \
  | docker login \
  --username AWS \
  --password-stdin "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com"

// tag the image
docker tag test-image:latest "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-
repository:latest"

// push the image to ECR
docker push "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-repository:latest"
```

5. Use the Amazon ECR image that you created above to create the Lambda function. For more information about a Lambda function as a container, see Create a Lambda function defined as a container image.
6. Ensure that the Lambda function execution role has the appconfig:GetConfiguration permission set.

Integrating with OpenAPI

You can use the following YAML specification for OpenAPI to create an SDK using a tool like OpenAPI Generator. You can update this specification to include hardcoded values for Application, Environment, or Configuration. You can also add additional paths (if you have multiple configuration types) and include configuration schemas to generate configuration-specific typed models for your SDK clients. For more information about OpenAPI (which is also known as Swagger), see the OpenAPI specification.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: AppConfig Agent Lambda extension API
  description: An API model for the AppConfig Agent Lambda extension.
servers:
  - url: https://localhost:{port}/
    variables:
      port:
        default:
          '2772'
paths:
  /applications/{Application}/environments/{Environment}/configurations/
{Configuration}:
    get:
      operationId: getConfiguration
      tags:
        - configuration
      parameters:
        - in: path
          name: Application
          description: The application for the configuration to get. Specify either the
 application name or the application ID.
          required: true
          schema:
            type: string
        - in: path
          name: Environment
          description: The environment for the configuration to get. Specify either the
 environment name or the environment ID.
          required: true
          schema:
            type: string
        - in: path
          name: Configuration
          description: The configuration to get. Specify either the configuration name
 or the configuration ID.
          required: true
          schema:
            type: string
      responses:
        200:
          headers:
            ConfigurationVersion:
              schema:
                type: string
          content:
            application/octet-stream:
              schema:
                type: string
                format: binary
          description: successful config retrieval
        400:
          description: BadRequestException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        404:
          description: ResourceNotFoundException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        500:
          description: InternalServerException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        502:
          description: BadGatewayException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        504:
          description: GatewayTimeoutException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'

components:
  schemas:
    Error:
      type: string
      description: The response error
```

Retrieving configuration data from Amazon EC2 instances

You can integrate AWS AppConfig with applications running on your Amazon Elastic Compute Cloud (Amazon EC2) Linux instances by using AWS AppConfig Agent. The agent enhances application processing and management in the following ways:

- The agent calls AWS AppConfig on your behalf by using an AWS Identity and Access Management (IAM) role and managing a local cache of configuration data. By pulling configuration data from the local cache, your application requires fewer code updates to manage configuration data, retrieves configuration data in milliseconds, and isn't affected by network issues that can disrupt calls for such data.*
- The agent offers a native experience for retrieving and resolving AWS AppConfig feature flags.
- Out of the box, the agent provides best practices for caching strategies, polling intervals, and availability of local configuration data while tracking the configuration tokens needed for subsequent service calls.
- While running in the background, the agent periodically polls the AWS AppConfig data plane for configuration data updates. Your application can retrieve the data by connecting to localhost on port 2772 (a customizable default port value) and calling HTTP GET to retrieve the data.

*AWS AppConfig Agent caches data the first time the service retrieves your configuration data. For this reason, the first call to retrieve data is slower than subsequent calls.

Topics

Step 1: (Required) Creating resources and configuring permissions

To integrate AWS AppConfig with applications running on your Amazon EC2 instances, you must create AWS AppConfig artifacts and configuration data, including feature flags or freeform configuration data. For more information, see Creating feature flags and free form configuration data in AWS AppConfig.

To retrieve configuration data hosted by AWS AppConfig, your applications must be configured with access to the AWS AppConfig data plane. To give your applications access, update the IAM permissions policy that is assigned to the Amazon EC2 instance role. Specifically, you must add the `appconfig:StartConfigurationSession` and `appconfig:GetLatestConfiguration` actions to the policy. Here is an example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appconfig:StartConfigurationSession",
                "appconfig:GetLatestConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about adding permissions to a policy, see Adding and removing IAM identity permissions in the *IAM User Guide*.

Step 2: (Required) Installing and starting AWS AppConfig Agent on Amazon EC2 instances

AWS AppConfig Agent is hosted in an Amazon Simple Storage Service (Amazon S3) bucket that is managed by AWS. Use the following procedure to install the latest version of the agent on your Linux instance. If your application is distributed across multiple instances, then you must perform this procedure on each instance that hosts the application.

Note

Note the following information:

- AWS AppConfig Agent is available for Linux operating systems running kernel version 4.15 or greater. Debian-based systems, such as Ubuntu, are not supported.
- The agent supports x86_64 and ARM64 architectures.
- For distributed applications, we recommend adding the install and startup commands to the Amazon EC2 user data of your Auto Scaling group. If you do, each instance runs the commands automatically. For more information, see Run commands on your Linux instance at launch in the *Amazon EC2 User Guide*. Additionally, see Tutorial: Configure

user data to retrieve the target lifecycle state through instance metadata in the *Amazon EC2 Auto Scaling User Guide*.

- The procedures throughout this topic describe how to perform actions like installing the agent by logging into the instance to run the command. You can run the commands from a local client machine and target one or more instances by using Run Command, which is a capability of AWS Systems Manager. For more information, see AWS Systems Manager Run Command in the *AWS Systems Manager User Guide*.
- AWS AppConfig Agent on Amazon EC2 Linux instances is a `systemd` service.

To install and start AWS AppConfig Agent on an instance

1. Log into your Linux instance.
2. Open a terminal and run the following command with Administrator permissions for x86_64 architectures:

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-
agent/linux/x86_64/latest/aws-appconfig-agent.rpm
```

   For ARM64 architectures, run the following command:

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-
agent/linux/arm64/latest/aws-appconfig-agent.rpm
```

   If you want to install a specific version of AWS AppConfig Agent, replace `latest` in the URL with a specific version number. Here's an example for x86_64:

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-
agent/linux/x86_64/2.0.2/aws-appconfig-agent.rpm
```

3. Run the following command to start the agent:

```
sudo systemctl start aws-appconfig-agent
```

4. Run the following command to verify the agent is running:

```
sudo systemctl status aws-appconfig-agent
```

If successful, the command returns information like the following:

```
aws-appconfig-agent.service - aws-appconfig-agent
  ...
  Active: active (running) since Mon 2023-07-26 00:00:00 UTC; 0s ago
  ...
```

Note

To stop the agent, run the following command:

```
sudo systemctl stop aws-appconfig-agent
```

Step 3: (Optional, but recommended) Sending log files to CloudWatch Logs

By default, AWS AppConfig Agent publishes logs to STDERR. Systemd redirects STDOUT and STDERR for all services running on the Linux instance to the systemd journal. You can view and manage log data in the systemd journal if you're running AWS AppConfig Agent on only one or two instances. A better solution, a solution we highly recommend for distributed applications, is to write log files to disk and then use Amazon CloudWatch agent to upload the log data to the AWS cloud. Additionally, you can configure the CloudWatch agent to delete old log files from your instance, which prevents your instance from running out of disk space.

To enable logging to disk, you must set the `LOG_PATH` environment variable, as described in Step 4: (Optional) Using environment variables to configure AWS AppConfig Agent for Amazon EC2.

To get started with the CloudWatch agent, see Collect metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch agent in the *Amazon CloudWatch User Guide*. You can use Quick Setup, a capability of Systems Manager to quickly install the CloudWatch agent. For more information, see Quick Setup Host Management in the *AWS Systems Manager User Guide*.

Warning

If you choose to write log files to disk without using the CloudWatch agent, you must delete old log files. AWS AppConfig Agent automatically rotates log files every hour. If you don't delete old log files, your instance can run out of disk space.

After you install the CloudWatch agent on your instance, create a CloudWatch agent configuration file. The configuration file instructs CloudWatch agent on how to work with AWS AppConfig Agent log files. For more information about creating a CloudWatch agent configuration file, see Create the CloudWatch agent configuration file.

Add the following logs section to the CloudWatch agent configuration file on the instance and save your changes:

```
"logs": {
  "logs_collected": {
    "files": {
      "collect_list": [
        {
          "file_path": "/path_you_specified_for_logging",
          "log_group_name": "${YOUR_LOG_GROUP_NAME}/aws-appconfig-agent.log",
          "auto_removal": true
        },
        ...
      ]
    },
    ...
  },
  ...
}
```

If the value of auto_removal is true, the CloudWatch agent automatically deletes rotated AWS AppConfig Agent log files.

Step 4: (Optional) Using environment variables to configure AWS AppConfig Agent for Amazon EC2

You can configure AWS AppConfig Agent for Amazon EC2 by using environment variables. To set environment variables for a systemd service, you create a drop-in unit file. The following example shows how to create drop-in unit file to set the AWS AppConfig Agent logging level to DEBUG.

Example of how to create a drop-in unit file for environment variables

1. Log into your Linux instance.
2. Open a terminal and run the following command with Administrator permissions. The command creates a configuration directory:

```
sudo mkdir /etc/systemd/system/aws-appconfig-agent.service.d
```

3. Run the following command to create the drop-in unit file. Replace *`file_name`* with a name for the file. The extension must be `.conf`:

```
sudo touch /etc/systemd/system/aws-appconfig-agent.service.d/file_name.conf
```

4. Enter information in the drop-in unit file. The following example adds a `Service` section that defines an environment variable. The example sets AWS AppConfig Agent log level to `DEBUG`.

```
[Service]
Environment=LOG_LEVEL=DEBUG
```

5. Run the following command to reload the systemd configuration:

```
sudo systemctl daemon-reload
```

6. Run the following command to restart AWS AppConfig Agent:

```
sudo systemctl restart aws-appconfig-agent
```

You can configure AWS AppConfig Agent for Amazon EC2 by specifying the following environment variables in a drop-in unit file.

Environment variable	Details	Default value
`ACCESS_TOKEN`	This environment variable defines a token that must be provided when requesting configuration data from the agent HTTP server. The value of the token must be set in the HTTP request authoriza tion header with an authoriza tion type of `Bearer`. Here is an example.	None

Environment variable	Details	Default value
	`GET /applications/my_app/... Host: localhost:2772 Authorization: Bearer <token value>`	
`BACKUP_DIRECTORY`	This environment variable enables AWS AppConfig Agent to save a backup of each configuration it retrieves to the specified directory. **Important** Configurations backed up to disk are not encrypted. If your configuration contains sensitive data, AWS AppConfig recommends that you practice the principle of least privilege with your filesystem permissions. For more information, see Security in AWS AppConfig.	None
`HTTP_PORT`	This environment variable specifies the port on which the HTTP server for the agent runs.	2772

Environment variable	Details	Default value
`LOG_LEVEL`	This environment variable specifies the level of detail that the agent logs. Each level includes the current level and all higher levels. The variables are case sensitive. From most to least detailed, the log levels are: `debug`, `info`, `warn`, `error`, and none. `Debug` includes detailed information, including timing information, about the agent.	`info`
`LOG_PATH`	The disk location where logs are written. If not specified, logs are written to stderr.	None

Environment variable	Details	Default value
`MANIFEST`	This environment variable configures AWS AppConfig Agent to take advantage of additional per-configuration features like *multi-account retrievals* and *save configuration to disk*. You can enter one of the following values: • `"app:env:manifest-config"` • `"file:/fully/qualified/path/to/manifest.json"` For more information about these features, see Additional retrieval features.	true
`MAX_CONNECTIONS`	This environment variable configures the maximum number of connections that the agent uses to retrieve configurations from AWS AppConfig.	3

Environment variable	Details	Default value
`POLL_INTERVAL`	This environment variable controls how often the agent polls AWS AppConfig for updated configuration data. You can specify a number of seconds for the interval. You can also specify a number with a time unit: s for seconds, m for minutes, and h for hours. If a unit isn't specified, the agent defaults to seconds. For example, 60, 60s, and 1m result in the same poll interval.	45 seconds
`PREFETCH_LIST`	This environment variable specifies the configuration data the agent requests from AWS AppConfig as soon as it starts.	None

Environment variable	Details	Default value
`PRELOAD_BACKUPS`	If set to `true`, AWS AppConfig Agent loads configuration backups found in the `BACKUP_DIRECTORY` into memory and immediately checks to see if a newer version exists from the service. If set to `false`, AWS AppConfig Agent only loads the contents from a configuration backup if it cannot retrieve configuration data from the service, for example if there is a problem with your network.	true
`PROXY_HEADERS`	This environment variable specifies headers that are required by the proxy referenced in the `PROXY_URL` environment variable. The value is a comma-separated list of headers. Each header uses the following form. `"header: value"`	None
`PROXY_URL`	This environment variable specifies the proxy URL to use for connections from the agent to AWS services, including AWS AppConfig. HTTPS and HTTP URLs are supported.	None

Environment variable	Details	Default value
`REQUEST_TIMEOUT`	This environment variable controls the amount of time the agent waits for a response from AWS AppConfig. If the service does not respond, the request fails. If the request is for the initial data retrieval, the agent returns an error to your application. If the timeout occurs during a background check for updated data, the agent logs the error and tries again after a short delay. You can specify the number of milliseconds for the timeout. You can also specify a number with a time unit: ms for milliseconds and s for seconds. If a unit isn't specified, the agent defaults to milliseconds. As an example, 5000, 5000ms and 5s result in the same request timeout value.	3000 milliseconds

Environment variable	Details	Default value
`ROLE_ARN`	This environment variable specifies the Amazon Resource Name (ARN) of an IAM role. AWS AppConfig Agent assumes this role to retrieve configuration data.	None
`ROLE_EXTERNAL_ID`	This environment variable specifies the external ID to use with the assumed role ARN.	None
`ROLE_SESSION_NAME`	This environment variable specifies the session name to be associated with the credentials for the assumed IAM role.	None
`SERVICE_REGION`	This environment variable specifies an alternative AWS Region that AWS AppConfig Agent uses to call the AWS AppConfig service. If left undefined, the agent attempts to determine the current Region. If it can't, the agent fails to start.	None
`WAIT_ON_MANIFEST`	This environment variable configures AWS AppConfig Agent to wait until the manifest is processed before completing startup.	true

Step 5: (Required) Retrieving configuration data

You can retrieve configuration data from AWS AppConfig Agent by using an HTTP localhost call. The following examples use `curl` with an HTTP client. You can call the agent using any available HTTP client supported by your application language or available libraries, including an AWS SDK.

To retrieve the full content of any deployed configuration

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

To retrieve a single flag and its attributes from an AWS AppConfig configuration of type `Feature Flag`

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

To access multiple flags and their attributes from an AWS AppConfig configuration of type `Feature Flag`

```

$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

Step 6 (Optional, but recommended): Automating updates to AWS AppConfig Agent

AWS AppConfig Agent is updated periodically. To ensure you are running the latest version of AWS AppConfig Agent on your instances, we recommend that you add the following commands to your Amazon EC2 user data. You can add the commands to the user data on either the instance or the EC2 Auto Scaling group. The script installs and starts the latest version of the agent each time an instance starts or reboots.

```
#!/bin/bash
# install the latest version of the agent
yum install -y https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/
linux/x86_64/latest/aws-appconfig-agent.rpm
```

```
# optional: configure the agent
mkdir /etc/systemd/system/aws-appconfig-agent.service.d
echo "${MY_AGENT_CONFIG}" > /etc/systemd/system/aws-appconfig-agent.service.d/
overrides.conf
systemctl daemon-reload
# start the agent
systemctl start aws-appconfig-agent
```

Retrieving configuration data from Amazon ECS and Amazon EKS

You can integrate AWS AppConfig with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) by using AWS AppConfig Agent. The agent functions as a sidecar container running alongside your Amazon ECS and Amazon EKS container applications. The agent enhances containerized application processing and management in the following ways:

- The agent calls AWS AppConfig on your behalf by using an AWS Identity and Access Management (IAM) role and managing a local cache of configuration data. By pulling configuration data from the local cache, your application requires fewer code updates to manage configuration data, retrieves configuration data in milliseconds, and isn't affected by network issues that can disrupt calls for such data.*
- The agent offers a native experience for retrieving and resolving AWS AppConfig feature flags.
- Out of the box, the agent provides best practices for caching strategies, polling intervals, and local configuration data availability while tracking the configuration tokens needed for subsequent service calls.
- While running in the background, the agent periodically polls the AWS AppConfig data plane for configuration data updates. Your containerized application can retrieve the data by connecting to localhost on port 2772 (a customizable default port value) and calling HTTP GET to retrieve the data.
- AWS AppConfig Agent updates configuration data in your containers without having to restart or recycle those containers.

*AWS AppConfig Agent caches data the first time the service retrieves your configuration data. For this reason, the first call to retrieve data is slower than subsequent calls.

Topics

- Starting the AWS AppConfig agent for Amazon ECS integration
- Starting the AWS AppConfig agent for Amazon EKS integration
- Using environment variables to configure AWS AppConfig Agent for Amazon ECS and Amazon EKS
- Retrieving configuration data

Before you begin

To integrate AWS AppConfig with your container applications, you must create AWS AppConfig artifacts and configuration data, including feature flags or freeform configuration data. For more information, see Creating feature flags and free form configuration data in AWS AppConfig.

To retrieve configuration data hosted by AWS AppConfig, your container applications must be configured with access to the AWS AppConfig data plane. To give your applications access, update the IAM permissions policy that is used by your container service IAM role. Specifically, you must add the `appconfig:StartConfigurationSession` and `appconfig:GetLatestConfiguration` actions to the policy. Container service IAM roles include the following:

- The Amazon ECS task role
- The Amazon EKS node role
- The AWS Fargate (Fargate) pod execution role (if your Amazon EKS containers use Fargate for compute processing)

For more information about adding permissions to a policy, see Adding and removing IAM identity permissions in the *IAM User Guide*.

Starting the AWS AppConfig agent for Amazon ECS integration

The AWS AppConfig Agent sidecar container is automatically available in your Amazon ECS environment. To use the AWS AppConfig Agent sidecar container, you must start it.

To start Amazon ECS (console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose the task definition for your application, and then select the latest revision.

4. Choose **Create new revision**, **Create new revision**.
5. Choose **Add more containers**.
6. For **Name**, enter a unique name for the AWS AppConfig Agent container.
7. For **Image URI**, enter: **`public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x`**
8. For **Essential container**, choose **Yes**.
9. In the **Port mappings** section, choose **Add port mapping**.
10. For **Container port**, enter **2772**.

    Note

    AWS AppConfig Agent runs on port 2772, by default. You can specify a different port.

11. Choose **Create**. Amazon ECS creates a new container revision and displays the details.
12. In the navigation pane, choose **Clusters**, and then choose your application cluster in the list.
13. On the **Services** tab, select the service for your application.
14. Choose **Update**.
15. Under **Deployment configuration**, for **Revision**, choose the latest revision.
16. Choose **Update**. Amazon ECS deploys the latest task definition.
17. After the deployment finishes, you can verify that AWS AppConfig Agent is running on the **Configuration and tasks** tab. On the **Tasks** tab, choose the running task.
18. In the **Containers** section, verify that the AWS AppConfig Agent container is listed.
19. To verify that AWS AppConfig Agent started, choose the **Logs** tab. Locate a statement like the following for the AWS AppConfig Agent container: `[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

Note

You can adjust the default behavior of AWS AppConfig Agent by entering or changing environment variables. For information about the available environment variables, see Using environment variables to configure AWS AppConfig Agent for Amazon ECS and Amazon EKS. For information about how to change environment variables in Amazon ECS, see Passing environment variables to a container in the *Amazon Elastic Container Service Developer Guide*.

Starting the AWS AppConfig agent for Amazon EKS integration

The AWS AppConfig Agent sidecar container is automatically available in your Amazon EKS environment. To use the AWS AppConfig Agent sidecar container, you must start it. The following procedure describes how to use the Amazon EKS `kubectl` command line tool to make changes in the `kubeconfig` file for your container application. For more information about creating or editing a `kubeconfig` file, see Creating or updating a kubeconfig file for an Amazon EKS cluster in the Amazon EKS User Guide.

To start AWS AppConfig Agent (kubectl command line tool)

1. Open your `kubeconfig` file and verify that your Amazon EKS application is running as a single-container deployment. The contents of the file should look similar to the following.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-application-label
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-application-label
  template:
    metadata:
      labels:
        app: my-application-label
    spec:
      containers:
      - name: my-app
        image: my-repo/my-image
        imagePullPolicy: IfNotPresent
```

2. Add the AWS AppConfig Agent container definition details to your YAML deployment file.

```
- name: appconfig-agent
        image: public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x
        ports:
        - name: http
```

```
          containerPort: 2772
          protocol: TCP
        env:
        - name: SERVICE_REGION
          value: region
        imagePullPolicy: IfNotPresent
```

Note

Note the following information.

- AWS AppConfig Agent runs on port 2772, by default. You can specify a different port.
- You can adjust the default behavior of AWS AppConfig Agent by entering environment variables. For more information, see Using environment variables to configure AWS AppConfig Agent for Amazon ECS and Amazon EKS.
- For *SERVICE_REGION*, specify the AWS Region code (for example, `us-west-1`) where AWS AppConfig Agent retrieves configuration data.

3. Run the following command in the `kubectl` tool to apply the changes to your cluster.

```
kubectl apply -f my-deployment.yml
```

4. After the deployment finishes, verify that AWS AppConfig Agent is running. Use the following command to view the application pod log file.

```
kubectl logs -n my-namespace -c appconfig-agent my-pod
```

Locate a statement like the following for the AWS AppConfig Agent container: `[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

Note

You can adjust the default behavior of AWS AppConfig Agent by entering or changing environment variables. For information about the available environment variables, see Using environment variables to configure AWS AppConfig Agent for Amazon ECS and Amazon EKS.

Using environment variables to configure AWS AppConfig Agent for Amazon ECS and Amazon EKS

You can configure AWS AppConfig Agent by changing the following environment variables for your agent container.

Environment variable	Details	Default value

`ACCESS_TOKEN`	This environment variable defines a token that must be provided when requesting configuration data from the agent HTTP server. The value of the token must be set in the HTTP request authorization header with an authorization type of `Bearer`. Here is an example.
	
	`GET /applications/my_app/...`
	`Host: localhost:2772`
	
	`Authorization: Bearer <token value>`
		None

`BACKUP_DIRECTORY`	This environment variable enables AWS AppConfig Agent to save a backup of each configuration it retrieves to the specified directory. **Important** Configurations backed up to disk are not encrypted. If	None

Environment variable	Details	Default value
	your configuration contains sensitive data, AWS AppConfig recommends that you practice the principle of least privilege with your filesyste m permissions. For more information, see Security in AWS AppConfig.	
`HTTP_PORT`	This environment variable specifies the port on which the HTTP server for the agent runs.	2772
`LOG_LEVEL`	This environment variable specifies the level of detail that the agent logs. Each level includes the current level and all higher levels. The variables are case sensitive. From most to least detailed, the log levels are: `debug`, `info`, `warn`, `error`, and none. `Debug` includes detailed information, including timing information, about the agent.	`info`

Environment variable	Details	Default value
`MANIFEST`	This environment variable configures AWS AppConfig Agent to take advantage of additional per-configuration features like *multi-account retrievals* and *save configuration to disk*. You can enter one of the following values: • `"app:env:manifest-config"` • `"file:/fully/qualified/path/to/manifest.json"` For more information about these features, see Additional retrieval features.	true
`MAX_CONNECTIONS`	This environment variable configures the maximum number of connections that the agent uses to retrieve configurations from AWS AppConfig.	3

Environment variable	Details	Default value
`POLL_INTERVAL`	This environment variable controls how often the agent polls AWS AppConfig for updated configuration data. You can specify a number of seconds for the interval. You can also specify a number with a time unit: s for seconds, m for minutes, and h for hours. If a unit isn't specified, the agent defaults to seconds. For example, 60, 60s, and 1m result in the same poll interval.	45 seconds
`PREFETCH_LIST`	This environment variable specifies the configuration data the agent requests from AWS AppConfig as soon as it starts.	None

Environment variable	Details	Default value
`PRELOAD_BACKUPS`	If set to `true`, AWS AppConfig Agent loads configuration backups found in the `BACKUP_DIRECTORY` into memory and immediately checks to see if a newer version exists from the service. If set to `false`, AWS AppConfig Agent only loads the contents from a configuration backup if it cannot retrieve configuration data from the service, for example if there is a problem with your network.	true
`PROXY_HEADERS`	This environment variable specifies headers that are required by the proxy referenced in the `PROXY_URL` environment variable. The value is a comma-separated list of headers. Each header uses the following form. `"header: value"`	None
`PROXY_URL`	This environment variable specifies the proxy URL to use for connections from the agent to AWS services, including AWS AppConfig. HTTPS and HTTP URLs are supported.	None

Environment variable	Details	Default value
`REQUEST_TIMEOUT`	This environment variable controls the amount of time the agent waits for a response from AWS AppConfig. If the service does not respond, the request fails. If the request is for the initial data retrieval, the agent returns an error to your application. If the timeout occurs during a background check for updated data, the agent logs the error and tries again after a short delay. You can specify the number of milliseconds for the timeout. You can also specify a number with a time unit: ms for milliseconds and s for seconds. If a unit isn't specified, the agent defaults to milliseconds. As an example, 5000, 5000ms and 5s result in the same request timeout value.	3000 milliseconds

Environment variable	Details	Default value
`ROLE_ARN`	This environment variable specifies the Amazon Resource Name (ARN) of an IAM role. AWS AppConfig Agent assumes this role to retrieve configuration data.	None
`ROLE_EXTERNAL_ID`	This environment variable specifies the external ID to use with the assumed role ARN.	None
`ROLE_SESSION_NAME`	This environment variable specifies the session name to be associated with the credentials for the assumed IAM role.	None
`SERVICE_REGION`	This environment variable specifies an alternative AWS Region that AWS AppConfig Agent uses to call the AWS AppConfig service. If left undefined, the agent attempts to determine the current Region. If it can't, the agent fails to start.	None
`WAIT_ON_MANIFEST`	This environment variable configures AWS AppConfig Agent to wait until the manifest is processed before completing startup.	true

Retrieving configuration data

You can retrieve configuration data from AWS AppConfig Agent by using an HTTP localhost call. The following examples use `curl` with an HTTP client. You can call the agent using any available HTTP client supported by your application language or available libraries.

Note

To retrieve configuration data if your application uses a forward slash, for example "test-backend/test-service", you will need to use URL encoding.

To retrieve the full content of any deployed configuration

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

To retrieve a single flag and its attributes from an AWS AppConfig configuration of type `Feature Flag`

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

To access multiple flags and their attributes from an AWS AppConfig configuration of type `Feature Flag`

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

Additional retrieval features

AWS AppConfig Agent offers the following additional features to help you retrieve configurations for your applications.

- Multi-account retrieval: Use AWS AppConfig Agent from a primary or *retrieval* AWS account to retrieve configuration data from multiple vendor accounts.

- Write configuration copy to disk: Use AWS AppConfig Agent to write configuration data to disk. This feature enables customers with applications that read configuration data from disk to integrate with AWS AppConfig.

About agent manifests

To enable these AWS AppConfig Agent features, you create a manifest. A manifest is a set of configuration data that you provide to control actions the agent can perform. A manifest is written in JSON. It contains a set of top-level keys that correspond to different configurations you've deployed using AWS AppConfig.

A manifest can include multiple configurations. Furthermore, each configuration in the manifest can identify one or more agent features to use for the specified configuration. The content of the manifest uses the following format:

```
{
    "application_name:environment_name:configuration_name": {
        "agent_feature_to_enable_1": {
            "feature-setting-key": "feature-setting-value"
        },
        "agent_feature_to_enable_2": {
            "feature-setting-key": "feature-setting-value"
        }
    }
}
```

Here is example JSON for a manifest with two configurations. The first configuration (*MyApp*) doesn't use any AWS AppConfig Agent features. The second configuration (*My2ndApp*) uses the *write configuration copy to disk* and the *multi-account retrieval* features:

```
{
        "MyApp:Test:MyAllowListConfiguration": {},

        "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
            "credentials": {
                "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
                "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
                "roleSessionName": "AwsAppConfigAgent",
                "credentialsDuration": "2h"
            },
            "writeTo": {
                "path": "/tmp/aws-appconfig/my-2nd-app/beta/my-enable-payments-feature-
flag-configuration.json"
            }
        }
    }
```

How to supply an agent manifest

You can store the manifest as a file in a location where AWS AppConfig Agent can read it. Or, you can store the manifest as an AWS AppConfig configuration and point the agent to it. To supply an agent manifest, you must set a `MANIFEST` environment variable with one of the following values:

Manifest location	Environment variable value	Use case
File	file:/path/to/agent-manifest.json	Use this method if your manifest won't change often.
AWS AppConfig configuration	*application-name*:*environment-name*:*configuration-name*	Use this method for dynamic updates. You can update and deploy a manifest stored in AWS AppConfig as a configuration in the same ways you store other AWS AppConfig configurations.
Environment variable	Manifest content (JSON)	Use this method if your manifest won't change often. This method is useful in container environments where it's easier to set an environment variable than it is to expose a file.

For more information about setting variables for AWS AppConfig Agent, see the relevant topic for your use case:

- Configuring the AWS AppConfig Agent Lambda extension

- Using AWS AppConfig Agent with Amazon EC2
- Using AWS AppConfig Agent with Amazon ECS and Amazon EKS

Multi-account retrieval

You can configure AWS AppConfig Agent to retrieve configurations from multiple AWS accounts by entering credential overrides in the AWS AppConfig Agent manifest. *Credential overrides* include the Amazon Resource Name (ARN) of an AWS Identity and Access Management (IAM) role, a role ID, a session name, and a duration for how long the agent can assume the role.

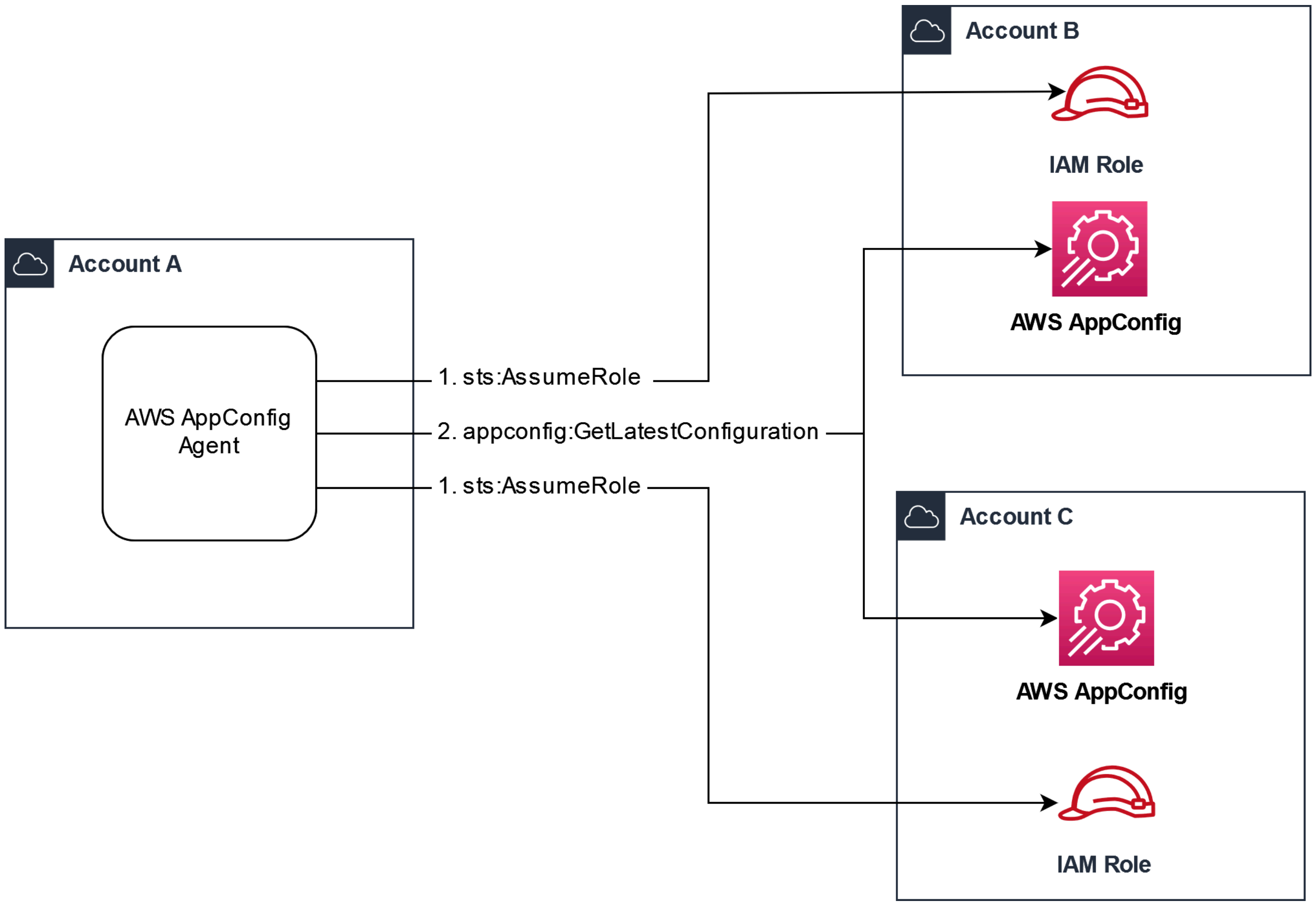
You enter these details in a "credentials" section in the manifest. The "credentials" section uses the following format:

```
{
    "application_name:environment_name:configuration_name": {
        "credentials": {
            "roleArn": "arn:partition:iam::account_ID:role/roleName",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

Here is an example:

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AWSAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

Before retrieving a configuration, the agent reads the credential details for the configuration from the manifest and then assumes the IAM role specified for that configuration. You can specify a different set of credential overrides for different configurations in a single manifest. The following diagram shows how AWS AppConfig Agent, while running in Account A (the retrieval account), assumes separate roles specified for Accounts B and C (the vendor accounts) and then calls the GetLatestConfiguration API operation to retrieve configuration data from AWS AppConfig running in those accounts:



Configure permissions to retrieve configuration data from vendor accounts

AWS AppConfig Agent running in the retrieval account needs permission to retrieve configuration data from the vendor accounts. You give the agent permission by creating an AWS Identity and Access Management (IAM) role in each of the vendor accounts. AWS AppConfig Agent in the retrieval account assumes this role to get data from vendor accounts. Complete the procedures in this section to create an IAM permissions policy, an IAM role, and add agent overrides to the manifest.

Before you begin

Collect the following information before you create a permission policy and a role in IAM.

- The IDs for each AWS account. The *retrieval* account is the account that will call other accounts for configuration data. The *vendor* accounts are the accounts that will vend configuration data to the retrieval account.
- The name of the IAM role used by AWS AppConfig in the retrieval account. Here's a list of the roles used by AWS AppConfig, by default:
  - For Amazon Elastic Compute Cloud (Amazon EC2), AWS AppConfig uses the instance role.
  - For AWS Lambda, AWS AppConfig uses the Lambda execution role.
  - For Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS), AWS AppConfig uses the container role.

  If you configured AWS AppConfig Agent to use a different IAM role by specifying the `ROLE_ARN` environment variable, make a note of that name.

Create the permissions policy

Use the following procedure to create a permissions policy using the IAM console. Complete the procedure in each AWS account that will vend configuration data for the retrieval account.

To create an IAM policy

1. Sign in to the AWS Management Console in a vendor account.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Policies**, and then choose **Create policy**.
4. Choose the **JSON** option.
5. In the **Policy editor**, replace the default JSON with the following policy statement. Update each *`example resource placeholder`* with vendor account details.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource":
 "arn:partition:appconfig:region:vendor_account_ID:application/
vendor_application_ID/environment/vendor_environment_ID/
configuration/vendor_configuration_ID"
    }
   ]
}
```

Here's an example:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource": "arn:aws:appconfig:us-east-2:111122223333:application/abc123/
environment/def456/configuration/hij789"
    }
   ]
}
```

6. Choose **Next**.
7. In the **Policy name** field, enter a name.
8. (Optional) For **Add tags**, add one or more tag-key value pairs to organize, track, or control access for this policy.
9. Choose **Create policy**. The system returns you to the **Policies** page.
10. Repeat this procedure in each AWS account that will vend configuration data for the retrieval account.

Create the IAM role

Use the following procedure to create an IAM role using the IAM console. Complete the procedure in each AWS account that will vend configuration data for the retrieval account.

To create an IAM role

1. Sign in to the AWS Management Console in a vendor account.
2. Open the IAM console at https://console.aws.amazon.com/iam/.

3. In the navigation pane, choose **Roles**, and then choose **Create policy**.
4. For **Trusted entity type**, choose **AWS account**.
5. In the **AWS account** section, choose **Another AWS account**.
6. In the **Account ID** field, enter the retrieval account ID.
7. (Optional) As a security best practice for this assume role, choose **Require external ID** and enter a string.
8. Choose **Next**.
9. On the **Add permissions** page, use the **Search** field to locate the policy you created in the previous procedure. Select the check box next to its name.
10. Choose **Next**.
11. For **Role name**, enter a name.
12. (Optional) For **Description**, enter a description.
13. For **Step 1: Select trusted entities**, choose **Edit**. Replace the default JSON trust policy with the following policy. Update each *example resource placeholder* with information from your retrieval account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS":
 "arn:aws:iam::retrieval_account_ID:role/appconfig_role_in_retrieval_account"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

14. (Optional) For **Tags**, add one or more tag-key value pairs to organize, track, or control access for this role.
15. Choose **Create role**. The system returns you to the **Roles** page.
16. Search for the role you just created. Choose it. In the **ARN** section, copy the ARN. You'll specify this information in the next procedure.

Add credential overrides to the manifest

After you create the IAM role in your vendor account, update the manifest in the retrieval account. Specifically, add the credentials block and the IAM role ARN for retrieving configuration data from the vendor account. Here is the JSON format:

```
{
    "vendor_application_name:vendor_environment_name:vendor_configuration_name": {
        "credentials": {
            "roleArn": 
  "arn:partition:iam::vendor_account_ID:role/name_of_role_created_in_vendor_account",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

Here is an example:

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AwsAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

Validate that multi-account retrieval is working

You can validate that that agent is able to retrieve configuration data from multiple accounts by reviewing the AWS AppConfig agent logs. The `INFO` level log for retrieved initial data for '`YourApplicationName:YourEnvironmentName:YourConfigurationName`' is the best indicator for successful retrievals. If retrievals are failing, you should see an `ERROR` level log indicating the failure reason. Here is an example for a successful retrieval from a vendor account:

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
```

```
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MyTestApplication:MyTestEnvironment:MyDenyListConfiguration' in XX.Xms
```

Write configuration copy to disk

You can configure AWS AppConfig Agent to automatically store a copy of a configuration to disk in plain text. This feature enables customers with applications that read configuration data from disk to integrate with AWS AppConfig.

This feature is not designed to be used as a configuration backup feature. AWS AppConfig Agent doesn't read from the configuration files copied to disk. If you want to back up configurations to disk, see the `BACKUP_DIRECTORY` and `PRELOAD_BACKUP` environment variables for Using AWS AppConfig Agent with Amazon EC2 or Using AWS AppConfig Agent with Amazon ECS and Amazon EKS.

Warning

Note the following important information about this feature:

- Configurations saved to disk are stored in *plain text* and are human readable. Don't enable this feature for configurations that include sensitive data.
- This feature writes to the local disk. Use the principle of least privilege for filesystem permissions. For more information, see Implement least privilege access.

To enable write configuration copy to disk

1. Edit the manifest.
2. Choose the configuration that you want AWS AppConfig to write to disk and add a `writeTo` element. Here is an example:

```
{
    "application_name:environment_name:configuration_name": {
        "writeTo": {
            "path": "path_to_configuration_file"
        }
    }
}
```

Here is an example:

```
{
    "MyTestApp:MyTestEnvironment:MyNewConfiguration": {
        "writeTo": {
            "path": "/tmp/aws-appconfig/mobile-app/beta/enable-mobile-payments"
        }
    }
}
```

3. Save your changes. The configuration.json file will be updated each time new configuration data is deployed.

Validate that write configuration copy to disk is working

You can validate that copies of a configuration are being written to disk by looking by reviewing the AWS AppConfig agent logs. The INFO log entry with the phrasing "INFO wrote configuration '*application*:*environment*:*configuration*' to *file_path*" indicates that AWS AppConfig Agent writes configuration copies to disk.

Here is an example:

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MobileApp:Beta:EnableMobilePayments' in XX.Xms
[appconfig agent] 2023/11/13 17:05:49 INFO wrote configuration
 'MobileApp:Beta:EnableMobilePayments' to /tmp/configs/your-app/your-env/your-
config.json
```

AWS AppConfig Agent local development

AWS AppConfig Agent supports a *local development mode*. If you enable local development mode, the agent reads configuration data from a specified directory on disk. It doesn't retrieve configuration data from AWS AppConfig. You can simulate configuration deployments by updating files in the specified directory. We recommend local development mode for the following use cases:

- Test different configuration versions before deploying them using AWS AppConfig.

- Test different configuration options for a new feature before committing changes to your code repository.
- Test different configuration scenarios to verify they work as expected.

Warning

Don't use local development mode in production environments. This mode doesn't support important AWS AppConfig safety features like deployment validation and automated rollbacks.

Use the following procedure to configure AWS AppConfig Agent for local development mode.

To configure AWS AppConfig Agent for local development mode

1. Install the agent using the method described for your compute environment. AWS AppConfig Agent works with the following AWS services:
   - AWS Lambda
   - Amazon EC2
   - Amazon ECS and Amazon EKS
2. If the agent is running, stop it.
3. Add `LOCAL_DEVELOPMENT_DIRECTORY` to the list of environment variables. Specify a directory on the filesystem that provides the agent with read permissions. For example, `/tmp/local_configs`.
4. Create a file in the directory. The file name must use the following format:

   ```
   application_name:environment_name:configuration_profile_name
   ```

   Here is an example:

   ```
   Mobile:Development:EnableMobilePaymentsFeatureFlagConfiguration
   ```

> **Note**
>
> (Optional) You can control the content type the agent returns for your configuration data based on the extension you give the file. For example, if you name the file with a .json extension, the agent returns a content type of `application/json` when your application requests it. If you omit the extension, the agent uses `application/octet-stream` for the content type. If you need precise control, you can provide an extension in the format `.`*`type`*`%`*`subtype`*. The agent will return a content type of `.type/subtype`.

5. Run the following command to restart the agent and request the configuration data.

```
curl http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name
```

The agent checks for changes to the local file at the poll interval specified for the agent. If the poll interval isn't specified, the agent uses the default interval of 45 seconds. This check at the poll interval ensures that the agent behaves the same in a local development environment as it does when configured to interact with the AWS AppConfig service.

> **Note**
>
> To deploy a new version of a local development configuration file, update the file with new data.

Retrieving configurations by directly calling APIs

Your application retrieves configuration data by first establishing a configuration session using the StartConfigurationSession API operation. Your session's client then makes periodic calls to GetLatestConfiguration to check for and retrieve the latest data available.

When calling `StartConfigurationSession`, your code sends the following information:

- Identifiers (ID or name) of an AWS AppConfig application, environment, and configuration profile that the session tracks.

- (Optional) The minimum amount of time the session's client must wait between calls to `GetLatestConfiguration`.

In response, AWS AppConfig provides an `InitialConfigurationToken` to be given to the session's client and used the first time it calls `GetLatestConfiguration` for that session.

Important

This token should only be used once in your first call to `GetLatestConfiguration`. You *must* use the new token in the `GetLatestConfiguration` response (`NextPollConfigurationToken`) in each subsequent call to `GetLatestConfiguration`. To support long poll use cases, the tokens are valid for up to 24 hours. If a `GetLatestConfiguration` call uses an expired token, the system returns `BadRequestException`.

When calling `GetLatestConfiguration`, your client code sends the most recent `ConfigurationToken` value it has and receives in response:

- `NextPollConfigurationToken`: the `ConfigurationToken` value to use on the next call to `GetLatestConfiguration`.
- `NextPollIntervalInSeconds`: the duration the client should wait before making its next call to `GetLatestConfiguration`.
- The configuration: the latest data intended for the session. This may be empty if the client already has the latest version of the configuration.

Important

Note the following important information.

- The StartConfigurationSession API should only be called once per application, environment, configuration profile, and client to establish a session with the service. This is typically done in the startup of your application or immediately prior to the first retrieval of a configuration.

- If your configuration is deployed using a `KmsKeyIdentifier`, your request to receive the configuration must include permission to call `kms:Decrypt`. For more information, see Decrypt in the *AWS Key Management Service API Reference*.
- The API operation previously used to retrieve configuration data, `GetConfiguration`, is deprecated. The `GetConfiguration` API operation does not support encrypted configurations.

Retrieving a configuration example

The following AWS CLI example demonstrates how to retrieve configuration data by using the AWS AppConfig Data `StartConfigurationSession` and `GetLatestConfiguration` API operations. The first command starts a configuration session. This call includes the IDs (or names) of the AWS AppConfig application, the environment, and the configuration profile. The API returns an `InitialConfigurationToken` used to fetch your configuration data.

```
aws appconfigdata start-configuration-session \
    --application-identifier application_name_or_ID \
    --environment-identifier environment_name_or_ID \
    --configuration-profile-identifier configuration_profile_name_or_ID
```

The system responds with information in the following format.

```
{
   "InitialConfigurationToken": initial configuration token
}
```

After starting a session, use InitialConfigurationToken to call GetLatestConfiguration to fetch your configuration data. The configuration data is saved to the `mydata.json` file.

```
aws appconfigdata get-latest-configuration \
    --configuration-token initial configuration token mydata.json
```

The first call to `GetLatestConfiguration` uses the `ConfigurationToken` obtained from `StartConfigurationSession`. The following information is returned.

```
{
    "NextPollConfigurationToken" : next configuration token,
    "ContentType" : content type of configuration,
    "NextPollIntervalInSeconds" : 60
}
```

Subsequent calls to `GetLatestConfiguration` *must* provide `NextPollConfigurationToken` from the previous response.

```
aws appconfigdata get-latest-configuration \
    --configuration-token next configuration token mydata.json
```

Important

Note the following important details about the `GetLatestConfiguration` API operation:

- The `GetLatestConfiguration` response includes a `Configuration` section that shows the configuration data. The `Configuration` section only appears if the system finds new or updated configuration data. If the system doesn't find new or updated configuration data, then the `Configuration` data is empty.
- You receive a new `ConfigurationToken` in every response from `GetLatestConfiguration`.
- We recommend tuning the polling frequency of your `GetLatestConfiguration` API calls based on your budget, the expected frequency of your configuration deployments, and the number of targets for a configuration.

Extending workflows using extensions

An extension augments your ability to inject logic or behavior at different points during the AWS AppConfig workflow of creating or deploying a configuration. For example, you can use extensions to perform the following types of tasks (to name a few):

- Send a notification to an Amazon Simple Notification Service (Amazon SNS) topic when a configuration profile is deployed.
- Scrub the contents of a configuration profile for sensitive data before a deployment starts.
- Create or update an Atlassian Jira issue whenever a change is made to a feature flag.
- Merge content from a service or data source into your configuration data when you start a deployment.
- Back up a configuration to an Amazon Simple Storage Service (Amazon S3) bucket whenever a configuration is deployed.

You can associate these types of tasks with AWS AppConfig applications, environments, and configuration profiles.

Contents

About AWS AppConfig extensions

This topic introduces AWS AppConfig extension concepts and terminology. The information is discussed in the context of each step required to set up and use AWS AppConfig extensions.

Topics

Step 1: Determine what you want to do with extensions

Do you want to receive a notification to a webhook that sends messages to Slack anytime an AWS AppConfig deployment completes? Do you want to back up a configuration profile to an Amazon Simple Storage Service (Amazon S3) bucket before a configuration is deployed? Do you want to scrub configuration data for sensitive information before the configuration is deployed? You can use extensions to perform these types of tasks and more. You can create custom extensions or use the AWS authored extensions included with AWS AppConfig.

Note

For most use cases, to create a custom extension, you must create an AWS Lambda function to perform any computation and processing defined in the extension. For more information, see Walkthrough: Creating custom AWS AppConfig extensions.

The following AWS authored extensions can help you quickly integrate configuration deployments with other services. You can use these extensions in the AWS AppConfig console or by calling extension API actions directly from the AWS CLI, AWS Tools for PowerShell, or the SDK.

Extension	Description
Amazon CloudWatch Evidently A/B testing	This extension allows your application to assign variations to user sessions locally instead of by calling the EvaluateFeature operation. For more information, see Working with the Amazon CloudWatch Evidently extension.
AWS AppConfig deployment events to EventBridge	This extension sends events to the EventBrid ge default event bus when a configuration is deployed.
AWS AppConfig deployment events to Amazon Simple Notification Service (Amazon SNS)	This extension sends messages to an Amazon SNS topic that you specify when a configura tion is deployed.

Extension	Description
AWS AppConfig deployment events to Amazon Simple Queue Service (Amazon SQS)	This extension enqueues messages into your Amazon SQS queue when a configuration is deployed.
Integration extension—Atlassian Jira	This extensions allows AWS AppConfig to create and update issues whenever you make changes to a feature flag.

Step 2: Determine when you want the extension to run

An extension defines one or more actions that it performs during an AWS AppConfig workflow. For example, the AWS authored `AWS AppConfig deployment events to Amazon SNS` extension includes an action to send a notification to an Amazon SNS topic. Each action is invoked either when you interact with AWS AppConfig or when AWS AppConfig is performing a process on your behalf. These are called *action points*. AWS AppConfig extensions support the following action points:

- `PRE_CREATE_HOSTED_CONFIGURATION_VERSION`
- `PRE_START_DEPLOYMENT`
- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_STEP`
- `ON_DEPLOYMENT_BAKING`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

Extension actions configured on PRE_* action points are applied after request validation, but before AWS AppConfig performs the activity that corresponds to the action point name. These action invocations are processed at the same time as a request. If more than one request is made, action invocations run sequentially. Also note that PRE_* action points receive and can change the contents of a configuration. PRE_* action points can also respond to an error and prevent an action from happening.

An extension can also run in parallel with an AWS AppConfig workflow by using an `ON_*` action point. `ON_*` action points are invoked asynchronously. `ON_*` action points don't receive the contents of a configuration. If an extension experiences an error during an `ON_*` action point, the service ignores the error and continues the workflow.

Step 3: Create an extension association

To create an extension, or configure an AWS authored extension, you define the action points that invoke an extension when a specific AWS AppConfig resource is used. For example, you can choose to run the `AWS AppConfig deployment events to Amazon SNS` extension and receive notifications on an Amazon SNS topic anytime a configuration deployment is started for a specific application. Defining which action points invoke an extension for a specific AWS AppConfig resource is called an *extension association*. An extension association is a specified relationship between an extension and an AWS AppConfig resource, such as an application or a configuration profile.

A single AWS AppConfig application can include multiple environments and configuration profiles. If you associate an extension to an application or an environment, AWS AppConfig invokes the extension for any workflows that relate to the application or environment resources, if applicable.

For example, say you have an AWS AppConfig application called MobileApps that includes a configuration profile called AccessList. And say the MobileApps application includes Beta, Integration, and Production environments. You create an extension association for the AWS authored Amazon SNS notification extension and associate the extension to the MobileApps application. The Amazon SNS notification extension is invoked anytime the configuration is deployed for the application to any of the three environments.

Note

You don't have to create an extension to use AWS authored extensions, but you do have to create an extension association.

Step 4: Deploy a configuration and verify the extension actions are performed

After you create an association, when a hosted configuration is created or a configuration is deployed, AWS AppConfig invokes the extension and performs the specified actions. When an extension is invoked, if the system experiences an error during a PRE-* action point, AWS AppConfig returns information about that error.

Working with AWS authored extensions

AWS AppConfig includes the following AWS authored extensions. These extensions can help you integrate the AWS AppConfig workflow with other services. You can use these extensions in the AWS Management Console or by calling extension API actions directly from the AWS CLI, AWS Tools for PowerShell, or the SDK.

Extension	Description
Amazon CloudWatch Evidently A/B testing	This extension allows your application to assign variations to user sessions locally instead of by calling the EvaluateFeature operation. For more information, see Working with the Amazon CloudWatch Evidently extension.
AWS AppConfig deployment events to EventBridge	This extension sends events to the EventBrid ge default event bus when a configuration is deployed.
AWS AppConfig deployment events to Amazon Simple Notification Service (Amazon SNS)	This extension sends messages to an Amazon SNS topic that you specify when a configura tion is deployed.
AWS AppConfig deployment events to Amazon Simple Queue Service (Amazon SQS)	This extension enqueues messages into your Amazon SQS queue when a configuration is deployed.
Integration extension—Atlassian Jira	This extensions allows AWS AppConfig to create and update issues whenever you make changes to a feature flag.

Working with the Amazon CloudWatch Evidently extension

You can use Amazon CloudWatch Evidently to safely validate new features by serving them to a specified percentage of your users while you roll out the feature. You can monitor the performance of the new feature to help you decide when to ramp up traffic to your users. This helps you reduce risk and identify unintended consequences before you fully launch the feature. You can also conduct A/B experiments to make feature design decisions based on evidence and data.

The AWS AppConfig extension for CloudWatch Evidently allows your application to assign variations to user sessions locally instead of by calling the EvaluateFeature operation. A local session mitigates the latency and availability risks that come with an API call. For information about how to configure and use the extension, see Perform launches and A/B experiments with CloudWatch Evidently in the *Amazon CloudWatch User Guide*.

Working with the `AWS AppConfig deployment events to Amazon EventBridge` extension

The `AWS AppConfig deployment events to Amazon EventBridge` extension is an AWS authored extension that helps you monitor and act on the AWS AppConfig configuration deployment workflow. The extension sends event notifications to the EventBridge default events bus whenever a configuration is deployed. After you've associated the extension to one of your AWS AppConfig applications, environments, or configuration profiles, AWS AppConfig sends event notifications to the event bus after every configuration deployment start, end, and rollback.

If you want more control over which action points send EventBridge notifications, you can create a custom extension and enter the EventBridge default events bus Amazon Resource Name (ARN) for the URI field. For information about creating an extension, see Walkthrough: Creating custom AWS AppConfig extensions.

Important

This extension supports only the EventBridge default events bus.

Using the extension

To use the `AWS AppConfig deployment events to Amazon EventBridge` extension, you first attach the extension to one of your AWS AppConfig resources by creating an extension association. You create the association by using the AWS AppConfig console or the

CreateExtensionAssociation API action. When you create the association, you specify the ARN of an AWS AppConfig application, environment, or configuration profile. If you associate the extension to an application or an environment, an event notification is sent for any configuration profile contained within the specified application or environment.

After you create the association, when a configuration for the specified AWS AppConfig resource is deployed, AWS AppConfig invokes the extension and sends notifications according to the action points specified in the extension.

Note

This extension is invoked by the following action points:

- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

You can't customize the actions points for this extension. To invoke different action points, you can create your own extension. For more information, see Walkthrough: Creating custom AWS AppConfig extensions.

Use the following procedures to create an AWS AppConfig extension association by using either the AWS Systems Manager console or the AWS CLI.

To create an extension association (console)

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **AWS AppConfig**.
3. On the **Extensions** tab, choose **Add to resource**.
4. In the **Extension resource details** section, for **Resource type**, choose an AWS AppConfig resource type. Depending on the resource you choose, AWS AppConfig prompts you to choose other resources.
5. Choose **Create association to resource**.

Here's a sample event sent to EventBridge when the extension is invoked.

```
{
   "version":"0",
   "id":"c53dbd72-c1a0-2302-9ed6-c076e9128277",
   "detail-type":"On Deployment Complete",
   "source":"aws.appconfig",
   "account":"111122223333",
   "time":"2022-07-09T01:44:15Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:appconfig:us-east-1:111122223333:extensionassociation/z763ff5"
   ],
   "detail":{
      "InvocationId":"5tfjcig",
       "Parameters":{

      },
      "Type":"OnDeploymentComplete",
      "Application":{
         "Id":"ba8toh7",
         "Name":"MyApp"
      },
      "Environment":{
         "Id":"pgil2o7",
         "Name":"MyEnv"
      },
      "ConfigurationProfile":{
         "Id":"ga3tqep",
         "Name":"MyConfigProfile"
      },
      "DeploymentNumber":1,
      "ConfigurationVersion":"1"
   }
}
```

Working with the `AWS AppConfig deployment events to Amazon SNS` extension

The `AWS AppConfig deployment events to Amazon SNS` extension is an AWS authored extension that helps you monitor and act on the AWS AppConfig configuration deployment workflow. The extension publishes messages to an Amazon SNS topic whenever a configuration is deployed. After you associate the extension to one of your AWS AppConfig applications, environments, or configuration profiles, AWS AppConfig publishes a message to the topic after every configuration deployment start, end, and rollback.

If you want more control over which action points send Amazon SNS notifications, you can create a custom extension and enter an Amazon SNS topic Amazon Resource Name (ARN) for the URI field. For information about creating an extension, see Walkthrough: Creating custom AWS AppConfig extensions.

Using the extension

This section describes how to use the `AWS AppConfig deployment events to Amazon SNS` extension.

Step 1: Configure AWS AppConfig to publish messages to a topic

Add an access control policy to your Amazon SNS topic granting AWS AppConfig (`appconfig.amazonaws.com`) publish permissions (`sns:Publish`). For more information, see Example cases for Amazon SNS access control.

Step 2: Create an extension association

Attach the extension to one of your AWS AppConfig resources by creating an extension association. You create the association by using the AWS AppConfig console or the CreateExtensionAssociation API action. When you create the association, you specify the ARN of an AWS AppConfig application, environment, or configuration profile. If you associate the extension to an application or an environment, a notification is sent for any configuration profile contained within the specified application or environment. When you create the association, you must enter a value for the `topicArn` parameter that contains the ARN of the Amazon SNS topic you want to use.

After you create the association, when a configuration for the specified AWS AppConfig resource is deployed, AWS AppConfig invokes the extension and sends notifications according to the action points specified in the extension.

Note

This extension is invoked by the following action points:

- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

You can't customize the actions points for this extension. To invoke different action points, you can create your own extension. For more information, see Walkthrough: Creating custom AWS AppConfig extensions.

Use the following procedures to create an AWS AppConfig extension association by using either the AWS Systems Manager console or the AWS CLI.

To create an extension association (console)

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **AWS AppConfig**.
3. On the **Extensions** tab, choose **Add to resource**.
4. In the **Extension resource details** section, for **Resource type**, choose an AWS AppConfig resource type. Depending on the resource you choose, AWS AppConfig prompts you to choose other resources.
5. Choose **Create association to resource**.

Here's a sample of the message sent to the Amazon SNS topic when the extension is invoked.

```
{
    "Type": "Notification",
    "MessageId": "ae9d702f-9a66-51b3-8586-2b17932a9f28",
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic",
    "Message": {
        "InvocationId": "7itcaxp",
        "Parameters": {
            "topicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic"
        },
        "Application": {
            "Id": "1a2b3c4d",
            "Name": MyApp
        },
        "Environment": {
            "Id": "1a2b3c4d",
            "Name": MyEnv
        },
        "ConfigurationProfile": {
            "Id": "1a2b3c4d",
            "Name": "MyConfigProfile"
        },
        "Description": null,
        "DeploymentNumber": "3",
        "ConfigurationVersion": "1",
        "Type": "OnDeploymentComplete"
    },
    "Timestamp": "2022-06-30T20:26:52.067Z",
    "SignatureVersion": "1",
    "Signature": "<...>",
    "SigningCertURL": "<...>",
    "UnsubscribeURL": "<...>",
    "MessageAttributes": {
        "MessageType": {
            "Type": "String",
            "Value": "OnDeploymentStart"
        }
    }
}
```

Working with the `AWS AppConfig deployment events to Amazon` SQS extension

The `AWS AppConfig deployment events to Amazon SQS` extension is an AWS authored extension that helps you monitor and act on the AWS AppConfig configuration deployment workflow. The extension enqueues messages into your Amazon Simple Queue Service (Amazon SQS) queue whenever a configuration is deployed. After you associate the extension to one of your AWS AppConfig applications, environments, or configuration profiles, AWS AppConfig enqueues a message into the queue after every configuration deployment start, end, and rollback.

If you want more control over which action points send Amazon SQS notifications, you can create a custom extension and enter an Amazon SQS queue Amazon Resource Name (ARN) for the URI field. For information about creating an extension, see Walkthrough: Creating custom AWS AppConfig extensions.

Using the extension

This section describes how to use the `AWS AppConfig deployment events to Amazon SQS` extension.

Step 1: Configure AWS AppConfig to enqueue messages

Add an Amazon SQS policy to your Amazon SQS queue granting AWS AppConfig (`appconfig.amazonaws.com`) send message permissions (`sqs:SendMessage`). For more information, see Basic examples of Amazon SQS policies.

Step 2: Create an extension association

Attach the extension to one of your AWS AppConfig resources by creating an extension association. You create the association by using the AWS AppConfig console or the CreateExtensionAssociation API action. When you create the association, you specify the ARN of an AWS AppConfig application, environment, or configuration profile. If you associate the extension to an application or an environment, a notification is sent for any configuration profile contained within the specified application or environment. When you create the association, you must enter a `Here` parameter that contains the ARN of the Amazon SQS queue you want to use.

After you create the association, when a configuration for the specified AWS AppConfig resource is created or deployed, AWS AppConfig invokes the extension and sends notifications according to the action points specified in the extension.

Note

This extension is invoked by the following action points:

- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

You can't customize the actions points for this extension. To invoke different action points, you can create your own extension. For more information, see Walkthrough: Creating custom AWS AppConfig extensions.

Use the following procedures to create an AWS AppConfig extension association by using either the AWS Systems Manager console or the AWS CLI.

To create an extension association (console)

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **AWS AppConfig**.
3. On the **Extensions** tab, choose **Add to resource**.
4. In the **Extension resource details** section, for **Resource type**, choose an AWS AppConfig resource type. Depending on the resource you choose, AWS AppConfig prompts you to choose other resources.
5. Choose **Create association to resource**.

Here's an example of the message sent to the Amazon SQS queue when the extension is invoked.

```
{
   "InvocationId":"7itcaxp",
   "Parameters":{
      "queueArn":"arn:aws:sqs:us-east-1:111122223333:MySQSQueue"
   },
   "Application":{
      "Id":"1a2b3c4d",
      "Name":MyApp
   },
   "Environment":{
      "Id":"1a2b3c4d",
      "Name":MyEnv
   },
   "ConfigurationProfile":{
      "Id":"1a2b3c4d",
      "Name":"MyConfigProfile"
   },
   "Description":null,
   "DeploymentNumber":"3",
   "ConfigurationVersion":"1",
   "Type":"OnDeploymentComplete"
}
```

Working with the Atlassian Jira extension for AWS AppConfig

By integrating with Atlassian Jira, AWS AppConfig can create and update issues in the Atlassian console whenever you make changes to a feature flag in your AWS account for the specified AWS

Region. Each Jira issue includes the flag name, application ID, configuration profile ID, and flag values. After you update, save, and deploy your flag changes, Jira updates the existing issues with the details of the change.

Note

Jira updates issues whenever you create or update a feature flag. Jira also updates issues when you delete a child-level flag attribute from a parent-level flag. Jira does not record information when you delete a parent-level flag.

To configure integration, you must do the following:

- Configuring permissions for AWS AppConfig Jira integration
- Configuring the AWS AppConfig Jira integration application

Configuring permissions for AWS AppConfig Jira integration

When you configure AWS AppConfig integration with Jira, you specify credentials for a user. Specifically, you enter the user's access key ID and secret key in the **AWS AppConfig for Jira** application. This user gives Jira permission to communicate with AWS AppConfig. AWS AppConfig uses these credentials one time to establish an association between AWS AppConfig and Jira. The credentials are not stored. You can remove the association by uninstalling the AWS AppConfig for Jira application.

The user account requires a permission policy that includes the following actions:

- `appconfig:CreateExtensionAssociation`
- `appconfig:GetConfigurationProfile`
- `appconfig:ListApplications`
- `appconfig:ListConfigurationProfiles`
- `appconfig:ListExtensionAssociations`
- `sts:GetCallerIdentity`

Complete the following tasks to create an IAM permission policy and a user for AWS AppConfig and Jira integration:

Tasks

Task 1: Create an IAM permission policy for AWS AppConfig and Jira integration

Use the following procedure to create an IAM permission policy that allows Atlassian Jira to communicate with AWS AppConfig. We recommend that you create a new policy and attach this policy to a new IAM role. Adding the required permission to an existing IAM policy and role goes against the principle of least privilege and is not recommended.

To create an IAM policy for AWS AppConfig and Jira integration

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **Create policy** page, choose the **JSON** tab and replace the default content with the following policy. In the following policy, replace *Region*, *account_ID*, *application_ID*, and *configuration_profile_ID* with information from your AWS AppConfig feature flag environment.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                        "appconfig:CreateExtensionAssociation",
                        "appconfig:ListExtensionAssociations",
                        "appconfig:GetConfigurationProfile"
             ],
            "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID",

 "arn:aws:appconfig:Region:account_ID:application/application_ID/
configurationprofile/configuration_profile_ID"
```

```
                ]
            },
           {
                "Effect": "Allow",
                "Action": [
                            "appconfig:ListApplications"

                 ],
                "Resource": [
                            "arn:aws:appconfig:Region:account_ID:*"
                 ]
            },
             {
                "Effect": "Allow",
                "Action": [
                            "appconfig:ListConfigurationProfiles"
                 ],
                "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID"
                 ]
            },
            {
                "Effect": "Allow",
                "Action": "sts:GetCallerIdentity",
                "Resource": "*"
            }
        ]
}
```

4. Choose **Next: Tags**.
5. (Optional) Add one or more tag-key value pairs to organize, track, or control access for this policy, and then choose **Next: Review**.
6. On the **Review policy** page, enter a name in the **Name** box, such as **`AppConfigJiraPolicy`**, and then enter an optional description.
7. Choose **Create policy**.

Task 2: Create a user for AWS AppConfig and Jira integration

Use the following procedure to create a user for AWS AppConfig and Atlassian Jira integration. After you create the user, you can copy the access key ID and secret key, which you will specify when you complete the integration.

To create a user for AWS AppConfig and Jira integration

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**, and then choose **Add users**.
3. In the **User name** field, enter a name, such as **`AppConfigJiraUser`**.
4. For **Select AWS credential type**, choose **Access key - Programmatic access**.
5. Choose **Next: Permissions**.
6. Under **Set permissions** page, choose **Attach existing policies directly**. Search for and select the check box for the policy that you created in Task 1: Create an IAM permission policy for AWS AppConfig and Jira integration, and then choose **Next: Tags**.
7. On the **Add tags (optional)** page, add one or more tag-key value pairs to organize, track, or control access for this user. Choose **Next: Review**.
8. On the **Review** page, verify the user details.
9. Choose **Create user**. The system displays the user's access key ID and secret key. Either download the .csv file or copy these credentials to a separate location. You will specify these credentials when you configure integration.

Configuring the AWS AppConfig Jira integration application

Use the following procedure to configure required options in the AWS AppConfig for Jira application. After you complete this procedure, Jira creates a new issue for each feature flag in your AWS account for the specified AWS Region. If you make changes to a feature flag in AWS AppConfig, Jira records the details in the existing issues.

Note

An AWS AppConfig feature flag can include multiple child-level flag attributes. Jira creates one issue for each parent-level feature flag. If you change a child-level flag attribute, you can view the details of that change in the Jira issue for the parent-level flag.

To configure integration

1. Log in to the Atlassian Marketplace.
2. Type **`AWS AppConfig`** in the search field and press **Enter**.
3. Install the application on your Jira instance.
4. In the Atlassian console, choose **Manage apps**, and then choose **AWS AppConfig for Jira**.
5. Choose **Configure**.
6. Under **Configuration details**, choose **Jira project** and then choose the project that you want to associate with your AWS AppConfig feature flag.
7. Choose **AWS Region**, and then choose the Region where your AWS AppConfig feature flag is located.
8. In the **Application ID** field, enter the name of the AWS AppConfig application that contains your feature flag.
9. In the **Configuration profile ID** field, enter the name of the AWS AppConfig configuration profile for your feature flag.
10. In the **Access key ID** and **Secret key** fields, enter the credentials you copied in Task 2: Create a user for AWS AppConfig and Jira integration. Optionally, you can also specify a session token.
11. Choose **Submit**.
12. In the Atlassian console, choose **Projects**, and then choose the project you selected for AWS AppConfig integration. The **Issues** page displays an issue for each feature flag in the specified AWS account and AWS Region.

Deleting the AWS AppConfig for Jira application and data

If you no longer want to use Jira integration with AWS AppConfig feature flags, you can delete the AWS AppConfig for Jira application in the Atlassian console. Deleting the integration application does the following:

- Deletes the association between your Jira instance and AWS AppConfig
- Deletes your Jira instance details from AWS AppConfig

To delete the AWS AppConfig for Jira application

1. In the Atlassian console, choose **Manage apps**.

2. Choose **AWS AppConfig for Jira**.
3. Choose **Uninstall**.

Walkthrough: Creating custom AWS AppConfig extensions

To create a custom AWS AppConfig extension, complete the following tasks. Each task is described in more detail in later topics.

Note

You can view samples of custom AWS AppConfig extensions on GitHub:

- Sample extension that prevents deployments with a `blocked day` moratorium calendar using Systems Manager Change Calendar
- Sample extension that prevents secrets from leaking into configuration data using git-secrets
- Sample extension that prevents personally identifiable information (PII) from leaking into configuration data using Amazon Comprehend

1. Create an AWS Lambda function

For most use cases, to create a custom extension, you must create an AWS Lambda function to perform any computation and processing defined in the extension. An exception to this rule is if you create *custom* versions of the AWS authored notification extensions to add or remove action points. For more details about this exception, see Creating a custom AWS AppConfig extension.

2. Configure permissions for your custom extension

To configure permissions for your custom extension, you can do one of the following:

- Create an AWS Identity and Access Management (IAM) service role that includes `InvokeFunction` permissions.
- Create a resource policy by using the Lambda AddPermission API action.

This walkthrough describes how to create the IAM service role.

3. Create an extension

You can create an extension by using the AWS AppConfig console or by calling the CreateExtension API action from the AWS CLI, AWS Tools for PowerShell, or the SDK. The walkthrough uses the console.

4. Create an extension association

You can create an extension association by using the AWS AppConfig console or by calling the CreateExtensionAssociation API action from the AWS CLI, AWS Tools for PowerShell, or the SDK. The walkthrough uses the console.

5. Perform an action that invokes the extension

After you create the association, AWS AppConfig invokes the extension when the action points defined by the extension occur for that resource. For example, if you associate an extension that contains a `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` action, the extension is invoked every time you create a new hosted configuration version.

The topics in this section describe each task involved in creating a custom AWS AppConfig extension. Each task is described in the context of a use case where a customer wants to create an extension that automatically backs up a configuration to an Amazon Simple Storage Service (Amazon S3) bucket. The extension runs whenever a hosted configuration is created (`PRE_CREATE_HOSTED_CONFIGURATION_VERSION`) or deployed (`PRE_START_DEPLOYMENT`).

Topics

Creating a Lambda function for a custom AWS AppConfig extension

For most use-cases, to create a custom extension, you must create an AWS Lambda function to perform any computation and processing defined in the extension. This section includes Lambda function sample code for a custom AWS AppConfig extension. This section also includes payload request and response reference details. For information about creating a Lambda function, see Getting started with Lambda in the *AWS Lambda Developer Guide*.

Sample code

The following sample code for a Lambda function, when invoked, automatically backs up an AWS AppConfig configuration to an Amazon S3 bucket. The configuration is backed up whenever a new configuration is created or deployed. The sample uses extension parameters so the bucket name doesn't have to be hardcoded in the Lambda function. By using extension parameters, the user can attach the extension to multiple applications and back up configurations to different buckets. The code sample includes comments to further explain the function.

Sample Lambda function for an AWS AppConfig extension

```
from datetime import datetime
import base64
import json

import boto3


def lambda_handler(event, context):
    print(event)

    # Extensions that use the PRE_CREATE_HOSTED_CONFIGURATION_VERSION and
 PRE_START_DEPLOYMENT
    # action points receive the contents of AWS AppConfig configurations in Lambda
 event parameters.
    # Configuration contents are received as a base64-encoded string, which the lambda
 needs to decode
    # in order to get the configuration data as bytes. For other action points, the
 content
    # of the configuration isn't present, so the code below will fail.
    config_data_bytes = base64.b64decode(event["Content"])

    # You can specify parameters for extensions. The CreateExtension API action lets
 you define
    # which parameters an extension supports. You supply the values for those
 parameters when you
    # create an extension association by calling the CreateExtensionAssociation API
 action.
    # The following code uses a parameter called S3_BUCKET to obtain the value
 specified in the
    # extension association. You can specify this parameter when you create the
 extension
    # later in this walkthrough.
    extension_association_params = event.get('Parameters', {})
    bucket_name = extension_association_params['S3_BUCKET']
    write_backup_to_s3(bucket_name, config_data_bytes)

    # The PRE_CREATE_HOSTED_CONFIGURATION_VERSION and PRE_START_DEPLOYMENT action
 points can
    # modify the contents of a configuration. The following code makes a minor change
    # for the purposes of a demonstration.
    old_config_data_string = config_data_bytes.decode('utf-8')
    new_config_data_string = old_config_data_string.replace('hello', 'hello!')
    new_config_data_bytes = new_config_data_string.encode('utf-8')

    # The lambda initially received the configuration data as a base64-encoded string
    # and must return it in the same format.
    new_config_data_base64string =
 base64.b64encode(new_config_data_bytes).decode('ascii')

    return {
        'statusCode': 200,
        # If you want to modify the contents of the configuration, you must include the
 new contents in the
        # Lambda response. If you don't want to modify the contents, you can omit the
 'Content' field shown here.
        'Content': new_config_data_base64string
    }


def write_backup_to_s3(bucket_name, config_data_bytes):
    s3 = boto3.resource('s3')
    new_object = s3.Object(bucket_name,
 f"config_backup_{datetime.now().isoformat()}.txt")
    new_object.put(Body=config_data_bytes)
```

If you want to use this sample during this walkthrough, save it with the name **`MyS3ConfigurationBackUpExtension`** and copy the Amazon Resource Name (ARN) for the function. You specify the ARN when you create the AWS Identity and Access Management (IAM) assume role in the next section. You specify the ARN and the name when you create the extension.

Payload reference

This section includes payload request and response reference details for working with custom AWS AppConfig extensions.

Request structure

PreCreateHostedConfigurationVersion

```
{
    'InvocationId': 'vlns753', // id for specific invocation
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh', // Base64 encoded content
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'Description': '',
    'Type': 'PreCreateHostedConfigurationVersion',
    'PreviousContent': {
        'ContentType': 'text/plain',
        'ContentVersion': '1',
        'Content': 'SGVsbG8gd29ybGQh'
    }
}
```

PreStartDeployment

```
{
    'InvocationId': '765ahdm',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh',
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'Environment': {
        'Id': 'ibpnqlq',
        'Name': 'EnvironmentName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'Type': 'PreStartDeployment'
}
```

Asynchronous events

OnStartDeployment, OnDeploymentStep, OnDeployment

```
{
    'InvocationId': 'o2xbtm7',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'Type': 'OnDeploymentStart',
    'Application': {
        'Id': 'abcd123'
    },
    'Environment': {
        'Id': 'efgh456'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'ConfigurationVersion': '2'
}
```

Response structure

The following examples show what your Lambda fuction returns in response to the request from a custom AWS AppConfig extension.

Synchronous events - successful response

If you want to transform the content, use the following:

```
"Content": "SomeBase64EncodedByteArray"
```

If you don't want to transform the content, return nothing.

Asynchronous events - successful response

Return nothing.

All error events

```
{
        "Error": "BadRequestError",
        "Message": "There was malformed stuff in here",
        "Details": [{
            "Type": "Malformed",
            "Name": "S3 pointer",
            "Reason": "S3 bucket did not exist"
        }]
    }
```

Configuring permissions for a custom AWS AppConfig extension

Use the following procedure to create and configure an AWS Identity and Access Management (IAM) service role (or *assume role*). AWS AppConfig uses this role to invoke the Lambda function.

To create an IAM service role and allow AWS AppConfig to assume it

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**, and then choose **Create role**.
3. Under **Select type of trusted entity**, choose **Custom trust policy**.
4. Paste the following JSON policy into the **Custom trust policy** field.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   Choose **Next**.

5. On the **Add permissions** page, choose **Create policy**. The **Create policy** page opens in a new tab.
6. Choose the **JSON** tab, and then paste the following permission policy into the editor. The `lambda:InvokeFunction` action is used for `PRE_*` action points. The `lambda:InvokeAsync` action is used for `ON_*` action points. Replace *`Your Lambda ARN`* with the Amazon Resource Name (ARN) of your Lambda.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:InvokeAsync"
      ],
      "Resource": "Your Lambda ARN"
    }
  ]
}
```

7. Choose **Next: Tags**.
8. On the **Add tags (Optional)** page, add one or more key-value pairs and then choose **Next: Review**.
9. On the **Review policy** page enter a name and a description, and then choose **Create policy**.
10. On the browser tab for your custom trust policy, choose the Refresh icon and then search for the permission policy you just created.

11. Select the check box for your permission policy and then choose **Next**.
12. On the **Name, review, and create** page, enter a name in the **Role name** box, and then enter a description.
13. Choose **Create role**. The system returns you to the **Roles** page. Choose **View role** in the banner.
14. Copy the ARN. You specify this ARN when you create the extension.

Creating a custom AWS AppConfig extension

An extension defines one or more actions that it performs during an AWS AppConfig workflow. For example, the AWS authored `AWS AppConfig deployment events to Amazon SNS` extension includes an action to send a notification to an Amazon SNS topic. Each action is invoked either when you interact with AWS AppConfig or when AWS AppConfig is performing a process on your behalf. These are called *action points*. AWS AppConfig extensions support the following action points:

- `PRE_CREATE_HOSTED_CONFIGURATION_VERSION`
- `PRE_START_DEPLOYMENT`
- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_STEP`
- `ON_DEPLOYMENT_BAKING`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

Extension actions configured on `PRE_*` action points are applied after request validation, but before AWS AppConfig performs the activity that corresponds to the action point name. These action invocations are processed at the same time as a request. If more than one request is made, action invocations run sequentially. Also note that `PRE_*` action points receive and can change the contents of a configuration. `PRE_*` action points can also respond to an error and prevent an action from happening.

An extension can also run in parallel with an AWS AppConfig workflow by using an `ON_*` action point. `ON_*` action points are invoked asynchronously. `ON_*` action points don't receive the contents of a configuration. If an extension experiences an error during an `ON_*` action point, the service ignores the error and continues the workflow.

The following sample extension defines one action that calls the `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` action point. In the `Uri` field, the action specifies the Amazon Resource Name (ARN) of the `MyS3ConfigurationBackUpExtension` Lambda function created earlier in this walkthrough. The action also specifies the AWS Identity and Access Management (IAM) assume role ARN created earlier in this walkthrough.

Sample AWS AppConfig extension

```
{
    "Name": "MySampleExtension",
    "Description": "A sample extension that backs up configurations to an S3 bucket.",
    "Actions": {
        "PRE_CREATE_HOSTED_CONFIGURATION_VERSION": [
            {
                "Name": "PreCreateHostedConfigVersionActionForS3Backup",
                "Uri": "arn:aws:lambda:aws-
region:111122223333:function:MyS3ConfigurationBackUpExtension",
                "RoleArn": "arn:aws:iam::111122223333:role/ExtensionsTestRole"
            }
        ]
    },
    "Parameters" : {
        "S3_BUCKET": {
            "Required": false
        }
    }
}
```

Note

To view request syntax and field descriptions when creating an extension, see the CreateExtension topic in the *AWS AppConfig API Reference*.

To create an extension (console)

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.

2. In the navigation pane, choose **AWS AppConfig**.
3. On the **Extensions** tab, choose **Create extension**.
4. For **Extension name**, enter a unique name. For the purposes of this walkthrough, enter **`MyS3ConfigurationBackUpExtension`**. Optionally, enter a description.
5. In the **Actions** section, choose **Add new action**.
6. For **Action name**, enter a unique name. For the purposes of this walkthrough, enter **`PreCreateHostedConfigVersionActionForS3Backup`**. This name describes the action point used by the action and the extension purpose.
7. In the **Action point** list, choose **PRE_CREATE_HOSTED_CONFIGURATION_VERSION**.
8. For **Uri**, choose **Lambda function** and then choose the function in the **Lambda function** list. If you don't see your function, verify that you are in the same AWS Region where you created the function.
9. For **IAM Role**, choose the role you created earlier in this walkthrough.
10. In the **Extension parameters (optional)** section, choose **Add new parameter**.
11. For **Parameter name**, enter a name. For the purposes of this walkthrough, enter **`S3_BUCKET`**.
12. Repeat steps 5–11 to create a second action for the `PRE_START_DEPLOYMENT` action point.
13. Choose **Create extension**.

Customizing AWS authored notification extensions

You don't have to create a Lambda or an extension to use AWS authored notification extensions. You can simply create an extension association and then perform an operation that calls one of the supported action points. By default, the AWS authored notification extensions support the following actions points:

- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

If you create custom versions of the `AWS AppConfig deployment events to Amazon SNS` extension and `AWS AppConfig deployment events to Amazon SQS` extensions, you can specify the action points for which you want to receive notifications.

Note

The `AWS AppConfig deployment events to EventBridge` extension doesn't support the `PRE_*` action points. You can create a custom version if you want to remove some of the default actions points assigned to the AWS authored version.

You don't need to create a Lambda function if you create custom versions of the AWS authored notification extensions. You only need to specify an Amazon Resource Name (ARN) in the `Uri` field for the new extension version.

- For a custom EventBridge notification extension, enter the ARN of the EventBridge default events in the `Uri` field.
- For a custom Amazon SNS notification extension, enter the ARN of an Amazon SNS topic in the `Uri` field.
- For a custom Amazon SQS notification extension, enter the ARN of an Amazon SQS message queue in the `Uri` field.

Creating an extension association for a custom AWS AppConfig extension

To create an extension, or configure an AWS authored extension, you define the action points that invoke an extension when a specific AWS AppConfig resource is used. For example, you can choose to run the `AWS AppConfig deployment events to Amazon SNS` extension and receive notifications on an Amazon SNS topic anytime a configuration deployment is started for a specific application. Defining which action points invoke an extension for a specific AWS AppConfig resource is called an *extension association*. An extension association is a specified relationship between an extension and an AWS AppConfig resource, such as an application or a configuration profile.

A single AWS AppConfig application can include multiple environments and configuration profiles. If you associate an extension to an application or an environment, AWS AppConfig invokes the extension for any workflows that relate to the application or environment resources, if applicable.

For example, say you have an AWS AppConfig application called MobileApps that includes a configuration profile called AccessList. And say the MobileApps application includes Beta, Integration, and Production environments. You create an extension association for the AWS authored Amazon SNS notification extension and associate the extension to the MobileApps application. The Amazon SNS notification extension is invoked anytime the configuration is deployed for the application to any of the three environments.

Use the following procedures to create an AWS AppConfig extension association by using the AWS AppConfig console.

To create an extension association (console)

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/appconfig/.
2. In the navigation pane, choose **AWS AppConfig**.
3. On the **Extensions** tab, choose an option button for an extension and then choose **Add to resource**. For the purposes of this walkthrough, choose **MyS3ConfigurationBackUpExtension**.
4. In the **Extension resource details** section, for **Resource type**, choose an AWS AppConfig resource type. Depending on the resource you choose, AWS AppConfig prompts you to choose other resources. For the purposes of this walkthrough, choose **Application**.
5. Choose an application in the list.
6. In the **Parameters** section, verify that **S3_BUCKET** is listed in the **Key** field. In the **Value** field, paste the ARN of the Lambda extensions. For example: `arn:aws:lambda:`*`aws-region`*`:111122223333:function:MyS3ConfigurationBackUpExtension`.
7. Choose **Create association to resource**.

After you create the association, you can invoke the `MyS3ConfigurationBackUpExtension` extension by creating a new configuration profile that specifies `hosted` for its `SourceUri`. As a part of the workflow to create the new configuration, AWS AppConfig encounters the `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` action point. Encountering this action point invokes the `MyS3ConfigurationBackUpExtension` extension, which automatically backs up the newly created configuration to the S3 bucket specified in the `Parameter` section of the extension association.

AWS AppConfig extension integration with Atlassian Jira

AWS AppConfig integrates with Atlassian Jira. Integration allows AWS AppConfig to create and update issues in the Atlassian console whenever you make changes to a feature flag in your AWS account for the specified AWS Region. Each Jira issue includes the flag name, application ID, configuration profile ID, and flag values. After you update, save, and deploy your flag changes, Jira updates the existing issues with the details of the change. For more information, see Working with the Atlassian Jira extension for AWS AppConfig.

Code samples for performing common AWS AppConfig tasks

This section includes code samples for programmatically performing common AWS AppConfig actions. We recommend you use these samples with the Java, Python, and JavaScript SDKs to perform the actions in a test environment. This section includes a code sample for cleaning up your test environment after you finish.

Topics

Creating or updating a freeform configuration stored in the hosted configuration store

Each of the following samples includes comments about the actions performed by the code. The samples in this section call the following APIs:

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

Java

```
public CreateHostedConfigurationVersionResponse createHostedConfigVersion() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));

        return hcv;
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a hosted, freeform configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a hosted, freeform configuration profile
const profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: profile.Id,
    ContentType: "text/plain",
    Content: "my config data",
  })
);
```

Creating a configuration profile for a secret stored in Secrets Manager

Each of the following samples includes comments about the actions performed by the code. The samples in this section call the following APIs:

- CreateApplication
- CreateConfigurationProfile

Java

```
private void createSecretsManagerConfigProfile() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a configuration profile for Secrets Manager Secret
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("secretsmanager://MySecret")
            .retrievalRoleArn("arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret")
            .type("AWS.Freeform"));
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a configuration profile for Secrets Manager Secret
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='secretsmanager://MySecret',
    RetrievalRoleArn='arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret',
    Type='AWS.Freeform')
```

JavaScript

```
import {
  AppConfigClient,
  CreateConfigurationProfileCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a configuration profile for Secrets Manager Secret
await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "secretsmanager://MySecret",
    RetrievalRoleArn: "arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret",
    Type: "AWS.Freeform",
  })
);
```

Deploying a configuration profile

Each of the following samples includes comments about the actions performed by the code. The samples in this section call the following APIs:

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

- CreateEnvironment
- StartDeployment
- GetDeployment

Java

```
private void createDeployment() throws InterruptedException {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));


        // Create an environment
        CreateEnvironmentResponse env = appconfig.createEnvironment(req -> req
            .applicationId(app.id())
            .name("Beta")
            // If you have CloudWatch alarms that monitor the health of your
 service, you can add them here and they
            // will trigger a rollback if they fire during an appconfig deployment
            //.monitors(Monitor.builder().alarmArn("arn:aws:cloudwatch:us-
east-1:520900602629:alarm:MyAlarm")
            //
  .alarmRoleArn("arn:aws:iam::520900602629:role/MyAppConfigAlarmRole").build())
        );
        // Start a deployment
        StartDeploymentResponse deploymentResponse = appconfig.startDeployment(req -
> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .environmentId(env.id())
            .configurationVersion(hcv.versionNumber().toString())
            .deploymentStrategyId("AppConfig.Linear50PercentEvery30Seconds")
        );

        // Wait for deployment to complete
        List<DeploymentState> nonFinalDeploymentStates = Arrays.asList(
            DeploymentState.DEPLOYING,
            DeploymentState.BAKING,
            DeploymentState.ROLLING_BACK,
            DeploymentState.VALIDATING);
        GetDeploymentRequest getDeploymentRequest =
 GetDeploymentRequest.builder().applicationId(app.id())

 .environmentId(env.id())

 .deploymentNumber(deploymentResponse.deploymentNumber()).build();
        GetDeploymentResponse deployment =
 appconfig.getDeployment(getDeploymentRequest);
        while (nonFinalDeploymentStates.contains(deployment.state())) {
            System.out.println("Waiting for deployment to complete: " + deployment);
            Thread.sleep(1000L);
            deployment = appconfig.getDeployment(getDeploymentRequest);
        }

        System.out.println("Deployment complete: " + deployment);
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')
```

```
# create an environment
environment = appconfig.create_environment(
    ApplicationId=application['Id'],
    Name='MyEnvironment')

# create a configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')

# start a deployment
deployment = appconfig.start_deployment(
    ApplicationId=application['Id'],
    EnvironmentId=environment['Id'],
    ConfigurationProfileId=config_profile['Id'],
    ConfigurationVersion=str(hcv['VersionNumber']),
    DeploymentStrategyId='AppConfig.Linear20PercentEvery6Minutes')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateEnvironmentCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
  StartDeploymentCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create an environment
const environment = await appconfig.send(
  new CreateEnvironmentCommand({
    ApplicationId: application.Id,
    Name: "MyEnvironment",
  })
);

// create a configuration profile
const config_profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
const hcv = await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: config_profile.Id,
    Content: "my config data",
    ContentType: "text/plain",
  })
);

// start a deployment
await appconfig.send(
  new StartDeploymentCommand({
    ApplicationId: application.Id,
    EnvironmentId: environment.Id,
    ConfigurationProfileId: config_profile.Id,
    ConfigurationVersion: hcv.VersionNumber.toString(),
    DeploymentStrategyId: "AppConfig.Linear20PercentEvery6Minutes",
  })
);
```

Using AWS AppConfig Agent to read a freeform configuration profile

Each of the following samples includes comments about the actions performed by the code.

Java

```
public void retrieveConfigFromAgent() throws Exception {
        /*
        In this sample, we will retrieve configuration data from the AWS AppConfig
 Agent.
        The agent is a sidecar process that handles retrieving configuration data
 from AppConfig
        for you in a way that implements best practices like configuration caching.

        For more information about the agent, see Simplified retrieval methods
        */

        // The agent runs a local HTTP server that serves configuration data
        // Make a GET request to the agent's local server to retrieve the
 configuration data
        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyConfigProfile");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("Configuration from agent via HTTP: " + content);
    }
```

Python

```
# in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
# the agent is a sidecar process that handles retrieving configuration data from AWS
 AppConfig
# for you in a way that implements best practices like configuration caching.
#
# for more information about the agent, see
# Simplified retrieval methods
#

import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

# the agent runs a local HTTP server that serves configuration data
# make a GET request to the agent's local server to retrieve the configuration data
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}")
config = response.content
```

JavaScript

```
// in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
// the agent is a sidecar process that handles retrieving configuration data from
 AppConfig
// for you in a way that implements best practices like configuration caching.

// for more information about the agent, see
// Simplified retrieval methods

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// the agent runs a local HTTP server that serves configuration data
// make a GET request to the agent's local server to retrieve the configuration data
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}`;
const response = await fetch(url);
const config = await response.text(); // (use `await response.json()` if your config
 is json)
```

Using AWS AppConfig Agent to read a specific feature flag

Each of the following samples includes comments about the actions performed by the code.

Java

```
public void retrieveSingleFlagFromAgent() throws Exception {
        /*
          You can retrieve a single flag's data from the agent by providing the
 "flag" query string parameter.
          Note: the configuration's type must be AWS.AppConfig.FeatureFlags
        */

        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyFlagsProfile?flag=myFlagKey");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("MyFlagName from agent: " + content);
    }
```

Python

```
import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'
flag_key = 'MyFlag'

# retrieve a single flag's data by providing the "flag" query string parameter
# note: the configuration's type must be AWS.AppConfig.FeatureFlags
```

```
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}?
flag={flag_key}")
config = response.content
```

JavaScript

```
const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";
const flag_name = "MyFlag";

// retrieve a single flag's data by providing the "flag" query string parameter
// note: the configuration's type must be AWS.AppConfig.FeatureFlags
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}?flag=${flag_name}`;
const response = await fetch(url);
const flag = await response.json(); // { "enabled": true/false }
```

Using the GetLatestConfig API action to read a freeform configuration profile

Each of the following samples includes comments about the actions performed by the code. The samples in this section call the following APIs:

- GetLatestConfiguration
- StartConfigurationSession

Java

```
public void retrieveConfigFromApi() {
        /*
        The example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
        For more information on these APIs, see AWS AppConfig Data         */
        AppConfigDataClient appConfigData = AppConfigDataClient.create();

        /*
        Start a new configuration session using the StartConfigurationSession API.
This operation does not return configuration data.
        Rather, it returns an initial configuration token that should be passed to
GetLatestConfiguration.
        IMPORTANT: This operation should only be performed once (per configuration),
prior to the first GetLatestConfiguration
        call you preform. Each GetLatestConfiguration will return a new
configuration token that you should then use in the
        next GetLatestConfiguration call.
        */
        StartConfigurationSessionResponse session =
            appConfigData.startConfigurationSession(req -> req
                .applicationIdentifier("MyDemoApp")
                .configurationProfileIdentifier("MyConfigProfile")
                .environmentIdentifier("Beta"));

        /*
        Retrieve configuration data using the GetLatestConfiguration API. The first
time you call this API your configuration
        data will be returned. You should cache that data (and the configuration
token) and update that cache asynchronously
        by regularly polling the GetLatestConfiguration API in a background thread.
If you already have the latest configuration
        data, subsequent GetLatestConfiguration calls will return an empty response.
If you then deploy updated configuration
        data the next time you call GetLatestConfiguration it will return that
updated data.

        You can also avoid all the complexity around writing this code yourself by
leveraging our agent instead.
        For more information about the agent, see Simplified retrieval methods
        */

        // The first getLatestConfiguration call uses the token from
StartConfigurationSession
        String configurationToken = session.initialConfigurationToken();
        GetLatestConfigurationResponse configuration =

appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        System.out.println("Configuration retrieved via API: " +
configuration.configuration().asUtf8String());
```

```
        // You'll want to hold on to the token in the getLatestConfiguration
 response because you'll need to use it
        // the next time you call
        configurationToken = configuration.nextPollConfigurationToken();
        configuration =

 appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        // Try creating a new deployment at this point to see how the output below
 changes.
        if (configuration.configuration().asByteArray().length != 0) {
            System.out.println("Configuration contents have changed
 since the last GetLatestConfiguration call, new contents = " +
 configuration.configuration().asUtf8String());
        } else {
            System.out.println("GetLatestConfiguration returned an empty response
 because we already have the latest configuration");
        }
    }
```

Python

```
# the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
#
# for more information on these APIs, see
# AWS AppConfig Data
#

import boto3

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

appconfigdata = boto3.client('appconfigdata')

# start a new configuration session.
# this operation does not return configuration data.
# rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
#
# note: this operation should only be performed once (per configuration).
#   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
scs = appconfigdata.start_configuration_session(
    ApplicationIdentifier=application_name,
    EnvironmentIdentifier=environment_name,
    ConfigurationProfileIdentifier=config_profile_name)
initial_token = scs['InitialConfigurationToken']

# retrieve configuration data from the session.
# this operation returns your configuration data.
# each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
#
# note: this operation does not always return configuration data after the first
 invocation.
#   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
#   therefore, you should cache the data returned by this call so that you can use
 it later.
glc = appconfigdata.get_latest_configuration(ConfigurationToken=initial_token)
config = glc['Configuration'].read()
```

JavaScript

```
// the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.

// for more information on these APIs, see
// AWS AppConfig Data

import {
  AppConfigDataClient,
  GetLatestConfigurationCommand,
  StartConfigurationSessionCommand,
} from "@aws-sdk/client-appconfigdata";

const appconfigdata = new AppConfigDataClient();

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// start a new configuration session.
// this operation does not return configuration data.
// rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
//
// note: this operation should only be performed once (per configuration).
//   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
const scs = await appconfigdata.send(
  new StartConfigurationSessionCommand({
    ApplicationIdentifier: application_name,
    EnvironmentIdentifier: environment_name,
    ConfigurationProfileIdentifier: config_profile_name,
  })
);
const { InitialConfigurationToken } = scs;

// retrieve configuration data from the session.
// this operation returns your configuration data.
// each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
//
// note: this operation does not always return configuration data after the first
 invocation.
//   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
//   therefore, you should cache the data returned by this call so that you can use
 it later.
const glc = await appconfigdata.send(
  new GetLatestConfigurationCommand({
    ConfigurationToken: InitialConfigurationToken,
  })
);
const config = glc.Configuration.transformToString();
```

Cleaning up your environment

If you ran one or more of the code samples in this section, we recommend you use one of the following samples to locate and delete the AWS AppConfig resources created by those code samples. The samples in this section call the following APIs:

- ListApplications
- DeleteApplication
- ListEnvironments

- DeleteEnvironments
- ListConfigurationProfiles
- DeleteConfigurationProfile
- ListHostedConfigurationVersions
- DeleteHostedConfigurationVersion

Java

```
/*
    This sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

    WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
    configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
    an application that you may need in the future.
    */

    public void cleanUpDemoResources() {
        AppConfigClient appconfig = AppConfigClient.create();

        // The name of the application to delete
        // IMPORTANT: verify this name corresponds to the application you wish to
 delete
        String applicationToDelete = "MyDemoApp";


 appconfig.listApplicationsPaginator(ListApplicationsRequest.builder().build()).items().forE
 -> {
            if (app.name().equals(applicationToDelete)) {
                System.out.println("Deleting App: " + app);
                appconfig.listConfigurationProfilesPaginator(req ->
 req.applicationId(app.id())).items().forEach(cp -> {
                    System.out.println("Deleting Profile: " + cp);
                    appconfig
                        .listHostedConfigurationVersionsPaginator(req -> req
                            .applicationId(app.id())
                            .configurationProfileId(cp.id()))
                        .items()
                        .forEach(hcv -> {
                            System.out.println("Deleting HCV: " + hcv);
                            appconfig.deleteHostedConfigurationVersion(req -> req
                                .applicationId(app.id())
                                .configurationProfileId(cp.id())
                                .versionNumber(hcv.versionNumber()));
                        });
                    appconfig.deleteConfigurationProfile(req -> req
                        .applicationId(app.id())
                        .configurationProfileId(cp.id()));
                });

                appconfig.listEnvironmentsPaginator(req-
>req.applicationId(app.id())).items().forEach(env -> {
                    System.out.println("Deleting Environment: " + env);
                    appconfig.deleteEnvironment(req-
>req.applicationId(app.id()).environmentId(env.id()));
                });

                appconfig.deleteApplication(req -> req.applicationId(app.id()));
            }
        });
    }
```

Python

```
# this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.
#
# WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
#   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
#   an application that you may need in the future.
#

import boto3

# the name of the application to delete
# IMPORTANT: verify this name corresponds to the application you wish to delete
application_name = 'MyDemoApp'

# create and iterate over a list paginator such that we end up with a list of pages,
 which are themselves lists of applications
```

```
# e.g. [ [{'Name':'MyApp1',...},{'Name':'MyApp2',...}], [{'Name':'MyApp3',...}] ]
list_of_app_lists = [page['Items'] for page in
 appconfig.get_paginator('list_applications').paginate()]
# retrieve the target application from the list of lists
application = [app for apps in list_of_app_lists for app in apps if app['Name'] ==
 application_name][0]
print(f"deleting application {application['Name']} (id={application['Id']})")

# delete all configuration profiles
list_of_config_lists = [page['Items'] for page in
 appconfig.get_paginator('list_configuration_profiles').paginate(ApplicationId=application['
for config_profile in [config for configs in list_of_config_lists for config in
 configs]:
    print(f"\tdeleting configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

    # delete all hosted configuration versions
    list_of_hcv_lists = [page['Items'] for page in
 appconfig.get_paginator('list_hosted_configuration_versions').paginate(ApplicationId=applic
 ConfigurationProfileId=config_profile['Id'])]
    for hcv in [hcv for hcvs in list_of_hcv_lists for hcv in hcvs]:

 appconfig.delete_hosted_configuration_version(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'], VersionNumber=hcv['VersionNumber'])
        print(f"\t\tdeleted hosted configuration version {hcv['VersionNumber']}")

    # delete the config profile itself
    appconfig.delete_configuration_profile(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'])
    print(f"\tdeleted configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

# delete all environments
list_of_env_lists = [page['Items'] for page in
 appconfig.get_paginator('list_environments').paginate(ApplicationId=application['Id'])]
for environment in [env for envs in list_of_env_lists for env in envs]:
    appconfig.delete_environment(ApplicationId=application['Id'],
 EnvironmentId=environment['Id'])
    print(f"\tdeleted environment {environment['Name']} (Id={environment['Id']})")

# delete the application itself
appconfig.delete_application(ApplicationId=application['Id'])
print(f"deleted application {application['Name']} (id={application['Id']})")
```

JavaScript

```
// this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

// WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
//   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
//   an application that you may need in the future.

import {
  AppConfigClient,
  paginateListApplications,
  DeleteApplicationCommand,
  paginateListConfigurationProfiles,
  DeleteConfigurationProfileCommand,
  paginateListHostedConfigurationVersions,
  DeleteHostedConfigurationVersionCommand,
  paginateListEnvironments,
  DeleteEnvironmentCommand,
} from "@aws-sdk/client-appconfig";

const client = new AppConfigClient();

// the name of the application to delete
// IMPORTANT: verify this name corresponds to the application you wish to delete
const application_name = "MyDemoApp";

// iterate over all applications, deleting ones that have the name defined above
for await (const app_page of paginateListApplications({ client }, {})) {
  for (const application of app_page.Items) {

    // skip applications that dont have the name thats set
    if (application.Name !== application_name) continue;

    console.log( `deleting application ${application.Name} (id=${application.Id})`);

    // delete all configuration profiles
    for await (const config_page of paginateListConfigurationProfiles({ client },
 { ApplicationId: application.Id })) {
      for (const config_profile of config_page.Items) {
        console.log(`\tdeleting configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`);
```

```
        // delete all hosted configuration versions
        for await (const hosted_page of
 paginateListHostedConfigurationVersions({ client },
          { ApplicationId: application.Id, ConfigurationProfileId:
 config_profile.Id }
        )) {
          for (const hosted_config_version of hosted_page.Items) {
            await client.send(
              new DeleteHostedConfigurationVersionCommand({
                ApplicationId: application.Id,
                ConfigurationProfileId: config_profile.Id,
                VersionNumber: hosted_config_version.VersionNumber,
              })
            );
            console.log(`\t\tdeleted hosted configuration version
 ${hosted_config_version.VersionNumber}`);
          }
        }

        // delete the config profile itself
        await client.send(
          new DeleteConfigurationProfileCommand({
            ApplicationId: application.Id,
            ConfigurationProfileId: config_profile.Id,
          })
        );
        console.log(`\tdeleted configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`)
      }

      // delete all environments
      for await (const env_page of paginateListEnvironments({ client },
 { ApplicationId: application.Id })) {
        for (const environment of env_page.Items) {
          await client.send(
            new DeleteEnvironmentCommand({
              ApplicationId: application.Id,
              EnvironmentId: environment.Id,
            })
          );
          console.log(`\tdeleted environment ${environment.Name} (Id=
${environment.Id})`)
        }
      }
    }

    // delete the application itself
    await client.send(
      new DeleteApplicationCommand({ ApplicationId: application.Id })
    );
    console.log(`deleted application ${application.Name} (id=${application.Id})`)
  }
}
```

Security in AWS AppConfig

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Systems Manager, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

AWS AppConfig is a capability of AWS Systems Manager. To understand how to apply the shared responsibility model when using AWS AppConfig, see Security in AWS Systems Manager. That section describes how to configure Systems Manager to meet the security and compliance objectives for AWS AppConfig.

Implement least privilege access

As a security best practice, grant the minimum required permissions that identities require to perform specific actions on specific resources under specific conditions. AWS AppConfig Agent offers two features that enable the agent to access the filesystem of an instance or container: *backup* and *write to disk*. If you enable these features, verify that only the AWS AppConfig Agent has permissions to write to the designated configuration files on the filesystem. Also verify that only the processes required to read from these configuration files have the ability to do so. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

For more information about implementing least privilege access, see SEC03-BP02 Grant least privilege access in the *AWS Well-Architected Tool User Guide*. For more information about the AWS AppConfig Agent features mentioned in this section, see Additional retrieval features.

Data encryption at rest for AWS AppConfig

AWS AppConfig provides encryption by default to protect customer data at rest using AWS owned keys.

AWS owned keys — AWS AppConfig uses these keys by default to automatically encrypt data deployed by the service and hosted in the AWS AppConfig data store. You can't view, manage, or use AWS owned keys, or audit their use. However, you don't have to take any action or change any programs to protect the keys that encrypt your data. For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.

While you can't disable this layer of encryption or select an alternate encryption type, you can specify a customer managed key to be used when you save configuration data hosted in the AWS AppConfig data store and when you deploy your configuration data.

Customer managed keys — AWS AppConfig supports the use of a symmetric customer managed key that you create, own, and manage to add a second layer of encryption over the existing AWS owned key. Because you have full control of this layer of encryption, you can perform such tasks as:

- Establishing and maintaining key policies and grants
- Establishing and maintaining IAM 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 key in the *AWS Key Management Service Developer Guide*.

AWS AppConfig supports customer managed keys

AWS AppConfig offers support for customer managed key encryption for configuration data. For configuration versions saved to the AWS AppConfig hosted data store, customers can set a `KmsKeyIdentifier` on the corresponding configuration profile. Each time a new version of configuration data is created using the `CreateHostedConfigurationVersion` API operation, AWS AppConfig generates an AWS KMS data key from the `KmsKeyIdentifier` to encrypt the data before storing it. When the data is later accessed, either during the `GetHostedConfigurationVersion` or `StartDeployment` API operations, AWS AppConfig decrypts the configuration data using information about the generated data key.

AWS AppConfig also offers support for customer managed key encryption for deployed configuration data. To encrypt configuration data, customers can provide a `KmsKeyIdentifier` to their deployment. AWS AppConfig generates the AWS KMS data key with this `KmsKeyIdentifier` to encrypt data on the `StartDeployment` API operation.

AWS AppConfig encryption access

When creating a customer managed key, use the following key policy to ensure that the key can be used.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "Allow use of the key",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::account_ID:role/role_name"
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
            ],
        "Resource": "*"
        }
 ]
```

To encrypt hosted configuration data with a customer managed key, the identity calling `CreateHostedConfigurationVersion` needs the following policy statement which can be assigned to a user, group, or role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

If you are using a Secrets Manager secret or any other configuration data encrypted with a customer managed key, your `retrievalRoleArn` will need `kms:Decrypt` to decrypt and retrieve the data.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:Region:account_ID:configuration source/object"
    }
  ]
}
```

When calling the AWS AppConfig [StartDeployment](https://docs.aws.amazon.com/appconfig/2019-10-09/APIReference/API_StartDeployment.html) API operation, the identity calling `StartDeployment` needs the following IAM policy which can be assigned to a user, group, or role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:Region:account_ID:key_ID"
    }
  ]
}
```

When calling the AWS AppConfig GetLatestConfiguration API operation, the identity calling `GetLatestConfiguration` needs the following policy which can be assigned to a user, group, or role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

Encryption context

An encryption context is an optional set of 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 include the same encryption context in the request.

AWS AppConfig encryption context: AWS AppConfig uses an encryption context in all AWS KMS cryptographic operations for encrypted hosted configuration data and deployments. The context contains a key corresponding to the type of data and a value that identifies the specific data item.

Monitoring your encryption keys for AWS

When you use an AWS KMS customer managed keys with AWS AppConfig, you can use AWS CloudTrail or Amazon CloudWatch Logs to track requests that AWS AppConfig sends to AWS KMS.

The following example is a CloudTrail event for `Decrypt` to monitor AWS KMS operations called by AWS AppConfig to access data encrypted by your customer managed key:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "appconfig.amazonaws.com"
    },
    "eventTime": "2023-01-03T02:22:28z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "Region",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:appconfig:deployment:arn":
 "arn:aws:appconfig:Region:account_ID:application/application_ID/
environment/environment_ID/deployment/deployment_ID"
        },
        "keyId": "arn:aws:kms:Region:account_ID:key/key_ID",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account_ID",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "account_ID",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```

Access AWS AppConfig using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and AWS AppConfig. You can access AWS AppConfig as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS AppConfig.

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 AppConfig.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for AWS AppConfig

Before you set up an interface endpoint for AWS AppConfig, review Considerations in the *AWS PrivateLink Guide*.

AWS AppConfig supports making calls to the `appconfig` and `appconfigdata` services through the interface endpoint.

Create an interface endpoint for AWS AppConfig

You can create an interface endpoint for AWS AppConfig 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 AppConfig using the following service names:

```
com.amazonaws.region.appconfig
```

```
com.amazonaws.region.appconfigdata
```

If you enable private DNS for the interface endpoint, you can make API requests to AWS AppConfig using its default Regional DNS name. For example, `appconfig.us-east-1.amazonaws.com` and `appconfigdata.us-east-1.amazonaws.com`.

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 AppConfig through the interface endpoint. To control the access allowed to AWS AppConfig 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*.

Example: VPC endpoint policy for AWS AppConfig actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed AWS AppConfig actions for all principals on all resources.

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "appconfig:CreateApplication",
            "appconfig:CreateEnvironment",
            "appconfig:CreateConfigurationProfile",
            "appconfig:StartDeployment",
            "appconfig:GetLatestConfiguration"
            "appconfig:StartConfigurationSession"
         ],
         "Resource":"*"
      }
   ]
}
```

Secrets Manager key rotation

This section describes important security information about AWS AppConfig integration with Secrets Manager. For information about Secrets Manager, see What is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*.

Setting up automatic rotation of Secrets Manager secrets deployed by AWS AppConfig

Rotation is the process of periodically updating a secret stored in Secrets Manager. When you rotate a secret, you update the credentials in both the secret and the database or service. You can configure automatic secrets rotation in Secrets Manager by using an AWS Lambda function to update the secret and the database. For more information, see Rotate AWS Secrets Manager secrets in the *AWS Secrets Manager User Guide*.

To enable key rotation of Secrets Manager secrets deployed by AWS AppConfig, update your rotation Lambda function and deploy the rotated secret.

Note

Deploy you AWS AppConfig configuration profile after your secret has been rotated and fully updated to the new version. You can determine if the secret rotated because the status of `VersionStage` changes from `AWSPENDING` to `AWSCURRENT`. Secret rotation completion occurs within the Secrets Manager Rotation Templates `finish_secret` function.

Here is an example function that starts an AWS AppConfig deployment after a secret is rotated.

```
import time
import boto3
client = boto3.client('appconfig')

def finish_secret(service_client, arn, new_version):
    """Finish the rotation by marking the pending secret as current
    This method finishes the secret rotation by staging the secret staged AWSPENDING
 with the AWSCURRENT stage.
    Args:
        service_client (client): The secrets manager service client
        arn (string): The secret ARN or other identifier
        new_version (string): The new version to be associated with the secret
    """
    # First describe the secret to get the current version
    metadata = service_client.describe_secret(SecretId=arn)
    current_version = None
    for version in metadata["VersionIdsToStages"]:
        if "AWSCURRENT" in metadata["VersionIdsToStages"][version]:
            if version == new_version:
                # The correct version is already marked as current, return
                logger.info("finishSecret: Version %s already marked as AWSCURRENT for
 %s" % (version, arn))
                return
            current_version = version
```

```
                break

        # Finalize by staging the secret version current
        service_client.update_secret_version_stage(SecretId=arn, VersionStage="AWSCURRENT", 
MoveToVersionId=new_version, RemoveFromVersionId=current_version)

        # Deploy rotated secret
        response = client.start_deployment(
                ApplicationId='TestApp',
                EnvironmentId='TestEnvironment',
                DeploymentStrategyId='TestStrategy',
                ConfigurationProfileId='ConfigurationProfileId',
                ConfigurationVersion=new_version,
                KmsKeyIdentifier=key,
                Description='Deploy secret rotated at ' + str(time.time())
            )

        logger.info("finishSecret: Successfully set AWSCURRENT stage to version %s for 
secret %s." % (new_version, arn))
```

Monitoring AWS AppConfig

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS AppConfig and your other AWS solutions. AWS provides the following monitoring tools to watch AWS AppConfig, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

Topics

Logging AWS AppConfig API calls using AWS CloudTrail

AWS AppConfig is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS AppConfig. CloudTrail captures all API calls for AWS AppConfig as events. The calls captured include calls from the AWS AppConfig console and code calls to the AWS AppConfig API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS AppConfig. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS AppConfig, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS AppConfig information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS AppConfig, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for AWS AppConfig, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All AWS AppConfig actions are logged by CloudTrail and are documented in the AWS AppConfig API Reference. For example, calls to the `CreateApplication`, `GetApplication` and `ListApplications` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

AWS AppConfig data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, retrieving the latest deployed configuration by calling GetLatestConfiguration). These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn’t log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the AWS AppConfig resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. The table in this section shows the resource types available for AWS AppConfig.

- To log data events using the CloudTrail console, create a trail or event data store to log data events, or update an existing trail or event data store to log data events.

  1. Choose **Data events** to log data events.
  2. From the **Data event type** list, choose **AWS AppConfig**.
  3. Choose the log selector template you want to use. You can log all data events for the resource type, log all `readOnly` events, log all `writeOnly` events, or create a custom log selector template to filter on the `readOnly`, `eventName`, and `resources.ARN` fields.
  4. For **Selector name**, enter **AppConfigDataEvents**. For information about enabling Amazon CloudWatch Logs for your data event trail, see Logging metrics for AWS AppConfig data plane calls.

- To log data events using the AWS CLI, configure the `--advanced-event-selectors` parameter to set the `eventCategory` field equal to `Data` and the `resources.type` field equal to the resource type value (see table). You can add conditions to filter on the values of the `readOnly`, `eventName`, and `resources.ARN` fields.

  - To configure a trail to log data events, run the put-event-selectors command. For more information, see Logging data events for trails with the AWS CLI.
  - To configure an event data store to log data events, run the create-event-data-store command to create a new event data store to log data events, or run the update-event-data-store command to update an existing event data store. For more information, see Logging data events for event data stores with the AWS CLI.

The following table lists the AWS AppConfig resource types. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail*
AWS AppConfig	`AWS::AppConfig::Configuration`	• GetLatestConfiguration • StartConfigurationSession

*You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector.

AWS AppConfig management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

AWS AppConfig logs all AWS AppConfig control plane operations as management events. For a list of the AWS AppConfig control plane operations that AWS AppConfig logs to CloudTrail, see the AWS AppConfig API Reference.

Understanding AWS AppConfig log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the StartConfigurationSession action.

```
{
      "eventVersion": "1.09",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "sessionIssuer": {},
          "attributes": {
            "creationDate": "2024-01-11T14:37:02Z",
            "mfaAuthenticated": "false"
          }
        }
      },
      "eventTime": "2024-01-11T14:45:15Z",
      "eventSource": "appconfig.amazonaws.com",
      "eventName": "StartConfigurationSession",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "203.0.113.0",
      "userAgent": "Boto3/1.34.11 md/Botocore#1.34.11 ua/2.0 os/macos#22.6.0
 md/arch#x86_64 lang/python#3.11.4 md/pyimpl#CPython cfg/retry-mode#legacy
 Botocore/1.34.11",
      "requestParameters": {
        "applicationIdentifier": "rrfexample",
        "environmentIdentifier": "mexampleqe0",
        "configurationProfileIdentifier": "3eexampleu1"
      },
      "responseElements": null,
      "requestID": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
      "eventID": "a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE",
      "readOnly": false,
      "resources": [
        {
          "accountId": "123456789012",
          "type": "AWS::AppConfig::Configuration",
          "ARN": "arn:aws:appconfig:us-east-1:123456789012:application/rrfexample/
environment/mexampleqe0/configuration/3eexampleu1"
        }
      ],
      "eventType": "AwsApiCall",
      "managementEvent": false,
      "recipientAccountId": "123456789012",
      "eventCategory": "Data",
      "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "appconfigdata.us-east-1.amazonaws.com"
      }
    }
```

Logging metrics for AWS AppConfig data plane calls

If you configured AWS CloudTrail to log AWS AppConfig data events, you can enable Amazon CloudWatch Logs to log metrics for calls to the AWS AppConfig data plane. You can then search and filter log data in CloudWatch Logs by creating one or more metric filters. Metric filters define the terms and patterns to look for in log data as it is sent to CloudWatch Logs. CloudWatch Logs uses metric filters to turn log data into numerical CloudWatch metrics. You can graph metrics or configure them with an alarm.

Before you begin

Enable logging of AWS AppConfig data events in AWS CloudTrail. The following procedure describes how to enable metric logging for an *existing AWS AppConfig trail* in CloudTrail. For information about how to enable CloudTrail logging for AWS AppConfig data plan calls, see AWS AppConfig data events in CloudTrail.

Use the following procedure to enable CloudWatch Logs to log metrics for calls to the AWS AppConfig data plane.

To enable CloudWatch Logs to log metrics for calls to the AWS AppConfig data plane

1. Open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. On the dashboard, choose your AWS AppConfig trail.
3. In the **CloudWatch Logs** section, choose **Edit**.
4. Choose **Enabled**.
5. For **Log group name**, either leave the default name or enter a name. Make a note of the name. You will choose the log group in the CloudWatch Logs console later.
6. For **Role name**, enter a name.
7. Choose **Save changes**.

Use the following procedure to create a metric and a metric filter for AWS AppConfig in CloudWatch Logs. The procedure describes how to create a metric filter for calls by `operation` and (optionally) calls by `operation` and `Amazon Resource Name (ARN)`.

To create a metric and a metric filter for AWS AppConfig in CloudWatch Logs

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Choose the checkbox beside the AWS AppConfig log group.
4. Choose **Actions**, and then choose **Create metric filter**.
5. For **Filter name**, enter a name.
6. For **Filter pattern**, enter the following:

   ```
   { $.eventSource = "appconfig.amazonaws.com" }
   ```
7. (Optional) In the **Test pattern** section, choose your log group from the **Select log data to test** list. If CloudTrail hasn't logged any calls, you can skip this step.
8. Choose **Next**.
9. For **Metric namespace**, enter **`AWS AppConfig`**.
10. For **Metric name**, enter **`Calls`**.
11. For **Metric value**, enter **1**.
12. Skip **Default value** and **Unit**.
13. For **Dimension name**, enter **`operation`**.
14. For **Dimension value**, enter **`$.eventName`**.

    (Optional) You can enter a second dimension that includes the Amazon Resource Name (ARN) making the call. To add a second dimension, for **Dimension name**, enter **`resource`**. For **Dimension value**, enter **`$.resources[0].ARN`**.

    Choose **Next**.
15. Review the details of the filter and **Create metric filter**.

(Optional) You can repeat this procedure to create a new metric filter for a specific error code like *AccessDenied*. If you do, enter the following details:

1. For **Filter name**, enter a name.

2. For **Filter pattern**, enter the following:

   ```
   { $.errorCode = "codename" }
   ```

   For example

   ```
   { $.errorCode = "AccessDenied" }
   ```

3. For **Metric namespace**, enter **`AWS AppConfig`**.
4. For **Metric name**, enter **`Errors`**.
5. For **Metric value**, enter **1**.
6. For **Default value**, enter a zero (0).
7. Skip **Unit**, **Dimensions**, and **Alarms**.

After CloudTrail logs API calls, you can view metrics in CloudWatch. For more information, see Viewing your metrics and logs in the console in the *Amazon CloudWatch User Guide*. For information about how to locate a metric you created, see Search for available metrics.

Note

If you set up the error metric with no dimension, as described here, you can view those metrics on the **Metrics with no dimension** page.

Creating an alarm for a CloudWatch metric

After you create metrics, you can create metric alarms in CloudWatch. For example, you can create an alarm for the *AWS AppConfig calls* metric you created in the previous procedure. Specifically, you can create an alarm for calls to the AWS AppConfig `StartConfigurationSession` API action that surpass a threshold. For information about how to create an alarm for a metric, see Create a CloudWatch alarm based on a static threshold in the *Amazon CloudWatch User Guide*. For information about default limits for calls to the AWS AppConfig data plane, see Data plane default limits in the *Amazon Web Services General Reference*.

AWS AppConfig User Guide document history

The following table describes the important changes to the documentation since the last release of AWS AppConfig.

Current API version: 2019-10-09

Change	Description	Date
AWS AppConfig custom extension samples	The Walkthrough: Creating custom AWS AppConfig extensions topic now includes links to the following sample extensions on GitHub: • Sample extension that prevents deployments with a `blocked day` moratorium calendar using Systems Manager Change Calendar • Sample extension that prevents secrets from leaking into configuration data using git-secrets • Sample extension that prevents personally identifiable information (PII) from leaking into configuration data using Amazon Comprehend	February 28, 2024
New topic: Logging AWS AppConfig API calls using AWS CloudTrail	AWS AppConfig is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS AppConfig. CloudTrai	January 18, 2024

	l captures all API calls for AWS AppConfig as events. This new topic provides AWS AppConfig-specific content rather than linking to the corresponding content in the *AWS Systems Manager User Guide*. For more information, see Logging AWS AppConfig API calls using AWS CloudTrail.	
AWS AppConfig now supports AWS PrivateLink	You can use AWS PrivateLink to create a private connection between your VPC and AWS AppConfig. You can access AWS AppConfig as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access AWS AppConfig. For more information, see Access AWS AppConfig using an interface endpoint (AWS PrivateLink).	December 6, 2023

Additional AWS AppConfig Agent retrieval features and a new local development mode	AWS AppConfig Agent offers the following additional features to help you retrieve configurations for your applications. Additional retrieval features • *Multi-account retrieval*: Use AWS AppConfig Agent from a primary or *retrieval* AWS account to retrieve configuration data from multiple vendor accounts. • *Write configuration copy to disk*: Use AWS AppConfig Agent to write configura tion data to disk. This feature enables customers with applications that read configuration data from disk to integrate with AWS AppConfig. **Note** *Write configuration to disk* is not designed as a configuration backup feature. AWS AppConfig Agent doesn't read from the configura tion files copied to disk. If you want to back up configura	December 1, 2023

tions to disk, see the `BACKUP_DIRECTORY` and `PRELOAD_BACKUP` environment variables for Using AWS AppConfig Agent with Amazon EC2 or Using AWS AppConfig Agent with Amazon ECS and Amazon EKS.

Local development mode

AWS AppConfig Agent supports a *local development mode*. If you enable local development mode, the agent reads configuration data from a specified directory on disk. It doesn't retrieve configuration data from AWS AppConfig. You can simulate configuration deployments by updating files in the specified directory. We recommend local development mode for the following use cases:

- Test different configuration versions before deploying them using AWS AppConfig.
- Test different configuration options for a new feature before commiting changes to your code repository.

	• Test different configuration scenarios to verify they work as expected.	
New code samples topic	Added a new code samples topic to this guide. The topic includes examples in Java, Python, and JavaScript for programmatically performing six common AWS AppConfig actions.	November 17, 2023
Revised table of contents to better reflect AWS AppConfig workflow	Content in this user guide is now grouped under the headings Creating, Deploying, Retrieving, and Extending workflows. This organization better reflects the workflow for using AWS AppConfig and aims to help make content more discoverable.	November 7, 2023
Payload reference added	The Creating a Lambda function for a custom AWS AppConfig extension topic now includes a request and response payload reference.	November 7, 2023
New AWS predefined deployment strategy	AWS AppConfig now offers and recommends the `AppConfig.Linear20PercentEvery6Minutes` predefined deployment strategy. For more information, see Predefined deployment strategies.	August 11, 2023

AWS AppConfig integration with Amazon EC2	You can integrate AWS AppConfig with applications running on your Amazon Elastic Compute Cloud (Amazon EC2) Linux instances by using AWS AppConfig Agent. The agent supports x86_64 and ARM64 architect ures for Amazon EC2. For more information, see AWS AppConfig integration with Amazon EC2.	July 20, 2023
AWS CloudFormation support for new AWS AppConfig resources and a feature flag example	AWS CloudFormation now supports the AWS::AppC onfig::Extension and AWS::AppConfig::ExtensionAs sociation resources to help you get started with AWS AppConfig extensions. The AWS::AppConfig::Co nfigurationProfile and AWS::AppConfig::HostedConfi gurationVersion resources now include an example for creating a feature flag configuration profile in the AWS AppConfig hosted configuration store.	April 12, 2023

AWS AppConfig integration with AWS Secrets Manager	AWS AppConfig integrates with AWS Secrets Manager. Secrets Manager helps you to securely encrypt, store, and retrieve credentials for your databases and other services. Instead of hardcoding credentials in your apps, you can make calls to Secrets Manager to retrieve your credentials whenever needed. Secrets Manager helps you protect access to your IT resources and data by enabling you to rotate and manage access to your secrets. When you create a freeform configuration profile, you can choose Secrets Manager as the source of your configuration data. You must onboard with Secrets Manager and create a secret before you create the configuration profile. For more information about Secrets Manager, see What is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*. For information about creating a configuration profile, see Creating a freeform configuration profile.	February 2, 2023

AWS AppConfig integration with Amazon ECS and Amazon EKS	You can integrate AWS AppConfig with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) by using the AWS AppConfig agent. The agent functions as a sidecar container running alongside your Amazon ECS and Amazon EKS container applications. The agent enhances containerized application processing and management in the following ways: • The agent calls AWS AppConfig on your behalf by using an AWS Identity and Access Management (IAM) role and managing a local cache of configuration data. By pulling configuration data from the local cache, your application requires fewer code updates to manage configuration data, retrieves configuration data in milliseconds, and isn't affected by network issues that can disrupt calls for such data. • The agent offers a native experience for retrievin	December 2, 2022

g and resolving AWS AppConfig feature flags.

- Out of the box, the agent provides best practices for caching strategies, polling intervals, and local configuration data availability while tracking the configuration tokens needed for subsequent service calls.
- While running in the background, the agent periodically polls the AWS AppConfig data plane for configuration data updates. Your containerized applicati on can retrieve the data by connecting to localhost on port 2772 (a customiza ble default port value) and calling HTTP GET to retrieve the data.
- The AWS AppConfig agent updates configuration data in your containers without having to restart or recycle those containers.

For more information, see AWS AppConfig integrati on with Amazon ECS and Amazon EKS.

New extension: AWS AppConfig extension for CloudWatch Evidently	You can use Amazon CloudWatch Evidently to safely validate new features by serving them to a specified percentage of your users while you roll out the feature. You can monitor the performance of the new feature to help you decide when to ramp up traffic to your users. This helps you reduce risk and identify unintended consequences before you fully launch the feature. You can also conduct A/B experiments to make feature design decisions based on evidence and data. The AWS AppConfig extension for CloudWatch Evidently allows your application to assign variations to user sessions locally instead of by calling the EvaluateFeature operation. A local session mitigates the latency and availability risks that come with an API call. For informati on about how to configure and use the extension, see Perform launches and A/B experiments with CloudWatc h Evidently in the *Amazon CloudWatch User Guide*.	September 13, 2022

Deprecation of the `GetConfiguration` API action	On Nov 18, 2021, AWS AppConfig released a new data plane service. This service replaces the previous process of retrievin g configuration data by using the `GetConfiguration` API action. The data plane service uses two new API actions, StartConfigurationSession and GetLatestConfiguration. The data plane service also uses new endpoints. For more information, see About the AWS AppConfig data plane service.	September 13, 2022
New version of the AWS AppConfig Agent Lambda extension	Version 2.0.122 of the AWS AppConfig Agent Lambda extension is now available . The new extension uses different Amazon Resource Names (ARNs). For more information, see AWS AppConfig Agent Lambda extension release notes.	August 23, 2022

Launch of AWS AppConfig extensions	An extension augments your ability to inject logic or behavior at different points during the AWS AppConfig workflow of creating or deploying a configuration. You can use AWS-authored extensions or create your own. For more informati on, see Working with AWS AppConfig extensions.	July 12, 2022
New version of the AWS AppConfig Agent Lambda extension	Version 2.0.58 of the AWS AppConfig Agent Lambda extension is now available. The new extension uses different Amazon Resource Names (ARNs). For more information, see Available versions of the AWS AppConfig Lambda extension.	May 3, 2022

AWS AppConfig integration with Atlassian Jira	Integrating with Atlassian Jira allows AWS AppConfig to create and update issues in the Atlassian console whenever you make changes to a feature flag in your AWS account for the specified AWS Region. Each Jira issue includes the flag name, application ID, configuration profile ID, and flag values. After you update, save, and deploy your flag changes, Jira updates the existing issues with the details of the change. For more informati on, see AWS AppConfig integration with Atlassian Jira.	April 7, 2022

General availability of feature flags and Lambda extension support for ARM64 (Graviton 2) processors	With AWS AppConfig feature flags, you can develop a new feature and deploy it to production while hiding the feature from users. You start by adding the flag to AWS AppConfig as configura tion data. Once the feature is ready to be released, you can update the flag configuration data without deploying any code. This feature improves the safety of your dev-ops environment because you don't need to deploy new code to release the feature. For more information, see Creating a feature flag configuration profile. General availability of feature flags in AWS AppConfig includes the following enhancements: • The console includes an option to designate a flag as a *short term* flag. You can filter and sort the list of flags on short-term flags. • For customers using feature flags in AWS Lambda, the new Lambda extension allows you to call individua l feature flags by using an HTTP endpoint. For more information, see see	March 15, 2022

	Retrieving one or more flags from a feature flag configuration. This update also provides support for AWS Lambda extensions developed for ARM64 (Graviton2) processor s. For more information, see see Available versions of the AWS AppConfig Lambda extension.	
The GetConfiguration API action is deprecated	The `GetConfiguration` API action is deprecated. Calls to receive configuration data should use the `StartConfigurationSession` and `GetLatestConfiguration` APIs instead. For more information about these APIs and how to use them, see Retrieving the configuration.	January 28, 2022
New region ARN for AWS AppConfig Lambda extension	AWS AppConfig Lambda extension is available in the new Asia Pacific (Osaka) region. The Amazon Resource Name (ARN) is required to create a Lambda in the region. For more information about the Asia Pacific (Osaka) region ARN, see Adding the AWS AppConfig Lambda extension.	March 4, 2021

AWS AppConfig Lambda extension	If you use AWS AppConfig to manage configurations for a Lambda function, then we recommend that you add the AWS AppConfig Lambda extension. This extension includes best practices that simplify using AWS AppConfig while reducing costs. Reduced costs result from fewer API calls to the AWS AppConfig service and, separately, reduced costs from shorter Lambda function processing times. For more information, see AWS AppConfig integrati on with Lambda extensions.	October 8, 2020
New section	Added a new section that provides instructions for setting up AWS AppConfig . For more information, see Setting up AWS AppConfig.	September 30, 2020
Added commandline procedures	Procedures in this user guide now include commandline steps for the AWS Command Line Interface (AWS CLI) and Tools for Windows PowerShel l. For more information, see Working with AWS AppConfig.	September 30, 2020

Launch of AWS AppConfig user guide	Use AWS AppConfig, a capability of AWS Systems Manager, to create, manage, and quickly deploy applicati on configurations. AWS AppConfig supports controlle d deployments to applicati ons of any size and includes built-in validation checks and monitoring. You can use AWS AppConfig with applications hosted on EC2 instances, AWS Lambda, containers, mobile applications, or IoT devices.	July 31, 2020