
AWS AppConfig

User Guide



AWS AppConfig: User Guide

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

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

Table of Contents

What Is AWS AppConfig?	1
Simplify tasks with AWS AppConfig	1
AWS AppConfig use cases	1
Benefits of using AWS AppConfig	1
Get started with AWS AppConfig	2
How AWS AppConfig works	2
Configure AWS AppConfig to work with your application	3
Enable your application code to check for and retrieve configuration data	4
Deploy a new or updated configuration	4
Pricing for AWS AppConfig	5
AWS AppConfig quotas	5
Getting started	6
Install or upgrade AWS command line tools	6
Configuring permissions for AWS AppConfig	7
(Optional) Configuring permissions for rollback based on CloudWatch alarms	9
Step 1: Create the permission policy for rollback based on CloudWatch alarms	9
Step 2: Create the IAM role for rollback based on CloudWatch alarms	10
Step 3: Add a trust relationship	10
Working with AWS AppConfig	11
Step 1: Creating an AWS AppConfig application	11
Creating an AWS AppConfig application (console)	11
Creating an AWS AppConfig application (commandline)	12
Step 2: Creating an environment	13
Creating an AWS AppConfig environment (console)	13
Creating an AWS AppConfig environment (commandline)	14
Step 3: Creating configuration profiles and feature flags	15
Example configurations	16
About the configuration profile IAM role	18
About configurations stored in Amazon S3	19
About validators	21
Creating a feature flag configuration profile	23
Creating a freeform configuration profile	32
Step 4: Creating a deployment strategy	38
Predefined deployment strategies	39
Create a deployment strategy	40
Step 5: Deploying a configuration	43
Deploy a configuration (console)	43
Deploy a configuration (commandline)	44
Step 6: Retrieving the configuration	46
Retrieving a configuration example	47
Services that integrate with AWS AppConfig	49
AWS Lambda extensions	49
How it works	49
Before you begin	51
Supported runtimes	51
Adding the AWS AppConfig Lambda extension	51
Configuring the AWS AppConfig Lambda extension	52
Retrieving one or more flags	54
Available versions of the AWS AppConfig Lambda extension	55
Using a container image to add the AWS AppConfig Lambda extension	63
Integrating with OpenAPI	65
Atlassian Jira	67
Configuring permissions	67
Configuring integration	69

Deleting integration	70
AWS CodePipeline	70
How integration works	71
Security	72
Monitoring	73
Document History	74
AWS glossary	77

What Is AWS AppConfig?

Use [AWS AppConfig](#), a capability of AWS Systems Manager, to create, manage, and quickly deploy application configurations. A *configuration* is a collection of settings that influence the behavior of your application. You can use AWS AppConfig with applications hosted on Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS Lambda, containers, mobile applications, or IoT devices. To view examples of the types of configurations you can manage by using AWS AppConfig, see [Example configurations](#) (p. 16).

Simplify tasks with AWS AppConfig

AWS AppConfig helps simplify the following tasks:

- **Configure**

Source your configurations from Amazon Simple Storage Service (Amazon S3), AWS AppConfig hosted configurations, Parameter Store, Systems Manager Document Store. Use AWS CodePipeline integration to source your configurations from Bitbucket Pipelines, GitHub, and AWS CodeCommit.

- **Validate**

While deploying application configurations, a simple typo could cause an unexpected outage. Prevent errors in production systems using AWS AppConfig validators. AWS AppConfig validators provide a syntactic check using a JSON schema or a semantic check using an AWS Lambda function to ensure that your configurations deploy as intended. Configuration deployments only proceed when the configuration data is valid.

- **Deploy and monitor**

Define deployment criteria and rate controls to determine how your targets retrieve the new configuration. Use AWS AppConfig deployment strategies to set deployment velocity, deployment time, and bake time. Monitor each deployment to proactively catch any errors using AWS AppConfig integration with Amazon CloudWatch. If AWS AppConfig encounters an error, the system rolls back the deployment to minimize impact on your application users.

AWS AppConfig use cases

AWS AppConfig can help you in the following use cases:

- **Application tuning** – Introduce changes carefully to your application that can be tested with production traffic.
- **Feature toggle** – Turn on new features that require a timely deployment, such as a product launch or announcement.
- **Allow list** – Allow premium subscribers to access paid content.
- **Operational issues** – Reduce stress on your application when a dependency or other external factor impacts the system.

Benefits of using AWS AppConfig

AWS AppConfig offers the following benefits for your organization:

- **Reduce errors in configuration changes**

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, you can call an AWS Lambda function that runs your configuration before you deploy it.

- **Deploy changes across a set of targets quickly**

AWS AppConfig simplifies the administration of applications at scale by deploying configuration changes from a central location. AWS AppConfig supports configurations stored in 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 AWS Identity and Access Management (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. You can use the rate controls of a deployment strategy to determine how fast you want your application targets to retrieve a configuration change.

Get started with AWS AppConfig

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

[AWS AppConfig Overview \(Video\)](#)

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:

- [Introduction to AWS AppConfig deployment](#) – Learn about safe deployment of application configuration settings with AWS AppConfig.
- [Automating feature release using AWS AppConfig deployment](#) – Learn how to automate feature release using AWS AppConfig integration with AWS CodePipeline.
- [Deploying application configuration to serverless workloads](#) – Learn how to use an AWS AppConfig Lambda extension to deploy application configuration to serverless workloads.

How AWS AppConfig works

At a high level, there are three processes for working with AWS AppConfig:

1. [Configure \(p. 3\)](#) AWS AppConfig to work with your application.
2. [Enable \(p. 4\)](#) your application code to periodically check for and retrieve configuration data from AWS AppConfig.
3. [Deploy \(p. 4\)](#) a new or updated configuration.

The following sections describe each step.

Configure AWS AppConfig to work with your application

To configure AWS AppConfig to work with your application, you set up three types of resources, which are described in the following table.

Resource	Details
Application	In AWS AppConfig, an application is simply an organizational construct like a folder. 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.
Environment	For each application, you define one or more environments. An environment is a logical deployment group of AWS AppConfig applications, such as applications in a Beta or Production environment. You can also define environments for application subcomponents such as the Web, Mobile, and Back-end components for your application. 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.
Configuration profile	<p>A <i>configuration profile</i> enables AWS AppConfig to access your configuration in its stored location. You can store configurations in the following formats and locations:</p> <ul style="list-style-type: none">• YAML, JSON, or text documents in the AWS AppConfig hosted configuration store• Objects in an Amazon S3 bucket• Documents in the Systems Manager document store• Parameters in Parameter Store <p>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</p>

Resource	Details
	detected, the deployment stops before making any changes to the targets of the configuration.

Enable your application code to check for and retrieve configuration data

Your application retrieves configuration data by first establishing a configuration session using the AWS AppConfig Data `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 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.

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`. This duration may vary over the course of the session, so it should be used instead of the value sent on the `StartConfigurationSession` call.
- The configuration: the latest data intended for the session. This may be empty if the client already has the latest version of the configuration.

For more information and to view example AWS CLI commands that show how to retrieve a configuration using the AWS AppConfig Data `StartConfigurationSession` and `GetLatestConfiguration` API actions, see [Receiving the configuration](#).

Deploy a new or updated configuration

AWS AppConfig enables you to deploy configurations in the manner that best suits the use case of your applications. You can deploy changes in seconds, or you can roll them out slowly to assess the impact of the changes. The AWS AppConfig resource that helps you control deployments is called a *deployment strategy*. A deployment strategy includes the following information:

- Total amount of time for a deployment to last. ([DeploymentDurationInMinutes](#)).
- The percentage of targets to retrieve a deployed configuration during each interval. ([GrowthFactor](#)).
- The amount of time AWS AppConfig monitors for alarms before considering the deployment to be complete and no longer eligible for automatic rollback. ([FinalBakeTimeInMinutes](#)).

You can use built-in deployment strategies that cover common scenarios, or you can create your own. After you create or choose a deployment strategy, you start the deployment. Starting the deployment calls the [StartDeployment](#) API action. The call includes the IDs of an application, environment,

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 rolls out.

Note

For information about AWS AppConfig language-specific SDKs, see [AWS AppConfig SDKs](#).

Pricing for AWS AppConfig

There is a charge to use AWS AppConfig. 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 \(p. 33\)](#).

Getting Started with AWS AppConfig

The following topics describe important tasks for getting started with AWS AppConfig.

Topics

- [Install or upgrade AWS command line tools](#) (p. 6)
- [Configuring permissions for AWS AppConfig](#) (p. 7)
- [\(Optional\) Configuring permissions for rollback based on CloudWatch alarms](#) (p. 9)

Install or upgrade AWS command line tools

This topic is for users who have *programmatic access* to use AWS AppConfig (or any other AWS service), and who want to run AWS CLI or AWS Tools for Windows PowerShell commands from their local machines.

Note

Programmatic access and *console access* are different permissions that can be granted to a user account by an AWS account administrator. A user can be granted one or both access types. For information, see [Create non-Admin IAM users and groups for Systems Manager](#) in the *AWS Systems Manager User Guide*.

For information about the AWS CLI, see the [AWS Command Line Interface User Guide](#). For information about the AWS Tools for Windows PowerShell, see the [AWS Tools for Windows PowerShell User Guide](#).

For information about all AWS AppConfig commands you can run using the AWS CLI, see [the AWS AppConfig section of the AWS CLI Command Reference](#). For information about all AWS AppConfig commands you can run using the AWS Tools for PowerShell, see [the AWS AppConfig section of the AWS Tools for PowerShell Cmdlet Reference](#).

AWS CLI

To install or upgrade and then configure the AWS CLI

1. Follow the instructions in [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide* to install or upgrade the AWS CLI on your local machine.

Tip

The AWS CLI is frequently updated with new functionality. Upgrade (reinstall) the CLI periodically to ensure that you have access to all the latest functionality.

2. To configure the AWS CLI, see [Configuring the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

In this step, you specify credentials that an AWS administrator in your organization has given you, in the following format:

```
AWS Access Key ID: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Important

When you configure the AWS CLI, you are prompted to specify an AWS Region. Choose one of the supported Regions listed for Systems Manager in the [AWS General Reference](#). If necessary, first verify with an administrator for your AWS account which Region you should choose.

For more information about access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*.

3. To verify the installation or upgrade, run the following command from the AWS CLI:

```
aws appconfig help
```

If successful, this command displays a list of available AWS AppConfig commands.

AWS Tools for PowerShell

To install or upgrade and then configure the AWS Tools for Windows PowerShell

1. Follow the instructions in [Setting up the AWS Tools for Windows PowerShell](#) or [AWS Tools for PowerShell Core](#) in the *AWS Tools for Windows PowerShell User Guide* to install or upgrade AWS Tools for PowerShell on your local machine.

Tip

AWS Tools for PowerShell is frequently updated with new functionality. Upgrade (reinstall) the AWS Tools for PowerShell periodically to ensure that you have access to all the latest functionality.

2. To configure AWS Tools for PowerShell, see [Using AWS Credentials](#) in the *AWS Tools for Windows PowerShell User Guide*.

In this step, you specify credentials that an AWS administrator in your organization has given you, using the following command.

```
Set-AWSCredential `
-AccessKey AKIAIOSFODNN7EXAMPLE `
-SecretKey wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY `
-StoreAs MyProfileName
```

Important

When you configure AWS Tools for PowerShell, you can run `Set-DefaultAWSRegion` to specify an AWS Region. Choose one of the supported Regions listed for AWS AppConfig in the [AWS General Reference](#). If necessary, first verify with an administrator for your AWS account which Region you should choose.

For more information about access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*.

3. To verify the installation or upgrade, run the following command from AWS Tools for PowerShell.

```
Get-AWSCmdletName -Service AppConfig
```

If successful, this command displays a list of available AWS AppConfig cmdlets.

Configuring permissions for AWS AppConfig

By default, only AWS account administrators have access to AWS AppConfig. You can grant AWS Identity and Access Management (IAM) users, groups, and roles access to AWS AppConfig by specifying resources, actions, and condition context keys in an IAM permission policy that you assign to the user, group, or role. For more information, see [Actions, resources, and condition keys for AWS AppConfig](#) in the *Service Authorization Reference*.

Important

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. To understand how to apply the shared responsibility model when using AWS AppConfig, see to [Security in AWS Systems Manager](#) in the *AWS Systems Manager User Guide*. AWS AppConfig is a capability of AWS Systems Manager.

We recommend that you create restrictive IAM permissions policies that grant users, groups, and roles the least privileges necessary to perform a desired action in AWS AppConfig.

For example, you can create a read-only IAM permissions policy that includes only the `Get` and `List` API actions used by AWS AppConfig, like the following.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetDocument",
        "ssm:ListDocuments",
        "appconfig:GetLatestConfiguration",
        "appconfig:StartConfigurationSession",
        "appconfig:ListApplications",
        "appconfig:GetApplication",
        "appconfig:ListEnvironments",
        "appconfig:GetEnvironment",
        "appconfig:ListConfigurationProfiles",
        "appconfig:GetConfigurationProfile",
        "appconfig:ListDeploymentStrategies",
        "appconfig:GetDeploymentStrategy",
        "appconfig:GetConfiguration",
        "appconfig:ListDeployments",
        "appconfig:GetDeployment"
      ],
      "Resource": "*"
    }
  ]
}
```

Important

Restrict access to the [StartDeployment](#) and [StopDeployment](#) API actions to trusted users who understand the responsibilities and consequences of deploying a new configuration to your targets.

For more information about creating and editing IAM policies, see [Creating IAM Policies](#) in the *IAM User Guide*. For information about how to assign this policy to an IAM group, see [Attaching a Policy to an IAM Group](#).

To configure an IAM user account with permission to use AWS AppConfig

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**.
3. In the list, choose a name.
4. Choose the **Permissions** tab.
5. On the right side of the page, under **Permission policies**, choose **Add inline policy**.
6. Choose the **JSON** tab.

7. Replace the default content with your custom permissions policy.
8. Choose **Review policy**.
9. On the **Review policy** page, for **Name**, enter a name for the inline policy. For example: AWSAppConfig-**<action>**-Access.
10. Choose **Create policy**.

(Optional) Configuring permissions for rollback based on CloudWatch alarms

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.

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 \(p. 9\)](#)
2. [Step 2: Create the IAM role for rollback based on CloudWatch alarms \(p. 10\)](#)
3. [Step 3: Add a trust relationship \(p. 10\)](#)

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**.

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

```
}  
  ]  
}
```

5. Enter tags for this role, and then choose **Next: Review**.
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": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "appconfig.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

3. Choose **Update Trust Policy**.

Working with AWS AppConfig

This section includes topics that describe how to use AWS AppConfig features to create and deploy a configuration to your hosts or targets. You can use AWS CloudFormation to create many of the AWS AppConfig artifacts described in this section. For more information, see [AWS AppConfig resource type reference](#) in the *AWS CloudFormation User Guide*.

Topics

- [Step 1: Creating an AWS AppConfig application](#) (p. 11)
- [Step 2: Creating an environment](#) (p. 13)
- [Step 3: Creating configuration profiles and feature flags](#) (p. 15)
- [Step 4: Creating a deployment strategy](#) (p. 38)
- [Step 5: Deploying a configuration](#) (p. 43)
- [Step 6: Retrieving the configuration](#) (p. 46)

Step 1: Creating an AWS AppConfig application

In AWS AppConfig, an application is simply an organizational construct like a folder. 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.

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*.

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 **AWS AppConfig**.
3. On the **Applications** tab, choose **Create application**.
4. For **Name**, enter a name for the application.
5. For **Description**, enter information about the application.
6. 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 [Step 2: Creating an environment](#) (p. 13). You can begin the procedure where it states, "On the **Environments** tab..."

Creating an AWS AppConfig application (commandline)

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. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.

For information, see [Install or upgrade AWS command line tools](#) (p. 6).

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
```

Step 2: Creating an environment

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. You can also define environments for application subcomponents such as the `Web`, `Mobile`, and `Back-end` components for your application. 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 [\(Optional\) Configuring permissions for rollback based on CloudWatch alarms](#) (p. 9).

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 **AWS AppConfig**.
3. On the **Applications** tab, choose the application you created in [Step 1: Creating an AWS AppConfig application](#) (p. 11) and then choose **View details**.
4. On the **Environments** tab, choose **Create environment**.
5. For **Name**, enter a name for the environment.
6. For **Description**, enter information about the environment.
7. In the **IAM role** list, choose the IAM role with permission to roll back a configuration when an alarm is triggered.
8. 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.
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 environment**.

AWS AppConfig creates the environment and then displays the **Environment details** page. Proceed to [Step 3: Creating configuration profiles and feature flags](#) (p. 15).

Creating an AWS AppConfig environment (commandline)

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. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.

For information, see [Install or upgrade AWS command line tools \(p. 6\)](#).

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
```

Step 3: Creating configuration profiles and feature flags

A *configuration* is a collection of settings that influence the behavior of your application. A *configuration profile* enables AWS AppConfig to access your configuration. Configuration profiles include the following information.

- The URI location where the configuration is stored.
- The AWS Identity and Access Management (IAM) role that provides access to the configuration.
- A validator for the configuration data. You can use either a JSON Schema or an AWS Lambda function to validate your configuration profile. A configuration profile can have a maximum of two validators.

AWS AppConfig supports the following types of configuration profiles.

- **Feature flag:** Use a feature flag configuration to turn on new features that require a timely deployment, such as a product launch or announcement.
- **Freeform:** Use a freeform configuration to carefully introduce changes to your application.

Contents

- [Example configurations \(p. 16\)](#)
- [About the configuration profile IAM role \(p. 18\)](#)
- [About configurations stored in Amazon S3 \(p. 19\)](#)
- [About validators \(p. 21\)](#)
- [Creating a feature flag configuration profile \(p. 23\)](#)
- [Creating a freeform configuration profile \(p. 32\)](#)

Example configurations

Use [AWS AppConfig](#), a capability of AWS Systems Manager, to create, manage, and quickly deploy application configurations. A *configuration* is a collection of settings that influence the behavior of your application. Here are some examples.

Feature flag configuration

The following feature flag configuration 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

The following operational configuration 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

The following access control list configuration 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
```

About the configuration profile IAM role

You can create the IAM role that provides access to the configuration data by using AWS AppConfig, as described in the following procedure. Or you can create the IAM role yourself and choose it from a list. 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 an SSM document

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

Configuration source is a Parameter Store parameter

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

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 only target an SSE-S3 encrypted object.
Storage classes	AWS AppConfig supports the following S3 storage classes: <code>STANDARD</code> , <code>INTELLIGENT_TIERING</code> , <code>REDUCED_REDUNDANCY</code> , <code>STANDARD_IA</code> , and <code>ONEZONE_IA</code> . The following classes are not supported: All S3 Glacier classes (<code>GLACIER</code> and <code>DEEP_ARCHIVE</code>).
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.

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

About validators

When you create a configuration and configuration profile, you can specify up to two validators. A validator ensures that your configuration data is syntactically and semantically correct. You can create validators in either JSON Schema or as an AWS Lambda function.

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.

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.

Note

Note the following information about JSON Schema validators:

- 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.

AWS Lambda 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 configuration profile Id of the configuration profile being validated",
  "ConfigurationVersion": "The configuration version of the configuration profile being validated",
  "Content": "Base64EncodedByteString",
  "Uri": "The uri of the configuration"
}
```

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 actions. 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.

Creating a feature flag configuration profile

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](#) (p. 34) section.

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`. For more information, see [GetLatestConfiguration](#) in the *AWS AppConfig API Reference*.

Topics

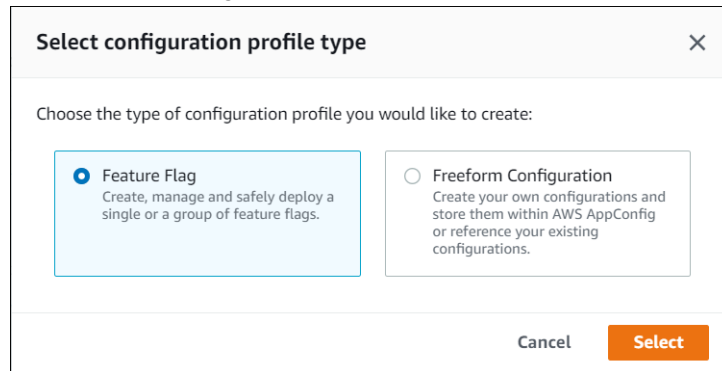
- [Creating a feature flag and a feature flag configuration profile \(console\)](#) (p. 23)
- [Creating a feature flag and a feature flag configuration profile \(commandline\)](#) (p. 25)
- [Type reference for AWS.AppConfig.FeatureFlags](#) (p. 27)

Creating a feature flag and a feature flag configuration profile (console)

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

To create a configuration profile

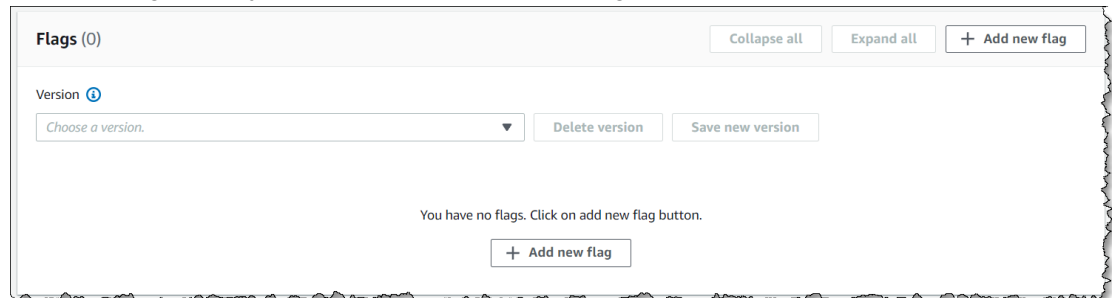
1. Open the AWS Systems Manager console at <https://console.aws.amazon.com/systems-manager/appconfig/>.
2. On the **Applications** tab, choose the application you created in [Create an AWS AppConfig configuration \(p. 11\)](#) and then choose the **Configuration profiles and feature flags** tab.
3. Choose **Create**.
4. Choose **Feature flag**.



The dialog box titled "Select configuration profile type" has a close button (X) in the top right corner. It contains the instruction "Choose the type of configuration profile you would like to create:". There are two radio button options: "Feature Flag" (selected) with the description "Create, manage and safely deploy a single or a group of feature flags." and "Freeform Configuration" with the description "Create your own configurations and store them within AWS AppConfig or reference your existing configurations." At the bottom, there are "Cancel" and "Select" buttons.

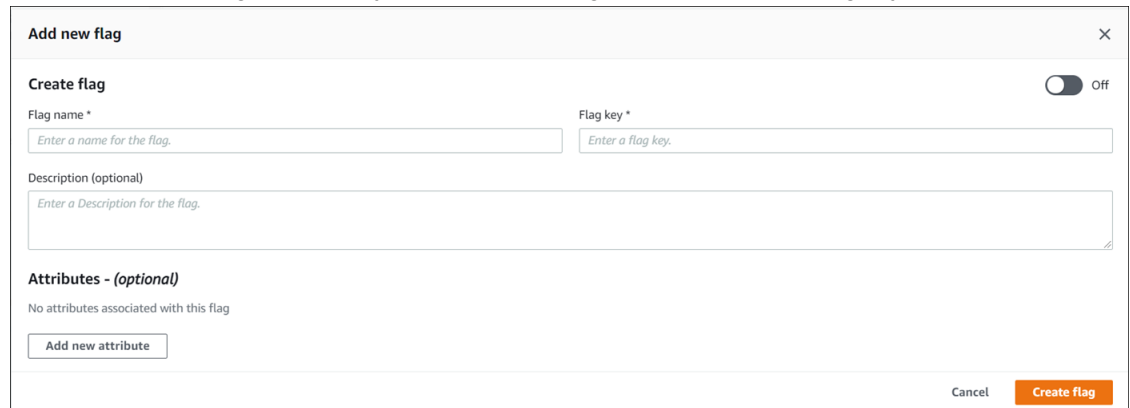
To create a feature flag

1. On the configuration you created, choose **Add new flag**.



The "Flags (0)" panel shows a header with "Collapse all", "Expand all", and "+ Add new flag" buttons. Below is a "Version" section with a dropdown menu labeled "Choose a version.", "Delete version", and "Save new version" buttons. A message states "You have no flags. Click on add new flag button." with a "+ Add new flag" button below it.

2. Provide a **Flag name** and (optional) **Description**. The **Flag key** auto populates by replacing spaces with underscores in the name you provided. You can edit the flag key if you want a different value or format. After the flag is created, you can edit the flag name, but not the flag key.



The "Add new flag" dialog box has a close button (X) in the top right corner. It contains a "Create flag" section with a toggle switch set to "Off". Below this are input fields for "Flag name *" (placeholder: "Enter a name for the flag."), "Flag key *" (placeholder: "Enter a flag key."), and "Description (optional)" (placeholder: "Enter a Description for the flag."). There is an "Attributes - (optional)" section with the text "No attributes associated with this flag" and an "Add new attribute" button. At the bottom, there are "Cancel" and "Create flag" buttons.

3. Specify whether the feature flag is **Enabled** or **Disabled** using the toggle button.

4. (Optional) Add **Attributes** and attribute **Constraints** to the feature flag. Attributes enable you to provide additional values within your flag. You can optionally validate attribute values against specified constraints. Constraints ensure that any unexpected values are not deployed to your application.

AWS AppConfig feature flags supports the following types of 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
	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

Note

Select **Required value** to specify whether the attribute value is required.

5. Choose **Save new version**.

Proceed to Step 4: Creating a deployment strategy.

Creating a feature flag and a feature flag configuration profile (commandline)

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. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.
For information, see [Install or upgrade AWS command line tools \(p. 6\)](#).
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-APPConfigurationProfile `\  
  -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](#) (p. 27).
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
```

The system returns information like the following.

Linux

```
{
  "ApplicationId"       : "The application ID",
  "ConfigurationProfileId" : "The configuration profile ID",
  "VersionNumber"       : "The configuration version number",
  "ContentType"         : "application/json"
}
```

Windows

```
{
  "ApplicationId"       : "The application ID",
  "ConfigurationProfileId" : "The configuration profile ID",
  "VersionNumber"       : "The configuration version number",
  "ContentType"         : "application/json"
}
```

PowerShell

```
ApplicationId       : The application ID
ConfigurationProfileId : The configuration profile ID
VersionNumber       : The configuration version number
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",
      "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`. 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
    }
  }
}
```

Creating a freeform configuration profile

A freeform configuration profile enables AWS AppConfig to access your configuration from a specified source location. You can store freeform configurations in the following formats and locations.

- YAML, JSON, or text documents in the AWS AppConfig hosted configuration store.
- Objects in an Amazon Simple Storage Service (Amazon S3) bucket.
- Documents in the Systems Manager document store.

- Any integration source action supported by AWS CodePipeline.

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 SSM parameters or in S3, you must create the parameter or object first and then add it to Parameter Store or S3. After you create the parameter or object, you can use the procedure in this topic to create the configuration profile. For information about creating a parameter in Parameter Store, see [Creating Systems Manager parameters](#) in the *AWS Systems Manager User Guide*.

Topics

- [About configuration store quotas and limitations \(p. 33\)](#)
- [About the AWS AppConfig hosted configuration store \(p. 34\)](#)
- [Creating a freeform configuration and configuration profile \(p. 34\)](#)

About configuration store quotas and limitations

AWS AppConfig-supported configuration store have the following quotas and limitations.

	AWS AppConfig hosted configuration store	S3	Parameter Store	Document store	AWS CodePipeline
Configuration size limit	1 MB	1 MB Enforced by AWS AppConfig, not S3	4 KB (free tier) / 8 KB (advanced parameters)	64 KB	1 MB Enforced by AWS AppConfig, not CodePipeline
Resource storage limit	1 GB	Unlimited	10,000 parameters (free tier) / 100,000 parameters (advanced parameters)	500 documents	Limited by the number of configuration profiles per application (100 profiles per application)
Server-side encryption	Yes	SSE-S3	No	No	Yes
AWS CloudFormation support	Yes	Not for creating or updating data	Yes	No	Yes
Validate create or update API actions	Not supported	Not supported	Regex supported	JSON Schema required for all put and update API actions	Not supported

	AWS AppConfig hosted configuration store	S3	Parameter Store	Document store	AWS CodePipeline
Pricing	Free	See Amazon S3 pricing	See AWS Systems 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 1 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.

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 \(p. 18\)](#).
- 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 \(p. 19\)](#).
- If you want to use validators, review the details and requirements for using them. For more information, see [About validators \(p. 21\)](#).

Topics

- [Creating an AWS AppConfig freeform configuration profile \(console\) \(p. 34\)](#)
- [Creating an AWS AppConfig freeform configuration profile \(commandline\) \(p. 37\)](#)

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 configuration profile

1. Open the AWS Systems Manager console at <https://console.aws.amazon.com/systems-manager/appconfig/>.
2. On the **Applications** tab, choose the application you created in [Create an AWS AppConfig configuration \(p. 11\)](#) and then choose the **Configuration profiles and feature flags** tab.
3. Choose **Create freeform configuration profile**.

Select configuration profile type ✕

Choose the type of configuration profile you would like to create:

☐ **Feature Flag**
Create, manage and safely deploy a single or a group of feature flags.

☒ **Freeform Configuration**
Create your own configurations and store them within AWS AppConfig or reference your existing configurations.

Cancel Select

4. For **Name**, enter a name for the configuration profile.
5. For **Description**, enter information about the configuration profile.
6. On the **Select configuration source** page, choose an option.
7.
 - If you selected **AWS AppConfig hosted configuration**, then choose either **YAML**, **JSON**, or **Text**, and enter your configuration in the field. Choose **Next** and go to Step 10 in this procedure.
 - If you selected **Amazon S3 object**, then enter the object URI. Choose **Next**.
 - If you selected **AWS Systems Manager parameter**, then choose the name of the parameter from the list. Choose **Next**.
 - If you selected **AWS CodePipeline**, then choose **Next** and go to Step 10 in this procedure.
 - If you selected **AWS Systems Manager document**, then complete the following steps.
 - a. In the **Document source** section, choose either **Saved document** or **New document**.
 - b. If you choose **Saved document**, then choose the SSM document from the list. If you choose **New document**, the **Details** and **Content** sections appear.
 - c. In the **Details** section, enter a name for the new application configuration.
 - d. For the **Application configuration schema** section, either choose the JSON schema using the list or choose **Create schema**. If you choose **Create schema**, Systems Manager opens the **Create schema** page. Enter the schema details in the **Content** section, and then choose **Create schema**.

Details

Name
MyAccessListConfiguration
Document names cannot contain special characters or spaces, and can be a maximum of 128 characters.

Application configuration schema
Choose a schema document for your application configuration document.
MyAccessListSchema or **Create schema**

Application configuration schema version
Choose a schema version.
1 or **Update schema**

- e. For **Application configuration schema version** either choose the version from the list or choose **Update schema** to edit the schema and create a new version.
- f. In the **Content** section, choose either **YAML** or **JSON** and then enter the configuration data in the field.

Content
Specify document content in YAML or JSON.

☐ YAML
Specify document content in JSON format.

☒ JSON
Specify document content in JSON format.

```
1 {  
2   "AccessList": [  
3     {  
4       "user_name": "Mateo_Jackson"  
5     },  
6     {  
7       "user_name": "Jane_Doe"  
8     }  
9   ]  
10 }
```

- g. Choose **Next**.
8. 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, to choose a role that already exists in IAM, choose **Existing service role**. Choose the role by using the **Role ARN** list.
9. 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.

10. In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
11. Choose **Create configuration profile**.

Important

If you created a configuration profile for AWS CodePipeline, then after you create a deployment strategy, as described in the next section, you must create a pipeline in CodePipeline that specifies AWS AppConfig as the *deploy provider*. 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 [Step 4: Creating a deployment strategy \(p. 38\)](#).

Creating an AWS AppConfig freeform configuration profile (commandline)

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

To create a configuration profile step by step

1. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.

For information, see [Install or upgrade AWS command line tools \(p. 6\)](#).

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_Location \
  --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_Location ^
  --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_the_spe
-
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"
```

Note

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 action. To view AWS CLI details and sample commands for this API action, see [create-hosted-configuration-version](#) in the *AWS CLI Command Reference*.

Step 4: Creating a deployment strategy

An AWS AppConfig deployment strategy defines the following important aspects of a configuration deployment.

Setting	Description
Deployment type	<p>Deployment type defines how the configuration deploys or <i>rolls out</i>. AWS AppConfig supports Linear and Exponential deployment types.</p> <ul style="list-style-type: none">Linear: For this type, AWS AppConfig processes the deployment by increments of the growth factor evenly distributed over the deployment. For example, a linear deployment that uses a step percentage of 20 initially makes the configuration available to 20 percent of the targets. After 1/5th of the deployment time has passed, the system updates the percentage to 40 percent. This continues until 100% of the targets are set to receive the deployed configuration.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 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:

Setting	Description
	<div>$2 * (2^0)$ $2 * (2^1)$ $2 * (2^2)$</div> <p>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.</p>
Step percentage (growth factor)	<p>This setting specifies the percentage of callers to target during each step of the deployment.</p> <p>Note In the SDK and the AWS AppConfig API Reference, <code>step percentage</code> is called <code>growth factor</code>.</p>
Deployment time	<p>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.</p>
Bake time	<p>This setting specifies the amount of time AWS AppConfig monitors for Amazon CloudWatch alarms before proceeding to the next step of a deployment or before considering the deployment to be complete. If an alarm is triggered during this time, AWS AppConfig rolls back the deployment. You must configure permissions for AWS AppConfig to roll back based on CloudWatch alarms. For more information, see (Optional) Configuring permissions for rollback based on CloudWatch alarms (p. 9).</p>

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.AllAtOnce	<p>Quick:</p> <p>This strategy deploys the configuration to all targets immediately. The system monitors for Amazon 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, AppConfig rolls back the deployment.</p>
AppConfig.Linear50PercentEvery30Seconds	<p>Testing/Demonstration:</p>

Deployment strategy	Description
	<p>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, AppConfig rolls back the deployment.</p> <p>We recommend using this strategy only for testing or demonstration purposes because it has a short duration and bake time.</p>
AppConfig.Canary10Percent20Minutes	<p>AWS Recommended:</p> <p>This strategy processes the deployment exponentially using a 10% growth factor over 20 minutes. The system monitors for Amazon 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, AppConfig rolls back the deployment.</p> <p>We recommend using this strategy for production deployments because it aligns with AWS best practices for configuration deployments.</p>

Create a deployment strategy

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 a 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 **AWS AppConfig**.
3. Choose the **Deployment Strategies** tab, and then choose **Create deployment strategy**.
4. For **Name**, enter a name for the deployment strategy.
5. For **Description**, enter information about the deployment strategy.
6. For **Deployment type**, choose a type.
7. For **Step percentage**, choose the percentage of callers to target during each step of the deployment.
8. For **Deployment time**, enter the total duration for the deployment in minutes or hours.
9. 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.
10. In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.

11. 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 [Step 5: Deploying a configuration](#) (p. 43). However, you must configure a client to receive application configuration updates as described in [Step 6: Retrieving the configuration](#) (p. 46). 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 [Step 5: Deploying a configuration](#) (p. 43).

Creating an AWS AppConfig deployment strategy (commandline)

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

To create a deployment strategy step by step

1. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.

For information, see [Install or upgrade AWS command line tools](#) (p. 6).

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_interval
\
  --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_interval
^
  --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_interval `
`
--
GrowthType The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time `
`
--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
```

Step 5: Deploying a configuration

Starting a deployment in AWS AppConfig calls the [StartDeployment](#) API action. 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.

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 **AWS AppConfig**.
3. On the **Applications** tab, choose an application, and then choose **View details**.
4. On the **Environments** tab, choose an environment, and then choose **View details**.
5. Choose **Start deployment**.
6. For **Configuration**, choose a configuration from the list.
7. Depending on the source of your configuration, use the **Document version** or **Parameter version** list to choose the version you want to deploy.
8. For **Deployment strategy**, choose a strategy from the list.
9. For **Deployment description**, enter a description.
10. In the **Tags** section, 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 a AWS AppConfig configuration.

To deploy a configuration step by step

1. Install and configure the AWS CLI or the AWS Tools for PowerShell, if you have not already.

For information, see [Install or upgrade AWS command line tools \(p. 6\)](#).

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
```

Step 6: Retrieving the configuration

Your application retrieves configuration data by first establishing 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 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](#).

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`. This duration may vary over the course of the session, so it should be used instead of the value sent on the `StartConfigurationSession` call.
- 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.
- The `InitialConfigurationToken` and `NextPollConfigurationToken` expire after 24 hours. If a `GetLatestConfiguration` call uses an expired token, the system returns `BadRequestException`.

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 actions. 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 action:

- 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.

Services that integrate with AWS AppConfig

AWS AppConfig integrates with other AWS services to help you manage your application configurations quickly and securely. The following information can help you configure AWS AppConfig to integrate with the products and services you use.

Contents

- [AWS AppConfig integration with Lambda extensions \(p. 49\)](#)
- [AWS AppConfig integration with Atlassian Jira \(p. 67\)](#)
- [AWS AppConfig integration with CodePipeline \(p. 70\)](#)

AWS AppConfig integration with Lambda extensions

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 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.

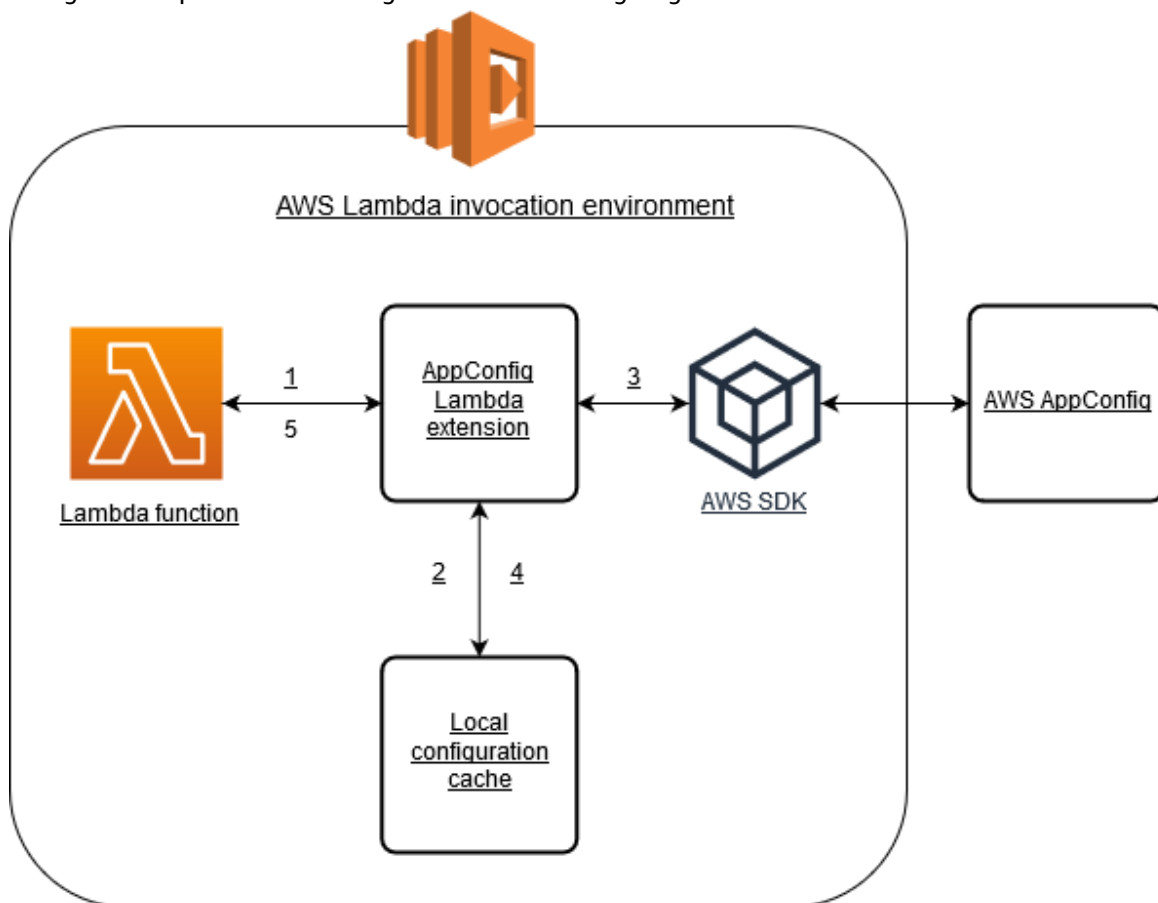
This topic includes information about the AWS AppConfig Lambda extension and the procedure for how to configure the extension to work with your Lambda function.

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 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.



1. You configure the AWS AppConfig 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 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 Lambda extension, do the following:

- Organize the configurations in your Lambda function so that you can externalize them into AWS AppConfig.
- Configure AWS AppConfig to manage your configuration updates. For more information, see [Working with AWS AppConfig \(p. 11\)](#).
- 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*.

Supported runtimes

The AWS AppConfig Lambda extension supports the following runtimes:

- Python 3.7, 3.8, 3.9
- Node.js 12.x, 14.x
- Ruby 2.7
- Java 8, 11 (Amazon Corretto)
- .NET Core 3.1
- Custom runtimes (Amazon Linux and Amazon Linux 2)

For more information about these runtimes, including the corresponding SDKs, operating systems, and architectures, see [Lambda runtimes](#) in the *AWS Lambda Developer Guide*.

Adding the AWS AppConfig Lambda extension

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

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

To use the AWS AppConfig 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 \(p. 55\)](#)
- [ARM64 platform \(p. 57\)](#)

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 Lambda extension as a container image, see [Using a container image to add the AWS AppConfig Lambda extension \(p. 63\)](#).

Configuring the AWS AppConfig 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 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 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 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 Lambda extension and the Lambda function by using Amazon CloudWatch Logs.

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_MILLISECONDS	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_HTTP_PORT	This environment variable specifies the port on which the local HTTP server that hosts the extension runs.	2772
AWS_APPCONFIG_EXTENSION_PRECONFIGURED_DATA	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
AWS_APPCONFIG_EXTENSION_MAX_CONNECTIONS	This environment variable configures the maximum number of connections the extension uses to retrieve configurations from AWS AppConfig.	3
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	info

Environment variable	Details	Default value
	detailed information, including timing information, about the extension.	
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
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_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

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 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 a 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
```

Available versions of the AWS AppConfig Lambda extension

This topic lists AWS AppConfig Lambda extension versions according to processor. The AWS AppConfig Lambda extension supports Lambda functions developed for the x86-64 and ARM64 (Graviton2) platforms. Each table includes the Amazon Resource Name (ARN) to use in each AWS Region.

Topics

- [x86-64 platform \(p. 55\)](#)
- [ARM64 platform \(p. 57\)](#)
- [Older extension versions \(p. 58\)](#)

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 2.0.58

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
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

Region	ARN
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
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

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 2.0.58

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

Region	ARN
Europe (London)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2
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

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 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 Lambda extension developed for the x86-64 platform.

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
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

Region	ARN
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
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

Region	ARN
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
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
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

Region	ARN
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 Lambda extension developed for the ARM64 platform.

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
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

Region	ARN
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 Lambda extension

You can package your AWS AppConfig 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 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 lambda get-layer-version-by-arn \  
  --region us-east-1 \  
  --arn arn:aws:lambda::layer:AWS-AppConfig-Extension:00
```

Following is the response.

```
{  
  "LayerVersionArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension:00",  
  "Description": "AWS AppConfig extension: Use dynamic configurations deployed via AWS  
AppConfig for your AWS Lambda functions",  
  "CreateDate": "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 arn:aws:lambda::layer:AWS-AppConfig-Extension:00 \  
| 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 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 arn:aws:lambda::layer:AWS-AppConfig-Extension:00 \  
| 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 hard-coded 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 Lambda extension API
  description: An API model for AppConfig 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: InternalServerErrorException
          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
```

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.

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:

- [Configure permissions](#)
- [Configure integration](#)

Configuring permissions for AWS AppConfig Jira integration

When you configure AWS AppConfig integration with Jira, you specify credentials for an AWS Identity and Access Management (IAM) 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 IAM user account requires a permissions policy that includes the following actions:

- `appconfig:AssociateExtension`
- `appconfig:GetConfigurationProfile`
- `appconfig:ListApplications`
- `appconfig:ListConfigurationProfiles`
- `appconfig:UpdateConfigurationProfile`

- `sts:GetCallerIdentity`

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

Tasks

- [Task 1: Create an IAM permissions policy for AWS AppConfig and Jira integration \(p. 68\)](#)
- [Task 2: Create an IAM user for AWS AppConfig and Jira integration \(p. 69\)](#)

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

Use the following procedure to create an IAM permissions 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 permissions 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:AssociateExtension",
        "appconfig:GetConfigurationProfile",
        "appconfig:UpdateConfigurationProfile"
      ],
      "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:*"
      ]
    }
  ]
}
```



```

        "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 an IAM user for AWS AppConfig and Jira integration

Use the following procedure to create an IAM 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 an IAM 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 permissions policy for AWS AppConfig and Jira integration \(p. 68\)](#), 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 an IAM user for AWS AppConfig and Jira integration \(p. 69\)](#). 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**.

AWS AppConfig 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 an application](#)
- [Create an environment](#)
- [Create an AWS CodePipeline 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*.

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, refer to [Security in AWS Systems Manager](#). It explains how to configure Systems Manager to meet the security and compliance objectives for AWS AppConfig.

Monitoring AWS AppConfig

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS solutions. AWS AppConfig is a capability of AWS Systems Manager. For information about monitoring Systems Manager capabilities, see [Monitoring AWS Systems Manager](#) in the *AWS Systems Manager User Guide*.

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

update-history-change	update-history-description	update-history-date
New version of the AWS AppConfig Lambda Extension (p. 74)	Version 2.0.58 of the AWS AppConfig 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 (p. 74)	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 information, see AWS AppConfig integration with Atlassian Jira .	April 7, 2022
General availability of feature flags and Lambda extension support for ARM64 (Graviton2) processors (p. 74)	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 configuration 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	March 15, 2022

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 individual feature flags by using an HTTP endpoint. For more information, see [Retrieving one or more flags from a feature flag configuration](#).

This update also provides support for AWS Lambda extensions developed for ARM64 (Graviton2) processors. For more information, see [Available versions of the AWS AppConfig Lambda extension](#).

The [GetConfiguration API action](#) is deprecated (p. 74)

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 (p. 74)

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 (p. 74)	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 integration with Lambda extensions .	October 8, 2020
New section (p. 74)	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 (p. 74)	Procedures in this user guide now include commandline steps for the AWS Command Line Interface (AWS CLI) and Tools for Windows PowerShell. For more information, see Working with AWS AppConfig .	September 30, 2020
Launch of AWS AppConfig user guide (p. 74)	Use AWS AppConfig, a capability of AWS Systems Manager, to create, manage, and quickly deploy application configurations. AWS AppConfig supports controlled deployments to applications 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

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.