AWS AppConfig

User Guide

AWS AppConfig: User Guide

Table of Contents

What Is AWS AppConfig?

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.

To prevent errors when deploying application configurations, especially for production systems where a simple typo could cause an unexpected outage, AWS AppConfig includes validators. A validator provides a syntactic or semantic check to ensure that the configuration you want to deploy works as intended. To validate your application configuration data, you provide a schema or Lambda function that runs against the configuration. The configuration deployment or update can only proceed when the configuration data is valid.

During a configuration deployment, AWS AppConfig monitors the application to ensure that the deployment is successful. If the system encounters an error, AWS AppConfig rolls back the change to minimize impact for your application users. You can configure a deployment strategy for each application or environment that includes deployment criteria, including velocity, bake time, and alarms to monitor. Similar to error monitoring, if a deployment triggers an alarm, AWS AppConfig automatically rolls back to the previous version.

AWS AppConfig supports multiple use cases. Here are some examples.

- **Application tuning**: Use AWS AppConfig to carefully introduce changes to your application that can only be tested with production traffic.
- **Feature toggle**: Use AWS AppConfig to turn on new features that require a timely deployment, such as a product launch or announcement.
- **Allow list**: Use AWS AppConfig to allow premium subscribers to access paid content.
- **Operational issues**: Use AWS AppConfig to reduce stress on your application when a dependency or other external factor impacts the system.

How can AWS AppConfig benefit my organization?

AWS AppConfig offers the following benefits.

- **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.
- **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 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.
- **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, 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 receive a configuration change.

What types of targets are supported?

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.

Is there a charge to use AWS AppConfig?

Yes. For more information, see AWS Systems Manager Pricing.

Do I have to change my application to work with AWS AppConfig?

Yes. You must configure your application to poll for new configuration updates by using the GetConfiguration API action. When a new or updated configuration is ready, AWS AppConfig deploys the configuration file to each target in your deployment strategy.

How does AWS AppConfig work?

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

1. Configure AWS AppConfig to work with your application.
2. Enable your application code to periodically check for and receive configuration data from AWS AppConfig.
3. Deploy a new or updated configuration.

Configure AWS AppConfig to work with your application

To configure AWS AppConfig to work with your application, you set up three types of resources.

Resource	Details
Application	An application in AWS AppConfig is a logical unit of code that provides capabilities for your customers. For example, an application can be a microservice that runs on EC2 instances, a mobile application installed by your users, a serverless application using Amazon API Gateway and AWS Lambda, or any system you run on behalf of others.
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	A *configuration profile* enables AWS AppConfig to access your configuration in its stored location. You can store 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 • Parameters in Parameter Store A configuration profile can also include optional validators to ensure your configuration data is syntactically and semantically correct. AWS AppConfig performs a check using the validators when you start a deployment. If any errors are detected, the deployment stops before making any changes to the targets of the configuration.

Enable your application code to periodically check for and receive configuration data from AWS AppConfig

Using the GetConfiguration API action, AWS AppConfig offers dynamic configuration updates in a controlled manner. You must configure your application code to call this API action on a periodic basis. When called, your code sends the following information:

- The IDs of an AWS AppConfig application, environment, and configuration profile.

- A unique application instance identifier called a client ID.
- The last configuration version known by your application code.

When a new configuration is deployed (which means there is a new version of the configuration), AWS AppConfig responds to the GetConfiguration request and returns the new configuration data.

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 receive 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.

What are the Service Quotas of AWS AppConfig?

Resource	Quota
AWS AppConfig applications	100
Deployment strategies	20
Environments per AWS AppConfig application	20
Configuration profiles per AWS AppConfig application	100

Note
For information about quotas for services that store AWS AppConfig configurations, see About configuration store quotas and limitations (p. 16).

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

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 Systems Manager service

endpoints in the *Amazon Web Services 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 Systems Manager service endpoints in the *Amazon Web Services 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

AWS AppConfig uses the following API actions.

AWS AppConfig resource	API actions
AWS AppConfig applications	CreateApplication, UpdateApplication, DeleteApplication, GetApplication, ListApplications For more information, see Step 1: Creating an AWS AppConfig application (p. 11)
Environments	CreateEnvironment, UpdateEnvironment, DeleteEnvironment, GetEnvironment, ListEnvironments For more information, see Step 2: Creating an environment (p. 13)
Configurations	GetConfiguration For more information, see Step 3: Creating a configuration and a configuration profile (p. 15)
Configuration profiles	CreateConfigurationProfile, UpdateConfigurationProfile, DeleteConfigurationProfile, GetConfigurationProfile, ValidateConfiguration, ListConfigurationProfiles, ValidateConfiguration For more information, see Step 3: Creating a configuration and a configuration profile (p. 15)
Deployment strategies	CreateDeploymentStrategy, UpdateDeploymentStrategy, DeleteDeploymentStrategy, GetDeploymentStrategy, ListDeploymentStrategies, CreateDocument, UpdateDocument, DeleteDocument, GetDocument, ListDocuments For more information, see Step 4: Creating a deployment strategy (p. 28)
Deployments	StartDeployment, StopDeployment, ListDeployments For more information, see Step 5: Deploying a configuration (p. 33)

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:ListApplications",
        "appconfig:GetApplication",
        "appconfig:ListEnvironments",
        "appconfig:GetEnvironment",
        "appconfig:ListConfigurationProfiles",
        "appconfig:GetConfigurationProfile",
        "appconfig:ListDeploymentStrategies",
        "appconfig:GetDeploymentStrategy",
        "appconfig:GetConfiguration",
        "appconfig:ListDeployments"

      ],
      "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: **`AWS AppConfig-<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 even if those alarms weren't created in the current AWS account.

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. 9)
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 **Review**.

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

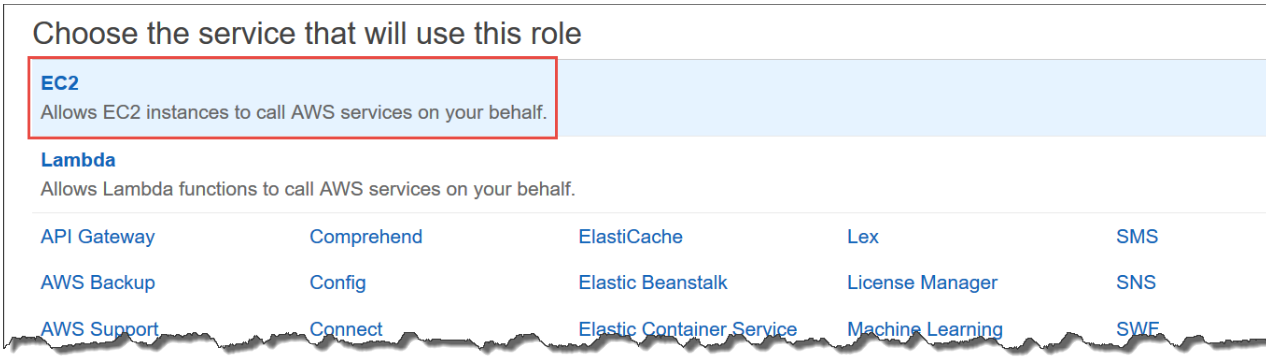
5. On the **Review** page, enter **`SSMCloudWatchAlarmDiscoveryRole`** in the **Role name** field.
6. Choose **Create policy**. The system returns you to the **Policies** page.

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**, and then choose **Next: Permissions**.



5. On the **Attached permissions policy** page, search for **SSMCloudWatchAlarmDiscoveryRole**.
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 a name 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.

Topics

Step 1: Creating an AWS AppConfig application

An application in AWS AppConfig is a logical unit of code that provides capabilities for your customers. For example, an application can be a microservice that runs on EC2 instances, a mobile application installed by your users, a serverless application using Amazon API Gateway and AWS Lambda, or any system you run on behalf of others.

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. 5).
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. 8).

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 **Monitors** section, choose **Enable rollback on CloudWatch Alarms** if you want AWS AppConfig to roll back a configuration when an alarm is triggered.
8. In the **IAM role** list, choose the IAM role with permission to roll back a configuration when an alarm is triggered.
9. In the **CloudWatch alarms** list, choose one or more alarms to monitor.
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 environment**.

AWS AppConfig creates the environment and then displays the **Environment details** page. Proceed to Step 3: Creating a configuration and a configuration profile (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. 5).
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 AWS 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 AWS 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
 AWS AppConfig to monitor AlarmArn}
Name              : Name of the environment
Response Metadata : Runtime Metadata
State             : State of the environment
```

Step 3: Creating a configuration and a configuration profile

A *configuration* is a collection of settings that influence the behavior of your application. For example, you can create and deploy configurations that carefully introduce changes to your application or turn on new features that require a timely deployment, such as a product launch or announcement. Here's a very simple example of an access list configuration.

```
{
    "AccessList": [
        {
            "user_name": "Mateo_Jackson"
        },
        {
            "user_name": "Jane_Doe"
        }
    ]
}
```

A *configuration profile* enables AWS AppConfig to access your configuration from a source location. You can store 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
- Parameters in Parameter Store
- Any integration source action supported by AWS CodePipeline

A configuration profile includes 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.

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

For 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*.

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	64 KB	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	No	No	No	Yes
AWS CloudFormation support	Yes	Not for creating or updating data	Yes	No	Yes
Validate create or	Not supported	Not supported	Regex supported	JSON Schema required for all	Not supported

	AWS AppConfig hosted configuration store	S3	Parameter Store	Document store	AWS CodePipeline
update API actions				put and update API actions	
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 64 KB 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 configuration and a configuration profile

Before you begin

Read the following related content before you complete the procedure in this section.

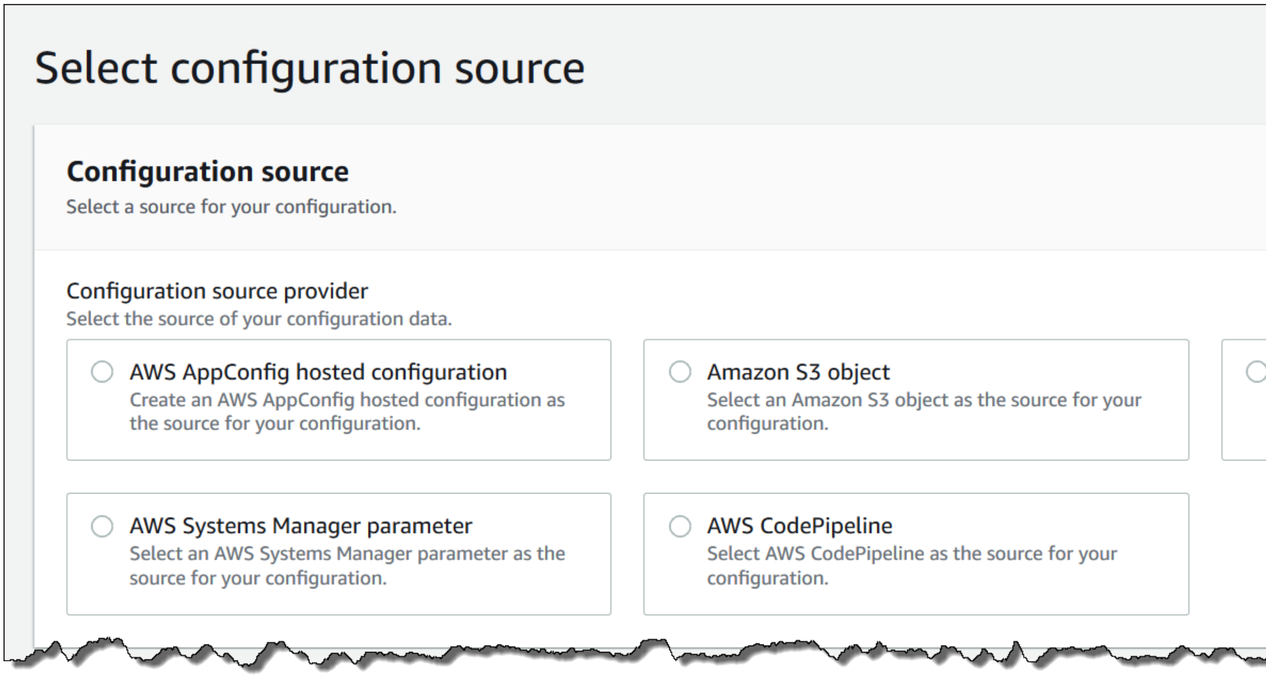
- The following procedure requires you to specify an IAM service role so that AWS AppConfig can access your configuration data in the configuration store you choose. This role is not required if you use the AWS AppConfig hosted configuration store. If you choose S3, Parameter Store, or the Systems Manager 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. 22).
- 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. 23).
- If you want to use validators, review the details and requirements for using them. For more information, see About validators (p. 27).

Creating an AWS AppConfig configuration profile (console)

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

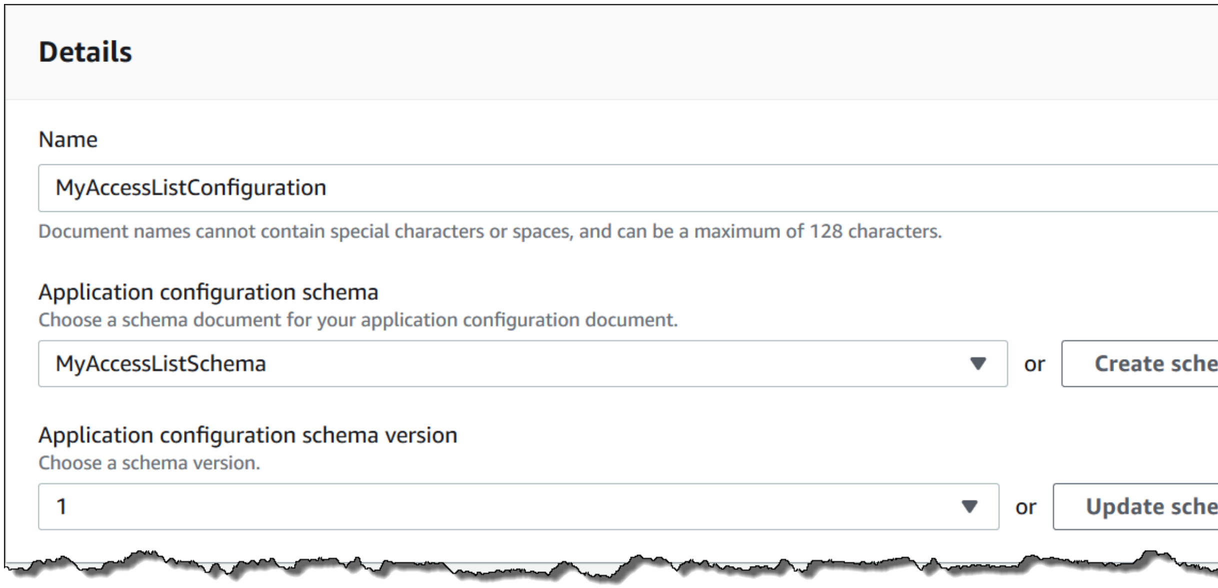
To create a configuration profile

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** tab.
3. Choose **Create configuration profile**.
4. For **Name**, enter a name for the configuration profile.
5. For **Description**, enter information about the configuration profile.
6. In the **Tags** section, enter a key and an optional value. You can specify a maximum of 50 tags for a resource.
7. On the **Select configuration source** page, choose an option.



8.
   - 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**.



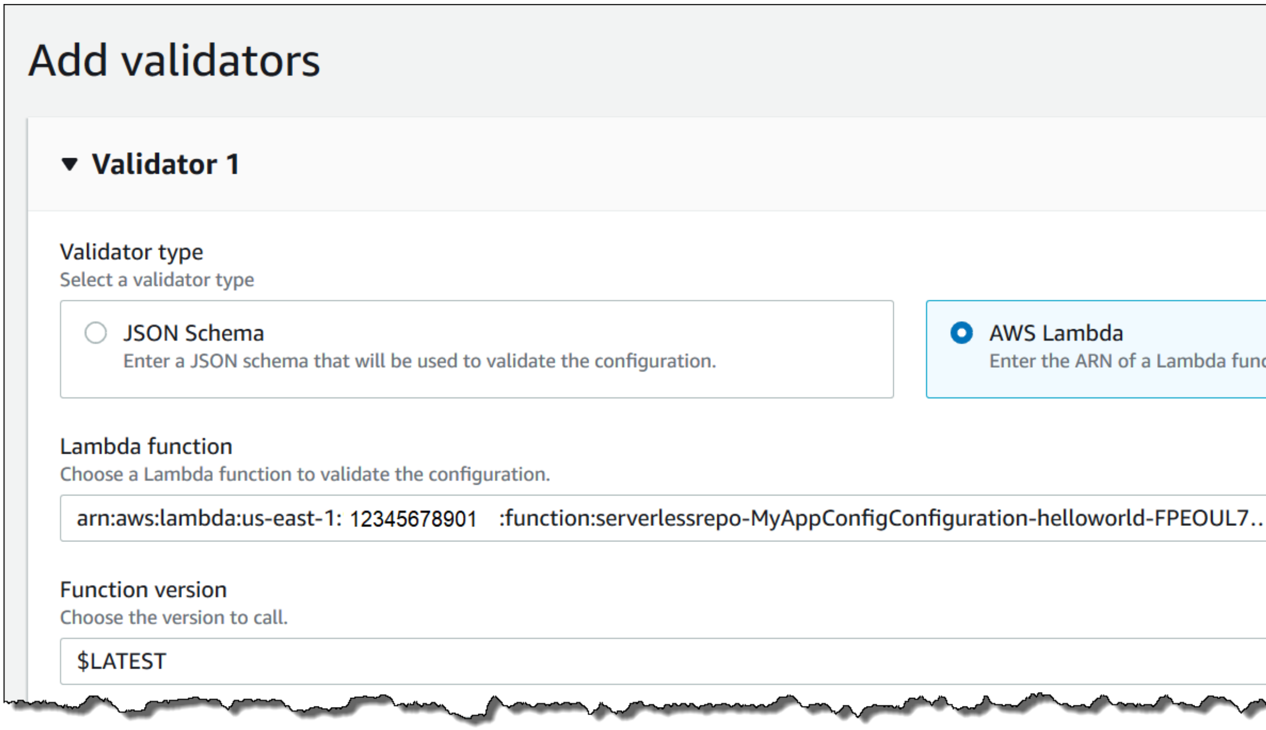
e. For **Application configuration schema version** either choose the version from the list or choose **Update schema** to edit the schema and create a new version.

f. In the **Content** section, choose either **YAML** or **JSON** and then enter the configuration data in the field.

g. Choose **Next**.

9. 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.

10. 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.

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. 28).

Creating an AWS AppConfig 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 configuration profile.

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. 5).

2. Run the following command to create a 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 \
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified_Locati
 \
  --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=validators_of_type_JSON
```

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 ^
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified_Locati
 ^
  --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=validators_of_type_JSON
```

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 `
  -
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=validators_of_type_JSON
```

The system returns information like the following.

Linux

```
{
   "ApplicationId": "The application ID",
   "Id": "The configuration profile ID",
   "Name": "The name of the configuration profile",
   "Description": "The configuration profile description",
   "LocationUri": "The URI location of the configuration",
   "RetrievalRoleArn": "The ARN of an IAM role with permission to access the
 configuration at the specified LocationUri",
   "Validators": [
      {
         "Content": "The JSON Schema content or the ARN of an AWS Lambda function",
```

```
            "Type": "Validators of type JSON_SCHEMA and LAMBDA"
        }
    ]
}
```

Windows

```
{
   "ApplicationId": "The application ID",
   "Id": "The configuration profile ID",
   "Name": "The name of the configuration profile",
   "Description": "The configuration profile description",
   "Id": "The configuration profile ID",
   "LocationUri": "The URI location of the configuration",
   "RetrievalRoleArn": "The ARN of an IAM role with permission to access the
 configuration at the specified LocationUri",
   "Validators": [
      {
         "Content": "The JSON Schema content or the ARN of an AWS Lambda function",
         "Type": "Validators of type JSON_SCHEMA and LAMBDA"
      }
   ]
}
```

PowerShell

```
ApplicationId    : The application ID
ContentLength    : Runtime of the command
Description      : The configuration profile description
HttpStatusCode   : HTTP Status of the runtime
Id               : The configuration profile ID
LocationUri      : The URI location of the configuration
Name             : The name of the configuration profile
ResponseMetadata : Runtime Metadata
RetrievalRoleArn : The ARN of an IAM role with permission to access the
 configuration at the specified LocationUri
Validators       : {Content: The JSON Schema content or the ARN of an AWS Lambda
 function, Type : Validators of type JSON_SCHEMA and LAMBDA}
```

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't target an encrypted S3 object.
Storage classes	AWS AppConfig supports the following S3 storage classes: `STANDARD`, `INTELLIGENT_TIERING`, `REDUCED_REDUNDANCY`, `STANDARD_IA`, and `ONEZONE_IA`. The following classes are not

Restriction	Details
	supported: All S3 Glacier classes (`GLACIER` and `DEEP_ARCHIVE`).
Versioning	AWS AppConfig requires that the S3 object use versioning.

Configuring permissions for a configuration stored as an Amazon S3 object

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

Permissions to access the S3 object

- s3:GetObject
- s3:GetObjectVersion

Permissions to list S3 buckets

s3:ListAllMyBuckets

Permissions to access the S3 bucket where the object is stored

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

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

Creating the IAM Policy for Accessing an S3 Object

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

To create an IAM policy for accessing an S3 object

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

```
{
  "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": "*"
    }
  ]
}
```

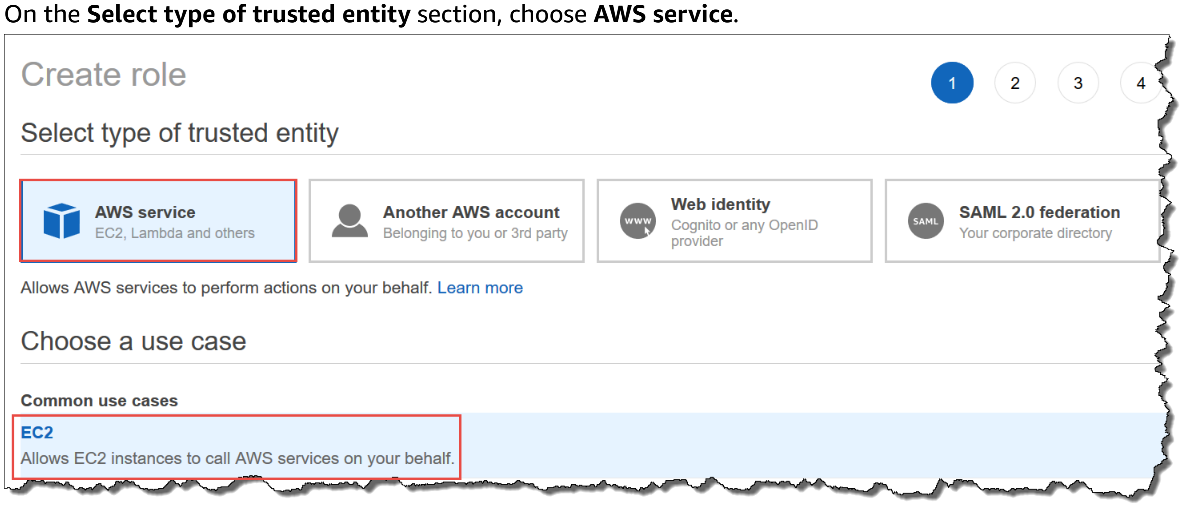
5. Choose **Review policy**.
6. On the **Review policy** page, type a name in the **Name** box, and then type a description.
7. Choose **Create policy**. The system returns you to the **Roles** page.

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.

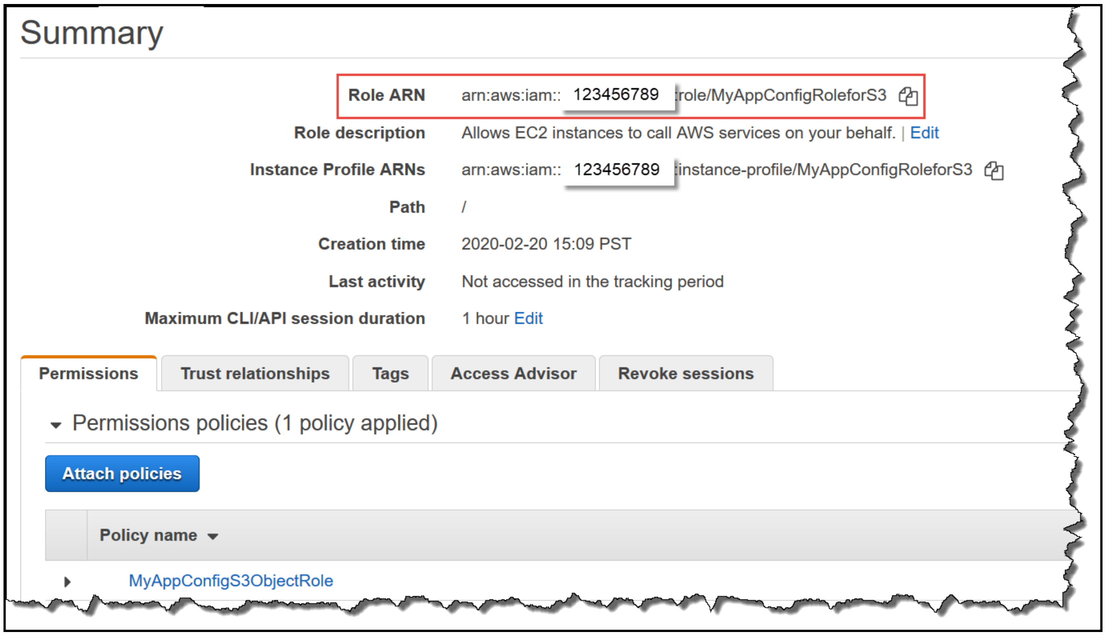
To create an IAM role for accessing an Amazon S3 object

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 the configuration by using the procedure in this topic, AWS AppConfig verifies that the configuration conforms to the schema requirements. If it doesn't, Systems Manager returns a validation error.

> **Note**
> 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.

```
{
   "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.

Step 4: Creating a deployment strategy

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

Setting	Description
Deployment type	Deployment type defines how the configuration deploys or *rolls out*. AWS AppConfig supports **Linear** and **Exponential** deployment types. • **Linear**: For this type, AWS AppConfig processes the deployment by increments of the growth factor evenly distributed over the deployment time. For example, a linear deployment that uses a step percentage of 20 initially makes the

Setting	Description

	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:
	`2*(2^0)`
	`2*(2^1)`
	`2*(2^2)`
	Expressed numerically, the deployment rolls out as follows: 2% of the targets, 4% of the targets, 8% of the targets, and continues until the configuration has been deployed to all targets.

Step percentage (growth factor)	This setting specifies the percentage of callers to target during each step of the deployment. **Note** In the SDK and the AWS AppConfig API Reference, `step percentage` is called `growth factor`.
Deployment time	This setting specifies an amount of time during which AWS AppConfig deploys to hosts. This is not a timeout value. It is a window of time during which the deployment is processed in intervals. >
Bake time	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. 8).

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	**Quick**: 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.
AppConfig.Linear50PercentEvery30Seconds	**Testing/Demonstration**: This strategy deploys the configuration to half of all targets every 30 seconds for a one-minute deployment. The system monitors for Amazon CloudWatch alarms for 1 minute. If no alarms are received in this time, the deployment is complete. If an alarm is triggered during this time, AppConfig rolls back the deployment. We recommend using this strategy only for testing or demonstration purposes because it has a short duration and bake time.
AppConfig.Canary10Percent20Minutes	**AWS Recommended**: 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. We recommend using this strategy for production deployments because it aligns with AWS best practices for configuration deployments.

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. 33). However, you must configure a client to receive application configuration updates as described in Step 6: Receiving the configuration (p. 36). 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. 33).

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. 5).
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_b
 \
  --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_b
 ^
  --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_
 `
  --
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.

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. 5).
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: Receiving the configuration

You must configure a client to receive configuration updates by integrating with the GetConfiguration API action. You can integrate using the AWS SDK. The following AWS CLI command demonstrates how to receive a configuration. This call includes the IDs of the AWS AppConfig application, the environment, the configuration profile, and a unique client ID. The configuration content is saved to the output filename.

Note

The `client-id` parameter in the following command is a unique, user-specified ID to identify the client for the configuration. This ID enables AWS AppConfig to deploy the configuration in intervals, as defined in the deployment strategy.

```
aws appconfig get-configuration \
    --application application_name_or_ID \
    --environment environment_name_or_ID \
    --configuration configuration_profile_name_or_ID \
    --client-id client_ID \
output_filename
```

The system responds with information in the following format.

```
{
   "ConfigurationVersion":"configuration version",
   "ContentType":"content type"
}
```

Important

Note the following important details about the `GetConfiguration` API action:

- The `GetConfiguration` response includes a `Content` section that shows the configuration data. The `Content` 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 `Content` section is not returned (`Null`).
- AWS AppConfig uses the value of the `ClientConfigurationVersion` parameter to identify the configuration version on your clients. If you don't send `ClientConfigurationVersion` with each call to `GetConfiguration`, your clients receive the current configuration. You are charged each time your clients receive a configuration.
- To avoid excess charges, we recommend that you include the `ClientConfigurationVersion` value with every call to `GetConfiguration`. This value must be saved on your client. Subsequent calls to `GetConfiguration` must pass this value by using the `ClientConfigurationVersion` parameter, as shown here.

Sending `ConfigurationVersion` during subsequent polling for configuration updates is similar to the concept of HTTP ETags.

```
aws appconfig get-configuration \
    --application application_name_or_ID \
    --environment environment_name_or_ID \
    --configuration configuration_profile_name_or_ID \
    --client-configuration-version previous_configuration_version_value \
    --client-id client_ID \
output_filename
```

Note

We recommend tuning the polling frequency of your `GetConfiguration` 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 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).

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*.

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 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.

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. For more information about Lambda extensions, see Lambda extensions in the *AWS Lambda Developer Guide*.

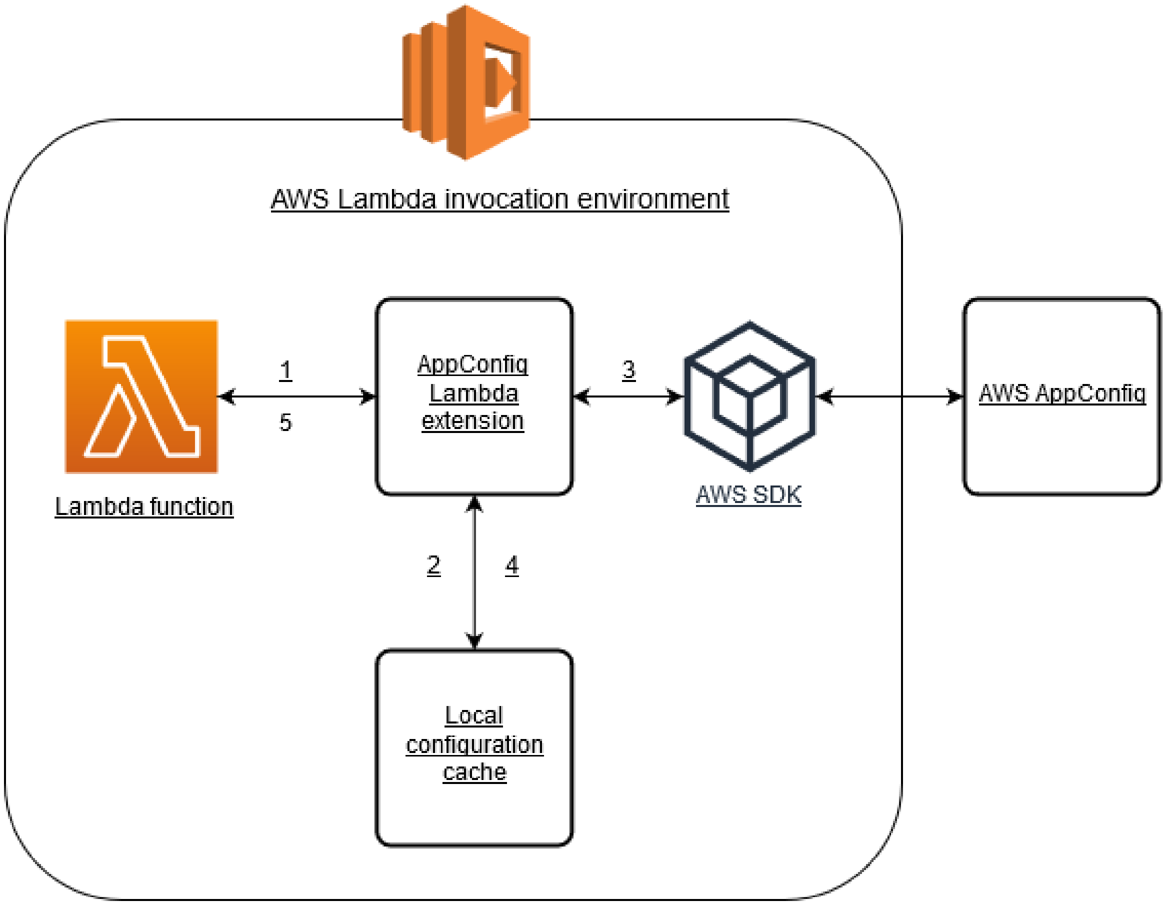
How it works

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 GetConfiguration API action. You integrate using the AWS SDK. The call to `GetConfiguration` includes the IDs of the AWS AppConfig application, the environment, the configuration profile, and a unique client ID.

You should also ensure that each call to `GetConfiguration` includes the `ClientConfigurationVersion` parameter. AWS AppConfig uses the value of this parameter to identify the configuration version used by AWS Lambda. If you don't include the `ClientConfigurationVersion` parameter, you will incur additional charges for using AWS AppConfig.

You are charged for each call to the `GetConfiguration` API. You are also charged for Lambda processing, rounded up to the nearest 100 ms.

Integrating the AWS AppConfig Lambda extension with your Lambda function simplifies this process and reduces costs. The following diagram shows how it works.



1. You configure the AWS AppConfig Lambda extension as a layer of your Lambda function.
2. The extension receives the `GetConfiguration` request from the function and checks the local cache. The local cache is an HTTP server.
3. The cache is empty so the extension passes the call to the SDK, which calls the AWS AppConfig service.
4. Upon receiving the configuration from the service, the extension stores it in the local cache and passes it to the Lambda function.
5. The extension locates and returns subsequent calls to `GetConfiguration` from the local cache. The AWS AppConfig Lambda extension manages configuration caching and updating so the configuration persists between Lambda handler invocations. When a configuration is needed, the Lambda handler makes a request to the AWS AppConfig Lambda extension instead of the AWS AppConfig service. The configuration is cached locally and returns much faster, which reduces the cost of using AWS Lambda and AWS AppConfig.

The next time the Lambda function is invoked, the extension checks the elapsed time since it retrieved a configuration from AWS AppConfig. If the elapsed time is greater than the poll interval, the extension retrieves the latest configuration from AWS AppConfig and updates the cache.

Before you begin

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

- Organize the dynamic 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).
- You must add `appconfig:GetConfiguration` 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 Configuring permissions for AWS AppConfig (p. 6).

Note the following details about using the AWS AppConfig Lambda extension.

> **Note**
>
> - 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.
> - AWS Lambda logs execution information about the AWS AppConfig Lambda extension along with the Lambda function by using Amazon CloudWatch Logs.
> - AWS Lambda records the `ExtensionDurationOverhead` metric. This metric measures extra time used by the extension after the function runs. For more information, see Monitoring and troubleshooting Lambda applications in the *AWS Lambda Developer Guide*.

Also, you can configure the extension by changing the following AWS Lambda environment variables, which are described in the following table. For more information, see Using AWS Lambda environment variables in the *AWS Lambda Developer Guide*.

Environment variable	Details	Default value
`AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS`	This environment variable controls how often the extension polls AWS AppConfig for an updated configuration in seconds.	45
`AWS_APPCONFIG_EXTENSION_POLL_TIMEOUT_MILLIS`	This environment variable controls the maximum amount of time the extension waits before retrieving configurations from the cache in milliseconds.	3000
`AWS_APPCONFIG_EXTENSION_HTTP_PORT`	This environment variable specifies the port on which the local HTTP server that hosts the extension runs.	2772

Adding the AWS AppConfig Lambda extension

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 the following list that corresponds with the AWS Region where you created the Lambda.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:1
US East (Ohio)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:1
US West (N. California)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:1
US West (Oregon)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:1
Canada (Central)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:1
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:1
Europe (Ireland)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:1
Europe (London)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:1
Europe (Paris)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:1
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:1
Europe (Milan)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:1
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:1
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:1

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:1
South America (São Paulo)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:1
Africa (Cape Town)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:1
Middle East (Bahrain)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:1

Note
The AWS AppConfig Lambda extension is only available in AWS Regions where AWS AppConfig is available.

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 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*.

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: AWS AppConfig Lambda extension API
  description: An API model for AWS 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: InternalServerException
          content:
            application/text:
              schema:
```

```
                $ref: '#/components/schemas/Error'
        502:
          description: BadGatewayException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        504:
          description: GatewayTimeoutException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'

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

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
AWS AppConfig Lambda Extension (p. 46)	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. 46)	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. 46)	Procedures in this user guide now include commandline steps for the 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. 46)	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	July 31, 2020

	hosted on EC2 instances, AWS Lambda, containers, mobile applications, or IoT devices.

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.