EC2 Image Builder

User Guide

EC2 Image Builder: User Guide

Table of Contents

What is EC2 Image Builder?

EC2 Image Builder is a fully managed AWS service that makes it easier to automate the creation, management, and deployment of customized, secure, and up-to-date server images that are pre-installed and pre-configured with software and settings to meet specific IT standards.

You can use the AWS Management Console, AWS CLI, or APIs to create custom images in your AWS account. When you use the AWS Management Console, the Image Builder wizard guides you through steps to:

- Provide starting artifacts
- Add and remove software
- Customize settings and scripts
- Run selected tests
- Distribute images to AWS Regions

The images you build are created in your account and you can configure them for operating system patches on an ongoing basis.

For troubleshooting and debugging your image deployment, you can configure build logs to be added to your Amazon Simple Storage Service (Amazon S3) bucket. You can also configure the instance-building application to send logs to CloudWatch. To receive notifications of image build status, and associate an Amazon Elastic Compute Cloud (Amazon EC2) key pair with your instance to perform manual debugging and inspection, you can configure an SNS topic.

Along with a final image, Image Builder creates an image recipe, which is a combination of the source image and components for building and testing. You can use the image recipe with existing source code version control systems and continuous integration/continuous deployment pipelines for repeatable automation.

Section Contents

Features of EC2 Image Builder

EC2 Image Builder provides the following features:

Increase productivity and reduce operations for building compliant and up-to-date images

Image Builder reduces the amount of work involved in creating and managing images at scale by automating your build pipelines. You can automate your builds by providing your build execution schedule preference. Automation reduces the operational cost of maintaining your software with the latest operating system patches.

Increase service uptime

Image Builder allows you to test your images before deployment with both AWS-provided and customized tests. AWS will distribute your image only if all of the configured tests have succeeded.

Raise the security bar for deployments

Image Builder allows you to create images that remove unnecessary exposure to component security vulnerabilities. You can apply AWS security settings to create secure, out-of-the-box images that meet industry and internal security criteria. Image Builder also provides collections of settings for companies in regulated industries. You can use these settings to help you quickly and easily build compliant images for STIG standards. For a complete list of STIG components available through Image Builder, see EC2 Image Builder STIG components (p. 96).

Centralized enforcement and lineage tracking

Using built-in integrations with AWS Organizations, Image Builder enables you to enforce policies that restrict accounts to run instances only from approved AMIs.

Simplified sharing of resources across AWS accounts

EC2 Image Builder integrates with AWS Resource Access Manager (AWS RAM) to allow you to share certain resources with any AWS account or through AWS Organizations. EC2 Image Builder resources that can be shared are:

- Components
- Images
- Image recipes
- Container recipes

For more information, see Share EC2 Image Builder resources (p. 146).

Supported operating systems

Image Builder supports the following operating systems:

- Amazon Linux 2
- Windows Server 2019/2016/2012 R2
- Windows Server version 2004, and 20H2
- Red Hat Enterprise Linux (RHEL) 8 and 7
- CentOS 8 and 7
- Ubuntu 20, 18, and 16
- SUSE Linux Enterprise Server (SUSE) 15

Supported image formats

For your custom AMI images, you can choose an existing AMI as a starting point. For Docker container images, you can choose from public images hosted on DockerHub, existing container images in Amazon ECR, or Amazon-managed container images.

Concepts

The following terms and concepts are central to your understanding and use of EC2 Image Builder.

AMI

An Amazon Machine Image (AMI) is the basic unit of deployment in Amazon EC2, and is one of the types of images you can create with Image Builder. An AMI is a pre-configured virtual machine image that contains the operating system (OS) and preinstalled software to deploy EC2 instances. For more information, see Amazon Machine Images (AMI).

Image pipeline

An image pipeline provides an automation framework for building secure AMIs and container images on AWS. The Image Builder image pipeline is associated with an image recipe or container recipe that defines the build, validation, and test phases for an image build lifecycle.

An image pipeline can be associated with an infrastructure configuration that defines where your image is built. You can define attributes, such as instance type, subnets, security groups, logging, and other infrastructure-related configurations. You can also associate your image pipeline with a distribution configuration to define how you would like to deploy your image.

Managed image

A managed image is a resource in Image Builder that consists of an AMI or container image, plus metadata, such as version and platform. The managed image is used by Image Builder pipelines to determine which base image to use for the build. In this guide, managed images are sometimes referred to as "images," however, an image is not the same as an AMI.

Image recipe

An Image Builder image recipe is a document that defines the base image and the components that are applied to the base image to produce the desired configuration for the output AMI image. You can use an image recipe to duplicate builds. Image Builder image recipes can be shared, branched, and edited using the console wizard, the AWS CLI, or the API. You can use image recipes with your version control software to maintain shareable, versioned image recipes.

Container recipe

An Image Builder container recipe is a document that defines the base image and the components that are applied to the base image to produce the desired configuration for the output container image. You can use a container recipe to duplicate builds. You can share, branch, and edit Image Builder image recipes by using the console wizard, the AWS CLI, or the API. You can use container recipes with your version control software to maintain shareable, versioned container recipes.

Base image

The base image is the selected image and operating system used in your image or container recipe document, along with the components. The base image and the component definitions combined produce the desired configuration for the output image.

Components

A component defines the sequence of steps required to either customize an instance prior to image creation (a **build component**), or to test an instance that was launched from the created image (a **test component**).

A component is created from a declarative, plain-text YAML or JSON document that describes the runtime configuration for building and validating, or testing an instance that is produced by your pipeline. Components run on the instance using a component management application. The component management application parses the documents and runs the desired steps.

After they are created, one or more components are grouped together using an image recipe or container recipe to define the plan for building and testing a virtual machine or container image. You

can use public components that are owned and managed by AWS, or you can create your own. For more information about components, see AWS Task Orchestrator and Executor component manager (p. 23).

Component document

A declarative, plain-text YAML or JSON document that describes configuration for a customization you can apply to your image. The document is used to create a build or test component.

Runtime stages

EC2 Image Builder has two runtime stages: **build** and **test**. Each runtime stage has one or more phases with configuration defined by the component document.

Configuration phases

The following list shows the phases that run during the **build** and **test** stages:

Build stage:

Build phase

An image pipeline begins with the build phase of the build stage when it runs. The base image is downloaded, and configuration that is specified for the build phase of the component is applied to build and launch an instance.

Validate phase

After the instance is launched and all of the build phase customizations are applied, the validation phase begins. During the validation phase, Image Builder validates that customizations are applied successfully, based on configuration that is specified for the validate phase of the component. If the instance validation is successful, the instance is turned off, an image is created, and Image Builder continues to the test stage.

Test stage:

Test phase

The test stage has only one phase – test. During this phase, an instance is launched from the image that was created after the validation phase completed successfully. Image Builder runs test components during this phase to verify that the instance is healthy, and is functioning as expected.

Pricing

There is no cost to use EC2 Image Builder to create custom AMI or container images. However, standard pricing applies for other services that are used in the process. The following list includes the usage of some AWS services that can incur costs when you create, build, store, and distribute your custom AMI or container images, depending on your configuration.

- Launching an EC2 instance
- Storing logs on Amazon S3
- Validating images with Amazon Inspector
- Storing Amazon EBS Snapshots for your AMIs
- Storing container images in Amazon ECR
- Pushing and pulling container images into and out of Amazon ECR
- If Systems Manager Advanced Tier is turned on, and Amazon EC2 instances run with on-premises activation, you might be charged for resources through Systems Manager

Related AWS services

EC2 Image Builder uses other AWS services to build images. Depending on your Image Builder image recipe or container recipe configuration, the following services might be used.

AWS License Manager

AWS License Manager allows you to create and apply license configurations from an account license configuration store. For each AMI, you can use Image Builder to attach to a preexisting license configuration that your AWS account has access to as part of the Image Builder workflow. License configurations can be applied only to AMIs. Image Builder can use only preexisting license configurations and cannot directly create or modify license configurations. License Manager settings will not replicate across AWS Regions that must be enabled in your account, for example, between the `ap-east-1` (Asia Pacific: Hong Kong) and the `me-south-1` (Middle East: Bahrain) Regions.

AWS Organizations

AWS Organizations allows you to apply Service Control Policies (SCP) on accounts in your organization. You can create, manage, enable, and disable individual policies. Similar to all other AWS artifacts and services, Image Builder honors the policies defined in AWS Organizations. AWS provides template SCPs for common scenarios, such as enforcing constraints on member accounts to launch instances with only approved AMIs.

Amazon Inspector

Image Builder uses Amazon Inspector as the default vulnerability scanning agent to establish security baselines for Amazon Linux 2, Windows Server 2012, and Windows Server 2016. For more information, see What is Amazon Inspector?

AWS Systems Manager Automation

An AWS Systems Manager automation document defines the actions that AWS Systems Manager performs on your managed instances and AWS resources. Systems Manager documents use JSON or YAML and include steps and parameters that you specify. The steps you specify run in sequential order. Automation documents are AWS Systems Manager documents of type Automation, as opposed to Command and Policy documents. For more information, see AWS Systems Manager Automation.

AWS Resource Access Manager

AWS Resource Access Manager (AWS RAM) lets you share your resources with any AWS account or through AWS Organizations. If you have multiple AWS accounts, you can create resources centrally and use AWS RAM to share those resources with other accounts. EC2 Image Builder allows sharing for the following resources: components, images, and image recipes. For more information about AWS RAM, see the AWS Resource Access Manager User Guide. For information about sharing Image Builder resources, see Share EC2 Image Builder resources (p. 146).

Amazon CloudWatch Logs

You can use Amazon CloudWatch Logs to monitor, store, and access your log files from EC2 instances, AWS CloudTrail, Amazon Route 53, and other sources.

Amazon Elastic Container Registry (Amazon ECR)

Amazon ECR is a managed AWS container image registry service that is secure, scalable, and reliable. Container images you create with Image Builder are stored in Amazon ECR in your default Region, and in any Regions where you distribute the container image. For more information about Amazon ECR, see the Amazon Elastic Container Registry User Guide.

How EC2 Image Builder works

When you use the EC2 Image Builder pipeline console wizard to create a custom image, a wizard guides you through the following steps.

1. **Specify pipeline details** – Enter information about your pipeline, such as a name, description, tags, and a schedule to run automated builds. You can choose manual builds, if you prefer.
2. **Choose recipe** – Choose between building an AMI, or building a container image. For both types of output images, you enter a name and version for your recipe, select a base image, and choose components to add for building and testing. You can also choose automatic versioning, to ensure that you always use the latest available Operating System (OS) version for your base image. Container recipes additionally define Dockerfiles, and the target Amazon ECR repository for your output Docker container image.

   Note
   Components are the building blocks that are consumed by an image recipe or a container recipe. For example, packages for installation, security hardening steps, and tests. The selected base image and components make up an image recipe.
3. **Define infrastructure configuration** – Image Builder launches EC2 instances in your account to customize images and run validation tests. The Infrastructure configuration settings specify infrastructure details for the instances that will run in your AWS account during the build process.
4. **Define distribution settings** – Choose the AWS Regions to distribute your image to after the build is complete and has passed all its tests. The pipeline automatically distributes your image to the Region where it runs the build, and you can add image distribution for other Regions.

The images that you build from your custom base image are in your AWS account. You can configure your image pipeline to produce updated and patched versions of your image by entering a build schedule. When the build is complete, you can receive notification through Amazon Simple Notification Service (SNS). In addition to producing a final image, the Image Builder console wizard generates a recipe that can be used with existing version control systems and continuous integration/continuous deployment (CI/CD) pipelines for repeatable automation. You can share and create new versions of your recipe.

Section contents

AMI elements

An Amazon Machine Image (AMI) is a preconfigured virtual machine (VM) image that contains the OS and software to deploy EC2 instances.

An AMI includes the following elements:

- A template for the root volume of the VM. When you launch an Amazon EC2 VM, the root device volume contains the image to boot the instance. When instance store is used, the root device is an instance store volume created from a template in Amazon S3. For more information, see Amazon EC2 Root Device Volume.
- When Amazon EBS is used, the root device is an EBS volume created from an EBS snapshot.
- Launch permissions that determine the AWS accounts that can launch VMs with the AMI.
- Block device mapping data that specifies the volumes to attach to the instance after launch.
- A unique resource identifier for each Region, for each account.
- Metadata payloads such as tags, and properties, such as Region, operating system, architecture, root device type, provider, launch permissions, storage for the root device, and signing status.
- An AMI signature for Windows images to protect against unauthorized tampering. For more information, see Instance Identity Documents.

Default quotas

To view the default quotas for Image Builder, see Image Builder Endpoints and Quotas.

AWS Regions and Endpoints

To view the service endpoints for Image Builder, see Image Builder Endpoints and Quotas.

Image Builder service integrations

EC2 Image Builder integrates with the following AWS services to provide detailed event metrics, logging, and monitoring to help you track your activity and troubleshoot image build issues.

- **Amazon CloudWatch Logs** – Monitor, store, and access your Image Builder log files. You can optionally save your logs to an S3 bucket. For more information about CloudWatch Logs, see What Is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*.
- **Amazon EventBridge** – Connect to a stream of real-time event data from Image Builder activities in your account. For more information about EventBridge, see What Is Amazon EventBridge? in the *Amazon EventBridge User Guide*.
- **AWS CloudTrail** – Monitor Image Builder events that are sent to CloudTrail. For more information about CloudTrail, see What Is AWS CloudTrail? in the *AWS CloudTrail User Guide*.

Service integrations

- Amazon CloudWatch Logs (p. 7)
- Amazon EventBridge (p. 8)
- AWS CloudTrail (p. 9)

Amazon CloudWatch Logs

CloudWatch Logs support is enabled by default. Logs are retained on the instance during the build process, and streamed to CloudWatch Logs. The instance logs are removed from the instance before image creation.

Build logs are streamed to following the Image Builder CloudWatch Logs group and stream:

- LogGroup: `"/aws/imagebuilder/<ImageName>`
- LogStream: `<ImageVersion>/<ImageBuildVersion>["x.x.x/x"]`

You can opt out of CloudWatch Logs streaming by removing the following permissions associated with the instance profile.

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:PutLogEvents"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/imagebuilder/*"
    }
]
```

For advanced troubleshooting, you can run predefined commands and scripts using AWS Systems Manager Run Command. For more information, see Troubleshoot EC2 Image Builder (p. 205).

Amazon EventBridge

Amazon EventBridge is a serverless event bus service that you can use to connect your Image Builder application with related data from other AWS services. In EventBridge, a rule matches incoming events and sends them to targets for processing. A single rule can send an event to multiple targets, which then run in parallel.

EventBridge enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can set up rules that react to incoming events to initiate actions; for example, sending an event to a Lambda function when the status of an EC2 instance changes from pending to running. These are called *patterns*. To create a rule based on an event pattern, see Creating Amazon EventBridge rules that react to events in the *Amazon EventBridge User Guide*.

Actions that can be automatically initiated include the following:

- Invoke an AWS Lambda function
- Invoke Amazon EC2 Run Command
- Relay the event to Amazon Kinesis Data Streams
- Activate an AWS Step Functions state machine
- Notify an Amazon SNS topic or an AWS SMS queue

You can also set up scheduling rules for the default event bus to perform an action at regular intervals, such as running an Image Builder pipeline to refresh an image on a quarterly basis. There are two types of schedule expressions:

- **cron expressions** – The following example of a cron expression schedules a task to run every day at noon UTC+0:

  ```
  cron(0 12 * * ? *)
  ```

  For more information about using cron expressions with EventBridge, see Cron expressions in the *Amazon EventBridge User Guide*.

- **rate expressions** – The following example of a rate expression schedules a task to run every 12 hours:

  ```
  rate(12 hour)
  ```

  For more information about using rate expressions with EventBridge, see Rate expressions in the *Amazon EventBridge User Guide*.

For more information about how EventBridge integrates with Image Builder image pipelines, see Use EventBridge rules with Image Builder pipelines (p. 170).

AWS CloudTrail

This service supports AWS CloudTrail, which is a service that records AWS calls for your AWS account and delivers log files to an Amazon S3 bucket. By using information collected by CloudTrail, you can determine what requests were successfully made to AWS services, who made the request, when it was made, and so on. For more information about CloudTrail integration with Image Builder, see Logging EC2 Image Builder API calls using AWS CloudTrail (p. 173).

To learn more about CloudTrail, including how to turn it on and find your log files, see the AWS CloudTrail User Guide.

Component management

EC2 Image Builder uses a component management application AWS Task Orchestrator and Executor (AWSTOE) that helps you orchestrate complex workflows, modify system configurations, and test your systems with YAML-based script components. Because AWSTOE is a standalone application, it does not require any additional setup. It can run on any cloud infrastructure and on premises. To get started using AWSTOE as a standalone application, see Get started with AWSTOE (p. 25).

Image Builder uses AWSTOE to perform all on-instance activities. These include building and validating your image before taking a snapshot, and testing the snapshot to ensure that it functions as expected before creating the final image. For more information about how Image Builder uses AWSTOE to manage its components, see Manage components with AWSTOE (p. 106). For more information about creating components with AWSTOE, see AWS Task Orchestrator and Executor component manager (p. 23).

Image testing

You can use AWSTOE test components to validate your image, and ensure that it functions as expected, prior to creating the final image.

Generally, each test component consists of a YAML document that contains a test script, a test binary, and test metadata. The test script contains the orchestration commands to start the test binary, which can be written in any language supported by the OS. Exit status codes indicate the test outcome. Test metadata describes the test and its behavior; for example, the name, description, paths to test binary, and expected duration.

Semantic versioning

Image Builder uses semantic versioning to organize resources and ensure that they have unique IDs. The semantic version has four nodes:

<major>.<minor>.<patch>/<build>

You can assign values for the first three, and can filter on all of them.

Semantic versioning is included in each object's Amazon Resource Name (ARN), at the level that applies to that object as follows:

1. Versionless ARNs and Name ARNs do not include specific values in any of the nodes. The nodes are either left off entirely, or they are specified as wildcards, for example: x.x.x.
2. Version ARNs have only the first three nodes: <major>.<minor>.<patch>
3. Build version ARNs have all four nodes, and point to a specific build for a specific version of an object.

Assignment: For the first three nodes you can assign any positive integer value, including zero, with an upper limit of 2^30-1, or 1073741823 for each node. Image Builder automatically assigns the build number to the fourth node.

Patterns: You can use any numeric pattern that adheres to the assignment requirements for the nodes that you can assign. For example, you might choose a software version pattern, such as 1.0.0, or a date, such as 2021.01.01.

Selection: With semantic versioning, you have the flexibility to use wildcards (x) to specify the most recent versions or nodes when selecting the base image or components for your recipe. When you use a wildcard in any node, all nodes to the right of the first wildcard must also be wildcards.

For example, given the following recent versions: 2.2.4, 1.7.8, and 1.6.8, version selection using wildcards produces the following results:

- `x.x.x` = 2.2.4
- `1.x.x` = 1.7.8
- `1.6.x` = 1.6.8
- `x.2.x` is not valid, and produces an error
- `1.x.8` is not valid, and produces an error

Resources created

When you create a pipeline, no resources external to Image Builder are created, unless the following is true:

- When an image is created through the pipeline schedule
- When you choose **Run Pipeline** from the **Actions** menu in the Image Builder console
- When you run either of these commands from the API or AWS CLI: **StartImagePipelineExecution** or **CreateImage**

The following resources are created during the image build process:

AMI image pipelines

- EC2 instance (*temporary*)
- Systems Manager Inventory Association (through Systems Manager State Manager) `EnhancedImageMetadata` is Enabled) on the EC2 instance
- Amazon EC2 AMI
- The Amazon EBS Snapshot associated with Amazon EC2 AMI

Container image pipelines

- Docker container running on an EC2 instance (*temporary*)

- Systems Manager Inventory Association (through Systems Manager State Manager) `EnhancedImageMetadata` is Enabled) on the EC2 instance
- Docker container image
- Dockerfile

After the image has been created, all of the temporary resources are deleted.

Distribution

EC2 Image Builder can distribute AMIs or container images to any AWS Region. The image is copied to each Region that you specify in the account used to build the image.

For AMI output images, you can define AMI launch permissions to control which AWS accounts are permitted to launch EC2 instances with the created AMI. For example, you can make the image private, public, or share with specific accounts. If you both distribute the AMI to other Regions, and define launch permissions for other accounts, the launch permissions are propagated to the AMIs in all of the Regions in which the AMI is distributed.

You can also use your AWS Organizations account to enforce limitations on member accounts to launch instances only with approved and compliant AMIs. For more information, see Managing the AWS accounts in Your Organization.

To update your distribution settings using the Image Builder console, follow the steps to Create a new image recipe version (console) (p. 121), or Create a new container recipe version (console) (p. 125).

Sharing Resources

To share components, recipes, or images with other accounts or within AWS Organizations, see Share EC2 Image Builder resources (p. 146).

Compliance

For CIS, EC2 Image Builder uses Amazon Inspector to perform assessments for exposure, vulnerabilities, and deviations from best practices and compliance standards. For example, it assesses unintended network accessibility, unpatched CVEs, public internet connectivity, and remote root login enablement. Amazon Inspector is offered as a test component that you can choose to add to your image recipe. For more information about Amazon Inspector, see the Amazon InspectorUser Guide. For hardening, EC2 Image Builder validates using STIG. For a complete list of STIG components available through Image Builder, see EC2 Image Builder STIG components (p. 96). For more information, see Center for Internet Security (CIS) Benchmarks.

Get started with EC2 Image Builder

This chapter helps you set up your environment and create an automated image pipeline or container pipeline for the first time, using the EC2 Image Builder **Create image pipeline** console wizard.

Contents

Prerequisites

Verify the following prerequisites to create an image pipeline with EC2 Image Builder. Unless specifically stated otherwise, prerequisites are required for all types of pipelines.

EC2 Image Builder service-linked role

EC2 Image Builder uses a service-linked role to grant permissions to other AWS services on your behalf. You don't need to manually create a service-linked role. When you create your first Image Builder resource in the AWS Management Console, the AWS CLI, or the AWS API, Image Builder creates the service-linked role for you. For more information about the service-linked role that Image Builder creates in your account, see Using service-linked roles for EC2 Image Builder (p. 196).

Configuration requirements

- Image Builder supports AWS PrivateLink.
- Image Builder supports EC2-Classic.
- The instances that Image Builder uses to build images and run tests must have access to the Systems Manager service. All build activity is orchestrated by Systems Manager Automation. Installation requirements depend on your operating system.

  To see the installation requirements for your base image, choose the tab that matches your base image operating system.

  Linux

  For Amazon EC2 Linux instances, Image Builder installs the Systems Manager Agent on the build instance if it is not already present, and removes it before creating the image.

  Windows

  Image Builder does not install the Systems Manager Agent on Amazon EC2 Windows Server build instances. If your base image did not come preinstalled with the Systems Manager Agent, you must launch an instance from your source image, manually install Systems Manager on the instance, and create a new base image from your instance.

  To manually install the Systems Manager agent on your Amazon EC2 Windows Server instance, see Manually install Systems Manager Agent on EC2 instances for Windows Server in the *AWS Systems Manager User Guide*.

Container repository (*container image pipelines*)

For container image pipelines, the recipe defines the configuration for the Docker images that are produced and stored in the target container repository. You must create the target repository before you create the container recipe for your Docker image.

The default target repository for Image Builder is Amazon ECR. To create an Amazon ECR repository, follow the steps described in Creating a repository in the *Amazon Elastic Container Registry User Guide*.

AWS Identity and Access Management (IAM)

The IAM role that you associate with your instance profile must have permissions to run the build and test components included in your image. The following IAM role policies must be attached to the IAM role that is associated with the instance profile:

- EC2InstanceProfileForImageBuilder
- EC2InstanceProfileForImageBuilderECRContainerBuilds
- AmazonSSMManagedInstanceCore

If you configure logging, the instance profile specified in your infrastructure configuration must have `s3:PutObject` permissions for the target bucket (`arn:aws:s3:::BucketName/*`). For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

Attach policy

The following steps guide you through the process of attaching the IAM policies to an IAM role to grant the preceding permissions.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Policies**.
3. Filter the list of policies with **EC2InstanceProfileForImageBuilder**
4. Select the bullet next to the policy, and from the **Policy actions** dropdown list, select **Attach**.
5. Select the name of the IAM role to which to attach the policy.
6. Choose **Attach policy**.
7. Repeat steps 3-6 for the **EC2InstanceProfileForImageBuilderECRContainerBuilds** and **AmazonSSMManagedInstanceCore** policies.

Note
If you want to copy an image created with Image Builder to another account, you must create the `EC2ImageBuilderDistributionCrossAccountRole` role in all of the target accounts, and attach the Ec2ImageBuilderCrossAccountDistributionAccess policy (p. 192) managed policy to the role. For more information, see Share EC2 Image Builder resources (p. 146).

Access EC2 Image Builder

You can manage EC2 Image Builder from one of the following interfaces.

- **EC2 Image Builder console landing page**. From the EC2 Image Builder console.
- **AWS Command Line Interface (AWS CLI)**. You can use the AWS CLI to access AWS API operations. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.
- **AWS Tools for SDKs**. You can use AWS SDKs and Tools to access and manage Image Builder using your preferred language.

Create an image pipeline using the EC2 Image Builder console wizard

This tutorial walks you through creating an automated pipeline to build and maintain a customized EC2 Image Builder image using the **Create image pipeline** console wizard. To help you move through the steps efficiently, default settings are used when they are available, and optional sections are skipped.

Create image pipeline workflow

Step 1: Specify pipeline details

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. To begin creating your pipeline, choose **Create image pipeline**.
3. In the **General** section, enter your **Pipeline name** (*required*).

   Tip
   Enhanced metadata collection is turned on by default. To ensure compatibility between components and base images, keep it turned on.
4. In the **Build schedule** section, you can keep the defaults for the **Schedule options**. Note that the **Time zone** shown for the default schedule is Universal Coordinated Time (UTC). For more information about UTC time, and to find the offset for your time zone, see Time Zone Abbreviations – Worldwide List.

   For **Dependency update settings**, choose the **Run pipeline at the scheduled time if there are dependency updates** option. This setting causes your pipeline to check for updates before starting the build. If there are no updates, it skips the scheduled pipeline build.

   Note
   To ensure that your pipeline recognizes dependency updates and builds as expected, you must use semantic versioning (x.x.x) for your base image and components. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).
5. Choose **Next** to proceed to the next step.

Step 2: Choose recipe

1. Image Builder defaults to **Use existing recipe** in the **Recipe** section. For your first time through, choose the **Create new recipe** option.
2. In the **Image type** section, choose the **Amazon Machine Image (AMI)** option to create an image pipeline that will produce and distribute an AMI.
3. In the **General** section, enter the following required boxes:

   - **Name** – your recipe name
   - **Version** – your recipe version (use the format *<major>.<minor>.<patch>*, where major, minor, and patch are integer values). New recipes generally start with `1.0.0`.
4. In the **Source image** section, keep the default values for **Select image**, **Image Operating System (OS)**, and **Image origin**. This results in a list of Amazon Linux 2 AMIs, managed by Amazon, for you to choose from for your base image.

   a. From the **Image name** dropdown, choose an image.

   b. Keep the default for **Auto-versioning options** (**Use latest available OS version**).

   > **Note**
   > This setting ensures that your pipeline uses semantic versioning for the base image, to detect dependency updates for automatically scheduled jobs. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).
5. In the **Instance configuration** section, keep the default values for the **Systems Manager agent**. This results in Image Builder removing the Systems Manager agent after the build and tests are complete, before creating the new image.

   Keep **User data** blank for this tutorial. You can use this area at other times to provide commands, or a command script to run when you launch your build instance. However, it replaces any commands that Image Builder might have added to ensure that Systems Manager is installed. When you do use it, make sure that the Systems Manager agent is preinstalled on your base image, or that you include the install in your user data.
6. In the **Components** section, you must choose at least one build component.

   In the **Build components – Amazon Linux** panel, you can browse through the components listed on the page. Use the pagination control in the upper right corner to navigate through additional components that are available for your base image OS. You can also search for specific components, or create your own build component using the Component manager.

   For this tutorial, choose a component that updates Linux with the latest security updates, as follows:

   a. Filter the results by entering the word `update` in the search bar that's located at the top of the panel.

   b. Select the check box for the `update-linux` build component.

   c. Scroll down, and in the upper right corner of the **Selected components** list, choose **Expand all** .

   d. Keep the default for **Versioning options** (**Use latest available component version**).

   > **Note**
   > This setting ensures that your pipeline uses semantic versioning for the selected component, to detect dependency updates for automatically scheduled jobs. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

   If you had selected a component that has input parameters, you would also see the parameters in this area. Parameters are not covered in this tutorial. For more information about using input parameters in your components, and setting them in your recipes, see Manage AWSTOE component parameters with EC2 Image Builder (p. 115).

Reorder components (optional)

If you have chosen more than one component to include in your image, you can use the drag-and-drop action to rearrange them into the order in which they should run during the build process.

1. Scroll back up to the list of available components.
2. Select the check box for the `update-linux-kernel-mainline` build component (or any other component of your choice).
3. Scroll down to the **Selected components** list, to see that there are at least two results.
4. Newly added components might not have their versioning or input parameter settings expanded. To expand **Versioning options** or **Input parameters** settings, you can either choose the arrow next to the name of the setting, or you can toggle the **Expand all** switch off and on to expand all of the settings for all of the selected components.
5. Choose one of the components, and drag it up or down to change the order in which the components will run.
6. To remove the `update-linux-kernel-mainline` component, choose `X` from the upper right corner of the component box.
7. Repeat the previous step to remove any other components you might have added, leaving only the `update-linux` component selected.

7. Choose **Next** to proceed to the next step.

Step 3: Define infrastructure configuration - optional

Image Builder launches EC2 instances in your account to customize images and run validation tests. The Infrastructure configuration settings specify infrastructure details for the instances that will run in your AWS account during the build process.

In the **Infrastructure configuration** section, the **Configuration options** default to `Create infrastructure configuration using service defaults`. This creates an IAM role and associated instance profile that are used by build instances to configure your EC2 AMIs. You can also create your own custom infrastructure configuration, or use settings that you have already created. For this tutorial, we are using the default settings.

- Choose **Next** to proceed to the next step.

Step 4: Define distribution settings - optional

Distribution settings include specific Region settings for encryption, launch permissions, accounts that can launch the output AMI, the output AMI name, and license configurations.

In the **Distribution settings** section, the **Configuration options** default to `Create distribution settings using service defaults`. This option will distribute the output AMI to the current Region. For this tutorial, we are using the default settings.

- Choose **Next** to proceed to the next step.

Step 5: Review

The **Review** section displays all of the settings you have configured. To edit information in any given section, choose the **Edit** button located in the top right corner of the step section. For example, if

you want to change your pipeline name, choose the **Edit** button in the top right corner of the **Step 1: Pipeline details** section.

1. When you have reviewed your settings, choose **Create pipeline** to create your pipeline.
2. You can see success or failure messages at the top of the page, as your resources are created for distribution settings, infrastructure configuration, your new recipe, and the pipeline. To see details for a resource, including the resource identifier, choose **View details**.
3. After you have viewed the details for a resource, you can view details about other resources by choosing the resource type from the navigation pane. For example, to see details for your new pipeline, choose **Image pipelines** from the navigation pane. If your build was successful, your new pipeline is displayed in the **Image pipelines** list.

Step 6: Clean up

Your Image Builder environment, just like your home, needs regular maintenance to help you find what you need, and complete your tasks without wading through clutter. Make sure to regularly clean up temporary resources that you created for testing. Otherwise, you might forget about those resources, and then later, not remember what they were used for. By then, it might not be clear if you can safely get rid of them.

> **Tip**
> To prevent dependency errors when you delete resources, make sure to delete your resources in the following order:
>
> 1. Image pipeline
> 2. Image recipe
> 3. All remaining resources

To clean up the resources that you created for this tutorial, follow these steps:

Delete the pipeline

1. To see a list of the build pipelines created under your account, choose **Image pipelines** from the navigation pane.
2. Select the check box next to **Pipeline name** to select the pipeline that you want to delete.
3. At the top of the **Image pipelines** panel, on the **Actions** menu, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete the recipe

1. To see a list of the recipes created under your account, choose **Image recipes** from the navigation pane.
2. Select the check box next to **Recipe name** to select the recipe that you want to delete.
3. At the top of the **Image recipes** panel, on the **Actions** menu, choose **Delete recipe**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete infrastructure configuration

1. To see a list of the infrastructure configurations created under your account, choose **Infrastructure configuration** from the navigation pane.
2. Select the check box next to **Configuration name** to select the infrastructure configuration that you want to delete.

3. At the top of the **Infrastructure configurations** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete distribution settings

1. To see a list of the distribution settings created under your account, choose **Distribution settings** from the navigation pane.
2. Select the check box next to **Configuration name** to select the distribution settings that you created for this tutorial.
3. At the top of the **Distribution settings** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete the image

Follow these steps to verify that you have deleted any image that was created from the tutorial pipeline. This tutorial is not likely to create an image unless enough time has elapsed since you created your pipeline that it runs, according to the build schedule.

1. To see a list of the images created under your account, choose **Images** from the navigation pane.
2. Choose the image **Version** for the image that you want to remove. This opens the **Image build versions** page.
3. Select the check box next to the **Version** for any image that you want to delete. You can select more than one image version at a time.
4. At the top of the **Image build versions** panel, choose **Delete version**.
5. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Create a container image pipeline using the EC2 Image Builder console wizard

This tutorial walks you through creating an automated pipeline to build and maintain a customized EC2 Image Builder Docker image using the **Create image pipeline** console wizard. To help you move through the steps efficiently, default settings are used when they are available, and optional sections are skipped.

Create image pipeline workflow

Step 1: Specify pipeline details

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. To begin creating your pipeline, choose **Create image pipeline**.

3. In the **General** section, enter your **Pipeline name** (*required*).

   Tip
   Enhanced metadata collection is turned on by default. To ensure compatibility between components and base images, keep it turned on.

4. In the **Build schedule** section, you can keep the defaults for the **Schedule options**. Note that the **Time zone** shown for the default schedule is Universal Coordinated Time (UTC). For more information about UTC time, and to find the offset for your time zone, see Time Zone Abbreviations – Worldwide List.

   For **Dependency update settings**, choose the **Run pipeline at the scheduled time if there are dependency updates** option. This setting causes your pipeline to check for updates before starting the build. If there are no updates, it skips the scheduled pipeline build.

   Note
   To ensure that your pipeline recognizes dependency updates and builds as expected, you must use semantic versioning (x.x.x) for your base image and components. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

5. Choose **Next** to proceed to the next step.

Step 2: Choose recipe

1. Image Builder defaults to **Use existing recipe** in the **Recipe** section. For your first time through, choose the **Create new recipe** option.
2. In the **Image type** section, choose the **Docker image** option to create a container pipeline that will produce a Docker image and distribute it to Amazon ECR repositories in target Regions.
3. In the **General** section, enter the following required boxes:

   - **Name** – your recipe name
   - **Version** – your recipe version (use the format *<major>.<minor>.<patch>*, where major, minor, and patch are integer values). New recipes generally start with `1.0.0`.

4. In the **Source image** section, keep the default values for **Select image**, **Image Operating System (OS)**, and **Image origin**. This results in a list of Amazon Linux 2 container images, managed by Amazon, for you to choose from for your base image.

   a. From the **Image name** dropdown, choose an image.

   b. Keep the default for **Auto-versioning options** (**Use latest available OS version**).

      Note
      This setting ensures that your pipeline uses semantic versioning for the base image, to detect dependency updates for automatically scheduled jobs. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

5. In the **Components** section, you must choose at least one build component.

   In the **Build components – Amazon Linux** panel, you can browse through the components listed on the page. Use the pagination control in the upper right corner to navigate through additional components that are available for your base image OS. You can also search for specific components, or create your own build component using the Component manager.

   For this tutorial, choose a component that updates Linux with the latest security updates, as follows:

   a. Filter the results by entering the word `update` in the search bar that's located at the top of the panel.

   b. Select the check box for the `update-linux` build component.

   c. Scroll down, and in the upper right corner of the **Selected components** list, choose **Expand all** .

d. Keep the default for **Versioning options** (**Use latest available component version**).

> **Note**
> This setting ensures that your pipeline uses semantic versioning for the selected component, to detect dependency updates for automatically scheduled jobs. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

If you had selected a component that has input parameters, you would also see the parameters in this area. Parameters are not covered in this tutorial. For more information about using input parameters in your components, and setting them in your recipes, see Manage AWSTOE component parameters with EC2 Image Builder (p. 115).

Reorder components (optional)

If you have chosen more than one component to include in your image, you can use the drag-and-drop action to rearrange them into the order in which they should run during the build process.

1. Scroll back up to the list of available components.
2. Select the check box for the `update-linux-kernel-mainline` build component (or any other component of your choice).
3. Scroll down to the **Selected components** list, to see that there are at least two results.
4. Newly added components might not have their versioning expanded. To expand **Versioning options**, you can either choose the arrow next to **Versioning options**, or you can toggle the **Expand all** switch off and on to expand versioning for all of the selected components.
5. Choose one of the components, and drag it up or down to change the order in which the components will run.
6. To remove the `update-linux-kernel-mainline` component, choose `X` from the upper right corner of the component box.
7. Repeat the previous step to remove any other components you might have added, leaving only the `update-linux` component selected.

6. In the **Dockerfile template** section, select the **Use example** option. In the **Content** panel, notice the contextual variables where Image Builder places build information or scripts, based on your container image recipe.
7. In the **Target repository** section, you must specify the name of an existing Amazon ECR repository that you created as a prerequisite for this tutorial.

> **Note**
> The target repository must exist in Amazon ECR for all target Regions prior to distribution.

8. Choose **Next** to proceed to the next step.

Step 3: Define infrastructure configuration - optional

Image Builder launches EC2 instances in your account to customize images and run validation tests. The Infrastructure configuration settings specify infrastructure details for the instances that will run in your AWS account during the build process.

In the **Infrastructure configuration** section, the **Configuration options** default to `Create infrastructure configuration using service defaults`. This creates an IAM role and associated instance profile that are used by build instances to configure your EC2 AMIs. You can also create your own custom infrastructure configuration, or use settings that you have already created. For this tutorial, we are using the default settings.

- Choose **Next** to proceed to the next step.

Step 4: Define distribution settings - optional

Distribution settings consist of the target Regions, and the target Amazon ECR repository name. Output Docker images are deployed to the named Amazon ECR repository in each Region.

In the **Distribution settings** section, the **Configuration options** default to `Create distribution settings using service defaults`. This option will distribute the output Docker image to the Amazon ECR repository in the current Region. For this tutorial, we are using the default settings.

- Choose **Next** to proceed to the next step.

Step 5: Review

The **Review** section displays all of the settings you have configured. To edit information in any given section, choose the **Edit** button located in the top right corner of the step section. For example, if you want to change your pipeline name, choose the **Edit** button in the top right corner of the **Step 1: Pipeline details** section.

1. When you have reviewed your settings, choose **Create pipeline** to create your pipeline.
2. You can see success or failure messages at the top of the page, as your resources are created for distribution settings, infrastructure configuration, your new recipe, and the pipeline. To see details for a resource, including the resource identifier, choose **View details**.
3. After you have viewed the details for a resource, you can view details about other resources by choosing the resource type from the navigation pane. For example, to see details for your new pipeline, choose **Image pipelines** from the navigation pane. If your build was successful, your new pipeline is displayed in the **Image pipelines** list.

Step 6: Clean up

Your Image Builder environment, just like your home, needs regular maintenance to help you find what you need, and complete your tasks without wading through clutter. Make sure to regularly clean up temporary resources that you created for testing. Otherwise, you might forget about those resources, and then later, not remember what they were used for. By then, it might not be clear if you can safely get rid of them.

Tip
To prevent dependency errors when you delete resources, make sure to delete your resources in the following order:

1. Image pipeline
2. Image recipe
3. All remaining resources

To clean up the resources that you created for this tutorial, follow these steps:

Delete the pipeline

1. To see a list of the build pipelines created under your account, choose **Image pipelines** from the navigation pane.
2. Select the check box next to **Pipeline name** to select the pipeline that you want to delete.
3. At the top of the **Image pipelines** panel, on the **Actions** menu, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete the container recipe

1. To see a list of the container recipes created under your account, choose **Container recipes** from the navigation pane.
2. Select the check box next to **Recipe name** to select the recipe that you want to delete.
3. At the top of the **Container recipes** panel, on the **Actions** menu, choose **Delete recipe**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete infrastructure configuration

1. To see a list of the infrastructure configurations created under your account, choose **Infrastructure configuration** from the navigation pane.
2. Select the check box next to **Configuration name** to select the infrastructure configuration that you want to delete.
3. At the top of the **Infrastructure configurations** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete distribution settings

1. To see a list of the distribution settings created under your account, choose **Distribution settings** from the navigation pane.
2. Select the check box next to **Configuration name** to select the distribution settings that you created for this tutorial.
3. At the top of the **Distribution settings** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete the image

Follow these steps to verify that you have deleted any image that was created from the tutorial pipeline. This tutorial is not likely to create an image unless enough time has elapsed since you created your pipeline that it runs, according to the build schedule.

1. To see a list of the images created under your account, choose **Images** from the navigation pane.
2. Choose the image **Version** for the image that you want to remove. This opens the **Image build versions** page.
3. Select the check box next to the **Version** for any image that you want to delete. You can select more than one image version at a time.
4. At the top of the **Image build versions** panel, choose **Delete version**.
5. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

AWS Task Orchestrator and Executor component manager

EC2 Image Builder uses the AWS Task Orchestrator and Executor (AWSTOE) application to orchestrate complex workflows, modify system configurations, and test your systems without writing code. This application uses a declarative document schema. Because it is a standalone application, it does not require additional server setup. It can run on any cloud infrastructure and on premises.

To install AWSTOE, choose the download link for your architecture and platform.

Contents

AWSTOE downloads

Architecture	Platform	Download link	Example
386	AL2, RHEL 8 and 7, Ubuntu 18 and 16, CentOS 7, SUSE 15	`https://awstoe-<region>.s3.<region>.amazonaws.com/latest/linux/386/awstoe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/386/awstoe
AMD64	Windows Server 2019/2016/2012 R2/ version 1909	`https://awstoe-<region>.s3.<region>.amazonaws.com/latest/windows/amd64/awstoe.exe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/windows/amd64/awstoe.exe
AMD64	AL2, RHEL 8 and 7, Ubuntu 18 and 16, CentOS 7, SUSE 15	`https://awstoe-<region>.s3.<region>.amazonaws.com/latest/linux/amd64/awstoe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/amd64/awstoe
ARM64	AL2, RHEL 8 and 7, Ubuntu 18 and 16, CentOS 7, SUSE 15	`https://awstoe-<region>.s3.<region>.amazonaws.com/`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/

Architecture	Platform	Download link	Example
		`latest/linux/arm64/awstoe`	latest/linux/arm64/awstoe

Supported Regions

AWSTOE is supported as a standalone application in the following Regions.

Region name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Milan)	eu-south-1
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1

Get started with AWSTOE

The AWS Task Orchestrator and Executor (AWSTOE) application is a standalone application that creates, validates, and runs commands within a component definition framework. AWS services can use AWSTOE to orchestrate workflows, install software, modify system configurations, and test image builds.

Follow these steps to install the AWSTOE application and use it for the first time.

Verify the signature of the AWSTOE installation download

This section describes the recommended process for verifying the validity of the installation download for AWSTOE on Linux- and Windows-based operating systems.

Topics

Verify the signature of the AWSTOE installation download on Linux

This topic describes the recommended process for verifying the validity of the installation download for the AWSTOE on Linux-based operating systems.

Whenever you download an application from the internet, we recommend that you authenticate the identity of the software publisher. Also, check that the application is not altered or corrupted since it was published. This protects you from installing a version of the application that contains a virus or other malicious code.

If, after running the steps in this topic, you determine that the software for the AWSTOE is altered or corrupted, do not run the installation file. Instead, contact AWS Support For more information about your support options, see AWS Support.

AWSTOE files for Linux-based operating systems are signed using `GnuPG`, an open source implementation of the Pretty Good Privacy (OpenPGP) standard for secure digital signatures. `GnuPG` (also known as `GPG`) provides authentication and integrity checking through a digital signature. Amazon EC2 publishes a public key and signatures that you can use to verify the downloaded Amazon EC2 CLI tools. For more information about `PGP` and `GnuPG` (`GPG`), see http://www.gnupg.org.

The first step is to establish trust with the software publisher. Download the public key of the software publisher, check that the owner of the public key is who they claim to be, and then add the public key to your *keyring*. Your keyring is a collection of known public keys. After you establish the authenticity of the public key, you can use it to verify the signature of the application.

Topics

Installing the GPG tools

If your operating system is Linux or Unix, the GPG tools are likely already installed. To test whether the tools are installed on your system, type **gpg** at a command prompt. If the GPG tools are installed, you

see a GPG command prompt. If the GPG tools are not installed, you see an error message stating that the command cannot be found. You can install the GnuPG package from a repository.

To install GPG tools on Debian-based Linux

- From a terminal, run the following command: **apt-get install gnupg**.

To install GPG tools on Red Hat–based Linux

- From a terminal, run the following command: **yum install gnupg**.

Authenticating and importing the public key

The next step in the process is to authenticate the AWSTOE public key and add it as a trusted key in your GPG keyring.

To authenticate and import the AWSTOE public key

1. Obtain a copy of our public GPG build key by doing one of the following:
   - Download the key from https://awstoe-**<region>**.s3.**<region>**.amazonaws.com/assets/awstoe.gpg. For example, https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/assets/awstoe.gpg.
   - Copy the key from the following text and paste it into a file called `awstoe.gpg`. Make sure to include everything that follows:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQENBF8UqwsBCACdiRF2bkZYaFSDPFC+LIkWLwFvtUCRwAHtD8KIwTJ6LVn3fHAU
GhuK0ZH9mRrqRT2bq/xJjGsnF9VqTj2AJqndGJdDjz75YCZYM+ocZ+r5HSJaeW9i
S5dykHj7Txti2zHe0G5+W0v7v5bPi2sPHsN7XWQ7+G2AMEPTz8PjxY//I0DvMQns
Sle3l9hz6wCClz1l9LbBzTyHfSm5ucTXvNe88XX5Gmt37OCDM7vfli0Ctv8WFoLN
6jbxuA/sV71yIkPm9IYp3+GvaKeT870+sn8/JOOKE/U4sJV1ppbqmuUzDfhrZUaw
8eW8IN9A1FTIuWiZED/5L83UZuQs1S7s2PjlABEBAAG0GkFXU1RPRSA8YXdzdG9l
QGFtYXpvbi5jb20+iQE5BBMBCAAjBQJfFKsLAhsDBwsJCAcDAgEGFQgCCQoLBBYC
AwECHgECF4AACgkQ3r3BVvWuvFJGiwf9EVmrBR77+Qe/DUeXZJYoaFr7If/fVDZl
6V3TC6p0J0Veme7uXleRUTFOjzbh+7e5sDX19HrnPquzCnzfMiqbp4lSoeUuNdOf
FcpuTCQH+M+sIEIgPno4PLl0Uj2uE1o++mxmonBl/Krk+hly8hB2L/9n/vW3L7BN
OMb1Ll9PmgGPbWipcT8KRdz4SUex9TXGYzjlWb3jU3uXetdaQY1M3kVKE1siRsRN
YYDtpcjmwbhjpu4xm19aFqNoAHCDctEsXJA/mkU3erwIRocPyjAZE2dnlkL9ZkFZ
z9DQkcIarbCnybDM5lemBbdhXJ6hezJE/b17VA0t1fY04MoEkn6oJg==
=oyze
-----END PGP PUBLIC KEY BLOCK-----
```

2. At a command prompt in the directory where you saved **awstoe.gpg**, use the following command to import the AWSTOE public key into your keyring.

```
gpg --import awstoe.gpg
```

   The command returns results that are similar to the following:

```
gpg: key F5AEBC52: public key "AWSTOE <awstoe@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

   Make a note of the key value; you need it in the next step. In the preceding example, the key value is `F5AEBC52`.

3. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
gpg --fingerprint key-value
```

This command returns results similar to the following:

```
pub   2048R/F5AEBC52 2020-07-19
      Key fingerprint = F6DD E01C 869F D639 15E5  5742 DEBD C156 F5AE BC52
uid       [ unknown] AWSTOE <awstoe@amazon.com>
```

Additionally, the fingerprint string should be identical to `F6DD E01C 869F D639 15E5 5742 DEBD C156 F5AE BC52`, as shown in the preceding example. Compare the key fingerprint that is returned to the one published on this page. They should match. If they don't match, do not install the AWSTOE installation script, and contact AWS Support.

Verify the signature of the package

After you install the `GPG` tools, authenticate and import the AWSTOE public key, and verify that the public key is trusted, you are ready to verify the signature of the installation script.

To verify the installation script signature

1. At a command prompt, run the following command to download the application binary:

```
curl -O https://awstoe-<region>.s3.<region>.amazonaws.com/latest/linux/<architecture>/
awstoe
```

For example:

```
curl -O https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/amd64/awstoe
```

Supported values for **`architecture`** can be `amd64`, `386`, and `arm64`.

2. At a command prompt, run the following command to download the signature file for the corresponding application binary from the same S3 key prefix path:

```
curl -O https://awstoe-<region>.s3.<region>.amazonaws.com/latest/linux/<architecture>/
awstoe.sig
```

For example:

```
curl -O https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/amd64/
awstoe.sig
```

Supported values for **`architecture`** can be `amd64`, `386`, and `arm64`.

3. Verify the signature by running the following command at a command prompt in the directory where you saved `awstoe.sig` and the AWSTOE installation file. Both files must be present.

```
gpg --verify ./awstoe.sig ~/awstoe
```

The output should look something like the following:

```
gpg: Signature made Mon 20 Jul 2020 08:54:55 AM IST using RSA key ID F5AEBC52
gpg: Good signature from "AWSTOE awstoe@amazon.com" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F6DD E01C 869F D639 15E5 5742 DEBD C156 F5AE BC52
```

If the output contains the phrase `Good signature from "AWSTOE <awstoe@amazon.com>"`, it means that the signature has successfully been verified, and you can proceed to run the AWSTOE installation script.

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, do not run the installation file that you downloaded previously, and contact AWS Support.

The following are details about the warnings that you might see:

- **WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.** Ideally, you would visit an AWS office and receive the key in person. However, you would most likely download it from a website. In this case, the website is an AWS website.
- **gpg: no ultimately trusted keys found.** This means that the specific key is not "ultimately trusted" by you, or by other people that you trust.

For more information, see http://www.gnupg.org.

Verify the signature of the AWSTOE installation download on Windows

This topic describes the recommended process for verifying the validity of the installation file for the AWS Task Orchestrator and Executor application on Windows-based operating systems.

Whenever you download an application from the internet, we recommend that you authenticate the identity of the software publisher and check that the application is not altered or corrupted since it was published. This protects you from installing a version of the application that contains a virus or other malicious code.

If, after running the steps in this topic, you determine that the software for the AWSTOE application is altered or corrupted, do not run the installation file. Instead, contact AWS Support.

To verify the validity of the downloaded awstoe binary on Windows-based operating systems, make sure that the thumbprint of its Amazon Services LLC signer certificate is equal to this value:

16 67 49 A7 B8 CC 5B 8A 57 1D DF 4B 7A 37 9D B1 6A 5E 65 80

To verify this value, perform the following procedure:

1. Right-click the downloaded `awstoe.exe`, and open the **Properties** window.
2. Choose the **Digital Signatures** tab.
3. From the **Signature List**, choose **Amazon Services LLC**, and then choose **Details**.
4. Choose the **General** tab, if not already selected, and then choose **View Certificate**.
5. Choose the **Details** tab, and then choose **All** in the **Show** dropdown list, if not already selected.
6. Scroll down until you see the **Thumbprint** field and then choose **Thumbprint**. This displays the entire thumbprint value in the lower window.

- If the thumbprint value in the lower window is identical to the following value:

  16 67 49 A7 B8 CC 5B 8A 57 1D DF 4B 7A 37 9D B1 6A 5E 65 80

  then your downloaded AWSTOE binary is authentic and can be safely installed.
- If the thumbprint value in the lower details window is not identical to the previous value, do not run `awstoe.exe`.

Get started steps

- Step 1: Install AWSTOE (p. 29)
- Step 2: Set AWS credentials (p. 29)
- Step 3: Develop component documents locally (p. 30)
- Step 4: Validate AWSTOE components (p. 31)
- Step 5: Run AWSTOE components (p. 31)

Step 1: Install AWSTOE

To develop components locally, download and install the AWSTOE application.

1. **Download the AWSTOE application**

   To install AWSTOE, choose the appropriate download link for your architecture and platform. For the full list of application download links, see AWSTOE downloads (p. 23)
2. **Verify the signature**

   The steps for verifying your download depend on the server platform where you run the AWSTOE application after you install it. To verify your download on a Linux server, see Verify the signature on Linux (p. 25). To verify your download on a Windows server, see Verify the signature on Windows (p. 28).

   Note
   AWSTOE is invoked directly from its download location. There is no need for a separate install step.

Step 2: Set AWS credentials

AWSTOE requires AWS credentials to connect to other AWS services, such as Amazon S3 and Amazon CloudWatch, when running tasks, such as:

- Downloading AWSTOE documents from a user-provided Amazon S3 path.
- Running `S3Download` or `S3Upload` action modules.
- Streaming logs to CloudWatch, when enabled.

If you are running AWSTOE on an EC2 instance, then running AWSTOE uses the same permissions as the IAM role attached to the EC2 instance.

For more information about IAM roles for EC2, see IAM roles for Amazon EC2.

The following examples show how to set AWS credentials using the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

To set these variables on Linux, macOS, or Unix, use `export`.

```
$ export AWS_ACCESS_KEY_ID=your_access_key_id
```

```
$ export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

To set these variables on Windows using PowerShell, use `$env`.

```
C:\> $env:AWS_ACCESS_KEY_ID=your_access_key_id
```

```
C:\> $env:AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

To set these variables on Windows using the command prompt, use `set`.

```
C:\> set AWS_ACCESS_KEY_ID=your_access_key_id
```

```
C:\> set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Step 3: Develop component documents locally

AWSTOE components are authored with plaintext YAML documents. For more information about document syntax, see Use component documents in AWSTOE (p. 32).

The following are example Hello World component documents that you can use to develop your documents locally.

`hello-world-windows.yml`.

```
name: Hello World
description: This is Hello World testing document for Windows.
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host 'Hello World from the build phase.'
  - name: validate
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host 'Hello World from the validate phase.'
  - name: test
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host 'Hello World from the test phase.'
```

`hello-world-linux.yml`.

```
name: Hello World
description: This is hello world testing document for Linux.
```

```
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: HelloWorldStep
        action: ExecuteBash
        inputs:
          commands:
            - echo 'Hello World from the build phase.'
  - name: validate
    steps:
      - name: HelloWorldStep
        action: ExecuteBash
        inputs:
          commands:
            - echo 'Hello World from the validate phase.'
  - name: test
    steps:
      - name: HelloWorldStep
        action: ExecuteBash
        inputs:
          commands:
            - echo 'Hello World from the test phase.'
```

Step 4: Validate AWSTOE components

You can validate the syntax of AWSTOE components locally with the AWSTOE application. The following examples show the AWSTOE application `validate` command to validate the syntax of a component without running it.

Note
The AWSTOE application can validate only the component syntax for the current operating system. For example, when running `awstoe.exe` on Windows, you cannot validate the syntax for a Linux document that uses the `ExecuteBash` action module.

Windows

```
C:\> awstoe.exe validate --documents C:\Users\user\Documents\hello-world.yml
```

Linux

```
$ awstoe validate --documents /home/user/hello-world.yml
```

Step 5: Run AWSTOE components

The AWSTOE application can run one or more phases of specified documents using the `--phases` command line argument. Supported values for `--phases` are `build`, `validate`, and `test`. Multiple phase values can be entered as comma separated values.

When you provide a list of phases, the AWSTOE application sequentially runs the specified phases of each document. For example, AWSTOE runs the `build` and `validate` phases of `document1.yaml`, followed by the `build` and `validate` phases of `document2.yaml`.

To ensure that your logs are stored securely and retained for troubleshooting, we recommend configuring log storage in Amazon S3. In Image Builder, the Amazon S3 location for publishing logs is specified in the infrastructure configuration. For more information about infrastructure configuration, see Manage EC2 Image Builder infrastructure configuration (p. 131)

If a list of phases is not provided, the AWSTOE application runs all phases in the order listed in the YAML document.

To run specific phases in single or multiple documents, use the following commands.

Single phase

```
awstoe run --documents hello-world.yml --phases build
```

Multiple phases

```
awstoe run --documents hello-world.yml --phases build,test
```

Document run

Run all phases in a single document

```
awstoe run --documents documentName.yaml
```

Run all phases in multiple documents

```
awstoe run --documents documentName1.yaml,documentName2.yaml
```

Enter Amazon S3 information to upload AWSTOE logs from a user-defined local path (recommended)

```
awstoe run --documents documentName.yaml --log-s3-bucket-name <S3Bucket> --log-s3-key-
prefix <S3KeyPrefix> --log-s3-bucket-owner <S3BucketOwner> --log-directory <local_path>
```

Run all phases in a single document, and display all logs on the console

```
awstoe run --documents documentName.yaml --trace
```

Example command

```
awstoe run --documents s3://bucket/key/doc.yaml --phases build,validate
```

Run document with unique ID

```
awstoe run --documents <documentName>.yaml --execution-id <user provided id> --phases
 <comma separated list of phases>
```

Get help with AWSTOE

```
awstoe --help
```

Use component documents in AWSTOE

To build a component using AWS Task Orchestrator and Executor (AWSTOE), you must provide a YAML-based document that represents the phases and steps that apply for the component you create. AWS services use your component when they create a new Amazon Machine Image (AMI) or container image.

Topics

- Document example schemas (p. 38)
- Define and reference variables in AWSTOE (p. 41)
- Use looping constructs in AWSTOE (p. 45)

Component document workflow

The AWSTOE component document uses phases and steps to group related tasks, and organize those tasks into a logical workflow for the component. Each component can contain phases that run during any stage in the image build process.

> **Tip**
> The service that uses your component to build an image might implement rules about what phases to use for their build process. This is important to consider when you design your component.

Phases

Phases represent the progression of your workflow through the image build process. For example, the Image Builder service uses `build` and `validate` phases during its *build stage* for the images it produces. It uses the `test` phase during its *test stage* to ensure that the image snapshot or container image produces the expected results before creating the final AMI or distributing the container image.

When the component runs, the associated commands for each phase are applied in the order that they appear in the component document.

Rules for phases

- Each phase name must be unique within a document.
- You can define many phases in your document.
- You must include at least one of the following phases in your document:
  - **build** – generally used during the *build stage*.
  - **validate** – generally used during the *build stage*.
  - **test** – generally used during the *test stage*.
- Phases always run in the order that they are defined in the document. The order in which they are specified for AWSTOE commands in the AWS CLI has no effect.

Steps

Steps are individual units of work that define the workflow within each phase. Steps run in sequential order. However, input or output for one step can also feed into a subsequent step as input. This is called "chaining".

Rules for steps

- The step name must be unique for the phase.
- The step must use a supported action (action module) that returns an exit code.

  For a complete list of supported action modules, how they work, input/output values, and examples, see Action modules supported by AWSTOE component manager (p. 52).

Component logging

AWSTOE creates a new log folder on the EC2 instances that are used for building and testing a new image, each time your component runs. For container images, the log folder is stored in the container.

To assist with troubleshooting if something goes wrong during the image creation process, the input document and all of the output files AWSTOE creates while running the component are stored in the log folder.

The log folder name is comprised of the following parts:

1. **Log directory** – when a service runs a AWSTOE component, it passes in the log directory, along with other settings for the command. For the following examples, we show the log file format that Image Builder uses.
   - **Linux**: `/var/lib/amazon/toe/`
   - **Windows**: `$env:ProgramFiles\Amazon\TaskOrchestratorAndExecutor\`
2. **File prefix** – This is a standard prefix used for all components: "`TOE_`".
3. **Run time** – This is a timestamp in YYYY-MM-DD_HH-MM-SS_UTC-0 format.
4. **Execution ID** – This is the GUID that is assigned when AWSTOE runs one or more components.

Example: *`/var/lib/amazon/toe/`*`TOE_`*`2021-07-01_12-34-56_UTC-0_a1bcd2e3-45f6-789a-bcde-0fa1b2c3def4`*

AWSTOE stores the following core files in the log folder:

Input files

- **document.yaml** – The document that is used as input for the command. After the component runs, this file is stored as an artifact.

Output files

- **application.log** – The application log contains timestamped debug level information from AWSTOE about what's happening as the component is running.
- **detailedoutput.json** – This JSON file has detailed information about run status, inputs, outputs, and failures for all documents, phases, and steps that apply for the component as it runs.
- **console.log** – The console log contains all of the standard out (stdout) and standard error (stderr) information that AWSTOE writes to the console while the component is running.
- **chaining.json** – This JSON file represents optimizations that AWSTOE applied to resolve chaining expressions.

> **Note**
> The log folder might also contain other temporary files that are not covered here.

Input and output chaining

The AWSTOE configuration management application provides a feature for chaining inputs and outputs by writing references in the following formats:

```
{{ phase_name.step_name.inputs/outputs.variable }}
```

or

```
{{ phase_name.step_name.inputs/outputs[index].variable }}
```

The chaining feature allows you to recycle code and improve the maintainability of the document.

Rules for chaining

- Chaining expressions can be used only in the inputs section of each step.

- Statements with chaining expressions must be enclosed in quotes. For example:
  - **Invalid expression**: `echo {{ phase.step.inputs.variable }}`
  - **Valid expression**: `"echo {{ phase.step.inputs.variable }}"`
  - **Valid expression**: `'echo {{ phase.step.inputs.variable }}'`
- Chaining expressions can reference variables from other steps and phases in the same document. However, the calling service might have rules that require chaining expressions to operate only within the context of a single stage. For example, Image Builder does not support chaining from the *build stage* to the *test stage*, as it runs each stage independently.
- Indexes in chaining expressions follow zero-based indexing. The index starts with zero (0) to reference the first element.

Examples

To refer to the source variable in the second entry of the following example step, the chaining pattern is `{{ build.SampleS3Download.inputs[1].source }}`.

```
phases:
  -
    name: 'build'
    steps:
      -
        name: SampleS3Download
        action: S3Download
        timeoutSeconds: 60
        onFailure: Abort
        maxAttempts: 3
        inputs:
          -
            source: 's3://sample-bucket/sample1.ps1'
            destination: 'C:\sample1.ps1'
          -
            source: 's3://sample-bucket/sample2.ps1'
            destination: 'C:\sample2.ps1'
```

To refer to the output variable (equal to "Hello") of the following example step, the chaining pattern is `{{ build.SamplePowerShellStep.outputs.stdout }}`.

```
phases:
  -
    name: 'build'
    steps:
      -
        name: SamplePowerShellStep
        action: ExecutePowerShell
        timeoutSeconds: 120
        onFailure: Abort
        maxAttempts: 3
        inputs:
          commands:
            - 'Write-Host "Hello"'
```

Document schema and definitions

The following is the YAML schema for a document.

```
name: (optional)
description: (optional)
schemaVersion: "string"
```

```
phases:
  - name: "string"
    steps:
        - name: "string"
          action: "string"
          timeoutSeconds: integer
          onFailure: "Abort|Continue|Ignore"
          maxAttempts: integer
          inputs:
```

The schema definitions for a document are as follows.

Field	Description	Type	Required
name	Name of the document.	String	No
description	Description of the document.	String	No
schemaVersion	Schema version of the document, currently 1.0.	String	Yes
phases	A list of phases with their steps.	List	Yes

The schema definitions for a phase are as follows.

Field	Description	Type	Required
name	Name of the phase.	String	Yes
steps	List of the steps in the phase.	List	Yes

The schema definitions for a step are as follows.

Field	Description	Type	Required	Default value
name	User-defined name for the step.	String		
action	Keyword pertaining to the module that runs the step.	String		
timeoutSeconds	Number of seconds that the step runs before failing or retrying. Also, supports -1 value, which indicates infinite	Integer	Yes	7,200 sec (120 mins)

Field	Description	Type	Required	Default value
	timeout. 0 and other negative values are not allowed.			
onFailure	Specifies what the step should do in case of failure. Valid values are as follows: • **Abort** – Fails the step after the maximum number of attempts, and stops running. Sets status for phase and document to `Failed`. • **Continue** – Fails the step after the maximum number of attempts, and continues to run remaining steps. Sets status for phase and document to `Failed`. • **Ignore** – Sets the step to `IgnoredFailure` after the the maximum number of failed attempts, and continues to run remaining steps. Sets status for phase and document to `SuccessWithIgnoredFailure`.	String	Yes	Abort
maxAttempts	Maximum number of attempts allowed before failing the step.	Integer	No	1

Field	Description	Type	Required	Default value
inputs	Contains parameters required by the action module to run the step.	Dict	Yes	

Document example schemas

The following is an example document schema to install all available Windows updates, run a configuration script, validate the changes before the AMI is created, and test the changes after the AMI is created.

```
name: RunConfig_UpdateWindows
description: 'This document will install all available Windows updates and run a config
 script. It will then validate the changes before an AMI is created. Then after AMI
 creation, it will test all the changes.'
schemaVersion: 1.0
phases:
  -
    name: build
    steps:
      -
        name: DownloadConfigScript
        action: S3Download
        timeoutSeconds: 60
        onFailure: Abort
        maxAttempts: 3
        inputs:
          -
            source: 's3://customer-bucket/config.ps1'
            destination: 'C:\config.ps1'
      -
        name: RunConfigScript
        action: ExecutePowerShell
        timeoutSeconds: 120
        onFailure: Abort
        maxAttempts: 3
        inputs:
          file: '{{build.DownloadConfigScript.inputs[0].destination}}'
      -
        name: Cleanup
        action: DeleteFile
        onFailure: Abort
        maxAttempts: 3
        inputs:
          - path: '{{build.DownloadConfigScript.inputs[0].destination}}'
      -
        name: RebootAfterConfigApplied
        action: Reboot
        inputs:
          delaySeconds: 60
      -
        name: InstallWindowsUpdates
        action: UpdateOS
  -
    name: validate
    steps:
      -
        name: DownloadTestConfigScript
```

```
        action: S3Download
        timeoutSeconds: 60
        onFailure: Abort
        maxAttempts: 3
        inputs:
          -
            source: 's3://customer-bucket/testConfig.ps1'
            destination: 'C:\testConfig.ps1'
      -
        name: ValidateConfigScript
        action: ExecutePowerShell
        timeoutSeconds: 120
        onFailure: Abort
        maxAttempts: 3
        inputs:
          file: '{{validate.DownloadTestConfigScript.inputs[0].destination}}'
      -
        name: Cleanup
        action: DeleteFile
        onFailure: Abort
        maxAttempts: 3
        inputs:
          - path: '{{validate.DownloadTestConfigScript.inputs[0].destination}}'
  -
    name: test
    steps:
      -
        name: DownloadTestConfigScript
        action: S3Download
        timeoutSeconds: 60
        onFailure: Abort
        maxAttempts: 3
        inputs:
          -
            source: 's3://customer-bucket/testConfig.ps1'
            destination: 'C:\testConfig.ps1'
      -
        name: ValidateConfigScript
        action: ExecutePowerShell
        timeoutSeconds: 120
        onFailure: Abort
        maxAttempts: 3
        inputs:
          file: '{{test.DownloadTestConfigScript.inputs[0].destination}}'
```

The following is an example document schema to download and run a custom Linux binary file.

```
name: LinuxBin
description: Download and run a custom Linux binary file.
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: Download
        action: S3Download
        inputs:
          - source: s3://mybucket/myapplication
            destination: /tmp/myapplication
      - name: Enable
        action: ExecuteBash
        onFailure: Continue
        inputs:
          commands:
            - 'chmod u+x {{ build.Download.inputs[0].destination }}'
      - name: Install
```

```
        action: ExecuteBinary
        onFailure: Continue
        inputs:
          path: '{{ build.Download.inputs[0].destination }}'
          arguments:
            - '--install'
      - name: Delete
        action: DeleteFile
        inputs:
          - path: '{{ build.Download.inputs[0].destination }}'
```

The following is an example document schema to install the AWS CLI using the setup file.

```
name: InstallCLISetUp
description: Install AWS CLI using the setup file
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: Download
        action: S3Download
        inputs:
          - source: s3://aws-cli/AWSCLISetup.exe
            destination: C:\Windows\temp\AWSCLISetup.exe
      - name: Install
        action: ExecuteBinary
        onFailure: Continue
        inputs:
          path: '{{ build.Download.inputs[0].destination }}'
          arguments:
            - '/install'
            - '/quiet'
            - '/norestart'
      - name: Delete
        action: DeleteFile
        inputs:
          - path: '{{ build.Download.inputs[0].destination }}'
```

The following is an example document schema to install the AWS CLI using the MSI installer.

```
name: InstallCLIMSI
description: Install AWS CLI using the MSI installer
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: Download
        action: S3Download
        inputs:
          - source: s3://aws-cli/AWSCLI64PY3.msi
            destination: C:\Windows\temp\AWSCLI64PY3.msi
      - name: Install
        action: ExecuteBinary
        onFailure: Continue
        inputs:
          path: 'C:\Windows\System32\msiexec.exe'
          arguments:
            - '/i'
            - '{{ build.Download.inputs[0].destination }}'
            - '/quiet'
            - '/norestart'
      - name: Delete
        action: DeleteFile
        inputs:
```

```
        - path: '{{ build.Download.inputs[0].destination }}'
```

Define and reference variables in AWSTOE

Variables provide a way to label data with meaningful names that can be used throughout an application. You can define custom variables with simple and readable formats for complex workflows, and reference them in the YAML application component document for an AWSTOE component.

This section provides information to help you define variables for your AWSTOE component in the YAML application component document, including syntax, name constraints, and examples.

Parameters

Parameters are mutable variables, with settings that the calling application can provide at runtime. You can define parameters in the `Parameters` section of the YAML document.

Rules for parameter names

- The name must be between 3 and 128 characters in length.
- The name can only contain alphanumeric characters (a-z, A-Z, 0-9), dashes (-), or underscores (_).
- The name must be unique within the document.
- The name must be specified as a YAML string.

Syntax

```
parameters:
  - <name>:
      type: <parameter type>
      default: <parameter value>
      description: <parameter description>
```

Key name	Required	Description
`name`	Yes	The name of the parameter. Must be unique for the document (it must not be the same as any other parameter names or constants).
`type`	Yes	The data type of the parameter. Supported types include: `string`.
`default`	No	The default value for the parameter.
`description`	No	Describes the parameter.

Reference parameter values in a document

You can reference parameters in step or loop inputs inside of your YAML document, as follows:

- Parameter references are case-sensitive, and the name must match exactly.
- The name must be enclosed within double curly braces `{{ MyParameter }}`.

- Spaces are allowed within the curly braces, and are automatically trimmed. For example, all of the following references are valid:

  `{{ MyParameter }}`, `{{ MyParameter}}`, `{{MyParameter }}`, `{{MyParameter}}`
- The reference in the YAML document must be specified as a string (enclosed in single or double quotes).

  For example: – `{{ MyParameter }}` is not valid, as it is not identified as a string.

  However, the following references are both valid: – `'{{ MyParameter }}'` and – `"{{ MyParameter }}"`.

Examples

The following examples show how to use parameters in your YAML document:

- Refer to a parameter in step inputs:

```
name: Download AWS CLI version 2
schemaVersion: 1.0
parameters:
  - Source:
      type: string
      default: 'https://awscli.amazonaws.com/AWSCLIV2.msi'
      description: The AWS CLI installer source URL.
phases:
  - name: build
    steps:
      - name: Download
        action: WebDownload
        inputs:
          - source: '{{ Source }}'
            destination: 'C:\Windows\Temp\AWSCLIV2.msi'
```

- Refer to a parameter in loop inputs:

```
name: PingHosts
schemaVersion: 1.0
parameters:
  - Hosts:
      type: string
      default: 127.0.0.1,amazon.com
      description: A comma separated list of hosts to ping.
phases:
  - name: build
    steps:
      - name: Ping
        action: ExecuteBash
        loop:
          forEach:
            list: '{{ Hosts }}'
            delimiter: ','
        inputs:
          commands:
            - ping -c 4 {{ loop.value }}
```

Override parameters at runtime

You can use the `--parameters` option from the AWS CLI with a key-value pair to set a parameter value at runtime.

- Specify the parameter key-value pair as the name and value, separated by an equals sign (<name>=<value>).
- Multiple parameters must be separated by a comma.
- Parameter names that are not found in the YAML component document are ignored.
- The parameter name and value are both required.

Syntax

```
--parameters name1=value1,name12=value2...
```

CLI option	Required	Description
--parameters *name*=*value*,...	No	This option takes list of key-value pairs, with the parameter name as the key.

Examples

The following examples show how to use parameters in your YAML document:

- The parameter key-value pair specified in this `--parameter` option is not valid:

```
--parameters ntp-server=
```

- Set one parameter key-value pair with the `--parameter` option in the AWS CLI:

```
--parameters ntp-server=ntp-server-windows-qe.us-east1.amazon.com
```

- Set multiple parameter key-value pairs with the `--parameter` option in the AWS CLI:

```
--parameters ntp-server=ntp-server.amazon.com,http-url=https://internal-us-
east1.amazon.com
```

Constants

Constants are immutable variables that cannot be modified or overridden once defined. Constants can be defined using values in the `constants` section of an AWSTOE document.

Rules for constant names

- The name must be between 3 and 128 characters in length.
- The name can only contain alphanumeric characters (a-z, A-Z, 0-9), dashes (-), or underscores (_).
- The name must be unique within the document.
- The name must be specified as a YAML string.

Syntax

```
constants:
  - <name>:
      type: <constant type>
      value: <constant value>
```

Key name	Required	Description
`name`	Yes	Name of the constant. Must be unique for the document (it must not be the same as any other parameter names or constants).
`value`	Yes	Value of the constant.
`type`	Yes	Type of the constant. Supported type is `string`.

Reference constant values in a document

You can reference constants in step or loop inputs inside of your YAML document, as follows:

- Constant references are case-sensitive, and the name must match exactly.
- The name must be enclosed within double curly braces `{{ MyConstant }}`.
- Spaces are allowed within the curly braces, and are automatically trimmed. For example, all of the following references are valid:

  `{{ MyConstant }}`, `{{ MyConstant}}`, `{{MyConstant }}`, `{{MyConstant}}`
- The reference in the YAML document must be specified as a string (enclosed in single or double quotes).

  For example: – `{{ MyConstant }}` is not valid, as it is not identified as a string.

  However, the following references are both valid: – `'{{ MyConstant }}'` and – `"{{ MyConstant }}"`.

Examples

Constant referenced in step inputs

```
name: Download AWS CLI version 2
schemaVersion: 1.0
constants:
  - Source:
      type: string
      value: https://awscli.amazonaws.com/AWSCLIV2.msi
phases:
  - name: build
    steps:
      - name: Download
        action: WebDownload
        inputs:
          - source: '{{ Source }}'
            destination: 'C:\Windows\Temp\AWSCLIV2.msi'
```

Constant referenced in loop inputs

```
name: PingHosts
schemaVersion: 1.0
constants:
  - Hosts:
      type: string
```

```
      value: 127.0.0.1,amazon.com
phases:
  - name: build
    steps:
      - name: Ping
        action: ExecuteBash
        loop:
          forEach:
            list: '{{ Hosts }}'
            delimiter: ','
        inputs:
          commands:
            - ping -c 4 {{ loop.value }}
```

Use looping constructs in AWSTOE

This section provides information to help you create looping constructs in the AWSTOE. Looping constructs define a repeated sequence of instructions. You can use the following types of looping constructs in AWSTOE:

- `for` constructs – Iterate over a bounded sequence of integers.
- `forEach` constructs
  - `forEach` loop with input list – Iterates over a finite collection of strings.
  - `forEach` loop with delimited list – Iterates over a finite collection of strings joined by a delimiter.

Note
Looping constructs support only string data types.

Looping construct topics

Reference iteration variables

To refer to the index and value of the current iteration variable, the reference expression `{{ loop.* }}` must be used within the input body of a step that contains a looping construct. This expression cannot be used to refer to the iteration variables of the looping construct of another step.

The reference expression consists of the following members:

- `{{ loop.index }}` – The ordinal position of the current iteration, which is indexed at `0`.
- `{{ loop.value }}` – The value associated with the current iteration variable.

Loop names

All looping constructs have an optional name field for identification. If a loop name is provided, it can be used to refer to iteration variables in the input body of the step. To refer to the iteration indices and values of a named loop, use `{{ <loop_name>.* }}` with `{{ loop.* }}` in the input body of the step. This expression cannot be used to refer to the named looping construct of another step.

The reference expression consists of the following members:

- `{{ <loop_name>.index }}` – The ordinal position of the current iteration of the named loop, which is indexed at `0`.
- `{{ <loop_name>.value }}` – The value associated with the current iteration variable of the named loop.

Resolve reference expressions

The AWSTOE resolves reference expressions as follows:

- `{{ <loop_name>.* }}` – AWSTOE resolves this expression using the following logic:
  - If the loop of the currently running step matches the `<loop_name>` value, then the reference expression resolves to the looping construct of the currently running step.
  - `<loop_name>` resolves to the named looping construct if it appears within the currently running step.
- `{{ loop.* }}` – AWSTOE resolves the expression using the looping construct defined in the currently running step.

If reference expressions are used within a step that does not contain a loop, then AWSTOE does not resolve the expressions and they appear in the step with no replacement.

> **Note**
> Reference expressions must be enclosed in double quotes to be correctly interpreted by the YAML compiler.

Types of looping constructs

This section provides information and examples about looping construct types that can be used in the AWSTOE.

Looping construct types

- for loop (p. 46)
- forEach loop with input list (p. 47)
- forEach loop with delimited list (p. 49)

`for` loop

The `for` loop iterates on a range of integers specified within a boundary outlined by the start and end of the variables. The iterating values are in the set `[start, end]` and includes boundary values.

AWSTOE verifies the `start`, `end`, and `updateBy` values to ensure that the combination does not result in an infinite loop.

`for` loop schema

```
name: "StepName"
action: "ActionModule"
loop:
  name: "string"
  for:
    start: int
    end: int
    updateBy: int
inputs:
  ...
```

`for` loop input

Field	Description	Type	Required	Default
`name`	Unique name of the loop. It must be unique compared to other loop names in the same phase.	String	No	""
`start`	Starting value of iteration. Does not accept chaining expressions.	Integer	Yes	n/a
`end`	Ending value of iteration. Does not accept chaining expressions.	Integer	Yes	n/a
`updateBy`	Difference by which an iterating value is updated through addition. It must be a negative or positive non-zero value. Does not accept chaining expressions.	Integer	Yes	n/a

`for` loop input example

```
name: "CalculateFileUploadLatencies"
action: "ExecutePowerShell"
loop:
  for:
    start: 100000
    end: 1000000
    updateBy: 100000
inputs:
  commands:
    - |
      $f = new-object System.IO.FileStream c:\temp\test{{ loop.index }}.txt, Create,
 ReadWrite
      $f.SetLength({{ loop.value }}MB)
      $f.Close()
    - c:\users\administrator\downloads\latencyTest.exe --file c:\temp
\test{{ loop.index }}.txt
    - AWS s3 cp c:\users\administrator\downloads\latencyMetrics.json s3://bucket/
latencyMetrics.json
    - |
      Remove-Item -Path c:\temp\test{{ loop.index }}.txt
      Remove-Item -Path c:\users\administrator\downloads\latencyMetrics.json
```

`forEach` loop with input list

The `forEach` loop iterates on an explicit list of values, which can be strings and chained expressions.

`forEach` loop with input list schema

```
name: "StepName"
action: "ActionModule"
loop:
  name: "string"
  forEach:
    - "string"
inputs:
  ...
```

`forEach` loop with input list input

Field	Description	Type	Required	Default
`name`	Unique name of the loop. It must be unique compared to other loop names in the same phase.	String	No	""
List of strings of `forEach` loop	List of strings for iteration. Accepts chained expressions as strings in the list. Chained expressions must be enclosed by double quotes for the YAML compiler to correctly interpret them.	List of strings	Yes	n/a

`forEach` loop with input list example 1

```
name: "ExecuteCustomScripts"
action: "ExecuteBash"
loop:
  name: BatchExecLoop
  forEach:
    - /tmp/script1.sh
    - /tmp/script2.sh
    - /tmp/script3.sh
inputs:
  commands:
    - echo "Count {{ BatchExecLoop.index }}"
    - sh "{{ loop.value }}"
    - |
      retVal=$?
      if [ $retVal -ne 0 ]; then
        echo "Failed"
      else
        echo "Passed"
      fi
```

`forEach` loop with input list example 2

```
name: "RunMSIWithDifferentArgs"
action: "ExecuteBinary"
loop:
  name: MultiArgLoop
  forEach:
    - "ARG1=C:\Users ARG2=1"
    - "ARG1=C:\Users"
    - "ARG1=C:\Users ARG3=C:\Users\Administrator\Documents\f1.txt"
inputs:
  commands:
    path: "c:\users\administrator\downloads\runner.exe"
    args:
      - "{{ MultiArgLoop.value }}"
```

`forEach` loop with input list example 3

```
name: "DownloadAllBinaries"
action: "S3Download"
loop:
  name: MultiArgLoop
  forEach:
    - "bin1.exe"
    - "bin10.exe"
    - "bin5.exe"
inputs:
  -
    source: "s3://bucket/{{ loop.value }}"
    destination: "c:\temp\{{ loop.value }}"
```

`forEach` loop with delimited list

The loop iterates over a string containing values separated by a delimiter. To iterate over the string's constituents, AWSTOE uses the delimiter to split the string into an array suitable for iteration.

`forEach` loop with delimited list schema

```
name: "StepName"
action: "ActionModule"
loop:
  name: "string"
  forEach:
    list: "string"
    delimiter: ".,;:\n\t -_"
inputs:
  ...
```

`forEach` loop with delimited list input

Field	Description	Type	Required	Default
`name`	Unique name given to the loop. It should be unique when compared to other loop names in the same phase.	String	No	""
`list`	A string that is composed	String	Yes	n/a

Field	Description	Type	Required	Default
	of constituent strings joined by a common delimiter character. Also accepts chained expressions. In case of chained expressions, ensure that those are enclosed by double quotes for correct interpretation by the YAML compiler.			
`delimiter`	Character used to separate out strings within a block. Default is the comma character. Only one delimiter character is allowed from the given list: • Dot: `"."` • Comma: `","` • Semicolon: `";"` • Colon: `":"` • New line: `"\n"` • Tab: `"\t"` • Space: `" "` • Hyphen: `"-"` • Underscore: `"_"` Chaining expressions cannot be used.	String	No	Comma: `","`

Note
The value of `list` is treated as an immutable string. If the source of `list` is changed during runtime, it will not be reflected during the run.

`forEach` loop with delimited list example 1

```
// Uses chaning expression ({{ <phase_name>.<step_name>.inputs/outputs.<var_name> }})
// to refer to another step's input/output variables for code re-use.
name: "RunMSIs"
action: "ExecuteBinary"
loop:
  forEach:
```

```
    list: "{{ build.GetAllMSIPathsForInstallation.outputs.stdout }}"
    delimiter: "\n"
inputs:
  commands:
    path: "{{ loop.value }}"
```

`forEach` loop with delimited list example 2

```
name: "UploadMetricFiles"
action: "S3Upload"
loop:
  forEach:
    list: "/tmp/m1.txt,/tmp/m2.txt,/tmp/m3.txt,..."
inputs:
  commands:
    -
      source: "{{ loop.value }}"
      destination: "s3://bucket/key/{{ loop.value }}"
```

Step fields

Loops are part of a step. Any field related to the running of a step is not applied to individual iterations. Step fields apply only at the step level, as follows:

- *timeoutSeconds* – All iterations of the loop must be run within the time period specified by this field. If the loop run times out, then AWSTOE runs the retry policy of the step and resets the timeout parameter for each new attempt. If the loop run exceeds the timeout value after reaching the maximum number of retries, the failure message of the step states that the loop run had timed out.
- *onFailure* – Failure handling is applied to the step as follows:
  - If *onFailure* is set to `Abort`, AWSTOE exits the loop and retries the step according to the retry policy. After the maximum number of retry attempts, AWSTOE marks the current step as failed, and stops running the process.

    AWSTOE sets the status code for the parent phase and document to `Failed`.

    Note
    No further steps run after the failed step.
  - If *onFailure* is set to `Continue`, AWSTOE exits the loop and retries the step according to the retry policy. After the maximum number of retry attempts, AWSTOE marks the current step as failed, and continues on to run the next step.

    AWSTOE sets the status code for the parent phase and document to `Failed`.
  - If *onFailure* is set to `Ignore`, AWSTOE exits the loop and retries the step according to the retry policy. After the maximum number of retry attempts, AWSTOE marks the current step as `IgnoredFailure`, and continues on to run the next step.

    AWSTOE sets the status code for the parent phase and document to `SuccessWithIgnoredFailure`.

    Note
    This is still considered a successful run, but includes information to let you know that one or more steps failed and were ignored.
- *maxAttempts* – For every retry, the entire step and all iterations are run from the beginning.
- *status* – The overall status of the running of a step.`status` does not represent the status of individual iterations. The status of a step with loops is determined as follows:
  - If a single iteration fails to run, the status of a step points to a failure.
  - If all iterations succeed, the status of a step points to a success.

- *startTime* – The overall start time of the running of a step. Does not represent the start time of individual iterations.
- *endTime* – The overall end time of the running of a step. Does not represent the end time of individual iterations.
- *failureMessage* – Includes the iteration indices that failed in case of non-timeout errors. In case of timeout errors, the message states that the loop run has failed. Individual error messages for each iteration are not provided to minimize the size of failure messages.

Step and iteration outputs

Every iteration contains an output. At the end of a loop run, AWSTOE consolidates all successful iteration outputs in `detailedOutput.json`. The consolidated outputs are a collation of values that belong to the corresponding output keys as defined in the output schema of the action module. The following example shows how the outputs are consolidated:

Output of `ExecuteBash` for Iteration 1

```
[{"stdout":"Hello"}]
```

Output of `ExecuteBash` for Iteration 2

```
[{"stdout":"World"}]
```

Output of `ExecuteBash` for Step

```
[{"stdout":"Hello\nWorld"}]
```

For example, `ExecuteBash`, `ExecutePowerShell`, and `ExecuteBinary` are action modules which return `STDOUT` as the action module output. `STDOUT` messages are joined with the new line character to produce the overall output of the step in `detailedOutput.json`.

AWSTOE will not consolidate the outputs of unsuccessful iterations.

Action modules supported by AWSTOE component manager

This section contains each action module that is supported by the AWSTOE component management application used by EC2 Image Builder to configure the instance that builds your image. Also included are the corresponding functionality details and input/output values of each action module.

AWSTOE components are authored with plaintext YAML documents. For more information about document syntax, see Use component documents in AWSTOE (p. 32).

> **Note**
> All action modules are run by using the same account as the Systems Manager agent, which is `root` on Linux and `NT Authority\SYSTEM` on Windows.

Action module types

- General execution modules (p. 53)

- File download and upload modules (p. 56)
- File system operation modules (p. 65)
- System action modules (p. 92)

General execution modules

The following section contains details for action modules that perform general execution commands and instructions.

General execution action modules

- ExecuteBash (p. 53)
- ExecuteBinary (p. 54)
- ExecutePowerShell (p. 55)

ExecuteBash

The **ExecuteBash** action module allows you to run bash scripts with inline shell code/commands. This module supports Linux.

All of the commands and instructions that you specify in the commands block are converted into a file (for example, `input.sh`) and run with the bash shell. The result of running the shell file is the exit code of the step.

The **ExecuteBash** module handles system restarts if the script exits with an exit code of `194`. When initiated, the application performs one of the following actions:

- The application hands the exit code to the caller if it is run by the Systems Manager Agent. The Systems Manager Agent handles the system reboot and runs the same step that initiated the restart, as described in Rebooting Managed Instance from Scripts.
- The application saves the current `executionstate`, configures a restart trigger to rerun the application, and restarts the system.

After system restart, the application runs the same step that initiated the restart. If you require this functionality, you must write idempotent scripts that can handle multiple invocations of the same shell command.

Input

Primitive	Description	Type	Required
`commands`	Contains a list of instructions or commands to run as per bash syntax. Multi-line YAML is allowed.	List	Yes

Input example: install and validate Corretto

```
name: InstallAndValidateCorretto
action: ExecuteBash
inputs:
```

```
commands:
  - sudo yum install java-11-amazon-corretto-headless -y
  - |
    function fail_with_message() {
        1>&2 echo $1
        exit 1
    }

    ARCH=`/usr/bin/arch`

    JAVA_PATH=/usr/lib/jvm/java-11-amazon-corretto.$ARCH/bin/java
    if [ -x $JAVA_PATH ]; then
        echo "Amazon Corretto 11 JRE is installed."
    else
        fail_with_message "Amazon Corretto 11 JRE is not installed. Failing."
    fi

    JAVAC_PATH=/usr/lib/jvm/java-11-amazon-corretto.$ARCH/bin/javac
    if [ -x $JAVAC_PATH ]; then
        echo "Amazon Corretto 11 JDK is installed."
    else
        fail_with_message "Amazon Corretto 11 JDK is not installed. Failing."
    fi
```

Output

Field	Description	Type
`stdout`	Standard output of command execution.	string

Output example

```
{
    “stdout”: “This is the standard output from running the shell\n"
}
```

If you start a reboot and return exit code `194` as part of the action module, the build will resume at the same action module step that initiated the reboot. If you start a reboot without the exit code, the build process may fail.

ExecuteBinary

The **ExecuteBinary** action module allows you to run binary files with a list of command-line arguments.

The **ExecuteBinary** module handles system restarts if the binary file exits with an exit code of `194` (Linux) or `3010` (Windows). When this happens, the application performs one of the following actions:

- The application hands the exit code to the caller if it is run by the Systems Manager Agent. The Systems Manager Agent handles restarting the system and runs the same step that initiated the restart, as described in Rebooting Managed Instance from Scripts.
- The application saves the current `executionstate`, configures a restart trigger to rerun the application, and restarts the system.

After the system restarts, the application runs the same step that initiated the restart. If you require this functionality, you must write idempotent scripts that can handle multiple invocations of the same shell command.

Input

Primitive	Description	Type	Required
`path`	The path to the binary file for execution.	String	Yes
`arguments`	Contains a list of command-line arguments to use when running the binary.	String List	No

Input example: install .NET

```
name: "InstallDotnet"
action: ExecuteBinary
inputs:
  path: C:\PathTo\dotnet_installer.exe
  arguments:
    - /qb
    - /norestart
```

Output

Field	Description	Type
`stdout`	Standard output of command execution.	string

Output example

```
{
      "stdout": "success"
}
```

ExecutePowerShell

The **ExecutePowerShell** action module allows you to run PowerShell scripts with inline shell code/commands. This module supports the Windows platform and Windows PowerShell.

All of the commands/instructions specified in the commands block are converted into a script file (for example, `input.ps1`) and run using Windows PowerShell. The result of running the shell file is the exit code.

The **ExecutePowerShell** module handles system restarts if the shell command exits with an exit code of `3010`. When initiated, the application performs one of the following actions:

- Hands the exit code to the caller if run by the Systems Manager Agent. The Systems Manager Agent handles the system reboot and runs the same step that initiated the restart, as described in Rebooting Managed Instance from Scripts.
- Saves the current `executionstate`, configures a restart trigger to rerun the application, and reboots the system.

After system restart, the application runs the same step that initiated the restart. If you require this functionality, you must write idempotent scripts that can handle multiple invocations of the same shell command.

Input

Primitive	Description	Type	Required
`commands`	Contains a list of instructions or commands to run as per PowerShell syntax. Multi-line YAML is allowed.	String List	Yes. Must specify `commands` or `file`, not both.
`file`	Contains the path to a PowerShell script file. PowerShell will run against this file using the `-file` command line argument. The path must point to a `.ps1` file.	String	Yes. Must specify `commands` or `file`, not both.

Input example: install software using PowerShell commands

```
name: InstallMySoftware
action: ExecutePowerShell
inputs:
  commands:
      - Set-SomeConfiguration -Value 10
      - Write-Host 'Successfully set the configuration.'

name: ExecuteMyScript
action: ExecutePowerShell
inputs:
  file: 'C:\PathTo\MyScript.ps1'
```

Output

Field	Description	Type
`stdout`	Standard output of command execution.	string

Output example

```
{
    “stdout”: “This is the standard output from the shell execution\n"
}
```

If you run a reboot and return exit code `3010` as part of the action module, the build will resume at the same action module step that initiated the reboot. If you run a reboot without the exit code, the build process may fail.

File download and upload modules

The following section contains details for action modules that perform download and upload commands and instructions.

Download and upload action modules

S3Download

With the `S3Download` action module, you can download an Amazon S3 object, or a set of objects, to a local file or folder that you specify with the `destination` path. If any file already exists in the specified location, and the `overwrite` flag is set to true, `S3Download` overwrites the file.

Your `source` location can point to a specific object in Amazon S3, or you can use a key prefix with an asterisk wildcard (`*`) to download a set of objects that match the key prefix path. When you specify a key prefix in your `source` location, the `S3Download` action module downloads everything that matches the prefix (files and folders included). Make sure that the key prefix ends with a forward-slash, followed by an asterisk (`/*`), so that you download everything that matches the prefix. For example: *`s3://my-bucket/my-folder/*`*.

> **Note**
> All folders in the destination path must exist prior to download, or the download fails.

If the `S3Download` action for a specified key prefix fails during a download, the folder content is not rolled back to its state prior to the failure. The destination folder remains as it was at the time of the failure.

Supported use cases

The `S3Download` action module supports the following use cases:

- The Amazon S3 object is downloaded to a local folder, as specified in the download path.
- Amazon S3 objects (with a key prefix in the Amazon S3 file path) are downloaded to the specified local folder, which recursively copies all Amazon S3 objects that match the key prefix to the local folder.

IAM requirements

The IAM role that you associate with your instance profile must have permissions to run the `S3Download` action module. The following IAM policies must be attached to the IAM role that is associated with the instance profile:

- **Single file**: `s3:GetObject` against the bucket/object (for example, `arn:aws:s3:::`**`BucketName`**`/*`).
- **Multiple files**: `s3:ListBucket` against the bucket/object (for example, `arn:aws:s3:::`**`BucketName`**) and `s3:GetObject` against the bucket/object (for example, `arn:aws:s3:::`**`BucketName`**`/*`).

Input

Primitive	Description	Type	Required	Default
`source`	The Amazon S3 bucket that is the source for your download. You can specify a path to a specific object, or use a key prefix, that ends with a forward-slash, followed by an	String	Yes	N/A

Primitive	Description	Type	Required	Default
	asterisk wildcard (`/*`), to download a set of objects that match the key prefix.			
`destination`	The local path where the Amazon S3 objects are downloaded.	String	Yes	N/A
`expectedBucketOwner`	Expected owner account ID of the bucket provided in the `source` path. We recommend that you verify the ownership of the Amazon S3 bucket specified in the source.	String	No	N/A
`overwrite`	When set to true, if a file of the same name already exists in the destination folder for the specified local path, the download file overwrites the local file. When set to false, the existing file on the local system is protected from being overwritten, and the action module fails with a download error. For example, `Error: S3Download: File already exists and "overwrite" property for "destination" file is set to false. Cannot download.`	Boolean	No	true

Note
For the following examples, the Windows folder path can be replaced with a Linux path. For example, *C:\myfolder\package.zip* can be replaced with */myfolder/package.zip*.

Input example: copy an Amazon S3 object to a local file

The following example shows how to copy an Amazon S3 object to a local file.

```
name: DownloadMyFile
action: S3Download
inputs:
    - source: s3://mybucket/path/to/package.zip
      destination: C:\myfolder\package.zip
      expectedBucketOwner: 123456789022
      overwrite: false
    - source: s3://mybucket/path/to/package.zip
      destination: C:\myfolder\package.zip
      expectedBucketOwner: 123456789022
      overwrite: true
    - source: s3://mybucket/path/to/package.zip
      destination: C:\myfolder\package.zip
      expectedBucketOwner: 123456789022
```

Input example: copy all Amazon S3 objects in an Amazon S3 bucket with key prefix to a local folder

The following example shows how to copy all Amazon S3 objects in an Amazon S3 bucket with the key prefix to a local folder. Amazon S3 has no concept of a folder, therefore all objects that match the key prefix are copied. The maximum number of objects that can be downloaded is 1000.

```
name: MyS3DownloadKeyprefix
action: S3Download
maxAttempts: 3
inputs:
    - source: s3://mybucket/path/to/*
      destination: C:\myfolder\
      expectedBucketOwner: 123456789022
      overwrite: false
    - source: s3://mybucket/path/to/*
      destination: C:\myfolder\
      expectedBucketOwner: 123456789022
      overwrite: true
    - source: s3://mybucket/path/to/*
      destination: C:\myfolder\
      expectedBucketOwner: 123456789022
```

Output

None.

S3Upload

With the **S3Upload** action module, you can upload a file from a source file or folder to an Amazon S3 location. You can use a wildcard (*) in the path specified for your source location to upload all of the files whose path matches the wildcard pattern.

If the recursive **S3Upload** action fails, any files that have already been uploaded will remain in the destination Amazon S3 bucket.

Supported use cases

- Local file to Amazon S3 object.

- Local files in folder (with wildcard) to Amazon S3 key prefix.
- Copy local folder (must have `recurse` set to `true`) to Amazon S3 key prefix.

IAM requirements

The IAM role that you associate with your instance profile must have permissions to run the `S3Upload` action module. The following IAM policy must be attached to the IAM role that is associated with the instance profile. The policy must grant `s3:PutObject` permissions to the target Amazon S3 bucket. For example, `arn:aws:s3:::BucketName/*`).

Input

Primitive	Description	Type	Required	Default
`source`	The local path where source files/folders originate. The `source` supports an asterisk wildcard (`*`).	String	Yes	N/A
`destination`	The path for the destination Amazon S3 bucket where source files/folders are uploaded.	String	Yes	N/A
`recurse`	When set to `true`, performs **S3Upload** recursively.	String	No	`false`
`expectedBucketOwner`	The expected owner account ID for the Amazon S3 bucket specified in the destination path. We recommend that you verify the ownership of the Amazon S3 bucket specified in the destination.	String	No	N/A

Input example: copy a local file to an Amazon S3 object

The following example shows how to copy a local file to an Amazon S3 object.

```
name: MyS3UploadFile
action: S3Upload
onFailure: Abort
maxAttempts: 3
inputs:
    - source: C:\myfolder\package.zip
      destination: s3://mybucket/path/to/package.zip
```

```
      expectedBucketOwner: 123456789022
```

Input example: copy all files in a local folder to an Amazon S3 bucket with key prefix

The following example shows how to copy all files in the local folder to an Amazon S3 bucket with key prefix. This example does not copy sub-folders or their contents because `recurse` is not specified, and it defaults to `false`.

```
name: MyS3UploadMultipleFiles
action: S3Upload
onFailure: Abort
maxAttempts: 3
inputs:
    - source: C:\myfolder\*
      destination: s3://mybucket/path/to/
      expectedBucketOwner: 123456789022
```

Input example: copy all files and folders recursively from a local folder to an Amazon S3 bucket

The following example shows how to copy all files and folders recursively from a local folder to an Amazon S3 bucket with key prefix.

```
name: MyS3UploadFolder
action: S3Upload
onFailure: Abort
maxAttempts: 3
inputs:
    - source: C:\myfolder\*
      destination: s3://mybucket/path/to/
      recurse: true
      expectedBucketOwner: 123456789022
```

Output

None.

WebDownload

The **WebDownload** action module allows you to download files and resources from a remote location over the HTTP/HTTPS protocol (*HTTPS is recommended*). There are no limits on the number or size of downloads. This module handles retry and exponential backoff logic.

Each download operation is allocated a maximum of 5 attempts to succeed according to user inputs. These attempts differ from those specified in the `maxAttempts` field of document `steps`, which are related to action module failures.

This action module implicitly handles redirects. All HTTP status codes, except for `200`, result in an error.

Input

Primitive	Description	Type	Required	Default
`source`	The valid HTTP/HTTPS URL (*HTTPS is recommended*), which follows the RFC 3986 standard. Chaining	String	Yes	N/A

Primitive	Description	Type	Required	Default
	expressions are permitted.			
`destination`	An absolute or relative file or folder path on the local system. Folder paths must end with `/`. If they do not end with `/`, they will be treated as file paths. The module creates any required file or folder for successful downloads. Chaining expressions are permitted.	String	Yes	N/A
`overwrite`	When enabled, overwrites any existing files on the local system with the downloaded file or resource. When not enabled, any existing files on the local system are not overwritten, and the action module fails with an error. When overwrite is enabled and checksum and algorithm are specified, then the action module downloads the file only if the checksum and the hash of any pre-existing files do not match.	Boolean	No	`true`

Primitive	Description	Type	Required	Default
`checksum`	When you specify the checksum, it is checked against the hash of the downloaded file that is generated with the supplied algorithm. For file verification to be enabled, both the checksum and the algorithm must be provided. Chaining expressions are permitted.	String	No	N/A
`algorithm`	The algorithm used to calculate the checksum. The options are MD5, SHA1, SHA256, and SHA512. For file verification to be enabled, both the checksum and the algorithm must be provided. Chaining expressions are permitted.	String	No	N/A
`ignoreCertificateErrors`	SSL certificate validation is ignored when enabled.	Boolean	No	`false`

Output

Primitive	Description	Type				
`destination`	Newline character-delimited string that specifies the destination path where the downloaded files or resources are stored.	String				

Input example: download remote file to local destination

```
name: DownloadRemoteFile
action: WebDownload
maxAttempts: 3
inputs:
  - source: https://testdomain/path/to/java14.zip
    destination: C:\testfolder\package.zip


Output:
{
  "destination": "C:\\testfolder\\package.zip"
}
```

Input example: download more than one remote file to more than one local destination

```
name: DownloadRemoteFiles
action: WebDownload
maxAttempts: 3
inputs:
  - source: https://testdomain/path/to/java14.zip
    destination: /tmp/java14_renamed.zip
  - source: https://testdomain/path/to/java14.zip
    destination: /tmp/create_new_folder_and_add_java14_as_zip/


Output:
{
  "destination": "/tmp/create_new_folder/java14_renamed.zip\n/tmp/
create_new_folder_and_add_java14_as_zip/java14.zip"
}
```

Input example: download one remote file without overwriting local destination, and download another remote file with file verification

```
name: DownloadRemoteMultipleProperties
action: WebDownload
maxAttempts: 3
inputs:
  - source: https://testdomain/path/to/java14.zip
    destination: C:\create_new_folder\java14_renamed.zip
    overwrite: false
  - source: https://testdomain/path/to/java14.zip
    destination: C:\create_new_folder_and_add_java14_as_zip\
    checksum: ac68bbf921d953d1cfab916cb6120864
    algorithm: MD5
    overwrite: true


Output:
{
  "destination": "C:\\create_new_folder\\java14_renamed.zip\nC:\
\create_new_folder_and_add_java14_as_zip\\java14.zip"
}
```

Input example: download remote file and ignore SSL certification validation

```
name: DownloadRemoteIgnoreValidation
action: WebDownload
maxAttempts: 3
```

```
inputs:
  - source: https://www.bad-ssl.com/resource
    destination: /tmp/downloads/
    ignoreCertificateErrors: true


Output:
{
  "destination": "/tmp/downloads/resource"
}
```

File system operation modules

The following section contains details for action modules that perform file system operation commands and instructions.

File system operation action modules

AppendFile

The **AppendFile** action module adds specified content to the preexisting content of a file.

If the file encoding value is different from the default encoding (`utf-8`) value, then you can specify the file encoding value by using the `encoding` option. By default, `utf-16` and `utf-32` are assumed to use little-endian encoding.

The action module returns an error when the following occurs:

- The specified file does not exist at runtime.
- You don't have write permissions to modify the file content.
- The module encounters an error during the file operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes
`content`	The content to be appended to the file.	String	No	Empty string	N/A	Yes
`encoding`	The encoding standard.	String	No	`utf8`	`utf8`, `utf-8`, `utf16`,`utf-16`, `utf16-LE`, `utf-16-LE` `utf16-BE`, `utf-16-BE`, `utf32`, `utf-32`, `utf32-LE`,`utf-32-LE`, `utf32-BE`, and `utf-32-BE`. The value of the encoding option is case insensitive.	Yes

Input example: append file without encoding (Linux)

```
name: AppendingFileWithOutEncodingLinux
action: AppendFile
inputs:
  - path: ./Sample.txt
    content: "The string to be appended to the file"
```

Input example: append file without encoding (Windows)

```
name: AppendingFileWithOutEncodingWindows
action: AppendFile
inputs:
  - path: C:\MyFolder\MyFile.txt
    content: "The string to be appended to the file"
```

Input example: append file with encoding (Linux)

```
name: AppendingFileWithEncodingLinux
action: AppendFile
inputs:
  - path: /FolderName/SampleFile.txt
```

```
    content: "The string to be appended to the file"
    encoding: UTF-32
```

Input example: append file with encoding (Windows)

```
name: AppendingFileWithEncodingWindows
action: AppendFile
inputs:
  - path: C:\MyFolderName\SampleFile.txt
    content: "The string to be appended to the file"
    encoding: UTF-32
```

Input example: append file with empty string (Linux)

```
name: AppendingEmptyStringLinux
action: AppendFile
inputs:
  - path: /FolderName/SampleFile.txt
```

Input example: append file with empty string (Windows)

```
name: AppendingEmptyStringWindows
action: AppendFile
inputs:
  - path: C:\MyFolderName\SampleFile.txt
```

Output

None.

CopyFile

The **CopyFile** action module copies files from the specified source to the specified destination. By default, the module recursively creates the destination folder if it does not exist at runtime.

If a file with the specified name already exists in the specified folder, the action module, by default, overwrites the existing file. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a file in the specified location with the specified name, the action module will return an error. This option works the same as the `cp` command in Linux, which overwrites by default.

The source file name can include a wildcard (`*`). Wildcard characters are accepted only after the last file path separator (`/` or `\`). If wildcard characters are included in the source file name, all of the files that match the wildcard are copied to the destination folder. If you want to move more than one file by using a wildcard character, the input to the `destination` option must end with a file path separator (`/` or `\`), which indicates that the destination input is a folder.

If the destination file name is different from the source file name, you can specify the destination file name using the `destination` option. If you do not specify a destination file name, the name of the source file is used to create the destination file. Any text that follows the last file path separator (`/` or `\`) is treated as the file name. If you want to use the same file name as the source file, then the input of the `destination` option must end with a file path separator (`/` or `\`).

The action module returns an error when the following occurs:

- You do not have permission to create a file in the specified folder.
- The source files do not exist at runtime.
- There is already a folder with the specified file name and the `overwrite` option is set to `false`.

- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`source`	The source file path.	String	Yes	N/A	N/A	Yes
`destination`	The destination file path.	String	Yes	N/A	N/A	Yes
`overwrite`	When set to false, the destination files will not be replaced when there is already a file in the specified location with the specified name.	Boolean	No	`true`	N/A	Yes

Input example: copy a file (Linux)

```
name: CopyingAFileLinux
action: CopyFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/destinationFile.txt
```

Input example: copy a file (Windows)

```
name: CopyingAFileWindows
action: CopyFile
inputs:
  - source: C:\MyFolder\Sample.txt
    destination: C:\MyFolder\destinationFile.txt
```

Input example: copy a file using the source file name (Linux)

```
name: CopyingFileWithSourceFileNameLinux
action: CopyFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/
```

Input example: copy a file using the source file name (Windows)

```
name: CopyingFileWithSourceFileNameWindows
action: CopyFile
```

```
inputs:
  - source: C:\Sample\MyFolder\Sample.txt
    destination: C:\MyFolder\
```

Input example: copy a file using the wildcard character (Linux)

```
name: CopyingFilesWithWildCardLinux
action: CopyFile
inputs:
  - source: /Sample/MyFolder/Sample*
    destination: /MyFolder/
```

Input example: copy a file using the wildcard character (Windows)

```
name: CopyingFilesWithWildCardWindows
action: CopyFile
inputs:
  - source: C:\Sample\MyFolder\Sample*
    destination: C:\MyFolder\
```

Input example: copy a file without overwriting (Linux)

```
name: CopyingFilesWithoutOverwriteLinux
action: CopyFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/destinationFile.txt
    overwrite: false
```

Input example: copy a file without overwriting (Windows)

```
name: CopyingFilesWithoutOverwriteWindows
action: CopyFile
inputs:
  - source: C:\Sample\MyFolder\Sample.txt
    destination: C:\MyFolder\destinationFile.txt
    overwrite: false
```

Output

None.

CopyFolder

The **CopyFolder** action module copies a folder from the specified source to the specified destination. The input for the `source` option is the folder to copy, and the input for the `destination` option is the folder where the contents of the source folder are copied. By default, the module recursively creates the destination folder if it does not exist at runtime.

If a folder with the specified name already exists in the specified folder, the action module, by default, overwrites the existing folder. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a folder in the specified location with the specified name, the action module will return an error.

The source folder name can include a wildcard (`*`). Wildcard characters are accepted only after the last file path separator (`/` or `\`). If wildcard characters are included in the source folder name, all of the folders that match the wildcard are copied to the destination folder. If you want to copy more than one folder by using a wildcard character, the input to the `destination` option must end with a file path separator (`/` or `\`), which indicates that the destination input is a folder.

If the destination folder name is different from the source folder name, you can specify the destination folder name using the `destination` option. If you do not specify a destination folder name, the name of the source folder is used to create the destination folder. Any text that follows the last file path separator (`/` or `\`) is treated as the folder name. If you want to use the same folder name as the source folder, then the input of the `destination` option must end with a file path separator (`/` or `\`).

The action module returns an error when the following occurs:

- You do not have permission to create a folder in the specified folder.
- The source folders do not exist at runtime.
- There is already a folder with the specified folder name and the `overwrite` option is set to `false`.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`source`	The source folder path.	String	Yes	N/A	N/A	Yes
`destination`	The destination folder path.	String	Yes	N/A	N/A	Yes
`overwrite`	When set to false, the destination folders will not be replaced when there is already a folder in the specified location with the specified name.	Boolean	No	`true`	N/A	Yes

Input example: copy a folder (Linux)

```
name: CopyingAFolderLinux
action: CopyFolder
inputs:
  - source: /Sample/MyFolder/SampleFolder
    destination: /MyFolder/destinationFolder
```

Input example: copy a folder (Windows)

```
name: CopyingAFolderWindows
action: CopyFolder
inputs:
  - source: C:\Sample\MyFolder\SampleFolder
    destination: C:\MyFolder\destinationFolder
```

Input example: copy a folder using the source folder name (Linux)

```
name: CopyingFolderSourceFolderNameLinux
action: CopyFolder
inputs:
  - source: /Sample/MyFolder/SourceFolder
    destination: /MyFolder/
```

Input example: copy a folder using the source folder name (Windows)

```
name: CopyingFolderSourceFolderNameWindows
action: CopyFolder
inputs:
  - source: C:\Sample\MyFolder\SampleFolder
    destination: C:\MyFolder\
```

Input example: copy a folder using the wildcard character (Linux)

```
name: CopyingFoldersWithWildCardLinux
action: CopyFolder
inputs:
  - source: /Sample/MyFolder/Sample*
    destination: /MyFolder/
```

Input example: copy a folder using the wildcard character (Windows)

```
name: CopyingFoldersWithWildCardWindows
action: CopyFolder
inputs:
  - source: C:\Sample\MyFolder\Sample*
    destination: C:\MyFolder\
```

Input example: copy a folder without overwriting (Linux)

```
name: CopyingFoldersWithoutOverwriteLinux
action: CopyFolder
inputs:
  - source: /Sample/MyFolder/SourceFolder
    destination: /MyFolder/destinationFolder
    overwrite: false
```

Input example: copy a folder without overwriting (Windows)

```
name: CopyingFoldersWithoutOverwrite
action: CopyFolder
inputs:
  - source: C:\Sample\MyFolder\SourceFolder
    destination: C:\MyFolder\destinationFolder
    overwrite: false
```

Output

None.

CreateFile

The **CreateFile** action module creates a file in a specified location. By default, if required, the module also recursively creates the parent folders.

If the file already exists in the specified folder, the action module, by default, truncates or overwrites the existing file. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a file in the specified location with the specified name, the action module will return an error.

If the file encoding value is different from the default encoding (`utf-8`) value, then you can specify the file encoding value by using the `encoding` option. By default, `utf-16` and `utf-32` are assumed to use little-endian encoding.

`owner`, `group`, and `permissions` are optional inputs. The input for `permissions` must be a string value. Files are created with default values when not provided. These options are not supported on Windows platforms. This action module validates and returns an error if the `owner`, `group`, and `permissions` options are used on Windows platforms.

This action module can create a file the with permissions defined by the default `umask` value of the operating system. You must set the `umask` value if you want to override the default value.

The action module returns an error when the following occurs:

- You do not have permission to create a file or a folder in the specified parent folder.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes
`content`	The content of the file.	String	No	N/A	N/A	Yes
`encoding`	The encoding standard.	String	No	`utf8`	`utf8`, `utf-8`, `utf16`,`utf-16`, `utf16-LE`, `utf-16-LE` `utf16-BE`, `utf-16-BE`, `utf32`, `utf-32`, `utf32-LE`,`utf-32-LE`, `utf32-BE`, and `utf-32-BE`. The value of the encoding option is case insensitive.	Yes
`owner`	The user name or ID.	String	No	N/A	N/A	Not supported on Windows.

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`group`	The group name or ID.	String	No	The current user.	N/A	Not supported on Windows.
`permissions`	The file permissions.	String	No	`0666`	N/A	Not supported on Windows.
`overwrite`	If the name of the specified file already exists, setting this value to `false` prevents the file from being truncated or overwritten by default.	Boolean	No	`true`	N/A	Yes

Input example: create a file without overwriting (Linux)

```
name: CreatingFileWithoutOverwriteLinux
action: CreateFile
inputs:
  - path: /home/UserName/Sample.txt
    content: ABCD\nRandom\tvalues
    overwrite: false
```

Input example: create a file without overwriting (Windows)

```
name: CreatingFileWithoutOverwriteWindows
action: CreateFile
inputs:
  - path: C:\Temp\Sample.txt
    content: ABCD\nRandom\tvalues
    overwrite: false
```

Input example: create a file with file properties

```
name: CreatingFileWithFileProperties
action: CreateFile
inputs:
  - path: SampleFolder/Sample.txt
    content: ABCD\nRandom\tvalues
    encoding: UTF-16
    owner: Ubuntu
    group: UbuntuGroup
    permissions: 0777
 - path: SampleFolder/SampleFile.txt
    permissions: 755
```

```
  - path: SampleFolder/TextFile.txt
    encoding: UTF-16
    owner: root
    group: rootUserGroup
```

Input example: create a file without file properties

```
name: CreatingFileWithoutFileProperties
action: CreateFile
inputs:
  - path: ./Sample.txt
  - path: Sample1.txt
```

Output

None.

CreateFolder

The **CreateFolder** action module creates a folder in a specified location. By default, if required, the module also recursively creates the parent folders.

If the folder already exists in the specified folder, the action module, by default, truncates or overwrites the existing folder. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a folder in the specified location with the specified name, the action module will return an error.

`owner`, `group`, and `permissions` are optional inputs. The input for `permissions` must be a string value. These options are not supported on Windows platforms. This action module validates and returns an error if the `owner`, `group`, and `permissions` options are used on Windows platforms.

This action module can create a folder the with permissions defined by the default `umask` value of the operating system. You must set the `umask` value if you want to override the default value.

The action module returns an error when the following occurs:

- You do not have permission to create a folder in the specified location.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The folder path.	String	Yes	N/A	N/A	Yes
`owner`	The user name or ID.	String	No	The current user.	N/A	Not supported on Windows.
`group`	The group name or ID.	String	No	The group of the current user.	N/A	Not supported on Windows.
`permissions`	The folder permissions.	String	No	`0777`	N/A	Not supported on Windows.

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`overwrite`	If the name of the specified file already exists, setting this value to `false` prevents the file from being truncated or overwritten by default.	Boolean	No	`true`	N/A	Yes

Input example: create a folder (Linux)

```
name: CreatingFolderLinux
action: CreateFolder
inputs:
  - path: /Sample/MyFolder/
```

Input example: create a folder (Windows)

```
name: CreatingFolderWindows
action: CreateFolder
inputs:
  - path: C:\MyFolder
```

Input example: create a folder specifying folder properties

```
name: CreatingFolderWithFolderProperties
action: CreateFolder
inputs:
  - path: /Sample/MyFolder/Sample/
    owner: SampleOwnerName
    group: SampleGroupName
    permissions: 0777
  - path: /Sample/MyFolder/SampleFoler/
    permissions: 777
```

Input example: create a folder that overwrites the existing folder, if there is one.

```
name: CreatingFolderWithOverwrite
action: CreateFolder
inputs:
  - path: /Sample/MyFolder/Sample/
    overwrite: true
```

Output

None.

CreateSymlink

The **CreateSymlink** action module creates symbolic links, or files that contain a reference to another file. This module is not supported on Windows platforms.

The input for the `path` and `target` options can be either an absolute or relative path. If the input for the `path` option is a relative path, it is replaced with the absolute path when the link is created.

By default, when a link with the specified name already exists in the specified folder, the action module returns an error. You can override this default behavior by setting the `force` option to `true`. When the `force` option is set to `true`, the module will overwrite the existing link.

If a parent folder does not exist, the action module creates the folder recursively, by default.

The action module returns an error when the following occurs:

- The target file does not exist at runtime.
- A nonsymbolic link file with the specified name already exists.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Not supported on Windows.
`target`	The target file path to which the symbolic link points.	String	Yes	N/A	N/A	Not supported on Windows.
`force`	Forces the creation of a link when a link with the same name already exists.	Boolean	No	`false`	N/A	Not supported on Windows.

Input example: create symbolic link that forces the creation of a link

```
name: CreatingSymbolicLinkWithForce
action: CreateSymlink
inputs:
  - path: /Folder2/Symboliclink.txt
    target: /Folder/Sample.txt
    force: true
```

Input example: create a symbolic link that does not force the creation of a link

```
name: CreatingSymbolicLinkWithOutForce
action: CreateSymlink
```

```
inputs:
  - path: Symboliclink.txt
    target: /Folder/Sample.txt
```

Output

None.

DeleteFile

The **DeleteFile** action module deletes a file or files in a specified location.

The input of `path` should be a valid file path or a file path with a wild card character (`*`) in the file name. When wildcard characters are specified in the file name, all of the files within the same folder that match the wildcard will be deleted.

The action module returns an error when the following occurs:

- You do not have permission to perform delete operations.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes

Input example: delete a single file (Linux)

```
name: DeletingSingleFileLinux
action: DeleteFile
inputs:
  - path: /SampleFolder/MyFolder/Sample.txt
```

Input example: delete a single file (Windows)

```
name: DeletingSingleFileWindows
action: DeleteFile
inputs:
  - path: C:\SampleFolder\MyFolder\Sample.txt
```

Input example: delete a file that ends with "log" (Linux)

```
name: DeletingFileEndingWithLogLinux
action: DeleteFile
inputs:
  - path: /SampleFolder/MyFolder/*log
```

Input example: delete a file that ends with "log" (Windows)

```
name: DeletingFileEndingWithLogWindows
action: DeleteFile
inputs:
```

```
  - path: C:\SampleFolder\MyFolder\*log
```

Input example: delete all files in a specified folder (Linux)

```
name: DeletingAllFilesInAFolderLinux
action: DeleteFile
inputs:
  - path: /SampleFolder/MyFolder/*
```

Input example: delete all files in a specified folder (Windows)

```
name: DeletingAllFilesInAFolderWindows
action: DeleteFile
inputs:
  - path: C:\SampleFolder\MyFolder\*
```

Output

None.

DeleteFolder

The **DeleteFolder** action module deletes folders.

If the folder is not empty, you must set the `force` option to `true` to remove the folder and its contents. If you do not set the `force` option to `true`, and the folder you are trying to delete is not empty, the action module returns an error. The default value of the `force` option is `false`.

The action module returns an error when the following occurs:

- You do not have permission to perform delete operations.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The folder path.	String	Yes	N/A	N/A	Yes
`force`	Removes the folder whether or not the folder is empty.	Boolean	No	`false`	N/A	Yes

Input example: delete a folder that is not empty using the `force` option (Linux)

```
name: DeletingFolderWithForceOptionLinux
action: DeleteFolder
inputs:
  - path: /Sample/MyFolder/Sample/
    force: true
```

Input example: delete a folder that is not empty using the `force` option (Windows)

```
name: DeletingFolderWithForceOptionWindows
action: DeleteFolder
inputs:
  - path: C:\Sample\MyFolder\Sample\
    force: true
```

Input example: delete a folder (Linux)

```
name: DeletingFolderWithOutForceLinux
action: DeleteFolder
inputs:
  - path: /Sample/MyFolder/Sample/
```

Input example: delete a folder (Windows)

```
name: DeletingFolderWithOutForce
action: DeleteFolder
inputs:
  - path: C:\Sample\MyFolder\Sample\
```

Output

None.

ListFiles

The **ListFiles** action module lists the files in a specified folder. When the recursive option is set to `true`, it lists the files in subfolders. This module does not list files in subfolders by default.

To list all of the files with names that match a specified pattern, use the `fileNamePattern` option to provide the pattern. The `fileNamePattern` option accepts the wildcard (`*`) value. When the `fileNamePattern` is provided, all of the files that match the specified file name format are returned.

The action module returns an error when the following occurs:

- The specified folder does not exist at runtime.
- You do not have permission to create a file or a folder in the specified parent folder.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The folder path.	String	Yes	N/A	N/A	Yes
`fileNamePattern`	The pattern to match to list all files with names that match the pattern.	String	No	N/A	N/A	Yes

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`recursive`	Lists files in the folder recursively.	Boolean	No	`false`	N/A	Yes

Input example: list files in specified folder (Linux)

```
name: ListingFilesInSampleFolderLinux
action: ListFiles
inputs:
  - path: /Sample/MyFolder/Sample
```

Input example: list files in specified folder (Windows)

```
name: ListingFilesInSampleFolderWindows
action: ListFiles
inputs:
  - path: C:\Sample\MyFolder\Sample
```

Input example: list files that end with "log" (Linux)

```
name: ListingFilesWithEndingWithLogLinux
action: ListFiles
inputs:
  - path: /Sample/MyFolder/
    fileNamePattern: *log
```

Input example: list files that end with "log" (Windows)

```
name: ListingFilesWithEndingWithLogWindows
action: ListFiles
inputs:
  - path: C:\Sample\MyFolder\
    fileNamePattern: *log
```

Input example: list files recursively

```
name: ListingFilesRecursively
action: ListFiles
inputs:
  - path: /Sample/MyFolder/
    recursive: true
```

Output

Primitive	Description	Type				
`files`	The list of files.	String				

Output example

```
{
  "files": "/sample1.txt,/sample2.txt,/sample3.txt"
}
```

MoveFile

The **MoveFile** action module moves files from the specified source to the specified destination.

If the file already exists in the specified folder, the action module, by default, overwrites the existing file. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a file in the specified location with the specified name, the action module will return an error. This option works the same as the `mv` command in Linux, which overwrites by default.

The source file name can include a wildcard (`*`). Wildcard characters are accepted only after the last file path separator (`/` or `\`). If wildcard characters are included in the source file name, all of the files that match the wildcard are copied to the destination folder. If you want to move more than one file by using a wildcard character, the input to the `destination` option must end with a file path separator (`/` or `\`), which indicates that the destination input is a folder.

If the destination file name is different from the source file name, you can specify the destination file name using the `destination` option. If you do not specify a destination file name, the name of the source file is used to create the destination file. Any text that follows the last file path separator (`/` or `\`) is treated as the file name. If you want to use the same file name as the source file, then the input of the `destination` option must end with a file path separator (`/` or `\`).

The action module returns an error when the following occurs:

- You do not have permission to create a file in the specified folder.
- The source files do not exist at runtime.
- There is already a folder with the specified file name and the `overwrite` option is set to `false`.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`source`	The source file path.	String	Yes	N/A	N/A	Yes
`destination`	The destination file path.	String	Yes	N/A	N/A	Yes
`overwrite`	When set to false, the destination files will not be replaced when there is already a file in the specified	Boolean	No	`true`	N/A	Yes

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
	location with the specified name.					

Input example: move a file (Linux)

```
name: MovingAFileLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/destinationFile.txt
```

Input example: move a file (Windows)

```
name: MovingAFileWindows
action: MoveFile
inputs:
  - source: C:\Sample\MyFolder\Sample.txt
    destination: C:\MyFolder\destinationFile.txt
```

Input example: move a file using the source file name (Linux)

```
name: MovingFileWithSourceFileNameLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/
```

Input example: move a file using the source file name (Windows)

```
name: MovingFileWithSourceFileNameWindows
action: MoveFile
inputs:
  - source: C:\Sample\MyFolder\Sample.txt
    destination: C:\MyFolder
```

Input example: move a file using a wildcard character (Linux)

```
name: MovingFilesWithWildCardLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample*
    destination: /MyFolder/
```

Input example: move a file using a wildcard character (Windows)

```
name: MovingFilesWithWildCardWindows
action: MoveFile
inputs:
  - source: C:\Sample\MyFolder\Sample*
```

```
    destination: C:\MyFolder
```

Input example: move a file without overwriting (Linux)

```
name: MovingFilesWithoutOverwriteLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
    destination: /MyFolder/destinationFile.txt
    overwrite: false
```

Input example: move a file without overwriting (Windows)

```
name: MovingFilesWithoutOverwrite
action: MoveFile
inputs:
  - source: C:\Sample\MyFolder\Sample.txt
    destination: C:\MyFolder\destinationFile.txt
    overwrite: false
```

Output

None.

MoveFolder

The **MoveFolder** action module moves folders from the specified source to the specified destination. The input for the `source` option is the folder to move, and the input to the `destination` option is the folder where the contents of the source folders are moved.

If the destination parent folder or the input to the `destination` option does not exist at runtime, the default behavior of the module is to recursively create the folder at the specified destination.

If a folder with the same as the source folder already exists in the destination folder, the action module, by default, overwrites the existing folder. You can override this default behavior by setting the overwrite option to `false`. When the overwrite option is set to `false`, and there is already a folder in the specified location with the specified name, the action module will return an error.

The source folder name can include a wildcard (`*`). Wildcard characters are accepted only after the last file path separator (`/` or `\`). If wildcard characters are included in the source folder name, all of the folders that match the wildcard are copied to the destination folder. If you want to move more than one folder by using a wildcard character, the input to the `destination` option must end with a file path separator (`/` or `\`), which indicates that the destination input is a folder.

If the destination folder name is different from the source folder name, you can specify the destination folder name using the `destination` option. If you do not specify a destination folder name, the name of the source folder is used to create the destination folder. Any text that follows the last file path separator (`/` or `\`) is treated as the folder name. If you want to use the same folder name as the source folder, then the input of the `destination` option must end with a file path separator (`/` or `\`).

The action module returns an error when the following occurs:

- You do not have permission to create a folder in the destination folder.
- The source folders do not exist at runtime.
- There is already a folder with the specified name and the `overwrite` option is set to `false`.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`source`	The source folder path.	String	Yes	N/A	N/A	Yes
`destination`	The destination folder path.	String	Yes	N/A	N/A	Yes
`overwrite`	When set to false, the destination folders will not be replaced when there is already a folder in the specified location with the specified name.	Boolean	No	`true`	N/A	Yes

Input example: move a folder (Linux)

```
name: MovingAFolderLinux
action: MoveFolder
inputs:
  - source: /Sample/MyFolder/SourceFolder
    destination: /MyFolder/destinationFolder
```

Input example: move a folder (Windows)

```
name: MovingAFolderWindows
action: MoveFolder
inputs:
  - source: C:\Sample\MyFolder\SourceFolder
    destination: C:\MyFolder\destinationFolder
```

Input example: move a folder using the source folder name (Linux)

```
name: MovingFolderWithSourceFolderNameLinux
action: MoveFolder
inputs:
  - source: /Sample/MyFolder/SampleFolder
    destination: /MyFolder/
```

Input example: move a folder using the source folder name (Windows)

```
name: MovingFolderWithSourceFolderNameWindows
action: MoveFolder
inputs:
  - source: C:\Sample\MyFolder\SampleFolder
```

```
    destination: C:\MyFolder\
```

Input example: move a folder using a wildcard character (Linux)

```
name: MovingFoldersWithWildCardLinux
action: MoveFolder
inputs:
  - source: /Sample/MyFolder/Sample*
    destination: /MyFolder/
```

Input example: move a folder using a wildcard character (Windows)

```
name: MovingFoldersWithWildCardWindows
action: MoveFolder
inputs:
  - source: C:\Sample\MyFolder\Sample*
    destination: C:\MyFolder\
```

Input example: move a folder without overwriting (Linux)

```
name: MovingFoldersWithoutOverwriteLinux
action: MoveFolder
inputs:
  - source: /Sample/MyFolder/SampleFolder
    destination: /MyFolder/destinationFolder
    overwrite: false
```

Input example: move a folder without overwriting (Windows)

```
name: MovingFoldersWithoutOverwriteWindows
action: MoveFolder
inputs:
  - source: C:\Sample\MyFolder\SampleFolder
    destination: C:\MyFolder\destinationFolder
    overwrite: false
```

Output

None.

ReadFile

The **ReadFile** action module reads the content of a text file of type string. This module can be used to read the content of a file for use in subsequent steps through chaining or for reading data to the `console.log` file. If the specified path is a symbolic link, this module returns the content of the target file. This module only supports text files.

If the file encoding value is different from the default encoding (`utf-8`) value, then you can specify the file encoding value by using the `encoding` option. By default, `utf-16` and `utf-32` are assumed to use little-endian encoding.

By default, this module cannot print the file content to the `console.log` file. You can override this setting by setting the `printFileContent` property to `true`.

This module can return only the content of a file. It cannot parse files, such as Excel or JSON files.

The action module returns an error when the following occurs:

- The file does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes
`encoding`	The encoding standard.	String	No	`utf8`	`utf8`, `utf-8`, `utf16`,`utf-16`, `utf16-LE`, `utf-16-LE` `utf16-BE`, `utf-16-BE`, `utf32`, `utf-32`, `utf32-LE`,`utf-32-LE`, `utf32-BE`, and `utf-32-BE`. The value of the encoding option is case insensitive.	Yes
`printFileContents`	Prints the file content to the `console.log` file.	Boolean	No	false	N/A	Yes.

Input example: read a file (Linux)

```
name: ReadingFileLinux
action: ReadFile
inputs:
  - path: /home/UserName/SampleFile.txt
```

Input example: read a file (Windows)

```
name: ReadingFileWindows
action: ReadFile
inputs:
  - path: C:\Windows\WindowsUpdate.log
```

Input example: read a file and specify encoding standard

```
name: ReadingFileWithFileEncoding
action: ReadFile
inputs:
  - path: /FolderName/SampleFile.txt
    encoding: UTF-32
```

Input example: read a file and print to the `console.log` file

```
name: ReadingFileToConsole
action: ReadFile
inputs:
  - path: /home/UserName/SampleFile.txt
    printFileContent: true
```

Output

Field	Description	Type
`content`	The file content.	string

Output example

```
{
    "content" : "The file content"
}
```

SetFileEncoding

The **SetFileEncoding** action module modifies the encoding property of an existing file. This module can convert file encoding from `utf-8` to a specified encoding standard. By default, `utf-16` and `utf-32` are assumed to be little-endian encoding.

The action module returns an error when the following occurs:

- You do not have permission to perform the specified modification.
- The file does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes
`encoding`	The encoding standard.	String	No	`utf8`	`utf8`, `utf-8`, `utf16`,`utf-16`, `utf16-LE`, `utf-16-LE` `utf16-BE`, `utf-16-BE`, `utf32`, `utf-32`, `utf32-LE`,`utf-32-LE`, `utf32-BE`, and `utf-32-`	Yes

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
					`BE`. The value of the encoding option is case insensitive.	

Input example: set file encoding property

```
name: SettingFileEncodingProperty
action: SetFileEncoding
inputs:
  - path: /home/UserName/SampleFile.txt
    encoding: UTF-16
```

Output

None.

SetFileOwner

The **SetFileOwner** action module modifies the `owner` and`group` owner properties of an existing file. If the specified file is a symbolic link, the module modifies the `owner` property of the source file. This module is not supported on Windows platforms.

This module accepts user and group names as inputs. If the group name is not provided, the module assigns the group owner of the file to the group that the user belongs to.

The action module returns an error when the following occurs:

- You do not have permission to perform the specified modification.
- The specified user or group name does not exist at runtime.
- The file does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Not supported on Windows.
`owner`	The user name.	string	Yes	N/A	N/A	Not supported on Windows.
`group`	The name of the user group.	String	No	The name of the group that the user belongs to.	N/A	Not supported on Windows.

Input example: set file owner property without specifying the name of the user group

```
name: SettingFileOwnerPropertyNoGroup
action: SetFileOwner
inputs:
  - path: /home/UserName/SampleText.txt
    owner: LinuxUser
```

Input example: set file owner property by specifying the owner and the user group

```
name: SettingFileOwnerProperty
action: SetFileOwner
inputs:
  - path: /home/UserName/SampleText.txt
    owner: LinuxUser
    group: LinuxUserGroup
```

Output

None.

SetFolderOwner

The **SetFolderOwner** action module recursively modifies the `owner` and`group` owner properties of an existing folder. By default, the module can modify ownership for all of the contents in a folder. You can set the `recursive` option to `false` to override this behavior. This module is not supported on Windows platforms.

This module accepts user and group names as inputs. If the group name is not provided, the module assigns the group owner of the file to the group that the user belongs to.

The action module returns an error when the following occurs:

- You do not have permission to perform the specified modification.
- The specified user or group name does not exist at runtime.
- The folder does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The folder path.	String	Yes	N/A	N/A	Not supported on Windows.
`owner`	The user name.	string	Yes	N/A	N/A	Not supported on Windows.
`group`	The name of the user group.	String	No	The name of the group that the user belongs to.	N/A	Not supported on Windows.

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`recursive`	Overrides the default behavior of modifying ownership for all of the contents of a folder when set to `false`.	Boolean	No	`true`	N/A	Not supported on Windows.

Input example: set folder owner property without specifying the name of the user group

```
name: SettingFolderPropertyWithOutGroup
action: SetFolderOwner
inputs:
  - path: /SampleFolder/
    owner: LinuxUser
```

Input example: set folder owner property without overriding the ownership of all of the contents in a folder

```
name: SettingFolderPropertyWithOutRecursively
action: SetFolderOwner
inputs:
  - path: /SampleFolder/
    owner: LinuxUser
    recursive: false
```

Input example: set file ownership property by specifying the name of the user group

```
name: SettingFolderPropertyWithGroup
action: SetFolderOwner
inputs:
  - path: /SampleFolder/
    owner: LinuxUser
    group: LinuxUserGroup
```

Output

None.

SetFilePermissions

The **SetFilePermissions** action module modifies the `permissions` of an existing file. This module is not supported on Windows platforms.

The input for `permissions` must be a string value.

This action module can create a file the with permissions defined by the default umask value of the operating system. You must set the `umask` value if you want to override the default value.

The action module returns an error when the following occurs:

- You do not have permission to perform the specified modification.
- The file does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Not supported on Windows.
`permissions`	The file permissions.	String	Yes	N/A	N/A	Not supported on Windows.

Input example: modify file permissions

```
name: ModifyingFilePermissions
action: SetFilePermissions
inputs:
  - path: /home/UserName/SampleFile.txt
    permissions: 766
```

Output

None.

SetFolderPermissions

The **SetFolderPermissions** action module recursively modifies the `permissions` of an existing folder and all of its subfiles and subfolders. By default, this module can modify permissions for all of the contents of the specified folder. You can set the `recursive` option to `false` to override this behavior. This module is not supported on Windows platforms.

The input for `permissions` must be a string value.

This action module can modify permissions according to the default umask value of the operating system. You must set the `umask` value if you want to override the default value.

The action module returns an error when the following occurs:

- You do not have permission to perform the specified modification.
- The folder does not exist at runtime.
- The action module encounters an error while performing the operation.

Input

Primitive	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The folder path.	String	Yes	N/A	N/A	Not supported on Windows.
`permissions`	The folder permissions.	String	Yes	N/A	N/A	Not supported on Windows.
`recursive`	Overrides the default behavior of modifying permissions for all of the contents of a folder when set to `false`.	Boolean	No	`true`	N/A	Not supported on Windows.

Input example: set folder permissions

```
name: SettingFolderPermissions
action: SetFolderPermissions
inputs:
  - path: SampleFolder/
    permissions: 0777
```

Input example: set folder permissions without modifying permissions for all of the contents of a folder

```
name: SettingFolderPermissionsNoRecursive
action: SetFolderPermissions
inputs:
  - path: /home/UserName/SampleFolder/
    permissions: 777
    recursive: false
```

Output

None.

System action modules

The following section contains details for action modules that perform file system action commands and instructions.

System action modules

Reboot

The **Reboot** action module reboots the instance. It has a configurable option to delay the start of the reboot. It does not support the step timeout value due to the instance getting rebooted. Default behavior is that `delaySeconds` is `0`, which means that there is no delay.

If the application is invoked by the Systems Manager Agent, it hands the exit code (`3010` for Windows, `194` for Linux) to the Systems Manager Agent. The Systems Manager Agent handles the system reboot as described in Rebooting Managed Instance from Scripts.

If the application is invoked on the host as a standalone process, it saves the current execution state, configures a post reboot auto-run trigger to re-execute the application, and then reboots the system.

Post-reboot auto-run trigger:

- **Windows**. Create a Task Scheduler entry with trigger At `SystemStartup`
- **Linux**. Add a job in crontab.

```
@reboot /download/path/awstoe run --document s3://bucket/key/doc.yaml
```

This trigger is cleaned up when the application starts.

To use the **Reboot** action module, for steps that contain reboot `exitcode` (for example, `3010`), you must run the application binary as `sudo user`.

Input

Primitive	Description	Type	Required	Default
`delaySeconds`	Delays a specific amount of time before initiating a reboot.	Integer	No	`0`

Input example: reboot step

```
name: RebootStep
action: Reboot
onFailure: Abort
maxAttempts: 2
inputs:
    delaySeconds: 60
```

Output

None.

When the **Reboot** module completes, Image Builder continues to the next step in the build.

SetRegistry

The **SetRegistry** action module accepts a list of inputs and allows you to set the value for the specified registry key. If a registry key does not exist, it is created in the defined path. This feature applies only to Windows.

Input

Primitive	Description	Type	Required
`path`	Path of registry key.	String	Yes
`name`	Name of registry key.	String	Yes
`value`	Value of registry key.	String/Number/Array	Yes
`type`	Value type of registry key.	String	Yes

Supported path prefixes

- `HKEY_CLASSES_ROOT / HKCR:`
- `HKEY_USERS / HKU:`
- `HKEY_LOCAL_MACHINE / HKLM:`
- `HKEY_CURRENT_CONFIG / HKCC:`
- `HKEY_CURRENT_USER / HKCU:`

Supported types

- `BINARY`
- `DWORD`
- `QWORD`
- `SZ`
- `EXPAND_SZ`
- `MULTI_SZ`

Input example: set registry key values

```
name: SetRegistryKeyValues
action: SetRegistry
maxAttempts: 3
inputs:
    - path: HKLM:\SOFTWARE\MySoftWare
      name: MyName
      value: FirstVersionSoftware
      type: SZ
    - path: HKEY_CURRENT_USER\Software\Test
      name: Version
      value: 1.1
      type: DWORD
```

Output

None.

UpdateOS

The **UpdateOS** action module adds support for installing Windows and Linux updates.

The **UpdateOS** action module installs all available updates by default. You can override this action by providing a list of one or more updates to include for installation and/or a list of one or more updates to exclude from installation.

If both “include" and "exclude" lists are provided, the resulting list of updates can include only those listed in the "include" list that are not listed in the "exclude" list.

- **Windows**. Updates are installed from the update source configured on the target machine.
- **Linux**. The application checks for the supported package manager in the Linux platform and uses either `yum` or `apt-get` package manager. If neither are supported, an error is returned. You should have `sudo` permissions to run the **UpdateOS** action module. If you do not have `sudo` permissions an `error.Input` is returned.

Input

Primitive	Description	Type	Required
`include`	For Windows, you can specify the following: • One or more Microsoft Knowledge Base (KB) article IDs to include in the list of updates that may be installed. Valid formats are `KB1234567` or `1234567`. • An update name using a wildcard value (`*`). Valid formats are `Security*` or `*Security*`. For Linux, you can specify one or more packages to be included in the list of updates for installation.	String List	No
`exclude`	For Windows, you can specify the following: • One or more Microsoft Knowledge Base (KB) article IDs to include in the list of updates to be excluded from the installation. Valid formats are `KB1234567` or `1234567`.	String List	No

Primitive	Description	Type	Required
	• An update name using a wildcard (*) value. Valid formats are: `Security*` or `*Security*`. For Linux, you can specify one or more packages to be excluded from the list of updates for installation.		

Input example: add support for installing Linux updates

```
name: UpdateMyLinux
action: UpdateOS
onFailure: Abort
maxAttempts: 3
inputs:
    exclude:
        - ec2-hibinit-agent
```

Input example: add support for installing Windows updates

```
name: UpdateWindowsOperatingSystem
action: UpdateOS
onFailure: Abort
maxAttempts: 3
inputs:
  include:
    - KB1234567
    - '*Security*'
```

Output

None.

EC2 Image Builder STIG components

Security Technical Implementation Guides (STIGs) are the configuration standards created by the Defense Information Systems Agency (DISA) to secure information systems and software. To make your systems compliant with STIG standards, you must install, configure, and test a variety of security settings.

AWS Task Orchestrator and Executor provides STIG components to help you more efficiently build compliant images for STIG standards. These STIG components scan for misconfigurations and run a remediation script. There are no additional charges for using STIG-compliant components.

Compliance levels

- **High (Category I)**

The most severe risk. Includes any vulnerability that can result in loss of confidentiality, availability, or integrity.

- **Medium (Category II)**

  Includes any vulnerability that can result in loss of confidentiality, availability, or integrity, but the risks can be mitigated.

- **Low (Category III)**

  Any vulnerability that degrades measures to protect against loss of confidentiality, availability, or integrity.

Topics

Windows STIG components

Windows STIG components are designed for standalone servers and they apply Local Group Policy. STIG-compliant components install InstallRoot on Windows AMIs from the Department of Defense (DoD) to install and update the DoD certificates. They also remove unnecessary certificates to maintain STIG compliance.

STIG-Build-Windows-Low version 1.4.0

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Windows AMIs, you can download our spreadsheet.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

- **Windows Server 2019 STIG Version 2 Release 2**

  V-205691, V-205819, V-205858, V-205859, V-205860, V-205870, V-205871, and V-205923

- **Windows Server 2016 STIG Version 2 Release 2**

  V-224916, V-224917, V-224918, V-224919, V-224931, V-224942, and V-225060

- **Windows Server 2012 R2 STIG Version 3 Release 2**

  V-225537, V-225536, V-225526, V-225525, V-225514, V-225511, V-225490, V-225489, V-225488, V-225487, V-225485, V-225484, V-225483, V-225482, V-225481, V-225480, V-225479, V-225476, V-225473, V-225468, V-225462, V-225460, V-225459, V-225412, V-225394, V-225392, V-225376, V-225363, V-225362, V-225360, V-225359, V-225358, V-225357, V-225355, V-225343, V-225342, V-225336, V-225335, V-225334, V-225333, V-225332, V-225331, V-225330, V-225328, V-225327, V-225324, V-225319, V-225318, and V-225250

- **Microsoft .NET Framework 4.0 STIG Version 2 Release 1**

  No STIG settings are applied to the Microsoft .NET Framework for Category III vulnerabilities.

- **Windows Firewall STIG Version 1 Release 7**

  V-17425, V-17426, V-17427, V-17435, V-17436, V-17437, V-17445, V-17446, and V-17447

- **Internet Explorer 11 STIG Version 1 Release 19**

  V-46477, V-46629, and V-97527

STIG-Build-Windows-Medium version 1.4.0

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Windows AMIs, you can download our spreadsheet.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

> **Note**
> The STIG-Build-Windows-Medium components include all STIG settings that Image Builder applies to STIG-Build-Windows-Low components, in addition to the STIG settings that are applied specifically for Category II vulnerabilities.

- **Windows Server 2019 STIG Version 2 Release 2**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

  V-205625, V-205626, V-205627, V-205629, V-205630, V-205633, V-205634, V-205635, V-205636, V-205637, V-205638, V-205639, V-205643, V-205644, V-205648, V-205649, V-205650, V-205651, V-205652, V-205655, V-205656, V-205659, V-205660, V-205662, V-205671, V-205672, V-205673, V-205675, V-205676, V-205678, V-205679, V-205680, V-205681, V-205682, V-205683, V-205684, V-205685, V-205686, V-205687, V-205688, V-205689, V-205690, V-205692, V-205693, V-205694, V-205697, V-205698, V-205708, V-205709, V-205712, V-205714, V-205716, V-205717, V-205718, V-205719, V-205720, V-205722, V-205729, V-205730, V-205733, V-205747, V-205751, V-205752, V-205754, V-205756, V-205758, V-205759, V-205760, V-205761, V-205762, V-205764, V-205765, V-205766, V-205767, V-205768, V-205769, V-205770, V-205771, V-205772, V-205773, V-205774, V-205775, V-205776, V-205777, V-205778, V-205779, V-205780, V-205781, V-205782, V-205783, V-205784, V-205795, V-205796, V-205797, V-205798, V-205801, V-205808, V-205809, V-205810, V-205811, V-205812, V-205813, V-205814, V-205815, V-205816, V-205817, V-205821, V-205822, V-205823, V-205824, V-205825, V-205826, V-205827, V-205828, V-205830, V-205831, V-205832, V-205833, V-205834, V-205835, V-205836, V-205837, V-205838, V-205839, V-205840, V-205841, V-205861, V-205863, V-205865, V-205866, V-205867, V-205868, V-205869, V-205872, V-205873, V-205874, V-205878, V-205879, V-205880, V-205881, V-205889, V-205891, V-205905, V-205911, V-205912, V-205915, V-205916, V-205917, V-205918, V-205920, V-205921, V-205922, V-205924, V-205925, and V-221930
- **Windows Server 2016 STIG Version 2 Release 2**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

  V-224850, V-224852, V-224853, V-224854, V-224855, V-224856, V-224857, V-224858, V-224859, V-224866, V-224867, V-224868, V-224869, V-224870, V-224871, V-224872, V-224873, V-224881, V-224882, V-224883, V-224884, V-224885, V-224886, V-224887, V-224888, V-224889, V-224890, V-224891, V-224892, V-224893, V-224894, V-224895, V-224896, V-224897, V-224898, V-224899, V-224900, V-224901, V-224902, V-224903, V-224904, V-224905, V-224906, V-224907, V-224908, V-224909, V-224910, V-224911, V-224912, V-224913, V-224914, V-224915, V-224920, V-224922, V-224924, V-224925, V-224926, V-224927, V-224928, V-224929, V-224930, V-224935, V-224936, V-224937, V-224938, V-224939, V-224940, V-224941, V-224943, V-224944, V-224945, V-224946, V-224947, V-224948, V-224949, V-224950, V-224951, V-224952, V-224953, V-224955, V-224956, V-224957, V-224959, V-224960, V-224962, V-224963, V-225010, V-225013, V-225014, V-225015, V-225016, V-225017, V-225018, V-225019, V-225021, V-225022, V-225023, V-225024, V-225028, V-225029, V-225030, V-225031, V-225032, V-225033, V-225034, V-225035, V-225038, V-225039,

V-225040, V-225041, V-225042, V-225043, V-225047, V-225049, V-225050, V-225051, V-225052, V-225055, V-225056, V-225057, V-225058, V-225061, V-225062, V-225063, V-225064, V-225065, V-225066, V-225067, V-225068, V-225069, V-225072, V-225073, V-225074, V-225076, V-225078, V-225080, V-225081, V-225082, V-225083, V-225084, V-225086, V-225087, V-225088, V-225089, V-225092, and V-225093

- **Windows Server 2012 R2 STIG Version 3 Release 2**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

  V-225574, V-225573, V-225572, V-225571, V-225570, V-225569, V-225568, V-225567, V-225566, V-225565, V-225564, V-225563, V-225562, V-225561, V-225560, V-225559, V-225558, V-225557, V-225555, V-225554, V-225553, V-225551, V-225550, V-225549, V-225548, V-225546, V-225545, V-225544, V-225543, V-225542, V-225541, V-225540, V-225539, V-225538, V-225535, V-225534, V-225533, V-225532, V-225531, V-225530, V-225529, V-225528, V-225527, V-225524, V-225523, V-225522, V-225521, V-225520, V-225519, V-225518, V-225517, V-225516, V-225515, V-225513, V-225510, V-225509, V-225508, V-225506, V-225504, V-225503, V-225502, V-225501, V-225500, V-225494, V-225486, V-225478, V-225477, V-225475, V-225474, V-225472, V-225471, V-225470, V-225469, V-225464, V-225463, V-225461, V-225458, V-225457, V-225456, V-225455, V-225454, V-225453, V-225452, V-225448, V-225443, V-225442, V-225441, V-225415, V-225414, V-225413, V-225411, V-225410, V-225409, V-225408, V-225407, V-225406, V-225405, V-225404, V-225402, V-225401, V-225400, V-225398, V-225397, V-225395, V-225393, V-225391, V-225389, V-225386, V-225385, V-225384, V-225383, V-225382, V-225381, V-225380, V-225379, V-225378, V-225377, V-225375, V-225374, V-225373, V-225372, V-225371, V-225370, V-225369, V-225368, V-225367, V-225356, V-225353, V-225352, V-225351, V-225350, V-225349, V-225348, V-225347, V-225346, V-225345, V-225344, V-225341, V-225340, V-225339, V-225338, V-225337, V-225329, V-225326, V-225325, V-225323, V-225322, V-225321, V-225320, V-225317, V-225316, V-225315, V-225314, V-225305, V-225304, V-225303, V-225302, V-225301, V-225300, V-225299, V-225298, V-225297, V-225296, V-225295, V-225294, V-225293, V-225292, V-225291, V-225290, V-225289, V-225288, V-225287, V-225286, V-225285, V-225284, V-225283, V-225282, V-225281, V-225280, V-225279, V-225278, V-225277, V-225276, V-225275, V-225273, V-225272, V-225271, V-225270, V-225269, V-225268, V-225267, V-225266, V-225265, V-225264, V-225263, V-225261, V-225260, V-225259, and V-225239

- **Microsoft .NET Framework 4.0 STIG Version 2 Release 1**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus V-225238

- **Windows Firewall STIG Version 1 Release 7**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

  V-17415, V-17416, V-17417, V-17419, V-17429, and V-17439

- **Internet Explorer 11 STIG Version 1 Release 19**

  Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

  V-46473, V-46475, V-46481, V-46483, V-46501, V-46507, V-46509, V-46511, V-46513, V-46515, V-46517, V-46521, V-46523, V-46525, V-46543, V-46545, V-46547, V-46549, V-46553, V-46555, V-46573, V-46575, V-46577, V-46579, V-46581, V-46583, V-46587, V-46589, V-46591, V-46593, V-46597, V-46599, V-46601, V-46603, V-46605, V-46607, V-46609, V-46615, V-46617, V-46619, V-46621, V-46625, V-46633, V-46635, V-46637, V-46639, V-46641, V-46643, V-46645, V-46647, V-46649, V-46653, V-46663, V-46665, V-46669, V-46681, V-46685, V-46689, V-46691, V-46693, V-46695, V-46701, V-46705, V-46709, V-46711, V-46713, V-46715, V-46717, V-46719, V-46721, V-46723, V-46725, V-46727, V-46729, V-46731, V-46733, V-46779, V-46781, V-46787, V-46789, V-46791, V-46797, V-46799, V-46801, V-46807, V-46811, V-46815, V-46819, V-46829, V-46841, V-46847, V-46849, V-46853, V-46857, V-46859, V-46861, V-46865, V-46869, V-46879, V-46883, V-46885, V-46889, V-46893, V-46895, V-46897, V-46903, V-46907, V-46921, V-46927, V-46939, V-46975, V-46981, V-46987, V-46995, V-46997, V-46999, V-47003, V-47005, V-47009, V-64711,

V-64713, V-64715, V-64717, V-64719, V-64721, V-64723, V-64725, V-64729, V-72757, V-72759, V-72761, V-72763, V-75169, and V-75171

STIG-Build-Windows-High version 1.4.0

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Windows AMIs, you can download our spreadsheet.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

> **Note**
> The STIG-Build-Windows-High components include all STIG settings that Image Builder applies to STIG-Build-Windows-Low and STIG-Build-Windows-Medium components, in addition to the STIG settings that are applied specifically for Category I vulnerabilities.

- **Windows Server 2019 STIG Version 2 Release 2**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-205653, V-205654, V-205711, V-205713, V-205724, V-205725, V-205757, V-205802, V-205804, V-205805, V-205806, V-205849, V-205908, V-205913, V-205914, and V-205919
- **Windows Server 2016 STIG Version 2 Release 2**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-224874, V-224932, V-224933, V-224934, V-224954, V-224958, V-224961, V-225025, V-225044, V-225045, V-225046, V-225048, V-225053, V-225054, and V-225079
- **Windows Server 2012 R2 STIG Version 3 Release 2**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-225556, V-225552, V-225547, V-225507, V-225505, V-225498, V-225497, V-225496, V-225493, V-225492, V-225491, V-225449, V-225444, V-225399, V-225396, V-225390, V-225366, V-225365, V-225364, V-225354, and V-225274
- **Microsoft .NET Framework 4.0 STIG Version 2 Release 1**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities for the Microsoft .NET Framework. No additional STIG settings are applied for Category I vulnerabilities.
- **Windows Firewall STIG Version 1 Release 7**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-17418, V-17428, and V-17438
- **Internet Explorer 11 STIG Version 1 Release 19**

  Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities for Internet Explorer 11. No additional STIG settings are applied for Category I vulnerabilities.

Linux STIG components

This section contains information about Linux STIG components. If the Linux distribution does not have STIG settings of its own, RHEL settings are applied. STIG settings are applied to components based on the Linux distribution, as follows:

Red Hat Enterprise Linux (RHEL) 7 STIG settings

- RHEL 7
- CentOS 7
- Amazon Linux 2 (AL2)

RHEL 8 STIG settings

- RHEL 8
- CentOS 8

STIG-Build-Linux-Low version 3.4.2

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Linux AMIs, you can download our spreadsheet.

For a complete list, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

RHEL 7 STIG Version 3 Release 4

RHEL 7/CentOS 7

V-204452, V-204576, and V-204605

AL2

V-204452, V-204576, and V-204605

RHEL 8 STIG Version 1 Release 3

RHEL 8/CentOS 8

V-230241, V-230253, V-230269, V-230270, V-230281, V-230285, V-230346, V-230381, V-230395, V-230468, V-230469, V-230485, V-230486, V-230491, V-230494, V-230495, V-230496, V-230497, V-230498, and V-230499

STIG-Build-Linux-Medium version 3.4.2

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Linux AMIs, you can download our spreadsheet.

For a complete list, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

> **Note**
> The STIG-Build-Linux-Medium components include all STIG settings that Image Builder applies to STIG-Build-Linux-Low components, in addition to the STIG settings that are applied specifically for Category II vulnerabilities.

RHEL 7 STIG Version 3 Release 4

Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

RHEL 7/CentOS 7

V-204405, V-204406, V-204407, V-204408, V-204409, V-204410, V-204411, V-204412, V-204413, V-204414, V-204415, V-204416, V-204417, V-204418, V-204422, V-204423, V-204426, V-204427, V-204428, V-204431, V-204435, V-204437, V-204449, V-204450, V-204451, V-204457, V-204466, V-204503, V-204516, V-204517, V-204518, V-204519, V-204520, V-204521, V-204522, V-204523, V-204524, V-204525, V-204526, V-204527, V-204528, V-204529, V-204530, V-204531, V-204532, V-204533, V-204534, V-204535, V-204536, V-204537, V-204538, V-204539, V-204540, V-204541, V-204542, V-204543, V-204544, V-204545, V-204546, V-204547, V-204548, V-204549, V-204550, V-204551, V-204552, V-204553, V-204554, V-204555, V-204556, V-204557, V-204558, V-204559, V-204560, V-204561, V-204562, V-204563, V-204564, V-204565, V-204566, V-204567, V-204568, V-204569, V-204570, V-204571, V-204572, V-204573, V-204579, V-204584, V-204585, V-204586, V-204587, V-204589, V-204590, V-204591, V-204592, V-204593, V-204598, V-204599, V-204600, V-204601, V-204602, V-204609, V-204610, V-204611, V-204612, V-204613, V-204614, V-204615, V-204616, V-204617, V-204619, V-204622, V-204624, V-204625, V-204630, V-204631, V-204633, V-233307, V-237634, and V-237635

AL2:

V-204405, V-204406, V-204407, V-204408, V-204409, V-204410, V-204411, V-204412, V-204413, V-204414, V-204415, V-204416, V-204417, V-204418, V-204422, V-204423, V-204426, V-204427, V-204428, V-204431, V-204435, V-204437, V-204449, V-204450, V-204451, V-204457, V-204466, V-204503, V-204516, V-204517, V-204518, V-204519, V-204520, V-204521, V-204522, V-204523, V-204524, V-204525, V-204526, V-204527, V-204528, V-204529, V-204530, V-204531, V-204532, V-204533, V-204534, V-204535, V-204536, V-204537, V-204538, V-204539, V-204540, V-204541, V-204542, V-204543, V-204544, V-204545, V-204546, V-204547, V-204548, V-204549, V-204550, V-204551, V-204552, V-204553, V-204554, V-204555, V-204556, V-204557, V-204558, V-204559, V-204560, V-204561, V-204562, V-204563, V-204564, V-204565, V-204566, V-204567, V-204568, V-204569, V-204570, V-204571, V-204572, V-204573, V-204578, V-204579, V-204584, V-204585, V-204586, V-204587, V-204589, V-204590, V-204591, V-204592, V-204593, V-204595, V-204598, V-204599, V-204600, V-204601, V-204602, V-204609, V-204610, V-204611, V-204612, V-204613, V-204614, V-204615, V-204616, V-204617, V-204619, V-204622, V-204624, V-204625, V-204630, V-204631, V-204633, V-233307, V-237634, and V-237635

RHEL 8 STIG Version 1 Release 3

Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

RHEL 8/CentOS 8

V-230228, V-230231, V-230233, V-230236, V-230237, V-230239, V-230240, V-230244, V-230255, V-230266, V-230267, V-230268, V-230273, V-230275, V-230277, V-230278, V-230279, V-230280, V-230282, V-230288, V-230289, V-230290, V-230291, V-230296, V-230297, V-230298, V-230310, V-230311, V-230312, V-230313, V-230314, V-230315, V-230324, V-230330, V-230332, V-230333, V-230334, V-230335, V-230336, V-230337, V-230338, V-230339, V-230340, V-230341, V-230342, V-230343, V-230344, V-230345, V-230348, V-230349, V-230353, V-230356, V-230357, V-230358, V-230359, V-230360, V-230361, V-230362, V-230363, V-230365, V-230368, V-230369, V-230370,

V-230375, V-230377, V-230378, V-230382, V-230383, V-230386, V-230387, V-230390, V-230392, V-230402, V-230403, V-230404, V-230405, V-230406, V-230407, V-230408, V-230409, V-230410, V-230411, V-230412, V-230413, V-230414, V-230415, V-230416, V-230417, V-230418, V-230419, V-230420, V-230421, V-230422, V-230423, V-230424, V-230425, V-230426, V-230427, V-230428, V-230429, V-230430, V-230431, V-230432, V-230433, V-230434, V-230435, V-230436, V-230437, V-230438, V-230439, V-230440, V-230441, V-230442, V-230443, V-230444, V-230445, V-230446, V-230447, V-230448, V-230449, V-230450, V-230451, V-230452, V-230453, V-230454, V-230455, V-230456, V-230457, V-230458, V-230459, V-230460, V-230461, V-230462, V-230463, V-230464, V-230465, V-230466, V-230467, V-230478, V-230480, V-230488, V-230489, V-230502, V-230503, V-230526, V-230527, V-230528, V-230532, V-230535, V-230536, V-230537, V-230538, V-230539, V-230540, V-230541, V-230542, V-230543, V-230544, V-230545, V-230546, V-230547, V-230548, V-230549, V-230555, V-230556, V-230559, V-230560, V-230561, V-237640, V-237642, V-237643, V-244520, V-244523, V-244524, V-244525, V-244526, V-244528, V-244533, V-244534, V-244537, V-244542, V-244549, V-244550, V-244551, V-244552, V-244553, and V-244554

STIG-Build-Linux-High version 3.4.2

The following list contains STIG settings that are applied to your image. Some STIG settings are not automatically applied. This can be due to technical limitations – for instance, the STIG setting might not be applicable for standalone servers. Organization-specific policies can also prevent automatic application of STIG settings, such as a requirement for administrators to review document settings. For more details about which STIGs are applied to Linux AMIs, you can download our spreadsheet.

For a complete list, see the STIGs Document Library. For information about how to view the complete list, see How to View SRGs and STIGs .

> **Note**
> The STIG-Build-Linux-High components include all STIG settings that Image Builder applies to STIG-Build-Linux-Low and STIG-Build-Linux-Medium components, in addition to the STIG settings that are applied specifically for Category I vulnerabilities.

RHEL 7 STIG Version 3 Release 4

Includes all STIG settings that Image Builder applies for Categories II and III (Medium and Low) vulnerabilities, plus:

RHEL 7/CentOS 7

V-204425, V-204442, V-204443, V-204447, V-204448, V-204455, V-204502, V-204620, and V-204621

AL2:

V-204425, V-204442, V-204443, V-204447, V-204448, V-204455, V-204502, V-204620, and V-204621

RHEL 8 STIG Version 1 Release 3

Includes all STIG settings that Image Builder applies for Category III (Low) vulnerabilities, plus:

RHEL 8/CentOS 8

V-230264, V-230265, V-230487, V-230492, V-230529, V-230531, V-230533, and V-230558

AWSTOE command reference

Use the following AWS CLI arguments to perform operations with the component management application.

CLI action/flag	Shorthand argument notation	Requirement status	Description
`validate`	N/A	Not required	Validates document syntax and checks for multiple component management application documents.
`run`	N/A	Not required	Runs multiple component management application documents.
`--documents`	`-d`	Required	The paths of the component management application documents as a comma-separated list. Valid options include local file paths, Amazon S3 URIs, or Image Builder component build version ARNs.
`--parameters`	`N/A`	Not required	Parameters are mutable variables that are defined in the component document, with settings that the calling application can provide at runtime.
`--log-s3-bucket-name`	`-b`	Not required	The Amazon S3 bucket name to use to upload the execution logs (recommended).
`--log-s3-key-prefix`	`-k`	Not required	An Amazon S3 object key prefix to use to upload logs (recommended).
`--log-s3-bucket-owner`	`N/A`	Not required	Account ID of Amazon S3 bucket for storing logs (recommended).
`--log-directory`	`-l`	Not required	All log files from running the application are generated in this path under directory `TOE_<DATETIME>_<EXECUTIONID>`. If not provided, "`.`" (current working directory) is used.

CLI action/flag	Shorthand argument notation	Requirement status	Description
`--phases`	`-p`	Not required	The names of the phases to be run from specified documents.
`--execution-id`	`-i`	Not required	This option takes id in a string that serves as the unique ID when AWSTOE runs. This unique ID is used to identify log files and detailed outputs about current AWSTOE execution. If this is not passed in, AWSTOE auto-generates a GUID.
`--version`	`-v`	Not required	Displays the application version.
`--cw-log-group`	`N/A`	Not required	CloudWatch Log group name.
`--cw-log-stream`	`N/A`	Not required	CloudWatch Log stream name, where `console.log` file is streamed.
`--cw-log-region`	`N/A`	Not required	CloudWatch Logs Region.
`--cw-ignore-failures`	`N/A`	Not required	Ignore CloudWatch logging failures.
`--help`	`-h`	Not required	Prints a manual about how to use the component management application options correctly. Provided by the Golang flag library, by default.
`--trace`	`-t`	Not required	Enables application logs to be logged to the console.

Manage EC2 Image Builder resources

Resources are the building blocks that make up image pipelines, as well as the images those pipelines produce. This chapter covers creating, maintaining, and sharing Image Builder resources, including components, recipes, and images, along with infrastructure configuration and distribution settings.

> **Note**
> To help you manage your Image Builder resources, you can assign your own metadata to each resource in the form of tags. You use tags to categorize your AWS resources in different ways; for example, by purpose, owner, or environment. This is useful when you have many resources of the same type. You can more readily identify a specific resource based on the tags you've assigned to it.
> For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

Contents

Manage components with AWSTOE

Image Builder uses the AWS Task Orchestrator and Executor (AWSTOE) component management application to orchestrate complex workflows. AWSTOE components are based on YAML documents that define the scripts to customize or test your image. For AMI images, Image Builder installs AWSTOE components and the component application on its Amazon EC2 build and test instances. For container images, the components and component application are installed inside of the running container.

Image Builder uses AWSTOE to perform all on-instance activities. There is no additional setup required to interact with AWSTOE when you run Image Builder commands or use the Image Builder console.

Workflow stages for building a new image

The Image Builder workflow for building new images includes the following two distinct stages.

1. **Build stage** (pre-snapshot) – During the build stage, you make changes to the Amazon EC2 build instance that's running your base image, to create the baseline for your new image. For example, your image or container recipe can include components that install an application, or modify the operating system firewall settings.

   The following phases run during the build stage:

   - build
   - validate

After this stage completes successfully, Image Builder creates a snapshot or container image that it uses for the test stage and beyond.

2. **Test stage** (post-snapshot) – During the test stage, Image Builder launches an EC2 instance from the snapshot or container image that was created as the final step of the build stage. Tests run on the new instance to validate settings and ensure that the instance is functioning as expected.

   The following phase runs for every component that is included in the recipe during the test stage:

   - test

   This phase applies to both Build and Test component types. After this stage completes successfully, Image Builder can create and distribute your final image from the snapshot or the container image.

   Note
   While AWSTOE allows you to define many phases in a component document, Image Builder has strict rules about what phases it runs, and during which stages it runs them. For a component to run during the build stage, the component document must define at least one of these phases: `build` or `validate`. For a component to run during the test stage, the component document must define the `test` phase.
   Since Image Builder runs the stages independently, chaining references in AWSTOE documents cannot cross stage boundaries. You cannot chain a value from a phase that runs in the build stage to a phase that runs in the test stage. You can, however, define input parameters to the intended target, and pass in values through the command line. For more information about setting component parameters in your Image Builder recipes, see Manage AWSTOE component parameters with EC2 Image Builder (p. 115).

To assist with troubleshooting on your build or test instance AWSTOE creates a log folder that contains the input document and log files to track what's happening each time a component runs. If you configured an Amazon S3 bucket in your pipeline configuration, the logs are also written there. For more information about YAML documents and log output, see Use component documents in AWSTOE (p. 32).

Tip
When you have many components to keep track of, tagging helps you to identify a specific component or version based on the tags you've assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

This section covers how to list, view, create, and import AWSTOE components, using the Image Builder console, and commands in the AWS CLI.

Contents

List and view component details

This section describes how you can find information and view details for the AWSTOE components that you use in your EC2 Image Builder recipes.

Component details

List components (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to list all of the component semantic versions that you have access to. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

```
aws imagebuilder list-components
```

You can optionally filter on whether you want to view components owned by you, by Amazon, or those shared with you by other accounts. By default, this request shows only components owned by your account.

```
aws imagebuilder list-components --owner Self
```

```
aws imagebuilder list-components --owner Amazon
```

```
aws imagebuilder list-components --owner Shared
```

List component build versions (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to list component build versions that have a specific semantic version. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

```
aws imagebuilder list-component-build-versions --component-version-arn
 arn:aws:imagebuilder:us-west-2:123456789012:component/my-example-component/2019.12.03
```

Get component details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of a component by specifying its Amazon Resource Name (ARN).

```
aws imagebuilder get-component --component-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.02/1
```

Get component policy details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of a component policy by specifying its ARN.

```
aws imagebuilder get-component-policy --component-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.02
```

Create a component using the Image Builder console

To create an AWSTOE application component using the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Select **Components** from the navigation pane. Then select **Create component**.
3. On the **Create component** page, under **Component details**, enter the following:

   a. **Image Operating system (OS)**. Specify the operating system that the component is compatible with.
   b. **Component category**. From the dropdown, select the type of build or test component that you are creating.
   c. **Component name**. Enter a name for the component.
   d. **Component version**. Enter the version number of the component.
   e. **Description**. Provide an optional description to help you identify the component.
   f. **Change description**. Provide an optional description to help you understand the changes made to this version of the component.
4. Under **Definition document**, which is the document that defines the actions that Image Builder performs on the build and test instances to create your image, enter the document content in YAML format in the provided box. You can optionally use the example provided by AWS (auto-filled when you select **Use example**) and edit the content inline. For more information about the phases, steps, and syntax for AWSTOE YAML application component documents, see Use documents in AWSTOE.
5. After you have entered the component details, select **Create component**.

   Note
   To see your new component when you create or update a recipe, apply the **Owned by me** filter to the build or test component list. The filter is located at the top of the component list, next to the search box.
6. To delete a component, from the **Components** page, select the check box next to the component that you want to delete. From the **Actions** dropdown, select **Delete component**.

To create a new component version, follow these steps:

1. Depending on where you start:

   - From the **Components** list page – Select the check box next to the component name, then select **Create new version** from the **Actions** menu.
   - From the component detail page – Choose the **Create new version** button in the upper right corner of the heading.
2. The component information is already populated with the current values when the **Create Component** page displays. Follow the create a component steps to update the component. This ensures that you enter a unique semantic version in the **Component version**. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

Create a component (AWS CLI)

In this section, we'll cover creating AWSTOE components by using Image Builder commands in the AWS CLI.

Create a YAML component document

To build a component, you must provide a YAML application component document, which represents the phases and steps to create the component.

The examples that we use in this section create a build component that calls the `UpdateOS` action module in the AWSTOE component management application to update the operating system. For more information about the `UpdateOS` action module, see UpdateOS (p. 94).

Note
Component types in Image Builder are based on the pipeline workflow. This workflow corresponds to the *Build stage* and the *Test stage* in the build process. Image Builder determines the component type as follows:

- **Build** – This is the default component type. Anything that is not classified as a test component, is a build component. This type of component runs during the *Build stage*, and if it has a `test` phase defined, that phase runs during the *Test stage*.
- **Test** – To be classified as a test component, the component document must include only one phase, named `test`. For tests that are related to build component configurations, we recommend that you use the `test` phase in the associated build component, rather than creating a standalone test component.

For more information about how Image Builder uses stages and phases to manage component workflow in its build process, see Manage components with AWSTOE (p. 106).

To create a YAML application component document for a sample application, follow the steps that are shown on the tab that matches your image operating system.

Linux

Create a YAML component file

Use your favorite file editing tool to create a file named *`update-linux-os.yaml`*, that has the following content:

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of this
# software and associated documentation files (the "Software"), to deal in the Software
# without restriction, including without limitation the rights to use, copy, modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
name: update-linux-os
description: Updates Linux with the latest security updates.
schemaVersion: 1
phases:
  - name: build
    steps:
    - name: UpdateOS
      action: UpdateOS
# Document End
```

Tip
Use a tool like this online YAML Validator, or a YAML lint extension in your code environment to verify that your YAML is well-formed.

Windows

Create a YAML component file

Use your favorite file editing tool to create a file named *`update-windows-os.yaml`*, that has the following content:

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of this
# software and associated documentation files (the "Software"), to deal in the Software
# without restriction, including without limitation the rights to use, copy, modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
name: update-windows-os
description: Updates Windows with the latest security updates.
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: UpdateOS
        action: UpdateOS
# Document End
```

Tip
Use a tool like this online YAML Validator, or a YAML lint extension in your code environment to verify that your YAML is well-formed.

For more information about the phases, steps, and syntax for AWSTOE YAML application component documents, see Use documents in AWSTOE.

Create AWSTOE components using Image Builder (AWS CLI)

This section walks you through the following steps to create an AWSTOE application component, using **imagebuilder** commands in the AWS CLI:

- Upload your YAML component document to an S3 bucket that you can reference from the command line.
- Create the AWSTOE application component using the **imagebuilder create-component** command.
- List component versions using the **imagebuilder list-components** command with a name filter to determine the next version for updates.

To create an AWSTOE application component from the YAML document that you created in the prior section, follow the steps that match your image operating system.

Linux

Store your application component document in Amazon S3

You can use an S3 bucket as a repository for the YAML application component document you created in a prior section. Follow these steps to store your AWSTOE application component source document:

- **Upload the document to Amazon S3**

  If your document is smaller than 64 KB, you can skip this step. Documents that are 64 KB or larger in size must be stored in Amazon S3.

```
aws s3 cp update-linux-os.yaml s3://my-s3-bucket/my-path/update-linux-os.yaml
```

Create a component from the YAML document

To streamline the **imagebuilder create-component** command that is used in the AWS CLI, we create a JSON file that contains all of the component parameters that we want to pass into the command, including the location of the *update-linux-os.yaml* document created in the prior steps. The `uri` key-value pair contains the file reference.

> **Note**
> The naming convention for the data points in the JSON file follows the pattern that is specified for the Image Builder API command request parameters. To review the API command request parameters, see the CreateComponent command in the *EC2 Image Builder API Reference*.
> Do not use the naming convention that is specified for providing these datapoints directly to the **imagebuilder create-component** command as options.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a file named *create-update-linux-os-component.json*, that has the following content:

```
{
 "name": "update-linux-os",
 "semanticVersion": "1.1.2",
 "description": "An example component that updates the Linux operating system",
 "changeDescription": "Initial version.",
 "platform": "Linux",
 "uri": "s3://my-s3-bucket/my-path/update-linux-os.yaml",
 "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/98765432-
b123-456b-7f89-0123456f789c",
 "tags": {
  "MyTagKey-purpose": "security-updates"
 }
}
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. **Create the component**

   Use the following command to create the component, referencing the file name for the JSON file that you created in the prior step:

```
aws imagebuilder create-component --cli-input-json file://create-update-linux-os-
component.json
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

Windows

Store your application component document in Amazon S3

You can use an S3 bucket as a repository for the YAML application component document you created in a prior section. Follow these steps to store your AWSTOE application component source document:

- **Upload the document to Amazon S3**

  If your document is smaller than 64 KB, you can skip this step. Documents that are 64 KB or larger in size must be stored in Amazon S3.

```
aws s3 cp update-windows-os.yaml s3://my-s3-bucket/my-path/update-windows-os.yaml
```

Create a component from the YAML document

To streamline the **imagebuilder create-component** command that is used in the AWS CLI, we create a JSON file that contains all of the component parameters that we want to pass into the command, including the location of the *update-windows-os.yaml* document created in the prior steps. The `uri` key-value pair contains the file reference.

> **Note**
> The naming convention for the data points in the JSON file follows the pattern that is specified for the Image Builder API command request parameters. To review the API command request parameters, see the CreateComponent command in the *EC2 Image Builder API Reference*.
> Do not use the naming convention that is specified for providing these datapoints directly to the **imagebuilder create-component** command as options.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a file named *create-update-windows-os-component.json*, that has the following content:

```
{
 "name": "update-windows-os",
 "semanticVersion": "1.1.2",
 "description": "An example component that updates the Windows operating system.",
 "changeDescription": "Initial version.",
 "platform": "Windows",
 "uri": "s3://my-s3-bucket/my-path/update-windows-os.yaml",
 "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/98765432-
b123-456b-7f89-0123456f789c",
 "tags": {
  "MyTagKey-purpose": "security-updates"
 }
}
```

   > **Note**
   >
   > - You must include the `file://` notation at the beginning of the JSON file path.
   > - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. **Create the component**

   Use the following command to create the component, referencing the file name for the JSON file that you created in the prior step:

   ```
   aws imagebuilder create-component --cli-input-json file://create-update-windows-os-component.json
   ```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

AWSTOE component versioning for updates (AWS CLI)

AWSTOE component names and versions are embedded in the component's Amazon Resource Name (ARN), after the component prefix. Each new version of a component has its own unique ARN. The steps to create a new version are exactly the same as the steps to create a new component, as long as the semantic version is unique for that component name. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

To ensure that you assign the next logical version, first get a list of the existing versions for the component that you want to change, using the **imagebuilder list-components**, command in the AWS CLI, and filtering on the name.

For this example, we are filtering on the name of the component that was created in the prior Linux examples. To list the component that you created, use the value of the `name` parameter from the JSON file you used in the **imagebuilder create-component** command.

```
aws imagebuilder list-components --filters name="name",values="update-linux-os"
{
    "requestId": "123a4567-b890-123c-45d6-ef789ab0cd1e",
    "componentVersionList": [
        {
            "arn": "arn:aws:imagebuilder:us-west-2:1234560087789012:component/update-linux-os/1.0.0",
            "name": "update-linux-os",
            "version": "1.0.0",
            "platform": "Linux",
            "type": "BUILD",
            "owner": "123456789012",
            "dateCreated": "2020-09-24T16:58:24.444Z"
        },
        {
            "arn": "arn:aws:imagebuilder:us-west-2:1234560087789012:component/update-linux-os/1.0.1",
            "name": "update-linux-os",
            "version": "1.0.1",
            "platform": "Linux",
            "type": "BUILD",
            "owner": "123456789012",
            "dateCreated": "2021-07-10T03:38:46.091Z"
        }
    ]
}
```

Based on your results, you can determine what the next version should be.

Manage AWSTOE component parameters with EC2 Image Builder

You can manage AWSTOE components, including creating and setting component parameters, directly from the EC2 Image Builder console, or by using AWS CLI commands, or one of the Image Builder SDKs. In this section, we'll cover creating and using parameters in your component, and setting component parameters through the Image Builder console and AWS CLI commands.

Use parameters in your YAML component document

To build a component, you must provide a YAML application component document, which represents the phases and steps to create the component. The recipe that references the component can set the parameters to customize the values at runtime, with default values that take effect if the parameter is not set to a specific value.

Create a component document with input parameters

This section shows you how to define and use input parameters in your YAML component document.

To create a YAML application component document that uses parameters and runs commands in your Image Builder build or test instances, follow the steps that match your image operating system:

Linux

Create a YAML component document

Use your favorite file editing tool to create a file named *hello-world-test.yaml*, that has the following content:

```
# Document Start
#
name: "HelloWorldTestingDocument-Linux"
description: "Hello world document to demonstrate parameters."
schemaVersion: 1.0
parameters:
  - MyInputParameter:
      type: string
      default: "It's me!"
      description: This is an input parameter.
phases:
  - name: build
    steps:
      - name: HelloWorldStep
        action: ExecuteBash
        inputs:
          commands:
            - echo "Hello World! Build phase. My input parameter value is
 {{ MyInputParameter }}"

  - name: validate
    steps:
      - name: HelloWorldStep
        action: ExecuteBash
        inputs:
          commands:
            - echo "Hello World! Validate phase. My input parameter value is
 {{ MyInputParameter }}"

  - name: test
    steps:
      - name: HelloWorldStep
```

```
        action: ExecuteBash
        inputs:
          commands:
            - echo "Hello World! Test phase. My input parameter value is
 {{ MyInputParameter }}"
# Document End
```

Tip
Use a tool like this online YAML Validator, or a YAML lint extension in your code environment to verify that your YAML is well-formed.

Windows

Create a YAML component document

Use your favorite file editing tool to create a file named *hello-world-test.yaml*, that has the following content:

```
# Document Start
#
name: "HelloWorldTestingDocument-Windows"
description: "Hello world document to demonstrate parameters."
schemaVersion: 1.0
parameters:
  - MyInputParameter:
      type: string
      default: "It's me!"
      description: This is an input parameter.
phases:
  - name: build
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host "Hello World! Build phase. My input parameter value is
 {{ MyInputParameter }}"

  - name: validate
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host "Hello World! Validate phase. My input parameter value is
 {{ MyInputParameter }}"

  - name: test
    steps:
      - name: HelloWorldStep
        action: ExecutePowerShell
        inputs:
          commands:
            - Write-Host "Hello World! Test phase. My input parameter value is
 {{ MyInputParameter }}"
# Document End
```

Tip
Use a tool like this online YAML Validator, or a YAML lint extension in your code environment to verify that your YAML is well-formed.

For more information about the phases, steps, and syntax for AWSTOE YAML application component documents, see Use documents in AWSTOE. For more information about parameters and their

requirements, see the Parameters (p. 41) section of the **Define and reference variables in AWSTOE** page.

Create a component from the YAML component document

Whatever method you use to create an AWSTOE component, the YAML application component document is always required as a baseline.

- To use the Image Builder console to create a component directly from your YAML document, see Create a component using the Image Builder console (p. 108).
- To use Image Builder commands in the AWS CLI to create your component, see Create AWSTOE components using Image Builder (AWS CLI) (p. 111). Replace the YAML document name in those examples with the name of your Hello World YAML document (*`hello-world-test.yaml`*).

Set component parameters in an Image Builder recipe (console)

Setting component parameters works the same for image recipes and container recipes. When you create a new recipe, or a new version of a recipe, you choose which components to include from the **Build components** and **Test components** lists. The component lists include components that are applicable for the base operating system you chose for your image.

After you select a component, it is displayed in the **Selected components** section, directly under the component lists. Configuration options are shown for each component that is selected. If your component has input parameters defined, they are displayed as an expandable section called **Input parameters**.

The following parameter settings are shown for each parameter that's defined for your component:

- **Parameter name** (*not editable*) – The name of the parameter.
- **Description** (*not editable*) – The parameter description
- **Type** (*not editable*) – The data type for the parameter value.
- **Value** – The value for the parameter. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the **Value** box with greyed-out text. If no other value is entered, AWSTOE uses the default value.

Import a component (AWS CLI)

For some scenarios, it might be easier to start with a pre-existing script. For this scenario, you can use the following example.

This example assumes that you have a file called *`import-component.json`* (as shown). Note that the file directly references a PowerShell script called `AdminConfig.ps1` that is already uploaded to *`my-s3-bucket`*. Currently, `SHELL` is supported for the component `format`.

```
{
"name": "MyImportedComponent",
"semanticVersion": "1.0.0",
"description": "An example of how to import a component",
"changeDescription": "First commit message.",
"format": "SHELL",
"platform": "Windows",
"type": "BUILD",
"uri": "s3://my-s3-bucket/AdminConfig.ps1",
"kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/60763706-b131-418b-8f85-3420912f020c"
}
```

To import the component, run the following command.

```
aws imagebuilder import-component --cli-input-json file://import-component.json
```

Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 152).

Manage recipes

An EC2 Image Builder recipe defines the base image to use as your starting point to create a new image, along with the set of components that you add to customize your image and verify that everything is working as expected. Automatic version choices are provided for each component. A maximum of 20 components, which include build and test, can be applied to a recipe.

After you create an image recipe, or a container recipe, you cannot modify or replace the recipe. To update components after a recipe is created, you must create a new recipe or recipe version. You can, however, always apply tags to your recipe. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

> **Tip**
> You can streamline the process of putting together Image Builder recipes by developing components locally, so that they are ready when you need them. To get started developing components locally, with AWSTOE, see Get started with AWSTOE (p. 25).

This section covers how to list, view, and create recipes.

Contents

List and view image recipe details

This section describes the various ways you can find information and view details for your EC2 Image Builder image recipes.

Image recipe details

List image recipes (console)

To see a list of the image recipes created under your account in the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image recipes** from the navigation pane. This shows a list of the image recipes that are created under your account.
3. To view details or create a new recipe version, choose the **Recipe name** link. This opens the detail view for the recipe.

   Note
   You can also select the check box next to the **Recipe name**, then choose **View details**.

List image recipes (AWS CLI)

The following example shows how to list all of your image recipes, using the AWS CLI.

```
aws imagebuilder list-image-recipes
```

View image recipe details (console)

To view details for a specific image recipe using the Image Builder console, select the image recipe to review, using the steps described in List image recipes (console) (p. 118).

On the recipe detail page, you can:

- Delete the recipe. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 152).
- Create a new version.
- Create a pipeline from the recipe. After choosing **Create pipeline from this recipe**, you are taken to the pipeline wizard. For more information about creating an Image Builder pipeline using the pipeline wizard, see Create an image pipeline using the EC2 Image Builder console wizard (p. 14)

  Note
  When you create a pipeline from an existing recipe, the option to create a new recipe is not available.

Get image recipe details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of an image recipe by specifying its Amazon Resource Name (ARN).

```
aws imagebuilder get-image-recipe --image-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-recipe/2020.12.03
```

Get image recipe policy details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of an image recipe policy by specifying its ARN.

```
aws imagebuilder get-image-recipe-policy --image-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-recipe/2020.12.03
```

List and view container recipe details

This section describes the various ways you can find information and view details for your EC2 Image Builder container recipes.

Container recipe details

List container recipes (console)

To see a list of the container recipes created under your account in the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Container recipes** from the navigation pane. This shows a list of the container recipes that are created under your account.
3. To view details or create a new recipe version, choose the **Recipe name** link. This opens the detail view for the recipe.

   Note
   You can also select the check box next to the **Recipe name**, then choose **View details**.

List container recipes (AWS CLI)

The following example shows how to list all of your container recipes, using the AWS CLI.

```
aws imagebuilder list-container-recipes
```

View container recipe details (console)

To view details for a specific container recipe using the Image Builder console, select the container recipe to review, using the steps described in List container recipes (console) (p. 120).

On the recipe detail page, you can:

- Delete the recipe. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 152).
- Create a new version.
- Create a pipeline from the recipe. After choosing **Create pipeline from this recipe**, you are taken to the pipeline wizard. For more information about creating an Image Builder pipeline using the pipeline wizard, see Create an image pipeline using the EC2 Image Builder console wizard (p. 14)

  Note
  When you create a pipeline from an existing recipe, the option to create a new recipe is not available.

Get container recipe details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of a container recipe by specifying its ARN.

```
aws imagebuilder get-container-recipe --container-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:container-recipe/my-example-recipe/2020.12.03
```

Get container recipe policy details (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to get the details of a container recipe policy by specifying its ARN.

```
aws imagebuilder get-container-recipe-policy --container-recipe-arn
 arn:aws:imagebuilder:us-west-2:123456789012:container-recipe/my-example-recipe/2020.12.03
```

Create image recipes and versions

This section describes creating new image recipes and recipe versions.

Contents

Create a new image recipe version (console)

Creating a new version is virtually the same as creating a new recipe. The difference is that certain details are pre-selected to match the base recipe, in most cases. The following list describes the differences between creating a new recipe and creating a new version of an existing recipe.

Base recipe details in the new version

- **Name** – *not editable*.
- **Version** – Required. It is not pre-filled with the current version or any kind of a sequence. Enter the version number you want to create, using the format *major.minor.patch*. If the version already exists, you will encounter an error.
- The **Select image** option – Pre-selected, but you can edit it. If you change your choice for the source of your base image, you might lose other details that depend on the original option you chose.

  To see details that are associated with your base image selection, choose the tab that matches your selection.

  Managed images

  - **Image Operating System (OS)** – *not editable*.
  - **Image name** – Pre-selected, based on the combination of base image choices you made for the existing recipe. However, if you change the **Select image** option, you lose the pre-selected **Image name**.
  - **Auto-versioning options** – does *not* match your base recipe. This defaults to the **Use selected OS version** option.

    Important
    If you are using semantic versioning to kick off pipeline builds, make sure you change this value to **Use latest available OS version**. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

  Custom AMI

  - **AMI ID** – Required. However, it is not pre-filled with your original entry. You must enter the AMI ID for your base image.
- **Instance configuration** – Settings are pre-selected, but you can edit them.
  - **Systems Manager agent** – You can select or clear this check box to control installation of the Systems Manager agent on the new image. The check box is cleared by default, to include the

Systems Manager agent in your new image. To remove the Systems Manager agent from the final image, so that it will not be included in your AMI, elect the check box.

- **User data** – You can use this area to provide commands, or a command script to run when you launch your build instance. However, it replaces any commands that Image Builder might have added to ensure that Systems Manager is installed.

  Important
  If you enter user data, make sure that the Systems Manager agent is pre-installed on your base image, or that you include the install in your user data.

- **Working directory** – Pre-selected, but you can edit it.
- **Components** – Components that are already included in the recipe are displayed in the **Selected components** section at the end of each of the component lists (build and test). You can remove or reorder the selected components to suit your needs.

  You can configure the following settings for your selected component:

  - **Versioning options** – Pre-selected, but you can change them. We recommend that you choose the **Use latest available component version** option to ensure that your image builds always pick up the latest version of the component. If you need to use a specific component version in your recipe, you can choose **Specify component version**, and enter the version in the **Component version** box that appears.
  - **Input parameters** – Displays input parameters that the component accepts. The **Value** is pre-filled with the value from the prior version of the recipe. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the **Value** box with greyed-out text. If no other value is entered, AWSTOE uses the default value.

  To expand **Versioning options** or **Input parameters** settings, you can either choose the arrow next to the name of the setting, or you can toggle the **Expand all** switch off and on to expand all of the settings for all of the selected components.

- **Storage (volumes)** – are pre-filled. The root volume **Device name**, **Snapshot**, and **IOPS** selections are not editable. However, you can change all of the remaining settings, such as the **Size**. You can also add new volumes.

To create a new image recipe version:

1. At the top of the recipe details page, choose **Create new version**. This takes you to the **Create image recipe** page.
2. To create the new version, make your changes, then choose **Create image recipe**.

For more information about creating an image recipe, within the context of creating an image pipeline, see Step 2: Choose recipe (p. 15) in the **Get started** section of this guide.

Create an image recipe (AWS CLI)

To create an Image Builder image recipe, using the `imagebuilder create-image-recipe` command in the AWS CLI, follow these steps:

Prerequisites

Before you run the Image Builder commands in this section to create an image recipe using the AWS CLI, you must have created the components that the recipe will use. The image recipe example in the following step refers to example components that are created in the Create a component (AWS CLI) (p. 109) section of this guide.

After you create your components, or if you are using existing components, take note of the ARNs that you want to include in the recipe.

1. **Create a CLI input JSON file**

   To streamline the **imagebuilder create-image-recipe** command that is used in the AWS CLI, we create a JSON file that contains all of the recipe attributes that we want to pass into the command.

   > **Note**
   > The naming convention for the data points in the JSON file follows the pattern that is specified for the Image Builder API command request parameters. To review the API command request parameters, see the CreateImageRecipe command in the *EC2 Image Builder API Reference*.
   > Do not use the naming convention that is specified for providing these datapoints directly to the **imagebuilder create-image-recipe** command as options.

   Here is a summary of the parameters that we specify in these examples:

   - **name** (string, required) – The name of the image recipe.
   - **description** (string) – The description of the image recipe.
   - **parentImage** (string, required) – The source (parent) image that the image recipe uses as its base environment. The value can be the parent image ARN or an Image Builder AMI ID.

     > **Note**
     > The Linux example uses an Image Builder AMI, and the Windows example uses an ARN.

   - **semanticVersion** (string, required) – The semantic version of the image recipe, which specifies the version in the following format, with numeric values in each position to indicate a specific version: <major>.<minor>.<patch>. For example, `1.0.0`. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).
   - **components** (array, required) – Contains an array of `ComponentConfiguration` objects. At least one build component must be specified:

     > **Important**
     > Components are installed in the order in which they are specified.

     - **componentARN** (string, required) – The component ARN.

       > **Note**
       > To use one of the examples to create your own image recipe, you must replace the example ARNs with the ARNs for the components that you are using for your recipe, including the AWS Region, name, and version number for each.

     - **parameters** (array of objects) – Contains an array of `ComponentParameter` objects.
       - **name** (string, required) – The name of the component parameter to set.
       - **value** (array of strings, required) – Contains an array of strings to set the value for the named component parameter. If there is a default value defined for the component, and no other value is provided, AWSTOE uses the default value.
   - **additionalInstanceConfiguration** (object) – Specify additional settings and launch scripts for your build instances.
     - **systemsManagerAgent** (object) – Contains settings for the Systems Manager agent on your build instance.
       - **uninstallAfterBuild** (Boolean) – Controls whether the Systems Manager agent is removed from your final build image, prior to creating the new AMI. If this is set to `true`, then the agent is removed from the final image. If it's set to `false`, then the agent is left in, so that it is included in the new AMI. The default value is `false`.

         > **Note**
         > If the `uninstallAfterBuild` attribute is not included in the JSON file, and the following conditions are true, then Image Builder removes the Systems Manager agent from the final image, so that it is not available in the AMI:
         >
         > - The `userDataOverride` is empty, or has been left out of the JSON file.

    - Image Builder automatically installed the Systems Manager agent on the build instance for an operating system that did not have the agent pre-installed on the source (parent) image.

- **userDataOverride** (string) – Provide commands or a command script to run when you launch your build instance.

  Note
  The user data is always base 64 encoded. For example, the following commands are encoded as `IyEvYmluL2Jhc2gKbWtkaXIgLXAgL3Zhci9iYi8KdG91Y2ggL3Zhci$`:

  ```
  #!/bin/bash
  mkdir -p /var/bb/
  touch /var
  ```

  The Linux example uses this encoded value.

Linux

The source (parent) image in this example is an Image Builder AMI. When you use an Image Builder AMI, you must have access to the AMI, and the AMI must be in the same Region in which you are using Image Builder. Save the file as `create-image-recipe.json`, to use in the **imagebuilder create-image-recipe** command.

```
{
    "name": "BB Ubuntu Image recipe",
    "description": "Hello World image recipe for Linux.",
 "parentImage": "ami-0a01b234c5de6fabc",
    "semanticVersion": "1.0.0",
    "components": [
        {
            "componentArn": "arn:aws:imagebuilder:us-
west-2:123456789012:component/bb$"
        }
    ],
    "additionalInstanceConfiguration": {
     "systemsManagerAgent": {
       "uninstallAfterBuild": true
     },
     "userDataOverride": "IyEvYmluL2Jhc2gKbWtkaXIgLXAgL3Zhci9iYi8KdG91Y2ggL3Zhci$
    }
}
```

Windows

This example refers to the latest version of the Windows Server 2016 English Full Base image. The ARN in this example references the latest image in the SKU based on the semantic version filters that you have specified: `arn:aws:imagebuilder:us-west-2:aws:image/windows-server-2016-english-full-base-x86/x.x.x`.

```
{
    "name": "MyBasicRecipe",
    "description": "This example image recipe creates a Windows 2016 image.",
    "parentImage": "arn:aws:imagebuilder:us-west-2:aws:image/windows-server-2016-
english-full-base-x86/x.x.x",
    "semanticVersion": "1.0.0",
    "components": [
        {
            "componentArn": "arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.02/1"
```

```
        },
        {
            "componentArn": "arn:aws:imagebuilder:us-
west-2:123456789012:component/my-imported-component/1.0.0/1"
        }
    ]
}
```

Note
To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

2. **Create the recipe**

   Use the following command to create the recipe, referencing the file name for the JSON file that you created in the prior step:

```
aws imagebuilder create-image-recipe --cli-input-json file://create-image-recipe.json
```

Note

- You must include the `file://` notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

Create container recipes and versions

This section covers creating new container recipes and recipe versions.

Contents

Create a new container recipe version (console)

Creating a new version is virtually the same as creating a new recipe. The difference is that certain details are pre-selected to match the base recipe, in most cases. The following list describes the differences between creating a new recipe and creating a new version of an existing recipe.

Recipe details

- **Name** – *not editable*.
- **Version** – Required. It is not pre-filled with the current version or any kind of a sequence. Enter the version number you want to create, using the format *major.minor.patch*. If the version already exists, you will encounter an error.

Base image

- The **Select image** option – Pre-selected, but editable. If you change your choice for the source of your base image, you might lose other details that depend on the original option you chose.

  To see details that are associated with your base image selection, choose the tab that matches your selection.

Managed images

- **Image Operating System (OS)** – *not editable*.
- The **Image name** – Pre-selected, based on the combination of base image choices you made for the existing recipe. However, if you change the **Select image** option, you lose the pre-selected **Image name**.
- **Auto-versioning options** – does *not* match your base recipe. This defaults to the **Use selected OS version** option.

  Important
  If you are using semantic versioning to kick off pipeline builds, make sure you change this value to **Use latest available OS version**. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

ECR image

- **Image Operating System (OS)** – Pre-selected, but editable.
- **OS version** – Pre-selected, but editable.
- **ECR image ID** – Pre-filled, but editable.

Docker Hub image

- **Image Operating System (OS)** – *not editable*.
- **OS version** – Pre-selected, but editable.
- **Docker image ID** – Pre-filled, but editable.

Instance configuration

- **AMI ID** – Pre-filled, but editable.
- **Storage (volumes)**

  EBS volume 1 (AMI root) – Pre-filled. You cannot edit the root volume **Device name**, **Snapshot**, and **IOPS** selections. However, you can change all of the remaining settings, such as the **Size**. You can also add new volumes.

Working directory

- **Working directory path** – Pre-filled, but editable.

Components

- **Components** – Components that are already included in the recipe are displayed in the **Selected components** section at the end of each of the component lists (build and test). You can remove or reorder the selected components to suit your needs.

  You can configure the following settings for your selected component:

  - **Versioning options** – Pre-selected, but you can change them. We recommend that you choose the **Use latest available component version** option to ensure that your image builds always pick up the latest version of the component. If you need to use a specific component version in your recipe, you can choose **Specify component version**, and enter the version in the **Component version** box that appears.
  - **Input parameters** – Displays input parameters that the component accepts. The **Value** is pre-filled with the value from the prior version of the recipe. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the **Value** box with greyed-out text. If no other value is entered, AWSTOE uses the default value.

To expand **Versioning options** or **Input parameters** settings, you can either choose the arrow next to the name of the setting, or you can toggle the **Expand all** switch off and on to expand all of the settings for all of the selected components.

Target repository

- **Target repository name** – Pre-filled, but editable.

To create a new container recipe version:

1. At the top of the container recipe details page, choose **Create new version**. You are taken to the **Create recipe** page for container recipes.
2. To create the new version, make your changes, then choose **Create recipe**.

For more information about creating a container recipe, within the context of creating an image pipeline, see Step 2: Choose recipe (p. 19) in the **Get started** section of this guide.

Create a container recipe (AWS CLI)

To create an Image Builder container recipe, using the `imagebuilder create-container-recipe` command in the AWS CLI, follow these steps:

Prerequisites

Before you run the Image Builder commands in this section to create a container recipe using the AWS CLI, you must have created the components that the recipe will use. The container recipe example in the following step refers to example components that are created in the Create a component (AWS CLI) (p. 109) section of this guide.

After you create your components, or if you are using existing components, take note of the ARNs that you want to include in the recipe.

1. **Create a CLI input JSON file**

   To streamline the **imagebuilder create-container-recipe** command that is used in the AWS CLI, we create a JSON file that contains all of the recipe parameters that we want to pass into the command. Save the file as `create-container-recipe.json`, to use in the **imagebuilder create-container-recipe** command.

   > **Note**
   > The naming convention for the data points in the JSON file follows the pattern that is specified for the Image Builder API command request parameters. To review the API command request parameters, see the CreateContainerRecipe command in the *EC2 Image Builder API Reference*.
   > Do not use the naming convention that is specified for providing these datapoints directly to the **imagebuilder create-container-recipe** command as options.

   Here is a summary of the parameters that we specify in is example:

   - **components** (array of objects, required) – Contains an array of `ComponentConfiguration` objects. At least one build component must be specified:

     > **Important**
     > Components are installed in the order in which they are specified.

     - **componentARN** (string, required) – The component ARN.

Note
To use the example to create your own container recipe, you must replace the example ARNs with the ARNs for the components that you are using for your recipe, including the AWS Region, name, and version number for each.

- **parameters** (array of objects) – Contains an array of `ComponentParameter` objects.
  - **name** (string, required) – The name of the component parameter to set.
  - **value** (array of strings, required) – Contains an array of strings to set the value for the named component parameter. If there is a default value defined for the component, and no other value is provided, AWSTOE uses the default value.
- **containerType** (string, required) – The type of container to create. Valid values include: "DOCKER".
- **dockerfileTemplateData** (string) – The Dockerfile template that is used to build your image, as an inline data blob.
- **name** (string, required) – The name of the container recipe.
- **description** (string) – The description of the container recipe.
- **parentImage** (string, required) – The source (parent) image that the container recipe uses as its base environment.
- **platformOverride** (string) – Specifies the operating system platform when you use a custom base image.
- **semanticVersion** (string, required) – The semantic version of the container recipe, which specifies the version in the following format, with numeric values in each position to indicate a specific version: <major>.<minor>.<patch>. For example, `1.0.0`. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).
- **tags** (string map) – Tags that are attached to the container recipe.
- **instanceConfiguration** (object) – A group of options that can be used to configure an instance for building and testing container images.
  - **image** (string) – The AMI ID to use as the base image for a container build and test instance. If not specified, Image Builder will use the appropriate Amazon ECS-optimized AMI as a base image.
  - **blockDeviceMappings** (array of objects) – Defines the block devices to attach for building an instance from the Image Builder AMI specified in the **image** parameter.
    - **deviceName** (string) – The device to which these mappings apply.
    - **ebs** (object) – Used to manage Amazon EBS-specific configuration for this mapping.
      - **deleteOnTermination** (Boolean) – Used to configure delete on termination of the associated device.
      - **encrypted** (Boolean) – Used to configure device encryption.
      - **volumeSize** (integer) – Used to override the device's volume size.
      - **volumeType** (string) – Used to override the device's volume type.
- **targetRepository** (object, required) – The destination repository for the container image.
  - **repositoryName** (string, required) – The name of the container repository where the output container image is stored. This name is prefixed by the repository location.
  - **service** (string, required) – Specifies the service in which this image was registered.
- **workingDirectory** (string) – The working directory for use during build and test workflows.

```
{
   "components": [
      {
         "componentArn": "arn:aws:imagebuilder:us-east-1:123456789012:component/
helloworldal2/x.x.x"
      }
   ],
```

```
   "containerType": "DOCKER",
   "description": "My Linux Docker container image",
   "dockerfileTemplateData": "FROM
{{{ imagebuilder:parentImage }}}\n{{{ imagebuilder:environments }}}\n{{{ imagebuilder:components }
   "name": "amazonlinux-container-recipe",
   "parentImage": "amazonlinux:latest",
   "platformOverride": "Linux",
   "semanticVersion": "1.0.2",
   "tags": {
      "sometag" : "Tag detail"
   },
   "instanceConfiguration": {
      "image": "ami-1234567890",
      "blockDeviceMappings": [
         {
            "deviceName": "/dev/xvda",
            "ebs": {
               "deleteOnTermination": true,
               "encrypted": false,
               "volumeSize": 8,
               "volumeType": "gp2"
             }
          }
      ]
   },
   "targetRepository": {
      "repositoryName": "myrepo",
      "service": "ECR"
   },
   "workingDirectory": "/tmp"
}
```

2. **Create the recipe**

   Use the following command to create the recipe, referencing the file name for the JSON file that you created in the prior step:

   ```
   aws imagebuilder create-container-recipe --cli-input-json file://create-container-recipe.json
   ```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 152).

Manage EC2 Image Builder images

After you have created AMI or container images with Image Builder, you can manage them using the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI.

Tip
When you have multiple resources of the same type, tagging helps you to identify a specific resource based on the tags you've assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

This section covers how to list, view, and create images.

Contents

List and view image details

This section describes the various ways that you can find information and view details for your EC2 Image Builder images.

Image recipe details

List images (AWS CLI)

The following example shows how to list all of the image versions that you have access to.

```
aws imagebuilder list-images
```

List image build versions (AWS CLI)

The following example shows how to list image build versions with a specific semantic version.

```
aws imagebuilder list-image-build-versions --image-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-image/2019.12.03
```

Note
To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

Get an AMI image (AWS CLI)

To check the progress of your image, use the `get-image` operation. `get-image` returns details about the image, metadata, current state, and output resources when they are available.

```
aws imagebuilder get-image --image-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-recipe/2019.12.03/1
```

Get image policy details (AWS CLI)

The following example shows how to get the details of an image policy by specifying its Amazon Resource Name (ARN).

```
aws imagebuilder get-image-policy --image-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-image/2019.12.02
```

Create images

This section covers creating AMI and container images.

Contents

Create an image (AWS CLI)

When you have a basic recipe and an infrastructure configuration, you can create an image using the **imagebuilder create-image** command.

```
aws imagebuilder create-image --image-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-recipe/2019.12.03 --infrastructure-
configuration-arn arn:aws:imagebuilder:us-west-2123456789012:infrastructure-configuration/
myexampleinfrastructure
```

Cancel an image creation (AWS CLI)

To cancel an in-progress image build, use the **imagebuilder cancel-image-creation** command.

```
aws imagebuilder cancel-image-creation --image-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-recipe/2019.12.03/1
```

Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 152).

Manage EC2 Image Builder infrastructure configuration

After you have created infrastructure configurations with Image Builder, you can manage them using the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI.

> **Tip**
> When you have multiple resources of the same type, tagging helps you to identify a specific resource based on the tags you've assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

This section covers how to list, view, and create infrastructure configurations.

Contents

List and view infrastructure configuration details

This section describes the various ways that you can find information and view details for your EC2 Image Builder infrastructure configurations.

Infrastructure configuration details

List infrastructure configurations (AWS CLI)

The following example shows how to list of all of your infrastructure configurations, using the **imagebuilder list-infrastructure-configurations** command in the AWS CLI.

```
aws imagebuilder list-infrastructure-configurations
```

Get infrastructure configuration details (AWS CLI)

The following example shows how use the **imagebuilder get-infrastructure-configurations** command in the AWS CLI to get the details of an infrastructure configuration by specifying its Amazon Resource Name (ARN).

```
aws imagebuilder get-infrastructure-configuration --infrastructure-configuration-arn
 arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/my-example-
infrastructure-configuration
```

Create and update infrastructure configurations

This section covers creating and updating EC2 Image Builder infrastructure configurations.

Contents

Create an infrastructure configuration (AWS CLI)

Infrastructure configurations allow you to specify the infrastructure within which to build and test your image. In the infrastructure configuration, you can specify instance types, subnets, and security groups to associate with your instance. You can also associate an Amazon EC2 key pair with the instance used to build your image. This allows you to log on to your instance to troubleshoot if your build fails and you set `terminateInstanceOnFailure` to `false`. If you configure logging, the instance profile specified in your infrastructure configuration must have `s3:PutObject` permissions for the target bucket (`arn:aws:s3:::BucketName/*`).

```
{
```

```
    "name": "MyExampleInfrastructure",
    "description": "An example that will retain instances of failed builds",
    "instanceTypes": [
        "m5.large", "m5.xlarge"
    ],
    "instanceProfileName": "myIAMInstanceProfileName",
    "securityGroupIds": [
        "sg-12345678"
    ],
    "subnetId": "sub-12345678",
    "logging": {
        "s3Logs": {
            "s3BucketName": "my-logging-bucket",
            "s3KeyPrefix": "my-path"
        }
    },
    "keyPair": "myKeyPairName",
    "terminateInstanceOnFailure": false,
    "snsTopicArn": "arn:aws:sns:us-west-2:123456789012:MyTopic"
}
```

The example infrastructure configuration is stored in a file called `create-infrastructure-configuration.json`.

The example configuration specifies two instance types, `m5.large` and `m5.xlarge`. We recommend specifying more than one instance type because this allows EC2 Image Builder to launch an instance from a pool with sufficient capacity. This can reduce your transient build failures.

The instance profile name is used to provide the instance with the permissions that are required to perform customization activities. For example, if you have a component that retrieves resources from Amazon S3, the instance profile requires permissions to access those files. This instance profile also requires a minimal set of permissions for EC2 Image Builder to successfully communicate with the instance. For more information, see Prerequisites (p. 12).

Use the JSON file to create the infrastructure configuration.

```
aws imagebuilder create-infrastructure-configuration --cli-input-json file://create-
infrastructure-configuration.json
```

Update an infrastructure configuration (AWS CLI)

The following example scenario consists of a JSON file, called `update-infrastructure-configuration.json`, followed by running the **imagebuilder update-infrastructure-configuration** command that uses the JSON file as input. The JSON file contains infrastructure configuration details for the update.

The example `update-infrastructure-configuration.json` contents are as follows.

```
{
    "infrastructureConfigurationArn": "arn:aws:imagebuilder:us-
west-2:123456789012:infrastructure-configuration/my-example-infrastructure-configuration",
    "description": "An example that will terminate instances of failed builds",
    "instanceTypes": [
        "m5.large", "m5.2xlarge"
    ],
    "instanceProfileName": "myIAMInstanceProfileName",
    "securityGroupIds": [
        "sg-12345678"
    ],
    "subnetId": "sub-12345678",
    "logging": {
```

```
        "s3Logs": {
            "s3BucketName": "my-logging-bucket",
            "s3KeyPrefix": "my-path"
        }
    },
    "terminateInstanceOnFailure": true,
    "snsTopicArn": "arn:aws:sns:us-west-2:123456789012:MyTopic"
}
```

Run the following command, which references the preceding update-infrastructure-configuration.json file.

```
aws imagebuilder update-infrastructure-configuration --cli-input-json file://update-
infrastructure-configuration.json
```

Manage EC2 Image Builder distribution settings

After you create distribution settings with Image Builder, you can manage them using the Image Builder console, the Image Builder API, or **imagebuilder** commands in the AWS CLI.

You can use your distribution settings in the following ways to deliver images to target Regions and accounts one time, or with every pipeline build:

- To automatically deliver updated images to specified Regions and accounts, use distribution settings with an Image Builder pipeline that runs on a schedule.
- To create a new image and deliver it to the specified Regions and accounts, use distribution settings with the **create-image** command using the AWS CLI.

> **Tip**
> When you have multiple resources of the same type, tagging helps you to identify a specific resource based on the tags you've assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 151) section of this guide.

This section covers how to list, view, and create distribution settings.

Contents

List and view distribution settings detail

This section describes the various ways that you can find information and view details for your EC2 Image Builder distribution settings.

Distribution settings detail

List distributions (AWS CLI)

The following example shows how to use the **imagebuilder list-distribution-configurations** command in the AWS CLI to list all of your distributions.

```
aws imagebuilder list-distribution-configurations
```

Get distribution settings detail (AWS CLI)

The following example shows how to use the **imagebuilder get-distribution-configuration** command in the AWS CLI to get the details of a distribution settings configuration by specifying its Amazon Resource Name (ARN).

```
aws imagebuilder get-distribution-configuration --distribution-configuration-arn
 arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-example-
distribution-configuration
```

Create and update distribution settings for AMIs

This section covers creating and updating distribution settings for an Image Builder AMI.

Contents

Create distribution settings for Image Builder AMIs (AWS CLI)

A distribution configuration allows you to specify the name and description of your output AMI, authorize other AWS accounts to launch the AMI, copy the AMI to other accounts, and replicate the AMI to other AWS Regions. It also allows you to export the AMI to Amazon Simple Storage Service (Amazon S3). To make an AMI public, set the launch permission authorized accounts to `all`. See the examples for making an AMI public at EC2 ModifyImageAttribute.

The following example shows how to use the **create-distribution-configuration** command to create distribution settings for your AMI, using the AWS CLI.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with keys shown in the following example, plus values that are valid for your environment. This example uses a file named `create-ami-distribution-configuration.json`:

```
{
    "name": "MyExampleDistribution",
    "description": "Copies AMI to eu-west-1 and exports to S3",
    "distributions": [
        {
            "region": "us-west-2",
            "amiDistributionConfiguration": {
                "name": "Name {{imagebuilder:buildDate}}",
                "description": "An example image name with parameter references",
                "amiTags": {
                    "KeyName": "Some Value"
                },
                "launchPermission": {
```

```
                    "userIds": [
                        "987654321012"
                    ]
                }
            }
        },
        {
            "region": "eu-west-1",
            "amiDistributionConfiguration": {
                "name": "My {{imagebuilder:buildVersion}} image
 {{imagebuilder:buildDate}}",
                "amiTags": {
                    "KeyName": "Some value"
                },
                "launchPermission": {
                    "userIds": [
                        "100000000001"
                    ]
                }
            }
        }
    ]
}
```

2. **Run the following command, using the file you created as input.**

```
aws imagebuilder create-distribution-configuration --cli-input-json file://create-ami-
distribution-configuration.json
```

Note

- You must include the `file://` notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see **create-distribution-configuration** in the *AWS CLI Command Reference*.

Update AMI distribution settings (AWS CLI)

The following example shows how to use the **update-distribution-configuration** command to update distribution settings for your AMI, using the AWS CLI.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the keys shown in the following example, plus values that are valid for your environment. This example uses a file named `update-ami-distribution-configuration.json`.

```
{
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-
west-2:123456789012:distribution-configuration/update-ami-distribution-
configuration.json",
    "description": "Copies AMI to eu-west-2 and exports to S3",
    "distributions": [
      {
            "region": "us-west-2",
            "amiDistributionConfiguration": {
```

```
                "name": "Name {{imagebuilder:buildDate}}",
                "description": "An example image name with parameter references",
                "launchPermissions": {
                    "userIds": [
                        "987654321012"
                    ]
                }
            }
        },
        {
            "region": "eu-west-2",
            "amiDistributionConfiguration": {
                "name": "My {{imagebuilder:buildVersion}} image
 {{imagebuilder:buildDate}}",
                "tags": {
                    "KeyName": "Some value"
                },
                "launchPermissions": {
                    "userIds": [
                        "100000000001"
                    ]
                }
            }
        }
    ]
}
```

2. **Run the following command, using the file you created as input.**

```
aws imagebuilder update-distribution-configuration --cli-input-json file://update-ami-distribution-configuration.json
```

Note

- You must include the `file://` notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see **update-distribution-configuration** in the *AWS CLI Command Reference*. To update tags for your distribution configuration resource, see the Tag resources (p. 151) section.

Create and update distribution settings for container images

This section covers creating and updating distribution settings for Image Builder container images.

Contents

Create distribution settings for Image Builder container images (AWS CLI)

A distribution configuration enables you to specify the name and description of your output container image and replicate the container image to other AWS Regions. You can also apply separate tags to the distribution configuration resource and to the container images within each Region.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the keys shown in the following example, plus values that are valid for your environment. This example uses a file named `create-container-distribution-configuration.json`:

```
{
 "name": "distribution-configuration-name",
 "description": "Distributes container image to Amazon ECR repository in two regions.",
 "distributions": [
     {
         "region": "us-west-2",
         "containerDistributionConfiguration": {
             "description": "My test image.",
             "targetRepository": {
                 "service": "ECR",
                 "repositoryName": "testrepo"
             },
             "containerTags": ["west2", "image1"]
         }
     },
     {
         "region": "us-east-1",
         "containerDistributionConfiguration": {
             "description": "My test image.",
             "targetRepository": {
                 "service": "ECR",
                 "repositoryName": "testrepo"
             },
            "containerTags": ["east1", "imagedist"]
         }
     }
 ],
 "tags": {
    "DistributionConfigurationTestTagKey1": "DistributionConfigurationTestTagValue1",
    "DistributionConfigurationTestTagKey2": "DistributionConfigurationTestTagValue2"
 }
}
```

2. **Run the following command, using the file you created as input.**

```
aws imagebuilder create-distribution-configuration --cli-input-json file://create-container-distribution-configuration.json
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see **create-distribution-configuration** in the *AWS CLI Command Reference*.

Update distribution settings for your container image (AWS CLI)

The following example shows how to use the **update-distribution-configuration** command to update distribution settings for your container image, using the AWS CLI. You can also update tags for the container images within each Region.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with keys shown in the following example, plus values that are valid for your environment. This example uses a file named `update-container-distribution-configuration.json`:

```
{
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-
west-2:123456789012:distribution-configuration/update-container-distribution-
configuration.json",
 "description": "Distributes container image to Amazon ECR repository in two regions.",
 "distributions": [
     {
         "region": "us-west-2",
         "containerDistributionConfiguration": {
             "description": "My test image.",
             "targetRepository": {
                 "service": "ECR",
                 "repositoryName": "testrepo"
             },
             "containerTags": ["west2", "image1"]
         }
     },
     {
         "region": "us-east-2",
         "containerDistributionConfiguration": {
             "description": "My test image.",
             "targetRepository": {
                 "service": "ECR",
                 "repositoryName": "testrepo"
             },
            "containerTags": ["east2", "imagedist"]
         }
     }
 ]
}
```

2. **Run the following command, using the file you created as input:**

```
aws imagebuilder update-distribution-configuration --cli-input-json file://update-
container-distribution-configuration.json
```

Note

- You must include the `file://` notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see **update-distribution-configuration** in the *AWS CLI Command Reference*. To update tags for your distribution configuration resource, see the Tag resources (p. 151) section.

Set up cross-account AMI distribution with Image Builder

This section describes how you can configure distribution settings to deliver an Image Builder AMI to other accounts that you specify.

The destination account can then launch or modify the AMI, as needed.

Note
AWS CLI command examples in this section assume that you have previously created image recipe and infrastructure configuration JSON files. To create the JSON file for an image recipe, see Create an image recipe (AWS CLI) (p. 122). To create the JSON file for an infrastructure configuration, see Create an infrastructure configuration (AWS CLI) (p. 132).

Prerequisites

To ensure that target accounts can successfully launch instances from your Image Builder image, you must configure the appropriate permissions for all destination accounts in all Regions.

If you encrypt your AMI using the AWS Key Management Service (AWS KMS), you must configure an AWS KMS key for your account that is used to encrypt the new image.

When Image Builder performs cross-account distribution for encrypted AMIs, the image in the source account is decrypted and pushed to the target Region, where it is re-encrypted using the designated key for that Region. Because Image Builder acts on behalf of the target account, and uses an IAM role that you create in the destination Region, that account must have access to keys in both the source and destination Regions.

Encryption keys

The following prerequisites are required if your image is encrypted using AWS KMS. IAM prerequisites are covered in the next section.

Source account requirements

- Create a KMS key in your account in all Regions where you build and distribute your AMI. You can also use an existing key.
- Update the key policy for all of those keys to allow destination accounts to use your key.

Destination account requirements

- Add an inline policy to `EC2ImageBuilderDistributionCrossAccountRole` that allows the role to perform the required actions to distribute an encrypted AMI. For IAM configuration steps, see the IAM policies (p. 141) prerequisites section.

For more information about cross-account access using AWS KMS, see Allowing users in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide*.

Specify your encryption key in the image recipe, as follows:

- If you are using the Image Builder console, choose your encryption key from the **Encryption (KMS alias)** dropdown list in the **Storage (volumes)** section of your recipe.
- If you are using the **CreateImageRecipe** API action, or the **create-image-recipe** command in the AWS CLI, configure your key in the `ebs` section under `blockDeviceMappings` in your JSON input.

  The following JSON snippet shows encryption settings for an image recipe. In addition to providing your encryption, you must also set the `encrypted` flag to `true`.

```
{
   ...
   "blockDeviceMappings": [
      {
         "deviceName": "Example root volume",
         "ebs": {
            "deleteOnTermination": true,
            "encrypted": true,
            "iops": 100,
            "kmsKeyId": "image-owner-key-id",
            ...
         },
         ...
      }
   ],
   ...
}
```

IAM policies

To configure cross-account distribution permissions in AWS Identity and Access Management (IAM), follow these steps:

1. To use Image Builder AMIs that are distributed across accounts, the destination account owner must create a new IAM role in their account called `EC2ImageBuilderDistributionCrossAccountRole`.
2. They must attach the Ec2ImageBuilderCrossAccountDistributionAccess policy (p. 192) to the role to enable cross-account distribution. For more information about managed policies, see Managed Policies and Inline Policies in the *AWS Identity and Access Management User Guide*.
3. Verify that the source account ID is added to the trust policy attached to the IAM role of the destination account. For more information about trust policies, see Resource-Based Policies in the *AWS Identity and Access Management User Guide*.
4. If the AMI you distribute is encrypted, the destination account owner must add the following inline policy to the `EC2ImageBuilderDistributionCrossAccountRole` in their account so that they can use your KMS keys. The `Principal` section contains their account number. This enables Image Builder to act on their behalf when it uses AWS KMS to encrypt and decrypt the AMI with the appropriate keys for each Region.

```
{
    "Version": "2012-10-17",
    "Statement": [
     {
      "Sid": "Allow the role to perform AWS KMS operations on behalf of the destination
 account",
      "Effect": "Allow",
      "Action": [
          "kms:Encrypt",
          "kms:Decrypt",
```

```
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey",
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
        ],
        "Resource": "*"
    }
  ]
}
```

For more information about inline policies, see Inline Policies in the *AWS Identity and Access Management User Guide*.

5. If you are using `launchTemplateConfigurations` to specify an Amazon EC2 launch template, you must also add the following policy to your `EC2ImageBuilderDistributionCrossAccountRole` in each destination account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateLaunchTemplateVersion",
                "ec2:ModifyLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/CreatedBy": "EC2 Image Builder"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeLaunchTemplates"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:launch-template/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CreatedBy": "EC2 Image Builder"
                }
            }
        }
    ]
}
```

Limits for cross-account distribution

There are some limitations when distributing Image Builder images across accounts:

- The destination account is limited to 50 concurrent AMI copies for each destination Region.

- If you want to copy a paravirtual (PV) virtualization AMI to another Region, the destination Region must support PV virtualization AMIs. For more information, see Linux AMI virtualization types.
- You cannot create an unencrypted copy of an encrypted snapshot. If you do not specify an AWS Key Management Service (AWS KMS) customer managed key for the `KmsKeyId` parameter, Image Builder uses the default key for Amazon Elastic Block Store (Amazon EBS). For more information, see Amazon EBS Encryption in the *Amazon Elastic Compute Cloud User Guide*.

For more information, see CreateDistributionConfiguration in the *EC2 Image Builder API Reference*.

Configure cross-account distribution for an Image Builder AMI (console)

This section describes how to create and configure distribution settings for cross-account distribution of your Image Builder AMIs using the AWS Management Console. Configuring cross-account distribution requires specific IAM permissions. You must complete the Prerequisites (p. 140) for this section before you continue.

To create distribution settings in the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Distribution settings** from the navigation pane. This shows a list of the distribution settings that are created under your account.
3. At the top of the **Distribution settings** page, choose **Create distribution settings**. This takes you to the **Create distribution settings** page.
4. In the **Image type** section, choose **Amazon Machine Image (AMI)** as the **Output type**. This is the default setting.
5. In the **General** section, enter the **Name** of the distribution settings resource that you want to create (*required*).
6. In the **Region settings** section, enter a 12-digit account ID that you want to distribute your AMI to in **Target accounts** for the selected Region, and press **Enter**. This checks for the correct formatting, and then displays the account ID that you entered below the box. Repeat the process to add more accounts.

   To remove an account that you entered, choose the **X** displayed to the right of the account ID.

   Enter the **Output AMI name** for each Region.
7. Continue specifying any additional settings that you require, and choose **Create settings** to create your new distribution settings resource.

Configure cross-account distribution for an Image Builder AMI (AWS CLI)

This section describes how to configure a distribution settings file and use the **create-image** command in the AWS CLI to build and distribute an Image Builder AMI across accounts.

Configuring cross-account distribution requires specific IAM permissions. You must complete the Prerequisites (p. 140) for this section before you run the **create-image** command.

1. **Configure a distribution settings file**

   Before you use the **create-image** command in the AWS CLI to create an Image Builder AMI that is distributed to another account, you must create a `DistributionConfiguration` JSON structure

that specifies the target account IDs in the `AmiDistributionConfiguration` settings. You must specify at least one `AmiDistributionConfiguration` in the source Region.

The following sample file, named `create-distribution-configuration.json`, shows configuration for cross-account image distribution in the source Region.

```
{
    "name": "cross-account-distribution-example",
    "description": "Cross Account Distribution Configuration Example",
    "distributions": [
        {
            "amiDistributionConfiguration": {
                "targetAccountIds": ["123456789012", "987654321098"],
                "name": "Name {{ imagebuilder:buildDate }}",
                "description": "ImageCopy Ami Copy Configuration"
            },
            "region": "us-west-2"
        }
    ]
}
```

2. **Create the distribution settings**

   To create an Image Builder distribution settings resource using the create-distribution-configuration command in the AWS CLI, provide the following parameters in the command:

   - Enter the name of the distribution in the `--name` parameter.
   - Attach the distribution configuration JSON file you created in the `--cli-input-json` parameter.

```
aws imagebuilder create-distribution-configuration --name my distribution name --cli-input-json file://create-distribution-configuration.json
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

You can also provide JSON directly in the command, using the `--distributions` parameter.

Configure AMI distribution settings to use an Amazon EC2 launch template

To help ensure a consistent launch experience for your Image Builder AMI in target accounts and Regions, you can specify an Amazon EC2 launch template in your distribution settings, using `launchTemplateConfigurations`. When `launchTemplateConfigurations` are present during the build process, Image Builder creates a new version of the launch template that includes all of the original settings from the template, and the new AMI ID from the build. For more information about launching an EC2 instance using a launch template, see one of the following links, depending on your target operating system.

- Launch a Linux instance from a launch template
- Launch a Windows instance from a launch template

Add an Amazon EC2 launch template to your AMI distribution settings (console)

To provide a launch template with your output AMI, follow these steps in the console:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Distribution settings** from the navigation pane. This shows a list of the distribution settings that are created under your account.
3. At the top of the **Distribution settings** page, choose **Create distribution settings**. This opens the **Create distribution settings** page.
4. In the **Image type** section, choose the **Amazon Machine Image (AMI) Output type**. This is the default setting.
5. In the **General** section, enter the **Name** of the distribution settings resource that you want to create (*required*).
6. In the **Region settings** section, select the name of an EC2 launch template from the list. If there are no launch templates in your account, choose **Create new launch template**, which opens the **Launch Templates** in the **EC2 Dashboard**.

   Select the **Set the default version** check box to update the launch template default version to the new version that Image Builder creates with your output AMI.

   To add another launch template to the selected Region, choose **Add launch template configuration**.

   To remove a launch template, choose **Remove**.
7. Continue specifying any additional settings that you require, and choose **Create settings** to create your new distribution settings resource.

Add an Amazon EC2 launch template to your AMI distribution settings (AWS CLI)

This section describes how to configure a distribution settings file with a launch template, and use the **create-image** command in the AWS CLI to build and distribute an Image Builder AMI and a new version of the launch template that uses it.

1. **Configure a distribution settings file**

   Before you can create an Image Builder AMI with a launch template, using the AWS CLI, you must create a distribution configuration JSON structure that specifies the `launchTemplateConfigurations` settings. You must specify at least one `launchTemplateConfigurations` entry in the source Region.

   The following sample file, named `create-distribution-config-launch-template.json`, shows a few possible scenarios for launch template configuration in the source Region.

```
{
    "name": "NewDistributionConfiguration",
    "description": "This is just a test",
    "distributions": [
        {
            "region": "us-west-2",
            "amiDistributionConfiguration": {
                "name": "test-{{imagebuilder:buildDate}}-
{{imagebuilder:buildVersion}}",
                "description": "description"
```

```
            },
            "launchTemplateConfigurations": [
                {
                    "launchTemplateId": "lt-0a1bcde2fgh34567",
                    "accountId": "935302948087",
                    "setDefaultVersion": true
                },
                {
                    "launchTemplateId": "lt-0aaa1bcde2ff3456"
                },
                {
                    "launchTemplateId": "lt-12345678901234567",
                    "accountId": "123456789012"
                }
            ]
        }
    ],
    "clientToken": "clientToken1"
}
```

2. **Create the distribution settings**

   To create an Image Builder distribution settings resource using the create-distribution-configuration command in the AWS CLI, provide the following parameters in the command:

   - Enter the name of the distribution in the `--name` parameter.
   - Attach the distribution configuration JSON file you created in the `--cli-input-json` parameter.

```
aws imagebuilder create-distribution-configuration --name my distribution name--cli-
input-json file://create-distribution-config-launch-template.json
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

You can also provide JSON directly in the command, using the `--distributions` parameter.

Share EC2 Image Builder resources

EC2 Image Builder integrates with AWS Resource Access Manager (AWS RAM) to allow you to share certain resources with any AWS account or through AWS Organizations. EC2 Image Builder resources that can be shared are:

- Components
- Images
- Recipes

This section provides information to help you share these EC2 Image Builder resources.

Section Contents

Working with shared components, images, and recipes in EC2 Image Builder

Component, image, and recipe sharing enables resource owners to share software configurations with other AWS accounts or within an AWS organization. You can manage resource sharing centrally, and define a set of accounts with which the configuration can be shared.

In this model, the AWS account that owns the component, image, or recipe (owners) shares it with other AWS accounts (consumers). Consumers can associate a shared component with their image pipelines to automatically consume updates to the shared component, image, or recipe.

A component, image, or recipe owner can share these resources with:

- Specific AWS accounts inside or outside of its organization in AWS Organizations.
- An organizational unit inside its organization in AWS Organizations.
- Its entire organization in AWS Organizations.

Prerequisites for sharing components, images, and image recipes

To share a component, image, or image recipe:

- You must own the component, image, or image recipe in your AWS account. You cannot share resources that have been shared with you.
- The AWS Key Management Service (AWS KMS) key associated with encrypted resources must be explicitly shared with the target accounts.
- You must enable sharing with AWS Organizations to share these resources with your organization or an organizational unit in AWS Organizations. For more information, see Enable Sharing with AWS Organizations in the *AWS Resource Access Manager User Guide*.
- You are responsible for ensuring that dependencies external to these resources, or underlying resources that are managed outside of AWS, are also shared with consumers. EC2 Image Builder does not manage dependencies external to EC2 Image Builder. This includes the Amazon EC2 AMIs associated with EC2 Image Builder images, which require account-level sharing. This can be configured in the distribution configuration associated with your Image Builder workflows.
- If you distribute an image encrypted with AWS KMS across accounts in different Regions, you must create an KMS key and alias in each target Region. Additionally, the people who will be launching instances in those Regions will need access to the KMS key specified via the Key Policy.

Related services

AWS Resource Access Manager

Component, image, and image recipe sharing integrates with AWS Resource Access Manager (AWS RAM). AWS RAM is a service that enables you to share your AWS resources with any AWS account or through AWS Organizations. With AWS RAM, you share resources that you own by creating a resource share. A resource share specifies the resources to share and the consumers with whom to share them. Consumers can be individual AWS accounts, organizational units, or an entire organization in AWS Organizations.

For more information about AWS RAM, see the *AWS RAM User Guide*.

Sharing across Regions

Shared components, images, and image recipes can be shared only in a specified AWS Region. When you share these resources, they will not replicate across Regions.

Sharing a component, image, or image recipe

To share a component, image, or image recipe, you must add it to a resource share. A resource share is an AWS Resource Access Manager resource that lets you share your resources across AWS accounts. A resource share specifies the resources to share and the consumers with whom they are shared. To add the component, image, or image recipe to a new resource share, you must first create the resource share using the AWS Resource Access Manager console.

If you are part of an organization in AWS Organizations and sharing within your organization is enabled, consumers in your organization are automatically granted access to the shared component, image, or image recipe. Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared resource after accepting the invitation.

You can share a component, image, or recipe that you own using the AWS Resource Access Manager console or the AWS CLI.

To share a component, image, or recipe that you own using the AWS Resource Access Manager console

See Creating a Resource Share in the AWS Resource Access Manager User Guide.

To share a component, image, or image recipe that you own using the AWS CLI

Use the `create-resource-share` command.

Using resource-based policies to share a component, image, or recipe

The `PutImagePolicy`, `PutComponentPolicy`, and `PutImageRecipePolicy` APIs provided by the Image Builder service allow you to define resource policies for images, components, and image recipes, respectively. AWS RAM uses these APIs to define the correct resource policies to allow the consumer with whom a resource is shared to make API calls involving the shared resource. For AWS RAM to set the correct policies to share and unshare a resource, the resource owner must have `imagebuilder:put*` permissions.

Apply a resource policy to an image

You can apply a resource policy to an image to allow other users to use the image in their image recipes. For the command to be successful, you must ensure that the account with which you are sharing has permission to access the underlying resource (for example, the Amazon EC2 AMI). We recommend that you use the RAM CLI command create-resource-share to share resources. If you use the EC2 Image

Builder CLI command put-image-policy, you must also use the RAM CLI command promote-resource-share-created-from-policy for the resource to be visible to all principals with whom the resource is shared. For more information, see Share EC2 Image Builder resources (p. 146).

```
aws imagebuilder put-image-policy --image-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-image/2019.12.03/1 --policy '{ "Version":
 "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": { "AWS":
 [ "123456789012" ] }, "Action": ["imagebuilder:GetImage", "imagebuilder:ListImages"],
 "Resource": [ "arn:aws:imagebuilder:us-west-2:123456789012:image/my-example-
image/2019.12.03/1" ] } ] }'
```

Apply a resource policy to a component

You can apply a resource policy to a build component to enable cross-account sharing of build components. This command gives other accounts permission to use your build component in their image recipes. For the command to be successful, you must ensure that the account with which you are sharing has permission to access any resources referenced by the shared build component, such as files hosted on private repositories. We recommend that you use the RAM CLI command create-resource-share to share resources. If you use the EC2 Image Builder CLI command put-component-policy, you must also use the RAM CLI command promote-resource-share-created-from-policy for the resource to be visible to all principals with whom the resource is shared. For more information, see Share EC2 Image Builder resources (p. 146).

```
aws imagebuilder put-component-policy --component-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.03/1 --policy
 '{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal":
 { "AWS": [ "123456789012" ] }, "Action": [ "imagebuilder:GetComponent",
 "imagebuilder:ListComponents" ], "Resource": [ "arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.03/1" ] } ] }'
```

Apply a resource policy to a recipe

You can apply a resource policy to an image recipe to enable cross-account sharing of image recipes. This command gives other accounts permission to use your image recipes to create images in their accounts. For the command to be successful, you must ensure that the account with which you are sharing has permission to access any images or components referenced by the image recipe. We recommend that you use the RAM CLI command create-resource-share to share resources. If you use the EC2 Image Builder CLI command put-image-recipe-policy, you must also use the RAM CLI command promote-resource-share-created-from-policy for the resource to be visible to all principals with whom the resource is shared. For more information, see Share EC2 Image Builder resources (p. 146).

```
aws imagebuilder put-image-recipe-policy --image-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-image-recipe/2019.12.03 --policy
 '{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal":
 { "AWS": [ "123456789012" ] }, "Action": [ "imagebuilder:GetImageRecipe",
 "imagebuilder:ListImageRecipes" ], "Resource": [ "arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-image-recipe/2019.12.03" ] } ] }'
```

Unsharing a shared component, image, or recipe

To unshare a shared component, image, or recipe that you own, you must remove it from the resource share. You can do this using the AWS Resource Access Manager console or the AWS CLI.

Note
To unshare a component, image, or recipe, the consumer cannot have any dependencies on them. The consumer must remove any dependencies on the shared resources before the owner can unshare them.

To unshare a shared component, image, or recipe that you own using the AWS Resource Access Manager console

See Updating a Resource Share in the AWS Resource Access Manager User Guide.

To unshare a shared component, image, or recipe that you own using the AWS CLI

Use the `disassociate-resource-share` command.

Identifying a shared component, image, or recipe

Owners and consumers can identify shared components, images, and image recipes using Image Builder commands in the AWS CLI.

Identify a shared component

Run the list-components command to get a list of the components that you own and the components that are shared with you. The get-component command shows the AWS account ID of the component owner.

Identify a shared image

Run the list-images command to get a list of the images that you own and images that are shared with you. The get-image command shows the AWS account ID of the image owner.

Identify a shared container image

Run the list-images command to get a list of the images that you own and images that are shared with you. The get-image command shows the AWS account ID of the image owner.

Identify a shared image recipe

Run the list-image-recipes command to get a list of the image recipes that you own and image recipes that are shared with you. The get-image-recipe command shows the AWS account ID of the image recipe owner.

Identify a shared container recipe

Run the list-container-recipes command to get a list of the container recipes that you own and container recipes that are shared with you. The get-container-recipe command shows the AWS account ID of the container recipe owner.

Shared component, image, and recipe permissions

Permissions for owners

Owners cannot delete a shared component, image, or image recipe until it is no longer shared. An owner cannot unshare these resources until none of the consumers depend on them.

Permissions for consumers

Consumers can read a component, image, or image recipe, but cannot modify them in any way. They cannot view or modify these resources if they are owned by other consumers or the owner of the resource. Consumers can use shared components and images in image recipes to create custom images. Consumers can use shared image recipes to create their own custom images.

Billing and metering

There is no charge to use EC2 Image Builder.

Instance limits

Shared components, images, and image recipes count toward the corresponding resource limits of the owner only. The resource limits of the consumers are not affected by the resources that have been shared with them.

Tag EC2 Image Builder resources

Tagging your resources can be useful for filtering and tracking resource costs, or other categories. You can also control access based on tags. For more information about tag-based authorization, see Authorization based on Image Builder tags (p. 181)

Image Builder supports the following dynamic tags:

- `- {{imagebuilder:buildDate}}`

  Resolves to the build date/time at build time.
- `- {{imagebuilder:buildVersion}}`

  Resolves to a build version, which is a number that is located at the end of an Image Builder ARN. For example, `"arn:aws:imagebuilder:us-west-2:123456789012:component/myexample-component/2019.12.02/1"` shows the build version as `1`.

Contents

Tag a resource (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to to add and tag a resource in EC2 Image Builder. You must provide the `resourceArn` and the tags to apply to it.

The example `tag-resource.json` contents are as follows:

```
{
    "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
    "tags": {
        "KeyName": "KeyValue"
    }
}
```

Run the following command, which references the preceding `tag-resource.json` file.

```
aws imagebuilder tag-resource --cli-input-json file://tag-resource.json
```

Untag a resource (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to remove a tag from a resource. You must provide the `resourceArn` and the keys to remove the tag.

The example `untag-resource.json` contents are as follows:

```
{
    "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
    "tagKeys": [
        "KeyName"
    ]
}
```

Run the following command, which references the preceding `untag-resource.json` file.

```
aws imagebuilder untag-resource --cli-input-json file://untag-resource.json
```

List all of the tags for a specific resource (AWS CLI)

The following example shows how to use an **imagebuilder** CLI command to list all the tags for a specific resource.

```
aws imagebuilder list-tags-for-resource --resource-arn arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline
```

Delete EC2 Image Builder resources

Your Image Builder environment, just like your home, needs regular maintenance to help you find what you need, and complete your tasks without wading through clutter. Make sure to regularly clean up temporary resources that you created for testing. Otherwise, you might forget about those resources, and then later, not remember what they were used for. By then, it might not be clear if you can safely get rid of them.

Tip
To prevent dependency errors when you delete resources, make sure to delete your resources in the following order:

1. Image pipeline
2. Image recipe
3. All remaining resources

Delete resources using the AWS Management Console

To delete an image pipeline and its resources, follow these steps:

Delete the pipeline

1. To see a list of the build pipelines created under your account, choose **Image pipelines** from the navigation pane.
2. Select the check box next to **Pipeline name** to select the pipeline that you want to delete.
3. At the top of the **Image pipelines** panel, on the **Actions** menu, choose **Delete**.

4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete the recipe

1. To see a list of the recipes created under your account, choose **Image recipes** from the navigation pane.
2. Select the check box next to **Recipe name** to select the recipe that you want to delete.
3. At the top of the **Image recipes** panel, on the **Actions** menu, choose **Delete recipe**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete infrastructure configuration

1. To see a list of the infrastructure configurations created under your account, choose **Infrastructure configuration** from the navigation pane.
2. Select the check box next to **Configuration name** to select the infrastructure configuration that you want to delete.
3. At the top of the **Infrastructure configurations** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete distribution settings

1. To see a list of the distribution settings created under your account, choose **Distribution settings** from the navigation pane.
2. Select the check box next to **Configuration name** to select the distribution settings that you created for this tutorial.
3. At the top of the **Distribution settings** panel, choose **Delete**.
4. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete an image

1. To see a list of the images created under your account, choose **Images** from the navigation pane.
2. Choose the image **Version** for the image that you want to remove. This opens the **Image build versions** page.
3. Select the check box next to the **Version** for any image that you want to delete. You can select more than one image version at a time.
4. At the top of the **Image build versions** panel, choose **Delete version**.
5. To confirm the deletion, enter `Delete` in the box, and choose **Delete**.

Delete an image pipeline using the AWS CLI

The following examples show how to delete Image Builder resources using the AWS CLI. As mentioned previously, resources must be deleted in the following order to avoid dependency errors:

1. Image pipeline
2. Image recipe
3. All remaining resources

Delete image pipeline (AWS CLI)

The following example shows how to delete an image pipeline by specifying its ARN.

```
aws imagebuilder delete-image-pipeline --image-pipeline-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-pipeline/my-example-pipeline
```

Delete image recipe (AWS CLI)

The following example shows how to delete an image recipe by specifying its ARN.

```
aws imagebuilder delete-image-recipe --image-recipe-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-recipe/my-example-recipe/2019.12.03
```

Delete an infrastructure configuration

The following example shows how to delete an infrastructure configuration resource by specifying its ARN.

```
aws imagebuilder delete-infrastructure-configuration --infrastructure-configuration-arn
 arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/my-example-
infrastructure-configuration
```

Delete distribution settings

The following example shows how to delete a distribution settings resource by specifying its ARN.

```
aws imagebuilder delete-distribution-configuration --distribution-configuration-arn
 arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-example-
distribution-configuration
```

Delete an image

The following example shows how to delete an image build version by specifying its ARN.

```
aws imagebuilder delete-image --image-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/my-example-image/2019.12.02/1
```

Delete a component

The following example shows how to use an **imagebuilder** CLI command to delete a component build version by specifying its ARN.

```
aws imagebuilder delete-component --component-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/my-example-component/2019.12.02/1
```

> **Important**
> Make sure there are no recipes that reference the component build version in any way before you delete it. Failing to do so could cause pipeline failures.

Manage EC2 Image Builder pipelines using the console

Image Builder image pipelines provide an automation framework for creating and maintaining custom AMIs and container images. Pipelines deliver the following functionality:

- Assemble the base image, components for building and testing, infrastructure configuration, and distribution settings.
- Facilitate scheduling for automated maintenance processes using the `Schedule builder` in the console wizard, or entering cron expressions for recurring updates to your images.
- Enable change detection for the base image and components, to automatically skip scheduled builds when there are no changes.
- Enable rule-based automation through Amazon EventBridge.

  Note
  For more information about using the EventBridge API to view or change rules, see the Amazon EventBridge API Reference. For more information about using EventBridge **events** commands in the AWS CLI to view or change rules, see events in the *AWS CLI Command Reference*.

Contents

List and view pipeline details

This section describes the various ways that you can find information and view details for your EC2 Image Builder image pipelines.

Pipeline details

List image pipelines (AWS CLI)

The following example shows how to use the **imagebuilder list-image-pipelines** command in the AWS CLI to list all of your image pipelines.

```
aws imagebuilder list-image-pipelines
```

Get image pipeline details (AWS CLI)

The following example shows how to use the **imagebuilder get-image-pipeline** command in the AWS CLI to get the details of an image pipeline by specifying its ARN.

```
aws imagebuilder get-image-pipeline --image-pipeline-arn arn:aws:imagebuilder:us-
west-2:123456789012:image-pipeline/my-example-pipeline
```

Create and update AMI image pipelines

You can set up, configure, and manage AMI image pipelines using the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI. The **Create image pipeline** console wizard provides starting artifacts, and guides you through steps to:

- Select a base image from quick-start managed images, or existing images in your account or that were shared with you from another account.
- Add and remove software
- Customize settings and scripts
- Run selected tests
- Distribute images to AWS Regions

For more information and a step-by-step tutorial about using the **Create image pipeline** console wizard, see Create an image pipeline using the EC2 Image Builder console wizard (p. 14).

Contents

Create an AMI image pipeline (AWS CLI)

You can create an AMI image pipeline using a JSON file as input to the **imagebuilder create-image-pipeline** command in the AWS CLI.

How often your pipeline builds a new image to incorporate any pending updates from your base image and components depends on the `schedule` that you have configured. A `schedule` has the following attributes:

- `scheduleExpression` – Sets the schedule for when your pipeline runs to evaluate the `pipelineExecutionStartCondition` and determine if it should start a build. The schedule is configured with cron expressions. For more information on how to format a cron expression in Image Builder, see Use cron expressions in EC2 Image Builder (p. 166).
- `pipelineExecutionStartCondition` – Determines if your pipeline should start the build. Valid values include:
  - `EXPRESSION_MATCH_ONLY` – your pipeline will build a new image every time the cron expression matches the current time.
  - `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE` – your pipeline will not start a new image build unless there are pending changes to your base image or components.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the following keys, plus values that are valid for your environment. This example uses a file named `create-image-pipeline.json`:

```
{
    "name": "MyWindows2019Pipeline",
    "description": "Builds Windows 2019 Images",
    "enhancedImageMetadataEnabled": true,
    "imageRecipeArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-recipe/my-example-recipe/2020.12.03",
    "infrastructureConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/my-example-infrastructure-configuration",
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-example-distribution-configuration",
    "imageTestsConfiguration": {
        "imageTestsEnabled": true,
        "timeoutMinutes": 60
    },
    "schedule": {
        "scheduleExpression": "cron(0 0 * * SUN *)",
        "pipelineExecutionStartCondition":
 "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
    },
    "status": "ENABLED"
}
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws imagebuilder create-image-pipeline --cli-input-json file://create-image-pipeline.json
```

Update AMI image pipelines (console)

After you have created an Image Builder image pipeline for your AMI image, you can make changes to the infrastructure configuration and distribution settings from the Image Builder console.

To update an image pipeline with a new image recipe, you must use the AWS CLI. For more information, see Update AMI image pipelines (AWS CLI) (p. 159) in this guide.

Choose an existing Image Builder pipeline

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. To see a list of the image pipelines created under your account, choose **Image pipelines** from the navigation pane.

   Note
   The list of image pipelines includes an indicator for the type of output image that is created by the pipeline – AMI or Docker.

3. To view details or edit a pipeline, choose the **Pipeline name** link. This opens the detail view for the pipeline.

Note
You can also select the check box next to the **Pipeline name**, then choose **View detail**.

Pipeline details

The pipeline details page includes the following sections:

Summary

The section at the top of the page summarizes key details for the pipeline that are visible with any of the detail tabs open. The details displayed in this section are editable only on their respective detail tabs.

Detail tabs

- **Output images** – Shows output images that the pipeline has produced.
- **Image recipe** – Shows recipe details. After you create a recipe, you cannot edit it. You must create a new version of the recipe from the **Image recipes** page in the Image Builder console, or by using Image Builder commands in the AWS CLI. For more information, see Manage recipes (p. 118).
- **Infrastructure configuration** – Shows editable information for configuring your build pipeline infrastructure.
- **Distribution settings** – Shows editable information for AMI distribution.
- **EventBridge rules** – For the selected **Event Bus**, shows EventBridge rules that target the current pipeline. Includes **Create event bus** and **Create rule** actions that link to the EventBridge console. For more information about this tab, see Use EventBridge rules (p. 170).

Edit infrastructure configuration for your pipeline

Infrastructure configuration includes the following details that you can edit after creating the pipeline:

- The **Description** for your infrastruction configuration.
- The **IAM role** to associate with the instance profile.
- **AWS infrastructure**, including the **Instance type** and an **SNS topic** for notifications.
- **VPC, subnet, and security groups**.
- **Troubleshooting settings**, including **Terminate instance on failure**, the **Key pair** for connecting, and an optional S3 bucket location for instance logs.

To edit infrastructure configuration from the pipeline details page, follow these steps:

1. Choose the **Infrastructure configuration** tab.
2. Choose **Edit** from the upper right corner of the **Configuration details** panel.
3. When you are ready to save updates you've made to your infrastructure configuration, choose **Save changes**.

Edit distribution settings for your pipeline

Distribution settings include the following details that you can edit after creating the pipeline:

- The **Description** for your distribution settings.
- **Region settings** for the Regions where you distribute your image. Region 1 defaults to the Region where you created the pipeline. You can add Regions for distribution with the **Add Region** button, and you can remove all Regions except Region 1.

Region settings include:

- Target **Region**
- The **Output AMI name**
- **Launch permissions**, and accounts to share them with
- Associated licenses (**Associate license configurations**)

> **Note**
> License Manager settings will not replicate across AWS Regions that must be enabled in your account, for example, between the `ap-east-1` (Hong Kong) and the `me-south-1` (Bahrain) Regions.

To edit your distribution settings from the pipeline details page, follow these steps:

1. Choose the **Distribution settings** tab.
2. Choose **Edit** from the upper right corner of the **Distribution details** panel.
3. When you are ready to save your updates, choose **Save changes**.

Edit the build schedule for your pipeline

The **Edit pipeline** page includes the following details that you can edit after creating the pipeline:

- The **Description** for your pipeline.
- **Enhanced metadata collection**. This is turned on by default. To turn it off, clear the **Enable enhanced metadata collection** check box.
- The **Build schedule** for your pipeline. You can change your **Schedule options** and all of the settings here.

To edit your pipeline from the pipeline details page, follow these steps:

1. In the upper right corner of the pipeline details page, choose **Actions**, and then **Edit pipeline**.
2. When you are ready to save your updates, choose **Save changes**.

> **Note**
> For more information about scheduling your build using cron expressions, see Use cron expressions in EC2 Image Builder (p. 166).

Update AMI image pipelines (AWS CLI)

You can update an AMI image pipeline using a JSON file as input to the **imagebuilder update-image-pipeline** command in the AWS CLI. To configure the JSON file, you must have Amazon Resource Names (ARNs) to reference the following existing resources:

- Image pipeline to update
- Image recipe
- Infrastructure configuration
- Distribution settings

Follow these steps to update an AMI image pipeline using the **imagebuilder update-image-pipeline** command in the AWS CLI:

Note
UpdateImagePipeline does not support selective updates for the pipeline. You must specify all of the required properties in the update request, not just the properties that have changed.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the following keys, plus values that are valid for your environment. This example uses a file named `create-component.json`:

```
    {
    "imagePipelineArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
    "imageRecipeArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-recipe/my-example-recipe/2019.12.08",
    "infrastructureConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/my-example-infrastructure-configuration",
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-example-distribution-configuration",
    "imageTestsConfiguration": {
        "imageTestsEnabled": true,
        "timeoutMinutes": 120
    },
    "schedule": {
        "scheduleExpression": "cron(0 0 * * MON *)",
        "pipelineExecutionStartCondition":
 "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
    },
    "status": "DISABLED"
}
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws imagebuilder update-image-pipeline --cli-input-json file://update-image-pipeline.json
```

Run your AMI image pipeline

If you chose the manual schedule option for your pipeline, it will only run when you manually kick off the build. If you chose one of the automatic scheduling options, you can also run it manually, in between regularly scheduled runs. For example, if you have a pipeline that normally runs once a month, but you need to incorporate an update to one of your components two weeks after the prior run, you can choose to run your pipeline manually.

Console

To run your pipeline from the pipeline details page in the Image Builder console, choose **Run pipeline** from the **Actions** menu at the top of the page. A status message appears at the top of the page to notify you that your pipeline has started, or if there is an error.

1. In the upper left corner of the pipeline details page, choose **Run pipeline**, from the **Actions** menu.

2. You can see the current status of your pipeline on the **Output images** tab, in the **Status** column.

AWS CLI

The following example shows how to use the **imagebuilder start-image-pipeline-execution** command in the AWS CLI to manually start an image pipeline. Running this command results in the pipeline creating a new image on demand.

```
aws imagebuilder start-image-pipeline-execution --image-pipeline-arn
 arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline
```

To see what resources are created when the build pipeline runs, see Resources created (p. 10).

Create and update container image pipelines

You can set up, configure, and manage container image pipelines using the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI. The **Create image pipeline** console wizard provides starting artifacts, and guides you through steps to:

- Select a base image from quick-start managed images, Amazon ECR, or Docker Hub repositories
- Add and remove software
- Customize settings and scripts
- Run selected tests
- Create a Dockerfile using pre-configured build-time variables.
- Distribute images to AWS Regions

For more information and a step-by-step tutorial about using the **Create image pipeline** console wizard, see Create a container image pipeline using the EC2 Image Builder console wizard (p. 18).

Contents

Create a container image pipeline (AWS CLI)

You can create a container image pipeline using a JSON file as input to the **imagebuilder create-image-pipeline** command in the AWS CLI.

How often your pipeline builds a new image to incorporate any pending updates from your base image and components depends on the `schedule` that you have configured. A `schedule` has the following attributes:

- `scheduleExpression` – Sets the schedule for when your pipeline runs to evaluate the `pipelineExecutionStartCondition` and determine if it should start a build. The schedule is configured with cron expressions. For more information on how to format a cron expression in Image Builder, see Use cron expressions in EC2 Image Builder (p. 166).
- `pipelineExecutionStartCondition` – Determines if your pipeline should start the build. Valid values include:

- `EXPRESSION_MATCH_ONLY` – your pipeline will build a new image every time the cron expression matches the current time.
- `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE` – your pipeline will not start a new image build unless there are pending changes to your base image or components.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the following keys, plus values that are valid for your environment. This example uses a file named `create-image-pipeline.json`:

```
{
    "name": "MyWindows2019Pipeline",
    "description": "Builds Windows 2019 Images",
    "enhancedImageMetadataEnabled": true,
    "containerRecipeArn": "arn:aws:imagebuilder:us-west-2:123456789012:container-
recipe/my-example-recipe/2020.12.03",
    "infrastructureConfigurationArn": "arn:aws:imagebuilder:us-
west-2:123456789012:infrastructure-configuration/my-example-infrastructure-
configuration",
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-
west-2:123456789012:distribution-configuration/my-example-distribution-configuration",
    "imageTestsConfiguration": {
        "imageTestsEnabled": true,
        "timeoutMinutes": 60
    },
    "schedule": {
        "scheduleExpression": "cron(0 0 * * SUN *)",
        "pipelineExecutionStartCondition":
 "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
    },
    "status": "ENABLED"
}
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws imagebuilder create-image-pipeline --cli-input-json file://create-image-
pipeline.json
```

Update a container image pipeline (console)

After you have created an Image Builder container image pipeline for your Docker image, you can make changes to the infrastructure configuration and distribution settings from the Image Builder console.

To update a container image pipeline with a new container recipe, you must use the AWS CLI. For more information, see Update container image pipelines (AWS CLI) (p. 164) in this guide.

Choose an existing Image Builder Docker image pipeline

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.

2. To see a list of the image pipelines created under your account, choose **Image pipelines** from the navigation pane.

   Note
   The list of image pipelines includes an indicator for the type of output image that is created by the pipeline – AMI or Docker.

3. To view details or edit a pipeline, choose the **Pipeline name** link. This opens the detail view for the pipeline.

   Note
   You can also select the check box next to the **Pipeline name**, then choose **View detail**.

Pipeline details

The EC2 Image Builder pipeline details page includes the following sections:

Summary

The section at the top of the page summarizes key details for the pipeline that are visible with any of the detail tabs open. The details displayed in this section are editable only on their respective detail tabs.

Detail tabs

- **Output images** – Shows output images that the pipeline has produced.
- **Container recipe** – Shows recipe details. After you create a recipe, you cannot edit it. You must create a new version of the recipe from the **Container recipes** page. For more information, see Create container recipes and versions (p. 125).
- **Infrastructure configuration** – Shows editable information for configuring your build pipeline infrastructure.
- **Distribution settings** – Shows editable information for Docker image distribution.
- **EventBridge rules** – For the selected **Event Bus**, shows EventBridge rules that target the current pipeline. Includes **Create event bus** and **Create rule** actions that link to the EventBridge console. For more information about this tab, see Use EventBridge rules (p. 170).

Edit infrastructure configuration for your pipeline

Infrastructure configuration includes the following details that you can edit after creating the pipeline:

- The **Description** for your infrastruction configuration.
- The **IAM role** to associate with the instance profile.
- **AWS infrastructure**, including the **Instance type** and an **SNS topic** for notifications.
- **VPC, subnet, and security groups**.
- **Troubleshooting settings**, including **Terminate instance on failure**, the **Key pair** for connecting, and an optional S3 bucket location for instance logs.

To edit infrastructure configuration from the pipeline details page, follow these steps:

1. Choose the **Infrastructure configuration** tab.
2. Choose **Edit** from the upper right corner of the **Configuration details** panel.
3. When you are ready to save updates you've made to your infrastructure configuration, choose **Save changes**.

Edit distribution settings for your pipeline

Distribution settings include the following details that you can edit after creating the pipeline:

- The **Description** for your distribution settings.
- **Region settings** for the Regions where you distribute your image. Region 1 defaults to the Region where you created the pipeline. You can add Regions for distribution with the **Add Region** button, and you can remove all Regions except Region 1.

  Region settings include:
  - Target **Region**
  - The **Service** defaults to "ECR", and is not editable.
  - **Repository name** – the name of your target repository (*not including the Amazon ECR location*). For example, the repository name with the location would look like the following pattern:

    `<account-id>.dkr.ecr.<region>.amazonaws.com/<repository-name>`

    Note
    If you change the **Repository name**, only the images created after the name change will be added under the new name. Any prior images that your pipeline created remain in their original repository.

To edit your distribution settings from the pipeline details page, follow these steps:

1. Choose the **Distribution settings** tab.
2. Choose **Edit** from the upper right corner of the **Distribution details** panel.
3. When you are ready to save updates you've made to your distribution settings, choose **Save changes**.

Edit the build schedule for your pipeline

The **Edit pipeline** page includes the following details that you can edit after creating the pipeline:

- The **Description** for your pipeline.
- **Enhanced metadata collection**. This is turned on by default. To turn it off, clear the **Enable enhanced metadata collection** check box.
- The **Build schedule** for your pipeline. You can change your **Schedule options** and all of the settings in this section.

To edit your pipeline from the pipeline details page, follow these steps:

1. In the upper right corner of the pipeline details page, choose **Actions**, and then **Edit pipeline**.
2. When you are ready to save your updates, choose **Save changes**.

Note
For more information about scheduling your build using cron expressions, see Use cron expressions in EC2 Image Builder (p. 166).

Update container image pipelines (AWS CLI)

You can update a container image pipeline using a JSON file as input to the **imagebuilder update-image-pipeline** command in the AWS CLI. To configure the JSON file, you must have Amazon Resource Names (ARNs) to reference the following existing resources:

- Image pipeline to update
- Container recipe
- Infrastructure configuration
- Distribution settings

Follow these steps to update a container image pipeline using the **imagebuilder update-image-pipeline** command in the AWS CLI:

Note
UpdateImagePipeline does not support selective updates for the pipeline. You must specify all of the required properties in the update request, not just the properties that have changed.

1. **Create a CLI input JSON file**

   Use your favorite file editing tool to create a JSON file with the following keys, plus values that are valid for your environment. This example uses a file named `create-component.json`:

```
    {
    "imagePipelineArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
    "containerRecipeArn": "arn:aws:imagebuilder:us-west-2:123456789012:container-recipe/my-example-recipe/2020.12.08",
    "infrastructureConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/my-example-infrastructure-configuration",
    "distributionConfigurationArn": "arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-example-distribution-configuration",
    "imageTestsConfiguration": {
        "imageTestsEnabled": true,
        "timeoutMinutes": 120
    },
    "schedule": {
        "scheduleExpression": "cron(0 0 * * MON *)",
        "pipelineExecutionStartCondition":
 "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
    },
    "status": "DISABLED"
}
```

   Note

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws imagebuilder update-image-pipeline --cli-input-json file://update-image-pipeline.json
```

Run your container image pipeline

If you chose the manual schedule option for your pipeline, it will only run when you manually kick off the build. If you chose one of the automatic scheduling options, you can also run it manually, in between regularly scheduled runs. For example, if you have a pipeline that normally runs once a month, but you

need to incorporate an update to one of your components two weeks after the prior run, you can choose to run your pipeline manually.

Console

To run your pipeline from the pipeline details page in the Image Builder console, choose **Run pipeline** from the **Actions** menu at the top of the page. A status message appears at the top of the page to notify you that your pipeline has started, or if there is an error.

1. In the upper left corner of the pipeline details page, choose **Run pipeline**, from the **Actions** menu.
2. You can see the current status of your pipeline on the **Output images** tab, in the **Status** column.

AWS CLI

The following example shows how to use the **imagebuilder start-image-pipeline-execution** command in the AWS CLI to manually start an image pipeline. Running this command results in the pipeline creating a new image on demand.

```
aws imagebuilder start-image-pipeline-execution --image-pipeline-arn
 arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline
```

To see what resources are created when the build pipeline runs, see Resources created (p. 10).

Use cron expressions in EC2 Image Builder

Use cron expressions for EC2 Image Builder to set up a time window to refresh your image with updates that apply to your pipeline's base image and components. The time window for your pipeline refresh starts with the time you set in the cron expression. You can set the time in your cron expression down to the minute. Your pipeline build can run on or after the start time.

It can sometimes take a few seconds, or up to a minute for your build to start running.

Note
Cron expressions use Universal Coordinated Time (UTC). For more information about UTC time, and to find the offset for your time zone, see Time Zone Abbreviations – Worldwide List.

Supported values for cron expressions in Image Builder

EC2 Image Builder uses a cron format that consists of six required fields. Each one is separated from the others by a space in between, with no leading or trailing spaces:

```
<Minute> <Hour> <Day> <Month> <Day of the week> <Year>
```

The following table shows supported values for required cron entries.

Supported values for cron expressions

Field	Values	Wildcards
Minute	0-59	, - * /
Hour	0-23	, - * /

Field	Values	Wildcards
Day	`1-31`	`, - * ? / L W`
Month	`1-12` or `jan-dec`	`, - * /`
Day of the week	`1-7` or `sun-sat`	`, - * ? L #`
Year	`1970-2199`	`, - * /`

Wildcards

The following table describes how Image Builder uses wildcards for cron expressions. Keep in mind that it can take up to a minute after the time you specify for the build to start.

Supported wildcards for cron expressions

Wildcard	Description
,	The **,** (comma) wildcard includes additional values. In the Month field, `jan,feb,mar` includes January, February, and March.
-	The **-** (dash) wildcard specifies ranges. In the day of the month field, `1-15` includes days 1 through 15 of the specified month.
*****	The ***** (asterisk) wildcard includes all valid values for the field.
?	The `?` (question mark) wildcard specifies that the field value depends on another setting. In the case of the Day and Day-of-week fields, when one is specified or includes all possible values (`*`), the other must be a `?`. You cannot specify both. For example, if you enter a `7` in the Day field (run the build on the seventh day of the month), the Day-of-week position must contain a `?`.
/	The **/** (forward slash) wildcard specifies increments. For example, if you want your build to run every other day, enter `*/2` in the day field.
L	The **L** wildcard in either of the day fields, specifies the *last* day: 28-31 for the day of the month, depending on what the month is, or Sunday, for the day of the week.
W	The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, if you enter a number prior to the `W`, that means you want to target the weekday that is closest to that day. For instance, if you specify `3W`, you want your build to run on the weekday closest to the third day of the month.
#	The **#** (hash) is allowed only for the day of the week field, and must be followed by a number between 1 and 5. The number specifies which

Wildcard	Description
	weeks in a given month apply for the build to run. For example, if you want your build to run on the second Friday of each month, use `fri#2` for the day of the week field. If you want your job to run twice a month, on the second and fourth Saturday of the month, you would use `sat#2,4` for the day of the week field.

Restrictions

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or `*` in one of these fields, you must use a `?` in the other.
- Cron expressions that lead to rates faster than one minute are not supported.

Examples of cron expressions in EC2 Image Builder

Cron expressions are entered differently for the Image Builder console, than they are for the API or CLI. To see examples, choose the tab that applies to you.

Image Builder console

The following examples show cron expressions that you can enter into the console for your build schedule. UTC time is specified using a 24-hour clock.

Run daily at 10:00 AM (UTC)

```
0 10 * * ? *
```

Run daily at 12:15 PM (UTC)

```
15 12 * * ? *
```

Run daily at midnight (UTC)

```
0 0 * * ? *
```

Run at 10:00 AM (UTC) every weekday morning

```
0 10 ? * 2-6 *
```

Run at 6 PM (UTC) every weekday evening

```
0 18 ? * mon-fri *
```

Run at 8:00 AM (UTC) on the first day of every month

```
0 8 1 * ? *
```

Run on the second Tuesday of every month at 10:30 PM (UTC)

```
30 22 ? * tue#2 *
```

Tip
If you don't want your pipeline job to extend into the next day while it's running, make sure that you factor in time for your build when you specify the start time.

API/CLI

The following examples show cron expressions that you can enter for your build schedule using CLI commands or API requests. Only the cron expression is shown.

Run daily at 10:00 AM (UTC)

```
cron(0 10 * * ? *)
```

Run daily at 12:15 PM (UTC)

```
cron(15 12 * * ? *)
```

Run daily at midnight (UTC)

```
cron(0 0 * * ? *)
```

Run at 10:00 AM (UTC) every weekday morning

```
cron(0 10 ? * 2-6 *)
```

Run at 6:00 PM (UTC) every weekday evening

```
cron(0 18 ? * mon-fri *)
```

Run at 8:00 AM (UTC) on the first day of every month

```
cron(0 8 1 * ? *)
```

Run on the second Tuesday of every month at 10:30 PM (UTC)

```
cron(30 22 ? * tue#2 *)
```

Tip
If you don't want your pipeline job to extend into the next day while it's running, make sure that you factor in time for your build when you specify the start time.

Rate expressions in EC2 Image Builder

A rate expression starts when you create the scheduled event rule, and then runs on its defined schedule.

Rate expressions have two required fields. Fields are separated by white space.

Syntax

```
rate(value unit)
```

value

A positive number.

unit

The unit of time. Different units are required for values of 1, such as `minute`, and values over 1, such as `minutes`.

Valid values: minute | minutes | hour | hours | day | days

Restrictions

If the value is equal to `1`, then the unit must be singular. Similarly, for values greater than `1`, the unit must be plural. For example, `rate(1 hours)` and `rate(5 hour)` are not valid, but `rate(1 hour)` and `rate(5 hours)` are valid.

Use EventBridge rules with Image Builder pipelines

Events from a wide range of AWS and partner services are streamed to Amazon EventBridge event buses in near real-time. You can also generate custom events, and send events from your own applications to EventBridge. The event buses use rules to determine where to route event data.

Image Builder pipelines are available as EventBridge rule targets, which means that you can run an Image Builder pipeline based on rules that you create to respond to events on the bus, or on a schedule.

Note
Event buses are specific to a Region. The rule and the target must be in the same Region.

Contents

EventBridge terms

This section contains a summary of terms to help you understand how EventBridge integrates with your Image Builder pipelines.

Event

Describes a change in an environment that might affect one or more application resources. The environment can be an AWS environment, a SaaS partner service or application, or one of your applications or services. You can also set up scheduled events on a timeline.

Event bus

A pipeline that receives event data from applications and services.

Source

The service or application that sent the event to the event bus.

Target

A resource or endpoint that EventBridge invokes when it matches a rule, delivering data from the event to the target.

Rule

A rule matches incoming events and sends them to targets for processing. A single rule can send an event to multiple targets, which can then run in parallel. Rules are based either on an event pattern or a schedule.

Pattern

An event pattern defines the event structure and the fields that a rule matches in order to initiate the target action.

Schedule

Schedule rules perform an action on a schedule, such as running an Image Builder pipeline to refresh an image on a quarterly basis. There are two types of schedule expressions:

- **Cron expressions** – Match specific scheduling criteria using the cron syntax that can outline simple criteria; for example, running weekly on a specific day. You can also establish more complex criteria, such as running quarterly on the fifth day of the month, between 2 AM and 4 AM.
- **Rate expressions** – Specify a regular interval when the target is invoked, such as every 12 hours.

View EventBridge rules for your Image Builder pipeline

The **EventBridge rules** tab in the Image Builder **Image pipelines** detail page displays EventBridge event buses that your account has access to, and the rules for the selected event bus that apply to the current pipeline. This tab also links directly to the EventBridge console for creating new resources.

Actions that link to the EventBridge console

- **Create event bus**
- **Create rule**

To learn more about EventBridge, see the following topics in the *Amazon EventBridge User Guide*.

- What is Amazon EventBridge
- Amazon EventBridge event buses
- Amazon EventBridge events
- Amazon EventBridge rules

Use EventBridge rules to schedule a pipeline build

For this example, we create a new schedule rule for the default event bus, using a rate expression. The rule in this example generates an event on the event bus every 90 days. The event initiates a pipeline build to refresh the image.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. To see a list of the image pipelines created under your account, choose **Image pipelines** from the navigation pane.

   Note
   The list of image pipelines includes an indicator for the type of output image that is created by the pipeline – AMI or Docker.
3. To view details or edit a pipeline, choose the **Pipeline name** link. This opens the detail view for the pipeline.

   Note
   You can also select the check box next to the **Pipeline name**, then choose **View detail**.
4. Open the **EventBridge rules** tab.
5. Keep the default event bus that is pre-selected in the **Event Bus** panel.
6. Choose **Create rule**. This takes you to the **Create rule** page in the Amazon EventBridge console.
7. Enter a name and description for the rule. The rule name must be unique within the event bus for the selected Region.
8. In the **Define pattern** panel, choose the **Schedule** option. This expands the panel, with the **Fixed rate every** option selected.
9. Enter `90` in the first box, and select **Days** from the drop-down list.
10. Perform the following actions in the **Select targets** panel:

a. Select `EC2 Image Builder` from the **Target** drop-down list.
b. To apply the rule to an Image Builder pipeline, select the target pipeline from the **Image Pipeline** drop-down list.
c. EventBridge needs permission to initiate a build for the selected pipeline. For this example, keep the default option to **Create a new role for this specific resource**.
d. Choose **Add target**.

11. Choose **Create**

Note
To learn more about settings for rate expression rules that are not covered in this example, see Rate expressions in the *Amazon EventBridge User Guide*.

Monitor events and logs in EC2 Image Builder

Monitoring is an important part of maintaining the reliability, availability, and performance of your EC2 Image Builder pipelines. Monitoring events and logs helps you to see the big picture and dive down into the details when an API call fails. Services that integrate with Image Builder allow you to send alerts and kick off automated responses when events match the criteria that you've configured.

The following topics describe monitoring techniques you can use through services that integrate with Image Builder.

Monitor events and logs

Logging EC2 Image Builder API calls using AWS CloudTrail

EC2 Image Builder is integrated with AWS CloudTrail, a service that provides a record of actions all API calls for taken by a user, role, or an AWS service through the Image Builder API. CloudTrail captures Image Builder as events. The calls captured include calls from the Image Builder console and code calls to the Image Builder API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for Image Builder. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Image Builder, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Image Builder information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Image Builder, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Image Builder, create a trail. A *trail* enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail.
- CloudTrail supported services and integrations.
- Configuring Amazon SNS notifications for CloudTrail.
- Receiving CloudTrail log files from multiple regions.

- Receiving CloudTrail log files from multiple accounts.

CloudTrail logs all Image Builder actions that are documented in the EC2 Image Builder API Reference. For example, calls to the `CreateImagePipeline`, `UpdateInfrastructureConfiguration`, and `StartImagePipelineExecution` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information about determining who requested an event, see the CloudTrail userIdentity element.

Security in EC2 Image Builder

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to EC2 Image Builder, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Image Builder. The following topics show you how to configure Image Builder to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Image Builder resources.

Topics

EC2 Image Builder and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and EC2 Image Builder by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Image Builder APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Image Builder APIs. Traffic between your VPC and Image Builder does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for Image Builder VPC endpoints

Before you set up an interface VPC endpoint for Image Builder, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

Image Builder supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for Image Builder

You can create a VPC endpoint for the Image Builder service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for Image Builder using the following service name:

- com.amazonaws.*region*.imagebuilder

If you enable private DNS for the endpoint, you can make API requests to Image Builder using its default DNS name for the Region, for example, `imagebuilder.us-east-1.amazonaws.com`. To look up the endpoint that applies to your target Region, see EC2 Image Builder endpoints and quotas in the *Amazon Web Services General Reference*.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Image Builder

You can attach an endpoint policy to your VPC endpoint that controls access to Image Builder. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

> **Important**
> When a non-default policy is applied to an interface VPC endpoint for EC2 Image Builder, certain failed API requests, such as those failing from `RequestLimitExceeded`, might not be logged to AWS CloudTrail or Amazon CloudWatch.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Image Builder actions

The following is an example of an endpoint policy for Image Builder that denies permission to delete Image Builder images and components. The example policy also grants permission to perform all other EC2 Image Builder actions.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Action": "imagebuilder:*",
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Action": [
            "imagebuilder: DeleteImage"
        ],
        "Effect": "Deny",
        "Resource": "*",
    },
    {
        "Action": [
```

```
            "imagebuilder: DeleteComponent"
        ],
        "Effect": "Deny",
        "Resource": "*",
    }]
}
```

Data protection in EC2 Image Builder

The AWS shared responsibility model applies to data protection in EC2 Image Builder. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Image Builder or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption and key management in EC2 Image Builder

Image Builder encrypts data in transit and at rest by default. Custom components defined in the service can be added to your image pipelines and shared with other customer accounts. You are not required to share your components to build images.

Custom components are encrypted with your KMS key or a KMS key owned by Image Builder. Image Builder does not store any of your logs in the service. All logs are saved on your Amazon EC2 instance that is used to build the image, or in your Systems Manager automation logs.

You can manage your keys through AWS KMS. You cannot manage the Image Builder KMS key owned by Image Builder.

For more information about managing your KMS keys with AWS Key Management Service, see Getting Started in the AWS Key Management Service Developer Guide.

Internetwork Traffic Privacy in EC2 Image Builder

Connections are secured between Image Builder and on-premises locations, between AZs within an AWS Region, and between AWS Regions through HTTPS. There are no direct connections between accounts.

Identity and Access Management for EC2 Image Builder

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in EC2 Image Builder.

Service user – If you use the EC2 Image Builder service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more EC2 Image Builder features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in EC2 Image Builder, see Troubleshooting EC2 Image Builder identity and access (p. 197).

Service administrator – If you're in charge of EC2 Image Builder resources at your company, you probably have full access to EC2 Image Builder. It's your job to determine which EC2 Image Builder features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with EC2 Image Builder, see How EC2 Image Builder works with IAM (p. 178).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to EC2 Image Builder. To view example EC2 Image Builder identity-based policies that you can use in IAM, see Image Builder identity-based policies (p. 179).

Authenticating with identities

For detailed information about how to provide authentication for people and processes in your AWS account, see Identities in the *IAM User Guide*.

How EC2 Image Builder works with IAM

Before you use IAM to manage access to Image Builder, you should understand what IAM features are available to use with Image Builder. To get a high-level view of how Image Builder and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Image Builder identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources, and also the conditions under which actions are allowed or denied. Image Builder supports specific actions, resources, and condition keys. For information about all of the elements that you use in a JSON policy, see Actions, Resources, and Condition Keys for Amazon EC2 Image Builder in the *IAM User Guide*.

Actions

Policy actions in Image Builder use the following prefix before the action: `imagebuilder:`. Policy statements must include either an `Action` or `NotAction` element. Image Builder defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "imagebuilder:action1",
      "imagebuilder:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "imagebuilder:List*"
```

To see a list of Image Builder actions, see Actions, Resources, and Condition Keys for AWS Services in the *IAM User Guide*.

Managing access using policies

For detailed information about how to manage access in AWS by creating policies and attaching them to IAM identities or AWS resources, see Policies and Permissions in the *IAM User Guide*.

The IAM role that you associate with your instance profile must have permissions to run the build and test components included in your image. The following IAM role policies must be attached to the IAM role that is associated with the instance profile:

- EC2InstanceProfileForImageBuilder
- EC2InstanceProfileForImageBuilderECRContainerBuilds
- AmazonSSMManagedInstanceCore

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

The Image Builder instance resource has the following Amazon Resource Name (ARN).

```
arn:aws:imagebuilder:region:account-id:resource:resource-id
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `i-1234567890abcdef0` instance in your statement, use the following ARN.

```
"Resource": "arn:aws:imagebuilder:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:imagebuilder:us-east-1:123456789012:instance/*"
```

Some Image Builder actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

Many EC2 Image Builder API actions involve multiple resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

Condition keys

Image Builder provides service-specific condition keys and supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*. The following service-specific condition keys are provided.

imagebuilder:CreatedResourceTagKeys

Works with string operators.

Use this key to filter access by the presence of tag keys in the request This allows you to manage the resources created by Image Builder through defined tags.

Availability - This key is available to only the `CreateInfrastrucutreConfiguration` and `UpdateInfrastructureConfiguration` APIs.

imagebuilder:CreatedResourceTag/<key>

Works with string operators.

Use this key to filter access by the tag key-value pairs attached to the resource created by Image Builder. This allows you to manage the resources created by Image Builder through defined tags.

Availability - This key is available to only the `CreateInfrastrucutreConfiguration` and `UpdateInfrastructureConfiguration` APIs.

Examples

To view examples of Image Builder identity-based policies, see EC2 Image Builder identity-based policies (p. 182).

Image Builder resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on the Image Builder resource and under what conditions. Image Builder supports resource-based permissions policies for components, images, and image recipes. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow an AWS service to access your components, images, and image recipes.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, you must also grant the principal entity permission to access the resource. Grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

For information about how to attach a resource-based policy to a component, image, or image recipe, see Share EC2 Image Builder resources (p. 146).

> **Note**
> When you update a resource policy using Image Builder, the update will appear in the RAM console.

Authorization based on Image Builder tags

You can attach tags to Image Builder resources or pass tags in a request to Image Builder. To control access based on tags, you provide tag information in the condition element of a policy using the `imagebuilder:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging Image Builder resources, see Tag a resource (AWS CLI) (p. 151).

Image Builder IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Image Builder

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Image Builder supports service-linked roles. For information about creating or managing Image Builder service-linked roles, see Using service-linked roles for EC2 Image Builder (p. 196).

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your

IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

EC2 Image Builder identity-based policies

Topics

Identity-based policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete EC2 Image Builder resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using EC2 Image Builder quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Image Builder console

To access the EC2 Image Builder console, you must have a minimum set of permissions. These permissions allow you to list and view details about the Image Builder resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that your IAM entities can use the Image Builder console, you must attach one of the following AWS managed policies to them:

- AWSImageBuilderReadOnlyAccess policy (p. 186)
- AWSImageBuilderFullAccess policy (p. 184)

For more information about Image Builder managed policies, see Using managed policies for EC2 Image Builder (p. 184).

> **Important**
> The **AWSImageBuilderFullAccess** policy is required to create the Image Builder service-linked role. When you attach this policy to an IAM entity, you must also attach the following custom policy and include the resources you want to use that do not have `imagebuilder` in the resource name:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "sns topic arn"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetInstanceProfile"
            ],
            "Resource": "instance profile role arn"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "instance profile role arn",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ec2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "bucket arn"
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls to only the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

EC2 Image Builder resource-based policies

For information about how to create a component, see Manage components with AWSTOE (p. 106).

Restricting Image Builder component access to specific IP addresses

The following example grants permissions to any user to perform any Image Builder operations on components. However, the request must originate from the range of IP addresses specified in the condition.

The condition in this statement identifies the 54.240.143.* range of allowed Internet Protocol version 4 (IPv4) IP addresses, with one exception: 54.240.143.188.

The `Condition` block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information about these condition keys, see Specifying Conditions in a Policy. The`aws:sourceIp` IPv4 values use the standard CIDR notation. For more information, see IP Address Condition Operators in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Id": "IBPolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "imagebuilder.GetComponent:*",
      "Resource": "arn:aws:imagebuilder:::examplecomponent/*",
      "Condition": {
         "IpAddress": {"aws:SourceIp": "54.240.143.0/24"},
         "NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}
      }
    }
  ]
}
```

Using managed policies for EC2 Image Builder

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ViewOnlyAccess** AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWSImageBuilderFullAccess policy

The **AWSImageBuilderFullAccess** policy grants full access to Image Builder resources for the role it's attached to, allowing the role to list, describe, create, update, and delete Image Builder resources. The policy also grants targeted permissions to related AWS services that are needed, for example, to verify resources, or to display current resources for the account in the AWS Management Console.

Permissions details

This policy includes the following permissions:

- **Image Builder** – Administrative access is granted, so that the role can list, describe, create, update, and delete Image Builder resources.
- **Amazon EC2** – Access is granted for Amazon EC2 Describe actions that are needed to verify resource existence or get lists of resources belonging to the account.
- **IAM** – Access is granted to get and use instance profiles whose name contains "imagebuilder", to verify the existence of the Image Builder service-linked role via the `iam:GetRole` API action, and to create the Image Builder service-linked role.
- **License Manager** – Access is granted to list license configurations or licenses for a resource.

- **Amazon S3** – Access is granted to list buckets belonging to the account, and also Image Builder buckets with "imagebuilder" in their names.
- **Amazon SNS** – Write permissions are granted to Amazon SNS to verify topic ownership for topics containing "imagebuilder".

Policy example

The following is an example of the AWSImageBuilderFullAccess policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:*imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "license-manager:ListLicenseConfigurations",
                "license-manager:ListLicenseSpecificationsForResource"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/imagebuilder.amazonaws.com/
AWSServiceRoleForImageBuilder"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/*imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListInstanceProfiles",
                "iam:ListRoles"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::*:instance-profile/*imagebuilder*",
                "arn:aws:iam::*:role/*imagebuilder*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ec2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3::*:*imagebuilder*"
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/aws-service-role/imagebuilder.amazonaws.com/
AWSServiceRoleForImageBuilder",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "imagebuilder.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:DescribeSnapshots",
                "ec2:DescribeVpcs",
                "ec2:DescribeRegions",
                "ec2:DescribeVolumes",
                "ec2:DescribeSubnets",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeInstanceTypeOfferings",
                "ec2:DescribeLaunchTemplates"
            ],
            "Resource": "*"
        }
    ]
}
```

AWSImageBuilderReadOnlyAccess policy

The **AWSImageBuilderReadOnlyAccess** policy provides read-only access to all Image Builder resources. Permissions are granted to verify that the Image Builder service-linked role exists via the `iam:GetRole` API action.

Permissions details

This policy includes the following permissions:

- **Image Builder** – Access is granted for read-only access to Image Builder resources.
- **IAM** – Access is granted to verify the existence of the Image Builder service-linked role via the `iam:GetRole` API action.

Policy example

The following is an example of the AWSImageBuilderReadOnlyAccess policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:Get*",
                "imagebuilder:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/imagebuilder.amazonaws.com/
AWSServiceRoleForImageBuilder"
        }
    ]
}
```

AWSServiceRoleForImageBuilder policy

The **AWSServiceRoleForImageBuilder** policy allows Image Builder to call AWS services on your behalf.

Permissions details

This policy is attached to the Image Builder service-linked role when the role is created through Systems Manager. The policy includes the following permissions:

- **CloudWatch Logs** – Access is granted to create and upload CloudWatch Logs to any log group whose name starts with `/aws/imagebuilder/`.
- **Amazon EC2** – Access is granted for Image Builder to launch EC2 instances in your account, using related snapshots, volumes, network interfaces, security groups, and key pairs as required, as long as the instance and volumes that are being created or used are tagged with `"Created By": "EC2 Image Builder"`.

  Image Builder can get information about Amazon EC2 images, instance attributes, instance status, the instance types that are available to your account, launch templates, subnets, and tags on your Amazon EC2 resources.

  Additionally, Image Builder can stop and terminate instances that are running in your account, share Amazon EBS snapshots, create and update images and launch templates, de-register existing images, add tags, and replicate images across accounts that you have granted permissions to via the **Ec2ImageBuilderCrossAccountDistributionAccess** policy. Image Builder tagging is required for all of these actions, as described previously.

To review specific permissions that are granted, see the policy example (p. 188) in this section.

- **IAM** – Access is granted for Image Builder to pass any role in your account to Amazon EC2.
- **AWS KMS** – Access is granted for Amazon EBS to encrypt, decrypt or re-encrypt Amazon EBS volumes. This is crucial to ensure that encrypted volumes work when Image Builder builds an image.
- **License Manager** – Access is granted for Image Builder to update License Manager specifications via `license-manager:UpdateLicenseSpecificationsForResource`.
- **Amazon SNS** – Write permissions are granted for any Amazon SNS topic in the account.
- **Systems Manager** – Access is granted for Image Builder to list Systems Manager commands and their invocations, instance information, inventory entries and automation execution statuses. Image Builder can also send automation signals, and stop automation exeuctions for any resource in your account.

  Image Builder is able to issue run command invocations to any instance that is tagged `"Created By": "EC2 Image Builder"` for the following script files: `AWS-RunPowerShellScript`, `AWS-RunShellScript`, or `AWSEC2-RunSysprep`. Image Builder is able to start an Systems Manager automation execution in your account for automation documents where the name starts with `ImageBuilder`.

  Image Builder is also able to create or delete State Manager associations for any instance in your account, as long as the association document is `AWS-GatherSoftwareInventory`, and to create the Systems Manager service-linked role in your account. For more information about the Image Builder service-linked role, see Using service-linked roles for EC2 Image Builder (p. 196).
- **AWS STS** – Access is granted for Image Builder to assume roles named **EC2ImageBuilderDistributionCrossAccountRole** from your account to any account where the Trust policy on the role permits it. This is used for cross-account image distribution.

Policy example

The following is an example of the AWSServiceRoleForImageBuilder policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*::image/*",
                "arn:aws:ec2:*::snapshot/*",
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:key-pair/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/CreatedBy": "EC2 Image Builder"
```

```
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": "iam:PassRole",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": [
                    "ec2.amazonaws.com",
                    "ec2.amazonaws.com.cn"
                ]
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:StopInstances",
            "ec2:TerminateInstances"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CopyImage",
            "ec2:CreateImage",
            "ec2:CreateLaunchTemplate",
            "ec2:DeregisterImage",
            "ec2:DescribeImages",
            "ec2:DescribeInstanceAttribute",
            "ec2:DescribeInstanceStatus",
            "ec2:DescribeInstances",
            "ec2:DescribeInstanceTypeOfferings",
            "ec2:DescribeInstanceTypes",
            "ec2:DescribeSubnets",
            "ec2:DescribeTags",
            "ec2:ModifyImageAttribute"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:ModifySnapshotAttribute"
        ],
        "Resource": "arn:aws:ec2:*::snapshot/*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags"
        ],
        "Resource": "arn:aws:ec2:*::image/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/CreatedBy": "EC2 Image Builder"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "license-manager:UpdateLicenseSpecificationsForResource"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:ListCommands",
                "ssm:ListCommandInvocations",
                "ssm:AddTagsToResource",
                "ssm:DescribeInstanceInformation",
                "ssm:GetAutomationExecution",
                "ssm:StopAutomationExecution",
                "ssm:ListInventoryEntries",
                "ssm:SendAutomationSignal",
                "ssm:DescribeInstanceAssociationsStatus",
                "ssm:DescribeAssociationExecutions"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:SendCommand",
            "Resource": [
                "arn:aws:ssm:*:*:document/AWS-RunPowerShellScript",
                "arn:aws:ssm:*:*:document/AWS-RunShellScript",
                "arn:aws:ssm:*:*:document/AWSEC2-RunSysprep",
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "ssm:resourceTag/CreatedBy": [
                        "EC2 Image Builder"
                    ]
```

```
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": "ssm:StartAutomationExecution",
        "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ssm:CreateAssociation",
            "ssm:DeleteAssociation"
        ],
        "Resource": [
            "arn:aws:ssm:*:*:document/AWS-GatherSoftwareInventory",
            "arn:aws:ssm:*:*:association/*",
            "arn:aws:ec2:*:*:instance/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncryptFrom",
            "kms:ReEncryptTo",
            "kms:GenerateDataKeyWithoutPlaintext",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "ForAllValues:StringEquals": {
                "kms:EncryptionContextKeys": [
                    "aws:ebs:id"
                ]
            },
            "StringLike": {
                "kms:ViaService": [
                    "ec2.*.amazonaws.com"
                ]
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": "kms:CreateGrant",
        "Resource": "*",
        "Condition": {
            "Bool": {
                "kms:GrantIsForAWSResource": true
            },
            "StringLike": {
                "kms:ViaService": [
                    "ec2.*.amazonaws.com"
                ]
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::*:role/EC2ImageBuilderDistributionCrossAccountRole"
    },
    {
        "Effect": "Allow",
```

```
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/imagebuilder/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateLaunchTemplateVersion",
                "ec2:DescribeLaunchTemplates",
                "ec2:ModifyLaunchTemplate"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "ssm.amazonaws.com"
                }
            }
        }
    ]
}
```

Ec2ImageBuilderCrossAccountDistributionAccess policy

The **Ec2ImageBuilderCrossAccountDistributionAccess** policy grants permissions for Image Builder to distribute images across accounts in target Regions. Additionally, Image Builder can describe, copy, and apply tags to any Amazon EC2 image in the account. The policy also grants the ability to modify AMI permissions via the `ec2:ModifyImageAttribute` API action.

Permissions details

This policy includes the following permissions:

- **Amazon EC2** – Access is granted for Amazon EC2 to describe, copy, and modify attributes for an image, and to create tags for any Amazon EC2 images in the account.

Policy example

The following is an example of the Ec2ImageBuilderCrossAccountDistributionAccess policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*::image/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:CopyImage",
                "ec2:ModifyImageAttribute"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

EC2InstanceProfileForImageBuilder policy

The **EC2InstanceProfileForImageBuilder** policy grants the minimum permissions required for an EC2 instance to work with Image Builder. This does not include permissions required to use the Systems Manager Agent.

Permissions details

This policy includes the following permissions:

- **CloudWatch Logs** – Access is granted to create and upload CloudWatch Logs to any log group whose name starts with `/aws/imagebuilder/`.
- **Image Builder** – Access is granted to get any Image Builder component.
- **AWS KMS** – Access is granted to decrypt an Image Builder component, if it was encrypted via AWS KMS.
- **Amazon S3** – Access is granted to get objects stored in an Amazon S3 bucket whose name starts with `ec2imagebuilder-`.

Policy example

The following is an example of the EC2InstanceProfileForImageBuilder policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:GetComponent"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "kms:EncryptionContextKeys": "aws:imagebuilder:arn",
                    "aws:CalledVia": [
                        "imagebuilder.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::ec2imagebuilder*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/imagebuilder/*"
        }
    ]
}
```

EC2InstanceProfileForImageBuilderECRContainerBuilds policy

The **EC2InstanceProfileForImageBuilderECRContainerBuilds** policy grants the minimum permissions required for an EC2 instance when working with Image Builder to build Docker images and then register and store the images in an Amazon ECR container repository. This does not include permissions required to use the Systems Manager Agent.

Permissions details

This policy includes the following permissions:

- **CloudWatch Logs** – Access is granted to create and upload CloudWatch Logs to any log group whose name starts with `/aws/imagebuilder/`.
- **Amazon ECR** – Access is granted for Amazon ECR to get, register, and store a container image, and to get an authorization token.
- **Image Builder** – Access is granted to get an Image Builder component or container recipe.
- **AWS KMS** – Access is granted to decrypt an Image Builder component or container recipe, if it was encrypted via AWS KMS.
- **Amazon S3** – Access is granted to get objects stored in an Amazon S3 bucket whose name starts with `ec2imagebuilder-`.

Policy example

The following is an example of the EC2InstanceProfileForImageBuilderECRContainerBuilds policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:GetComponent",
                "imagebuilder:GetContainerRecipe",
                "ecr:GetAuthorizationToken",
                "ecr:BatchGetImage",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:PutImage"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "kms:EncryptionContextKeys": "aws:imagebuilder:arn",
                    "aws:CalledVia": [
                        "imagebuilder.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::ec2imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/imagebuilder/*"
        }
    ]
}
```

Image Builder updates to AWS managed policies

This section provides information about updates to AWS managed policies for Image Builder since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Image Builder document history (p. 209) page.

Change	Description	Date
AWSServiceRoleForImageBuilder (p. 187) – Update to an existing policy	Image Builder added new permissions to fix issues where more than one inventory association causes the image build to get stuck.	August 11, 2021
AWSImageBuilderFullAccess (p. 184) – Update to an existing policy	Image Builder added permissions to allow `ec2:DescribeInstanceTypeOffereings`. Added permissions to call `ec2:DescribeInstanceTypeOffereings` to enable the Image Builder console to accurately reflect the instance types that are available in the account.	April 13, 2021
Image Builder started tracking changes	Image Builder started tracking changes for its AWS managed policies.	April 02, 2021

Using service-linked roles for EC2 Image Builder

EC2 Image Builder uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Image Builder. Service-linked roles are predefined by Image Builder and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Image Builder more efficient, because you don't have to add the necessary permissions manually. Image Builder defines the permissions of its service-linked roles, and unless defined otherwise, only Image Builder can assume its roles. The defined permissions include the trust policy and the permissions policy. The permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Image Builder

Image Builder uses the **AWSServiceRoleForImageBuilder** service-linked role to allow EC2 Image Builder to access AWS resources on your behalf. The service-linked role trusts the *imagebuilder.amazonaws.com* service to assume the role.

You don't need to manually create this service-linked role. When you create your first Image Builder resource in the AWS Management Console, the AWS CLI, or the AWS API, Image Builder creates the service-linked role for you, through Systems Manager.

> **Important**
> If the service-linked role is deleted from your account, you can use the same process to create it again. When you create your first EC2 Image Builder resource, Image Builder creates the service-linked role for you again.

To see permissions for the **AWSServiceRoleForImageBuilder**, see the AWSServiceRoleForImageBuilder policy (p. 187) page. To learn more about configuring permissions for a service-linked role, see Service-Linked Role Permissions in the *IAM User Guide*.

Removing an Image Builder service-linked role from your account

You can use the IAM console, the AWS CLI, or the AWS API to manually remove the service-linked role for Image Builder from your account. However, before you do this, you must ensure that there are no Image Builder resources enabled that refer to it.

> **Note**
> If the Image Builder service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait for a few minutes and try the operation again.

Clean up Image Builder resources used by the `AWSServiceRoleForImageBuilder` role

1. Verify that no pipeline builds are running before you start. To cancel a running build, use the `cancel-image-creation` command from the AWS CLI.

```
aws imagebuilder cancel-image-creation --image-build-version-
arn arn:aws:imagebuilder:us-east-1:123456789012:image-pipeline/sample-pipeline
```

2. Change all pipeline schedules to use a manual build process, or delete them if you won't be using them again. For more information about deleting resources, see Delete EC2 Image Builder resources (p. 152).

Delete the service-linked role using IAM

You can use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForImageBuilder` role from your account. For more information, see Deleting a Service-Linked Rolein the *IAM User Guide*.

Supported Regions for EC2 Image Builder service-linked roles

Image Builder supports using service-linked roles in all of the AWS Regions where the service is available. For the list of supported AWS Regions, see AWS Regions and Endpoints (p. 7).

Troubleshooting EC2 Image Builder identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Image Builder and IAM.

I am not authorized to perform an action in Image Builder

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a component but does not have `imagebuilder:ListComponents` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 imagebuilder:ListComponents on resource: Component
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `Component` resource using the `imagebuilder:ListComponents` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to EC2 Image Builder.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in EC2 Image Builder. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an Administrator and want to allow others to access Image Builder

To allow others to access EC2 Image Builder, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in EC2 Image Builder.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Image Builder resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies, you can use those policies to grant people access to your resources.

For more information about:

- How to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS account That You Own in the *IAM User Guide*.
- How to provide access to your resources to third-party AWS accounts, see Providing Access to AWS accounts Owned by Third Parties in the *IAM User Guide*.
- How to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- The difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Compliance validation for EC2 Image Builder

EC2 Image Builder is not in scope of any AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Image Builder is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

For STIG compliance, use the Amazon-provided STIG components provided in the Image Builder service to help you scan for misconfigurations and to run a remediation script. We cannot guarantee that images built using Image Builder are STIG compliant. You must confirm image compliance with your compliance teams. For a complete list of STIG components available through Image Builder, see EC2 Image Builder STIG components (p. 96).

Resilience in EC2 Image Builder

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

The EC2 Image Builder service allows you to distribute images built in one Region with other Regions, giving them multi-Region resiliency for AMIs. There is no mechanism to "back up" image pipelines, recipes, or components. You can store the recipe and component documents outside of the Image Builder service, such as in an Amazon S3 bucket.

The EC2 Image Builder cannot be configured for High Availability (HA). You can distribute images to multiple Regions to make the images more highly available.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in EC2 Image Builder

As a managed service, EC2 Image Builder is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Image Builder through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Instances used to build images and run tests using Image Builder must have access to the AWS Systems Manager service.

Patch Management in EC2 Image Builder

EC2 Image Builder provides the latest Amazon Linux 2, Red Hat Enterprise Linux (RHEL), CentOS, Ubuntu, SUSE Linux Enterprise Server, and Windows 2012 R2 and later AMIs as managed image sources. You maintain the Amazon EC2 system patching responsibility, per the shared responsibility model. If the EC2 instances in your application workload can be easily replaced, then it might be more efficient to update the base AMI and redeploy all compute nodes based on this image.

The following are two ways you can keep your Image Builder AMIs up to date.

- **AWS-provided patching components** – EC2 Image Builder provides two build components, `update-linux` and `update-windows`,which install all pending operating system updates. These components use the `UpdateOS` action module. For more information, see UpdateOS (p. 94). The components can be added to your image build pipelines by selecting them from the list of AWS-provided components.
- **Custom build components with patching operations** – To selectively install or update patches on operating systems of supported AMIs, you can author an Image Builder component to install the required patches. A custom component can install patches using shell scripts (Bash or PowerShell), or it can use the `UpdateOS` action module to specify patches for installation or exclusion. For more information, see Action modules supported by AWSTOE component manager (p. 52).

  Component that uses the `UpdateOS` action module (Linux and Windows)

```
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: UpdateOS
        action: UpdateOS
```

  Component that uses Bash to install yum updates

```
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: InstallYumUpdates
        action: ExecuteBash
        inputs:
          commands:
            - sudo yum update -y
```

Security best practices for EC2 Image Builder

EC2 Image Builder provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- Do not use overly-permissive security groups in Image Builder recipes.
- Do not share images with accounts that you do not trust.
- Do not make images public that have private or sensitive data.
- Apply all available Windows or Linux security patches during image builds.

Script Execution

When building Linux images using EC2 Image Builder, AWS will enforce the execution of a script that will run at the end of the image building process. Similarly, EC2 Image Builder will run Microsoft's Sysprep utility after customizing Windows images. These actions follow AWS best practices for hardening and cleaning the image. However, because additional customizations can be made during image customization, AWS does not guarantee the images produced to be compliant with any specific regulatory criteria.

AWS recommends that you test your images to validate the security posture and applicable security compliance levels. Solutions such as Amazon Inspector can help validate the security and compliance posture of images.

The following script is run as a mandatory step when Linux images are customized with EC2 Image Builder.

```
#!/bin/bash

FILES=(
        # Secure removal of list of sudo users
        "/etc/sudoers.d/90-cloud-init-users"

        # Secure removal of RSA encrypted SSH host keys.
        "/etc/ssh/ssh_host_rsa_key"
        "/etc/ssh/ssh_host_rsa_key.pub"

        # Secure removal of ECDSA encrypted SSH host keys.
        "/etc/ssh/ssh_host_ecdsa_key"
        "/etc/ssh/ssh_host_ecdsa_key.pub"

        # Secure removal of ED25519 encrypted SSH host keys.
        "/etc/ssh/ssh_host_ed25519_key"
        "/etc/ssh/ssh_host_ed25519_key.pub"

        # Secure removal of "root" user approved SSH keys list.
        "/root/.ssh/authorized_keys"

        # Secure removal of "ec2-user" user approved SSH keys list.
        "/home/ec2-user/.ssh/authorized_keys"

        # Secure removal of file which tracks system updates
        "/etc/.updated"
        "/var/.updated"

        # Secure removal of file with aliases for mailing lists
        "/etc/aliases.db"

        # Secure removal of file which contains the hostname of the system
        "/etc/hostname"

        # Secure removal of files with system-wide locale settings
        "/etc/locale.conf"

        # Secure removal of cached GPG signatures of yum repositories
        "/var/cache/yum/x86_64/2/.gpgkeyschecked.yum"

        # Secure removal of audit framework logs
        "/var/log/audit/audit.log"

        # Secure removal of boot logs
        "/var/log/boot.log"

        # Secure removal of kernel message logs
        "/var/log/dmesg"

        # Secure removal of cloud-init logs
```

```
        "/var/log/cloud-init.log"

        # Secure removal of cloud-init's output logs
        "/var/log/cloud-init-output.log"

        # Secure removal of cron logs
        "/var/log/cron"

        # Secure removal of aliases file for the Postfix mail transfer agent
        "/var/lib/misc/postfix.aliasesdb-stamp"

        # Secure removal of master lock for the Postfix mail transfer agent
        "/var/lib/postfix/master.lock"

        # Secure removal of spool data for the Postfix mail transfer agent
        "/var/spool/postfix/pid/master.pid"

        # Secure removal of history of Bash commands
        "/home/ec2-user/.bash_history"

)

for FILE in "${FILES[@]}"; do
      if [[ -f $FILE ]]; then
              echo "Deleting $FILE"
              sudo shred -zuf $FILE
      fi
      if [[ -f $FILE ]]; then
              echo "Failed to delete '$FILE'. Failing."
              exit 1
      fi
done

# Secure removal of TOE's log directories
if [[ $( sudo find {{workingDirectory}}/TOE_* -type f | sudo wc -l) -gt 0 ]]; then
      echo "Deleting files within {{workingDirectory}}/TOE_*"
      sudo find {{workingDirectory}}/TOE_* -type f -exec shred -zuf {} \;
fi
if [[ $( sudo find {{workingDirectory}}/TOE_* -type f | sudo wc -l) -gt 0 ]]; then
      echo "Failed to delete {{workingDirectory}}/TOE_*"
      exit 1
fi
if [[ $( sudo find {{workingDirectory}}/TOE_* -type d | sudo wc -l) -gt 0 ]]; then
      echo "Deleting {{workingDirectory}}/TOE_*"
      sudo rm -rf {{workingDirectory}}/TOE_*
fi
if [[ $( sudo find {{workingDirectory}}/TOE_* -type d | sudo wc -l) -gt 0 ]]; then
      echo "Failed to delete {{workingDirectory}}/TOE_*"
      exit 1
fi

# Secure removal of system activity reports/logs
if [[ $( sudo find /var/log/sa/sa* -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/log/sa/sa*"
      sudo shred -zuf /var/log/sa/sa*
fi
if [[ $( sudo find /var/log/sa/sa* -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Failed to delete /var/log/sa/sa*"
      exit 1
fi

# Secure removal of SSM logs
if [[ $( sudo find /var/log/amazon/ssm -type f | sudo wc -l) -gt 0 ]]; then
      echo "Deleting files within /var/log/amazon/ssm/*"
      sudo find /var/log/amazon/ssm -type f -exec shred -zuf {} \;
fi
```

```
if [[ $( sudo find /var/log/amazon/ssm -type f | sudo wc -l) -gt 0 ]]; then
      echo "Failed to delete /var/log/amazon/ssm"
      exit 1
fi
if [[ -d "/var/log/amazon/ssm" ]]; then
      echo "Deleting /var/log/amazon/ssm/*"
      sudo rm -rf /var/log/amazon/ssm
fi
if [[ -d "/var/log/amazon/ssm" ]]; then
      echo "Failed to delete /var/log/amazon/ssm"
      exit 1
fi

# Secure removal of DHCP client leases that have been acquired
if [[ $( sudo find /var/lib/dhclient/dhclient*.lease -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/lib/dhclient/dhclient*.lease"
      sudo shred -zuf /var/lib/dhclient/dhclient*.lease
fi
if [[ $( sudo find /var/lib/dhclient/dhclient*.lease -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Failed to delete /var/lib/dhclient/dhclient*.lease"
      exit 1
fi

# Secure removal of cloud-init files
if [[ $( sudo find /var/lib/cloud -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting files within /var/lib/cloud/*"
      sudo find /var/lib/cloud -type f -exec shred -zuf {} \;
fi
if [[ $( sudo find /var/lib/cloud -type f | sudo wc -l ) -gt 0 ]]; then
      echo "Failed to delete /var/lib/cloud"
      exit 1
fi
if [[ $( sudo ls /var/lib/cloud | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/lib/cloud/*"
      sudo rm -rf /var/lib/cloud/*
fi
if [[ $( sudo ls /var/lib/cloud | sudo wc -l ) -gt 0 ]]; then
      echo "Failed to delete /var/lib/cloud/*"
      exit 1
fi

# Secure removal of temporary files
if [[ $( sudo find /var/tmp -type f | sudo wc -l) -gt 0 ]]; then
      echo "Deleting files within /var/tmp/*"
      sudo find /var/tmp -type f -exec shred -zuf {} \;
fi
if [[ $( sudo find /var/tmp -type f | sudo wc -l) -gt 0 ]]; then
      echo "Failed to delete /var/tmp"
      exit 1
fi
if [[ $( sudo ls /var/tmp | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/tmp/*"
      sudo rm -rf /var/tmp/*
fi

# Shredding is not guaranteed to work well on rolling logs

# Removal of system logs
if [[ -f "/var/lib/rsyslog/imjournal.state" ]]; then
      echo "Deleting /var/lib/rsyslog/imjournal.state"
      sudo shred -zuf /var/lib/rsyslog/imjournal.state
      sudo rm -f /var/lib/rsyslog/imjournal.state
fi
if [[ -f "/var/lib/rsyslog/imjournal.state" ]]; then
      echo "Failed to delete /var/lib/rsyslog/imjournal.state"
      exit 1
```

```
fi

# Removal of journal logs
if [[ $( sudo ls /var/log/journal/ | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/log/journal/*"
      sudo find /var/log/journal/ -type f -exec shred -zuf {} \;
      sudo rm -rf /var/log/journal/*
fi
```

Troubleshoot EC2 Image Builder

EC2 Image Builder integrates with AWS services for monitoring and troubleshooting to help you troubleshoot image build issues. Image Builder tracks and displays the progress for each step in the image building process. Additionally, Image Builder can export logs to an Amazon S3 location that you provide.

For advanced troubleshooting, you can run predefined commands and scripts using AWS Systems Manager Run Command.

Contents

Troubleshoot pipeline builds

If an Image Builder pipeline build fails, Image Builder returns an error message that describes the failure. Image Builder also returns a Systems Manager execution ID in the failure message, such as the one in the following example.

```
Systems Manager execution 'aaaaaaaa-bbbb-cccc-dddd-example12345' failed with status…
```

Image Builder uses AWS Systems Manager Automation to orchestrate image build actions. To review additional details to help troubleshoot a build failure, search the Systems Manager Automation console for the execution ID provided by Image Builder and review the Automation execution.

All build activity is also logged in AWS CloudTrail if it is enabled in your account. Filter CloudTrail events by the source `imagebuilder.amazonaws.com`, or search for the Amazon EC2 instance ID that is returned in the execution log to see more details about the pipeline execution.

By default, Image Builder shuts down the Amazon EC2 build or test instance that is running when the pipeline fails. You can change the instance settings for the infrastructure configuration resource that your pipeline uses, to retain your build or test instance for troubleshooting.

To change the instance settings in the console, you must clear the **Terminate instance on failure** check box located in the **Troubleshooting settings** section of your infrastructure configuration resource.

To change the instance settings using the **imagebuilder update-infrastructure-configuration** command in the AWS CLI, set the `terminateInstanceOnFailure` value to `false` in the JSON file that the command references with the `--cli-input-json` parameter. For details, see Update an infrastructure configuration (AWS CLI) (p. 133).

The logs that you send to your S3 bucket show the steps and error messages for activity on the EC2 instance during the image build process. The logs include log outputs from the component manager, the definitions of the components that were run, and the detailed output (in JSON) of all of the steps taken on the instance. If you encounter an issue, you should review these files, starting with the `application.log`, to diagnose the cause of the problem on the instance.

Troubleshooting scenarios

This section lists the following detailed troubleshooting scenarios:

To see the details of a scenario, choose the scenario title to expand it. You can have multiple titles expanded at the same time.

Access denied – status code 403

Description

The pipeline build fails with "AccessDenied: Access Denied status code: 403".

Cause

Possible causes include:

- The instance profile does not have the required permissions (p. 13) to access APIs or component resources.
- The instance profile role is missing permissions that are required for logging to Amazon S3. Most commonly, this occurs when the instance profile role does not have **PutObject** permissions for your S3 buckets.

Solution

Depending on the cause, this issue can be resolved as follows:

- **Instance profile is missing managed policies** – Add the missing policies to your instance profile role. Then run the pipeline again.
- **Instance profile is missing write permissions for S3 bucket** – Add a policy to your instance profile role that grants **PutObject** permissions to write to your S3 bucket. Then run the pipeline again.

Build times out while verifying the Systems Manager Agent availability on the build instance

Description

The pipeline build fails with "status = 'TimedOut'" and "failure message = 'Step timed out while step is verifying the Systems Manager Agent availability on the target instance(s)'".

Cause

Possible causes include:

- The instance that was launched to perform the build operations and to run components was not able to access the Systems Manager endpoint.
- The instance profile does not have the required permissions (p. 13).

Solution

Depending on the possible cause, this issue can be resolved as follows:

- **Access issue, private subnet** – If you are building in a private subnet, make sure that you have set up PrivateLink endpoints for Systems Manager, Image Builder, and, if you want logging, Amazon S3/CloudWatch. For more information about setting up PrivateLink endpoints, see VPC endpoints concepts (AWS PrivateLink).
- **Missing permissions** – Add the following managed policies to your IAM service-linked role for Image Builder:
  - EC2InstanceProfileForImageBuilder
  - EC2InstanceProfileForImageBuilderECRContainerBuilds
  - AmazonSSMManagedInstanceCore

  For more information about the Image Builder service-linked role, see Using service-linked roles for EC2 Image Builder (p. 196).

Windows secondary disk is offline at launch

Description

When the instance type used to build an Image Builder Windows AMI does not match the instance type that is used to launch from the AMI, an issue can occur where non-root volumes are offline at launch. This primarily happens when the build instance is using a newer architecture than the launch instance.

The following example demonstrates what happens when an Image Builder AMI is built on an EC2 Nitro instance type and launched on an EC2 Xen instance:

Build instance type: m5.large (Nitro)

Launch instance type: t2.medium (Xen)

```
PS C:\Users\Administrator> get-disk
Number  Friendly Name  Serial Number         Health Status  Operational Status  Total Size
 Partition Style
------  -------------  -------------         -------------  ------------------  ----------
 ---------------
0       AWS PVDISK     vol0abc12d34e567f8a9  Healthy        Online                    30 GB
 MBR
1       AWS PVDISK     vol1bcd23e45f678a9b0  Healthy        Offline                    8 GB
 MBR
```

Cause

Because of Windows default settings, newly discovered disks are not automatically brought online and formatted. When the instance type is changed on EC2, Windows treats this as new disks being discovered. This is because of the underlying driver change.

Solution

We recommend that you use the same system of instance types when building your Windows AMI that you intend to launch from. Do not include instance types that are built on different systems in your infrastructure configuration. If any of the instance types you specify use the Nitro system, then they should all use the Nitro system.

For more information about instances that are built on the Nitro system, see Instances built on the Nitro System in the *Amazon EC2 User Guide for Windows Instances*.

Build fails with CIS hardened base image

Description

You are using a CIS hardened base image and the build fails.

Cause

When the `/tmp` directory is classified as `noexec`, it can cause Image Builder to fail.

Solution

Choose a different location for your working directory in the `workingDirectory` field of the image recipe. For more information, see the ImageRecipe data type description.

Document History for EC2 Image Builder User Guide

The following table describes important changes to the documentation by date. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version: 2021-07-06**

update-history-change	update-history-description	update-history-date
Document update: Add CloudTrail integration content (p. 209)	Added monitoring summary and CloudTrail integration content.	September 17, 2021
New STIG Versions	Updated STIG versions and applied STIGS for 2021 third quarter release.	September 10, 2021
Feature release: Amazon EventBridge integration (p. 209)	Added EventBridge support that enables you to connect Image Builder with events from related AWS services, and initiate events based on rules defined in EventBridge.	August 18, 2021
Managed policy update: AWSServiceRoleForImageBuilder	Updated managed policy for service role.	August 11, 2021
Document update: Reorder AWSTOE pages (p. 209)	Rearranged AWSTOE pages for clarity.	August 11, 2021
Feature release: Parameterized components and additional instance configuration (p. 209)	Added support for specifying parameters to customize components for recipes. Expanded configuration of the EC2 instances that are used for building and testing images, including the ability to specify commands to run on launch, and more control over installation and removal of the Systems Manager agent.	July 7, 2021
New STIG versions	Updated STIG versions and applied STIGS for 2021 second quarter release.	June 30, 2021
Enhancement: Tagging enhancements (p. 209)	Improved messaging around resource tagging.	June 25, 2021
Feature release: Launch template integration (p. 209)	Added support for using Amazon EC2 launch templates for AMI distribution in the Distribution settings.	April 7, 2021

Feature release: Container build enhancements (p. 209)	Added support for configuring block device mappings and specifying AMIs to use as the base image for container builds.	April 7, 2021
New STIG versions (p. 96)	Updated STIG versions and applied STIGS.	March 5, 2021
Update cron expressions	Image Builder cron processing is updated to increase cron expression granularity to the minute, and use a standard cron scheduling engine. Examples are updated with the new format.	February 8, 2021
Feature release: Container support (p. 209)	Added support for creating Docker container images using Image Builder, with registration and storage of the resulting images on Amazon Elastic Container Registry (Amazon ECR). Content has been rearranged to reflect new functionality and accomodate future growth.	December 17, 2020
Restructured cron documentation	This page now highlights more information about how cron works with Image Builder pipeline builds, and includes details about UTC time. Wildcards that are not allowed for specific fields have been removed. Examples now include expression samples for both console and CLI.	November 13, 2020
Console version 2.0: updated pipeline editing (p. 209)	Content changes in getting started and create pipeline tutorials, plus the manage image pipelines page, to incorporate new console features and flow.	November 13, 2020
New STIG versions (p. 96)	Updated STIG versions and applied STIGS. Note - list format changed to show STIGs that are applied by default.	October 15, 2020
Support for looping constructs in AWSTOE (p. 45)	Create looping constructs to define a repeated sequence of instructions in the AWSTOE application.	July 29, 2020
Support for local development of AWSTOE components (p. 23)	Develop and test image components locally with the AWSTOE application.	July 28, 2020
Encrypted AMIs (p. 209)	EC2 Image Builder adds support for encrypted AMI distribution.	July 1, 2020

AutoScaling deprecation (p. 209)	Deprecation of the use of AutoScaling to launch instances.	June 15, 2020
Support for connectivity through AWS PrivateLink	You can establish a private connection between your VPC and EC2 Image Builder by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Image Builder APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Image Builder APIs. Traffic between your VPC and Image Builder does not leave the Amazon network.	June 10, 2020
New STIG versions (p. 96)	Updated STIG versions and applied STIGS.	January 23, 2020
Troubleshooting (p. 209)	Added general troubleshooting scenarios.	January 22, 2020
STIG Components (p. 96)	You can create STIG-compliant images with Image Builder STIG components.	January 22, 2020

AWS glossary

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