User Guide

EC2 Image Builder

EC2 Image Builder: User Guide

Table of Contents

What is Image Builder?

EC2 Image Builder is a fully managed AWS service that helps you to automate the creation, management, and deployment of customized, secure, and up-to-date server images. You can use the AWS Management Console, AWS Command Line Interface, or APIs to create custom images in your AWS account.

You own the customized images that Image Builder creates in your account. You can configure pipelines to automate updates and system patching for the images that you own. You can also run a stand-alone command to create an image with the configuration resources that you've defined.

The Image Builder pipeline wizard can guide you through the steps to create a custom image, as follows:

Step 1: Specify pipeline details

- Name your pipeline and add tags.
- Define metadata and vulnerability scan settings.
- Set up a schedule for your pipeline.

Step 2: Customize your image

You can select an existing recipe or create a new one.

- Choose a base image for your customizations.
- Add to or remove software from your base image.
- Customize settings and scripts with build components.
- Select test components to run.

Step 3: Define your workflow

An image workflow defines the sequence of steps that Image Builder performs during the build and test stages of the image creation process. This is part of the overall Image Builder workflow framework.

Step 4: Configure build infrastructure

- Select an IAM role to associate with the instance profile for instances that Image Builder launches during the image creation process.
- Select one or more instance types that can be applied at launch.
- Select an Amazon Simple Notification Service (SNS) Topic to receive notifications from Image Builder.
- Specify VPC, subnet, and security groups that apply for the image creation process.
- Select troubleshooting settings such as where Image Builder writes logs, and whether to terminate the build instance on failure (default) or keep it running for further troubleshooting.

Step 5: Define image distribution

- Select AWS Regions where Image Builder distributes your Amazon Machine Image (AMI) or container image.
- If your Image Builder pipeline creates an AMI, Image Builder also supports the following configuration:
  - Select a KMS key to use for encryption.
  - Configure AMI sharing across AWS accounts and Organizations.
  - Associate a License Manager self-managed license with your distributed image.
  - Configure a launch template for your image.

Features of 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 provides access to test components that you can use to test your images before deployment. You can also create custom test components with AWS Task Orchestrator and Executor (AWSTOE), and use those. Image Builder distributes 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 Amazon managed STIG hardening components for Image Builder.

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 Image Builder resources with AWS RAM.

Supported operating systems

Image Builder supports the following operating system versions:

Operating system/distribution	Supported versions
Amazon Linux	2 and 2023

Operating system/distribution	Supported versions
CentOS	7 and 8
CentOS Stream	8
macOS	12.x (Monterey), 13.x (Ventura), 14.x (Sonoma), 15.x (Sequoia)
Red Hat Enterprise Linux (RHEL)	7, 8, 9, and 10
SUSE Linux Enterprise Server (SUSE)	12 and 15
Ubuntu	18.04 LTS, 20.04 LTS, 22.04 LTS, and 24.04 LTS
Windows Server	2012 R2, 2016, 2019, 2022, and 2025

Supported image formats

For your custom images that create an Amazon Machine Image (AMI), 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 as your starting point.

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.

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 How Image Builder uses the AWS Task Orchestrator and Executor application to manage components.

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 Image Builder launches the instance and applies all of the build phase customizations, the validation phase begins. During this phase, Image Builder ensures that all of the customizations work as expected, based on the configuration that the component specifies for the validate phase. If the instance validation succeeds, Image Builder stops the instance, creates an image, and then continues to the test stage.

Test stage:

Test phase

During this phase, Image Builder launches an instance from the image that it created after the validation phase completed successfully. Image Builder runs test components during this phase to verify that the instance is healthy and functions as expected.

Container host test phase

After Image Builder runs the test phase for all of the components that you selected in the container recipe, Image Builder runs this phase for container workflows. The container host test phase can run additional tests that validate container management and custom runtime configurations.

Workflow

Workflows define the sequence of steps that Image Builder performs when it creates a new image. All images have build and test workflows. Containers have an additional workflow for distribution.

Workflow types

`BUILD`

Covers build stage configuration for every image created.

`TEST`

Covers test stage configuration for every image created.

`DISTRIBUTION`

Covers distribution workflow for container images.

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 recipe configuration. For more information about product and service integration for your custom images, see Integrate products and services in Image Builder.

How EC2 Image Builder works

When you use the EC2 Image Builder console to create a custom image pipeline, the system 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.

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 Manual set up to develop custom components with AWSTOE.

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 Use components to customize your Image Builder image. For more information about creating components with AWSTOE, see How Image Builder uses the AWS Task Orchestrator and Executor application to manage components.

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.

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 if `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 from the console, or Create a new container recipe version with the console.

Sharing Resources

To share components, recipes, or images with other accounts or within AWS Organizations, see Share Image Builder resources with AWS RAM.

Compliance

For Center for Internet Security (CIS) Benchmarks, EC2 Image Builder uses Amazon Inspector to perform assessments for exposure, vulnerabilities, and deviations from best practices and compliance standards. For example, Image Builder assesses unintended network accessibility, unpatched CVEs, public internet connectivity, and remote root login activation. 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 Inspector User Guide*. For more information, see Center for Internet Security (CIS) Benchmarks.

Image Builder provides STIG hardening components to help you more efficiently build compliant images for baseline STIG standards. These STIG components scan for misconfigurations and run a remediation script. There are no additional charges for using STIG-compliant components. For a complete list of STIG components available through Image Builder, see Amazon managed STIG hardening components for Image Builder.

Semantic versioning in Image Builder

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, or 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

Get set up to build custom images with Image Builder

Before you build images with EC2 Image Builder, verify that you've met the following prerequisites to create an image pipeline. Unless specifically stated otherwise, these prerequisites are required for all types of pipelines.

Prerequisites

- Image Builder service-linked role
- Configuration requirements
- Container repository for container image pipelines
- Dedicated host for macOS images
- IAM prerequisites
- Systems Manager Agent prerequisites

After you've met the prerequisites, you can manage EC2 Image Builder from any of the following interfaces.

- EC2 Image Builder console
- Image Builder commands in the AWS CLI
- EC2 Image Builder API Reference
- AWS SDKs and Tools

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 Use IAM service-linked roles for Image Builder.

Configuration requirements

- Image Builder supports AWS PrivateLink. For more information about configuring VPC endpoints for Image Builder, see Image Builder and AWS PrivateLink interface VPC endpoints.

- The instances that Image Builder uses to build container images must have internet access to download the AWS CLI from Amazon S3, and to download a base image from the Docker Hub repository, if applicable. Image Builder uses the AWS CLI to get the Dockerfile from the container recipe, where it is stored as data.
- The instances that Image Builder uses to build images and run tests must have access to the Systems Manager service. 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*.

  macOS

  TBD - need something about the prerequisites

Container repository for 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.

Image Builder uses Amazon ECR as its target repository for container images. To create an Amazon ECR repository, follow the steps described in Creating a repository in the *Amazon Elastic Container Registry User Guide*.

Dedicated host for macOS images

Amazon EC2 Mac instances require a Dedicated Host running on a metal instance type. Before you create a custom macOS image, you must Allocate a Dedicated Host to your account. For more information about Mac instances and a list of instance types that natively support the macOS operating system, see Amazon EC2 Mac instances in the *Amazon EC2 User Guide*.

When you've created a Dedicated Host, you can configure settings in the infrastructure configuration resource for your image. The infrastructure configuration includes placement properties where you can specify the host, host placement group, or Availability Zone where the instances that launch from your image should go.

IAM prerequisites

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:

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "UploadFileToS3Bucket",
   "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 managed policy to the role. For more information, see Share Image Builder resources with AWS RAM.

Systems Manager Agent prerequisites

EC2 Image Builder runs AWS Systems Manager (Systems Manager) Agent on the EC2 instances it launches to build and test your image. Image Builder collects additional information about the instance used during the build phase with Systems Manager Inventory. This information includes the operating system (OS) name and version, as well as the list of packages and their respective versions as reported by your operating system.

To opt out of collecting this information, select the method that matches your preferred environment:

- **Image Builder console –** Deselect the **Enable enhanced metadata collection** check box.
- **AWS CLI –** Specify the `--no-enhanced-image-metadata-enabled` option
- **Image Builder API or SDKs –** Set the `enhancedImageMetadataEnabled` parameter to `false`.

Image Builder uses `RunCommand` to send actions to your build and test instance as part of the image build and test workflow. You can't opt out of the use of `RunCommand` to send actions to your build and test instance.

Learn how to create custom images with Image Builder tutorials

There are many ways to build custom images and components with EC2 Image Builder. Tutorials help you learn about key Image Builder concepts. Each tutorial presents a use case with steps that you can follow for the first time. The instructions use defaults where possible to assist with learning the overall process. After you use one of the tutorials, you can explore more ways to customize your own images.

Build your first image

The following tutorials show you how to build your first image with the Image Builder console wizard. At the end of the tutorial you'll have created the following set of Image Builder resources. The final step in the tutorial is to clean up the resources you created.

- An image recipe for your Amazon Machine Image (AMI) or a container recipe for your container image.
- An infrastructure configuration resource with default settings.
- A distribution settings resource with default settings that distributes the output to the source Region (your account in the Region that you use to run the console wizard).
- An image pipeline that uses the listed resources to build your output image with the default image build workflows.
- An output AMI or container image.

Console wizard tutorials

- Pipeline wizard: Create AMI
- Pipeline wizard: Create container image

Create a custom component with input parameters

The following tutorial shows you how to create a custom component that defines input parameters, and then set the values from your Image Builder recipe.

Custom component with parameters

Use Systems Manager parameters with Image Builder

The following tutorial shows you how to create an AWS Systems Manager Parameter Store parameter and use it in an image recipe.

Use a base image parameter in your recipe

You can also use Parameter Store parameters in AMI distribution settings to store your output image ID, and in custom components. For more information, see Create and update AMI distribution configurations for distributions, and Use Systems Manager Parameter Store parameters for custom components.

Tutorial: Create an image pipeline with output AMI from the 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
- Step 2: Choose recipe
- Step 3: Define infrastructure configuration - optional
- Step 4: Define distribution settings - optional
- Step 5: Review
- Step 6: Clean up

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 in Image Builder.

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 Linux AMIs that are managed by Amazon. For this tutorial, select the `Amazon Linux 2 x86` 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 in Image Builder](#).

5. In the **Instance configuration** section, keep the default values for the **Systems Manager agent**. This results in Image Builder keeping the Systems Manager agent after the build and tests are complete, to include the Systems Manager agent in your 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** panel, choose **Add build components**, and select `Amazon managed` from the component owner filter list. This opens a selection panel on the right side of the console interface where you can browse and filter the components that are available.

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

> **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 in Image Builder.

d. Choose **Add to recipe** to add the component to your recipe. This closes the component selection panel.

e. Back in the **Build components** panel, the component that you added is displayed.

7. **Reorder components (optional)**

If you've 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.

> **Note**
>
> CIS hardening components don't follow the standard component ordering rules in Image Builder recipes. The CIS hardening components always run last to ensure that the benchmark tests run against your output image.

a. Repeat the prior steps to add the `update-linux-kernel-5` component to your recipe.

b. The component you just added has an input parameter for the kernel version. To expand settings for **Versioning options** or **Input parameters**, you can choose the arrow next to the name of the setting. To expand all of the settings for all selected components, you can toggle the **Expand all** switch off and on. For more information about using input parameters in your components, and setting them in your recipes, see Tutorial: Create a custom component with input parameters.

c. Choose one of the components, and drag it up or down to change the order in which the components will run.

d. To remove the `update-linux-kernel-5` component, choose X from the upper right corner of the component box.

Repeat this step to remove any other components you might have added, leaving only the `update-linux` component selected.

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 for the EC2 build and test instances that are used to configure your image. For more information about infrastructure configuration settings, see CreateInfrastructureConfiguration in the *EC2 Image Builder API Reference*.

For this tutorial, we are using the default settings.

> **Note**
>
> To specify a subnet to use for a private VPC, you can create your own custom infrastructure configuration, or use settings that you have already created.

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

Step 4: Define distribution settings - optional

Distribution configurations include the output AMI name, specific Region settings for encryption, launch permissions, and AWS accounts, organizations, and organizational units (OUs) that can launch the output AMI, 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 more information about configuring your distribution settings, see Manage Image Builder distribution settings.

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**.

Tutorial: Create an image pipeline with output Docker container image from the 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
- Step 2: Choose recipe
- Step 3: Define infrastructure configuration - optional
- Step 4: Define distribution settings - optional
- Step 5: Review
- Step 6: Clean up

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*).
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 in Image Builder.

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 in Image Builder.

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 in Image Builder.

   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 [Tutorial: Create a custom component with input parameters](#).

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.

> **Note**
>
> CIS hardening components don't follow the standard component ordering rules in Image Builder recipes. The CIS hardening components always run last to ensure that the benchmark tests run against your output image.

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.

   By default, Image Builder uses the following contextual variables in your Dockerfile.

parentImage (required)

At build time, this variable resolves to the base image for your recipe.

Example:

```
FROM
{{{ imagebuilder:parentImage }}}
```

environments (required if components are specified)

This variable will resolves to a script that runs components.

Example:

```
{{{ imagebuilder:environments }}}
```

components (optional)

Image Builder resolves build and test component scripts for the components that the container recipe includes. This variable can be placed anywhere in the Dockerfile, after the environments variable.

Example:

```
{{{ imagebuilder:components }}}
```

7. In the **Target repository** section, specify the name of the Amazon ECR repository that you created as a prerequisite for this tutorial. This repository is used as the default setting for the distribution configuration in the Region where the pipeline runs (Region 1).

> **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 container images. You can also create your own custom infrastructure configuration, or use settings that you have already created. For more information about infrastructure configuration settings, see CreateInfrastructureConfiguration in the *EC2 Image Builder API Reference*.

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 specified in your container recipe for the Region where your pipeline runs (Region 1). If you choose `Create new distribution settings`, you can override the ECR repository for the current Region, and add more Regions for distribution.

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**.

Tutorial: Create a custom component with input parameters

You can manage Image Builder components, including creating and setting component parameters, directly from the EC2 Image Builder console, from the AWS CLI, or from the Image Builder API or 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 at runtime.

> **Important**
>
> Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets Manager, see What is Secrets Manager? in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

Use parameters in your YAML component document

To build a component, you must provide a YAML or JSON application component document. The document contains the code that runs during the phases and steps that you define to provide customization for your image. 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 a file editing tool to create a component document file. Documentation examples use a file named *hello-world-test.yaml* that includes 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 a file editing tool to create a component document file. Documentation examples use a file named *hello-world-test.yaml* that includes 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.

macOS

Create a YAML component document

Use a file editing tool to create a component document file. Documentation examples use a file named *hello-world-test.yaml* that includes the following content:

```
# Document Start
#
name: "HelloWorldTestingDocument-macOS"
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.

For more information about the phases, steps, and syntax for AWSTOE application component documents, see Use documents in AWSTOE. For more information about parameters and their requirements, see the Parameters 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 create a component directly from your YAML document with the Image Builder console, see [Create a custom component from the console](#).
- To create a component from the command line with the Image Builder **create-component** command, see [Create a custom component from the AWS CLI](#). 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 from the 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 input parameter, the default value appears in the **Value** box with greyed-out text. If no other value is entered, Image Builder uses the default value.

Use a base image parameter in your recipe

When you create a recipe for image customizations, there are several ways to identify the base image that you start with. If you specify the Amazon Machine Image (AMI) ID for your base image and that base image is updated, its AMI ID might change and you would need to update your recipe to match.

Instead of changing your recipe each time the base image ID changes, you can define an AWS Systems Manager Parameter Store parameter (SSM parameter) to store the value of your base image AMI ID, and then use the parameter to specify the base image in your recipe. For AWS managed AMIs, you can use a public parameter for the latest version.

This tutorial walks you through the process of creating an AMI ID parameter and using it in an image recipe. Image Builder steps in this tutorial are console-based.

Contents

Step 1: Find or create a Parameter Store parameter

The process for this step depends on the type of AMI that you specify for your base image. For AWS managed AMIs, you can use a public parameter that refers to the current version. Some parameters might not be available in all AWS Regions.

To begin, open the tab that corresponds to your AMI.

AWS managed AMI

If your base image is an AWS managed AMI, you can use public parameters to specify the AMI ID, rather than creating your own parameter. To find the public parameter for your AMI, see Discovering public parameters in the *AWS Systems Manager User Guide*.

Custom AMI

To create an AMI ID parameter, follow the instructions for Creating Parameter Store parameters in Systems Manager with the console, AWS CLI, or PowerShell. Provide the following values to ensure that the parameter value is an AMI ID.

Parameter tier: `Standard`

Type: `String`

Data type: Select `aws:ec2:image`. When you specify this type, the system validates the value that's entered to ensure that it's an AMI ID.

Value: Enter a valid AMI ID (for example, ***`ami-1234567890abcdef1`***).

Step 2: Configure IAM permissions (optional)

To use a Systems Manager Parameter Store parameter (SSM parameter), whether public or private, the following Systems Manager Parameter Store actions must be specified in an IAM role, with the parameter listed as a resource. The Image Builder service-linked role grants permission to get public parameters, or to get or update private parameters that have an `/imagebuilder/` prefix. For private parameters that don't have that prefix, you can add permission to your execution role.

- `ssm:GetParameter` – This action allows you to use an SSM parameter to specify the base image in your recipe.
- `ssm:PutParameter` – This action allows you to store the output AMI ID in an SSM parameter during distribution. Policy definition looks the same, but this tutorial does not include the put action in the example policy.

1. **Create a custom role (optional)**

   When you create a pipeline or use the create-image command in the AWS CLI, you can only specify one Image Builder execution role. If you have defined an Image Builder workflow execution role, you would add any additional feature permissions to that role. Otherwise, you would create a new custom role that includes the required permissions. If you already have a custom execution role defined, you can skip this step.

   Follow the process for Creating a role to delegate permissions to an AWS service in the *AWS Identity and Access Management User Guide*.

2. **Add permissions to your custom role**

   To add the SSM parameter permissions to your custom role, follow the Update the permissions policy for a role process in the *AWS Identity and Access Management User Guide*.

   The following policy example shows the `ssm:GetParameter` action with a parameter that's created in your account.

   JSON

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

```
      "Sid": "PrivateParameterCustomRole",
      "Effect": "Allow",
      "Action": "ssm:GetParameter",
      "Resource": "arn:aws:ssm:*:111122223333:parameter/parameter-name"
    }
  ]
}
```

For more information about public parameter resources, see Calling AMI public parameters in the *AWS Systems Manager User Guide*.

Step 3: Create an Image Recipe that uses the parameter

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image recipes**, then choose **Create image recipe** from the list page.
3. Fill out the **Base image** section, as follows:

   a. Choose the **Use custom AMI** option. This displays additional fields where you can enter the AMI ID or an SSM parameter that contains the AMI ID.
   b. Choose the **SSM parameter** option.
   c. In the **SSM parameter** field, enter the parameter name or Amazon Resource Name (ARN) of the parameter that you created in Step 1. If you enter the name, it will ***not*** have the prefix in the console.
4. Complete the remaining recipe configuration as needed.

> **Note**
>
> If you set the parent image through other interfaces, such as the AWS CLI, the parameter name must have a prefix of `ssm:` (for example, `ssm:/ImageBuilder-Tutorial/BaseAMI`.

Use components to customize your Image Builder image

Image Builder uses the AWS Task Orchestrator and Executor (AWSTOE) component management application to orchestrate complex workflows. Build and test components that work with the AWSTOE application are based on YAML documents that define the scripts to customize or test your image. For AMI images, Image Builder installs components and the AWSTOE component management application on its Amazon EC2 build and test instances. For container images, the components and AWSTOE component management 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.

> **Note**
>
> When a component that is managed by Amazon reaches the end of its support lifespan, it is no longer maintained. About four weeks before this occurs, any accounts that are using the component receive notification, and a list of the affected recipes in their account from their AWS Health Dashboard. To learn more about AWS Health, see AWS Health User Guide.

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 recipe can include components that install an application or modify the operating system firewall settings.

   The following phases from your component document 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, there are some differences between images that create AMIs and container images. For AMI workflows, Image Builder launches an EC2 instance from the snapshot that it 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. For container workflows, the tests run on the same instance that was used for building.

   The following phase from your component document runs for every component that is included in the recipe during the image build test stage:

   - test

   This component 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 the AWSTOE application framework 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 image build stage, the component document must define at least one of these phases: `build` or `validate`. For a component to run during the image test stage, the component document must define the `test` phase, and no other phases.
> Since Image Builder runs the stages independently, chaining references in component 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 Tutorial: Create a custom component with input parameters.

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 the AWSTOE component document framework for custom components.

> **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 section of this guide.

This section covers how to list, view, create, and import components, using the Image Builder console or commands in the AWS CLI.

Topics

- List and view component details
- Use AWS Marketplace components to customize your image
- Use managed components to customize your Image Builder image
- Develop custom components for your Image Builder image
- How Image Builder uses the AWS Task Orchestrator and Executor application to manage components

List and view component details

This section describes how you can find information and view details for the components that you use in your EC2 Image Builder recipes.

Component details

- List Image Builder components
- List component build versions from the AWS CLI
- Get component details from the AWS CLI
- Get component policy details from the AWS CLI

List Image Builder components

You can use one of the following methods to list and filter Image Builder components.

AWS Management Console

To display a list of components in the AWS Management 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. By default, Image Builder shows a list of components that your account owns.
3. You can optionally filter on component ownership. To see components that you don't own, but have access to, expand the owner type dropdown list and select one of the values. The owner type list is located in the search bar, next to the search text box. You can select from the following values:

   - **AWS Marketplace** – Components that are associated directly with an AWS Marketplace product subscription.
   - **Quick start (Amazon managed)** – Publicly available components that Amazon creates and maintains.
   - **Owned by me** – Components that you created. This is the default selection.
   - **Shared with me** – Components that others created and shared with you from their account.
   - **Third party managed** – Components that a third party owns that you subscribed to in AWS Marketplace.

AWS CLI

The following example shows how to use the **list-components** command to return a list of Image Builder components that your account owns.

```
aws imagebuilder list-components
```

You can optionally filter on component ownership. The owner attribute defines who owns the components that you want to list. By default, this request returns a list of components that your account owns. To filter the results by component owner, specify one of the following values with the `--owner` parameter when you run the **list-components** command.

Component owner values

- `AWSMarketplace`

- `Amazon`
- `Self`
- `Shared`
- `ThirdParty`

The following examples show the **list-components** command with the `--owner` parameter to filter results.

```
aws imagebuilder list-components --owner Self
{
    "requestId": "012a3456-b789-01cd-e234-fa5678b9012b",
    "componentVersionList": [
        {
            "arn": "arn:aws:imagebuilder:us-west-2:123456789012:component/sample-
component01/1.0.0",
            "name": "sample-component01",
            "version": "1.0.0",
            "platform": "Linux",
            "type": "BUILD",
            "owner": "123456789012",
            "dateCreated": "2020-09-24T16:58:24.444Z"
        },
        {
            "arn": "arn:aws:imagebuilder:us-west-2:123456789012:component/sample-
component01/1.0.1",
            "name": "sample-component01",
            "version": "1.0.1",
            "platform": "Linux",
            "type": "BUILD",
            "owner": "123456789012",
            "dateCreated": "2021-07-10T03:38:46.091Z"
        }
    ]
}
```

```
aws imagebuilder list-components --owner Amazon
```

```
aws imagebuilder list-components --owner Shared
```

```
aws imagebuilder list-components --owner ThirdParty
```

List component build versions from the AWS CLI

The following example shows how to use the **list-component-build-versions** 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 in Image Builder.

```
aws imagebuilder list-component-build-versions --component-version-arn
 arn:aws:imagebuilder:us-west-2:123456789012:component/example-component/1.0.1
{
    "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "componentSummaryList": [
        {
            "arn": "arn:aws:imagebuilder:us-west-2:123456789012:component/
examplecomponent/1.0.1/1",
            "name": "examplecomponent",
            "version": "1.0.1",
            "platform": "Linux",
            "type": "BUILD",
            "owner": "123456789012",
            "description": "An example component that builds, validates and tests an
 image",
            "changeDescription": "Updated version.",
            "dateCreated": "2020-02-19T18:53:45.940Z",
            "tags": {
                "KeyName": "KeyValue"
            }
        }
    ]
}
```

Get component details from the AWS CLI

The following example shows how to use the **get-component** command to get component details when you specify the component's Amazon Resource Name (ARN).

```
aws imagebuilder get-component --component-build-version-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/example-component/1.0.1/1
```

```
{
    "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11112",
    "component": {
        "arn": "arn:aws:imagebuilder:us-west-2:123456789012:component/
examplecomponent/1.0.1/1",
        "name": "examplecomponent",
        "version": "1.0.1",
        "type": "BUILD",
        "platform": "Linux",
        "owner": "123456789012",
        "data": "name: HelloWorldTestingDocument\ndescription: This is hello world
 testing document... etc.\"\n",
        "encrypted": true,
        "dateCreated": "2020-09-24T16:58:24.444Z",
        "tags": {}
    }
}
```

Get component policy details from the AWS CLI

The following example shows how to use the **get-component-policy** command to get details of a component policy when you specify the component's ARN.

```
aws imagebuilder get-component-policy --component-arn arn:aws:imagebuilder:us-
west-2:123456789012:component/example-component/1.0.1
```

Use AWS Marketplace components to customize your image

In addition to a large selection of images created by Independent Software Vendors (ISVs), the AWS Marketplace offers components that you can use to customize your own Image Builder images. You must subscribe to these AWS Marketplace components before you can use them in your image recipe to build a new image.

When you specify an AWS Marketplace component in an image recipe, Image Builder validates the subscription and performs dependency checks to ensure that you have the resources that you need to use it. When validation succeeds, Image Builder creates secure downloads for the component and its artifacts for use by image pipeline builds.

Discover AWS Marketplace components

You can discover AWS Marketplace software components to use in your recipes from the **Discover products** page in the Image Builder console, as follows.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. From the navigation pane, choose **Discover products** in the **AWS Marketplace** section.
3. Select the **Components** tab. This tab lists all of the AWS Marketplace products that use the delivery option that includes associated components in AWS Marketplace.
4. To search for specific software products that include components, you can enter part of the name in the search bar or filter by `Status`, `Operating System`, `Publisher`, or `Categories`. The search bar also contains pagination controls for your results.

Results

Each AWS Marketplace product has its own detail panel that includes the following information.

The AWS Marketplace product name and logo

The software product name is linked to the product detail in AWS Marketplace. You can select the link to learn more about the product in AWS Marketplace. Alternatively, you can view a summary of subscription options and subscribe directly from the search results with the **View subscription options** button if you've already done your research.

Version

This contains the primary version of the component.

Operating system

The operating system that the component is designed to run on.

Publisher

The publisher of the component. This is linked to the publisher detail page in AWS Marketplace. The publisher detail page opens in a new tab in your browser.

Categories

One or more AWS Marketplace product categories that apply for the component.

Status

Shows whether you are subscribed to this product. If you're not subscribed, you can choose **View subscription options** to see a summary of the subscription options for the AWS Marketplace product, and optionally subscribe directly from the Image Builder console.

Associated components

If the AWS Marketplace product has one or more versions that are included with your subscription, they are shown in the **Associated components** section. The section is collapsed initially, and displays a count of the associated components. You can expand the section to see more details.

> **Note**
>
> The Center for Internet Security (CIS) component that's associated with their AWS Marketplace image product is not shown in the **Discover products** results. If you subscribe to their image product, the associated component is shown in the **Subscriptions** page, and as a third-party component in the **Create image recipe** dialog. For more information about the component, see CIS hardening components.

Subscribe to AWS Marketplace components

After you've found an AWS Marketplace product with components that you want to use in your recipes, you can subscribe to it directly from the Image Builder console, as follows, or you can subscribe from the AWS Marketplace console.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. From the navigation pane, choose **Discover products** in the **AWS Marketplace** section.
3. Select the **Components** tab. This tab lists all of the AWS Marketplace products that use the delivery option that includes associated components in AWS Marketplace.
4. To search for a specific AWS Marketplace product, enter part of the name in the search bar. If you know the publisher, but not the exact product name or how to spell it, you can also filter by `Publisher` to get a list of products that the publisher has available.
5. Select the product that you want to subscribe to from the results list, and choose **View subscription options**. This shows a summary of subscription options for the AWS Marketplace product.

6. Select **Subscribe** to subscribe to the product without leaving the Image Builder console. You are notified that the subscription is being processed. After you're subscribed, the **Status** is updated to `Subscribed`.

For more information about the AWS Marketplace products that you're currently subscribed to, see the console steps described in AWS Marketplace subscriptions in Image Builder.

Use an AWS Marketplace component in an Image Builder image recipe

You can use AWS Marketplace components in your Image Builder image recipes the same way that you use other types of components. For most of the components that are associated with an AWS Marketplace image product, the ownership category is `AWS Marketplace`. For example, to use a build component from an AWS Marketplace product that you've subscribed to, choose **Add build components**, and select `AWS Marketplace` from the list. This opens a selection panel on the right side of the console interface that lists AWS Marketplace components.

> **Note**
>
> If you're looking for the CIS hardening component, select `Third party managed`, from the ownership list instead of `AWS Marketplace`.

For more information about how to select, arrange, and configure parameters for your components, see Create a new version of an image recipe.

Use managed components to customize your Image Builder image

Managed components are created by AWS, sometimes in partnership with a third-party organization, such as the Center for Internet Security (CIS), for example. When you use managed components in your image or container recipes, Amazon provides the latest component versions that have patches and other updates applied. To get a list of components, or to get component information, see List and view component details.

The following list of featured AWS managed components includes a component that's available for you to use when you subscribe to CIS hardened AMIs through the AWS Marketplace.

Featured components

- Distributor package managed component application install for Image Builder Windows images
- CIS hardening components
- Amazon managed STIG hardening components for Image Builder

Distributor package managed component application install for Image Builder Windows images

AWS Systems Manager Distributor helps you package and publish software to AWS Systems Manager managed nodes. You can package and publish your own software or use Distributor to find and publish AWS-provided agent software packages. For more information about Systems Manager Distributor, see AWS Systems Manager Distributor in the *AWS Systems Manager User Guide*.

Managed components for Distributor

The following Image Builder managed components use AWS Systems Manager Distributor to install application packages on Windows instances.

- The `distributor-package-windows` managed component uses AWS Systems Manager Distributor to install application packages that you specify on your Windows image build instance. To configure parameters when you include this component in your recipe, see Configure `distributor-package-windows` as a standalone component.
- The `aws-vss-components-windows` component uses AWS Systems Manager Distributor to install the `AwsVssComponents` package on your Windows image build instance. To configure parameters when you include this component in your recipe, see Configure `aws-vss-components-windows` as a standalone component.

For more information about how to use managed components in your Image Builder recipe, see Create a new version of an image recipe for image recipes or Create a new version of a container recipe for container recipes. For more information about the `AwsVssComponents` package, see Create a VSS application-consistent snapshot in the *Amazon EC2 User Guide*.

Prerequisites

Before you use Image Builder components that rely on Systems Manager Distributor to install application packages, you must ensure that the following prerequisites are met.

- Image Builder components that use Systems Manager Distributor to install application packages on your instance need permission to call the Systems Manager API. Before you use the components in an Image Builder recipe, you must create the IAM policy and role that grant permission. To configure permissions, see Configure Systems Manager Distributor permissions.

> **Note**
>
> Image Builder doesn't currently support Systems Manager Distributor packages that reboot the instance. For example, the `AWSNVMe`, `AWSPVDrivers`, and `AwsEnaNetworkDriver` Distributor packages reboot the instance, and so are not allowed.

Configure Systems Manager Distributor permissions

The `distributor-package-windows` component and other components that use it, such as `aws-vss-components-windows`, require additional permission on the build instance to run. The build instance must be able to call the Systems Manager API to begin a Distributor installation and poll for the result.

Follow these procedures in the AWS Management Console to create a custom IAM policy and role that grant permission for Image Builder components to install Systems Manager Distributor packages from the build instance.

Step 1: Create a policy

Create an IAM policy for Distributor permissions.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then choose **Create policy**.
3. On the **Create policy** page, choose the **JSON** tab, and then replace the default content with the following JSON policy, substituting partition, Region, and account ID as necessary, or using wildcards.

   JSON

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

```
        {
            "Sid": "AllowDistributorSendCommand",
            "Effect": "Allow",
            "Action": "ssm:SendCommand",
            "Resource": [
                "arn:aws:ssm:*::document/AWS-ConfigureAWSPackage",
                "arn:aws:ec2:*:111122223333:instance/*"
            ]
        },
        {
            "Sid": "AllowGetCommandInvocation",
            "Effect": "Allow",
            "Action": "ssm:GetCommandInvocation",
            "Resource": "*"
        }
    ]
}
```

4. Choose **Review policy**.
5. For **Name**, enter a name to identify the policy, such as *InvokeDistributor* or another name that you prefer.
6. (Optional) For **Description**, enter a description of the role's purpose.
7. Choose **Create policy**.

Step 2: Create a role

Create an IAM role for Distributor permissions.

1. From the IAM console navigation pane, choose **Roles**, and then choose **Create role**.
2. Under **Select type of trusted entity**, choose **AWS service**.
3. Immediately under **Choose the service that will use this role**, choose **EC2**, and then choose **Next: Permissions**.
4. Under **Select your use case**, choose **EC2**, and then choose **Next: Permissions**.
5. In the list of policies, select the check box next to **AmazonSSMManagedInstanceCore**. (Type `SSM` in the search box if you need to narrow the list.)
6. In this list of policies, choose the box next to **EC2InstanceProfileForImageBuilder**. (Type `ImageBuilder` in the search box if you need to narrow the list.)
7. Choose **Next: Tags**.

8. (Optional) Add one or more tag key value pairs to organize, track, or control access for this role, and then choose **Next: Review**.
9. For **Role name**, enter a name for the role, such as `InvokeDistributor` or another name that you prefer.
10. (Optional) For **Role description**, replace the default text with a description of this role's purpose.
11. Choose **Create role**. The system returns you to the **Roles** page.

Step 3: Attach the policy to the role

The final step to set up your Distributor permissions is to attach the IAM policy to the IAM role.

1. From the **Roles** page in the IAM console, choose the role that you just created. The role **Summary page** opens.
2. Choose **Attach policies**.
3. Search for the policy that you created in the previous procedure and select the check box next to the name.
4. Choose **Attach policy**.

Use this role in the Image Builder Infrastructure Configuration resource for any image that includes components that use Systems Manager Distributor. For more information, see Create an infrastructure configuration.

Configure `distributor-package-windows` as a standalone component

To use the `distributor-package-windows` component in a recipe, set the following parameters that configure the package to install.

> **Note**
>
> Before you use the `distributor-package-windows` component in a recipe, you must ensure that all of the Prerequisites are met.

- **Action** (Required) – Specify whether to install or uninstall the package. Valid values include `Install` and `Uninstall`. The value defaults to `Install`.

- **PackageName** (Required) – The name of the Distributor package to install or uninstall. For a list of valid package names, see Find Distributor packages.
- **PackageVersion** (Optional) – The version of the Distributor package to install. PackageVersion defaults to the recommended version.
- **AdditionalArguments** (Optional) – A JSON string that contains the additional parameters to provide to your script to install, uninstall, or update a package. For more information, see **additionalArguments** in the aws:configurePackage **Inputs** section of the **Systems Manager Command document plugin reference** page.

Configure `aws-vss-components-windows` as a standalone component

When you use the `aws-vss-components-windows` component in a recipe, you can optionally set the `PackageVersion` parameter to use a specific version of the `AwsVssComponents` package. When you leave out this parameter, the component defaults to use the recommended version of the `AwsVssComponents` package.

> **Note**
>
> Before you use the `aws-vss-components-windows` component in a recipe, you must ensure that all of the Prerequisites are met.

Find Distributor packages

Amazon and third parties provide public packages that you can install with Systems Manager Distributor.

To view available packages in the AWS Management Console, log into the AWS Systems Manager console and choose **Distributor** from the navigation pane. The **Distributor** page shows all of the packages that are available to you. For more information about listing available packages with the AWS CLI, see View packages (command line) in the *AWS Systems Manager User Guide*.

You can also create your own private Systems Manager Distributor packages. For more information, see Create a package in the *AWS Systems Manager User Guide*.

CIS hardening components

The Center for Internet Security (CIS) is a community-driven nonprofit organization. Their cybersecurity experts work together to develop IT security guidelines that safeguard public and private organizations against cyber threats. Their globally recognized set of best practices, known as CIS Benchmarks, help IT organizations around the world to securely configure their systems. For trending articles, blog posts, podcasts, webinars, and whitepapers, see CIS Insights on the *Center for Internet Security* website.

CIS Benchmarks

CIS creates and maintains a set of configuration guidelines, known as the CIS Benchmarks, which provide configuration best practices for specific technologies, including operating systems, cloud platforms, applications, databases, and more. CIS Benchmarks are recognized as an industry standard by organizations and standards such as PCI DSS, HIPAA, DoD Cloud Computing SRG, FISMA, DFARS, and FEDRAMP. To learn more, see CIS Benchmarks on the *Center for Internet Security* website.

CIS hardening components

When you subscribe to a CIS Hardened Image in AWS Marketplace, you also get access to the associated hardening component that runs a script to enforce CIS Benchmarks Level 1 guidelines for your configuration. The CIS organization owns and maintains CIS hardening components to ensure that they reflect the latest guidelines.

> **Note**
>
> CIS hardening components don't follow the standard component ordering rules in Image Builder recipes. The CIS hardening components always run last to ensure that the benchmark tests run against your output image.

Amazon managed STIG hardening components for Image Builder

Security Technical Implementation Guides (STIGs) are the configuration hardening 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.

Image Builder provides STIG hardening components to help you more efficiently build compliant images for baseline STIG standards. These STIG components scan for misconfigurations and run a remediation script. There are no additional charges for using STIG-compliant components.

> **Important**
>
> With few exceptions, STIG hardening components do not install third-party packages. If third-party packages are already installed on the instance, and if there are related STIGs that Image Builder supports for that package, the hardening component applies them.

This page lists all STIGs that Image Builder supports that are applied to the EC2 instances that Image Builder launches when you build and test a new image. If you want to apply additional STIG settings to your image, you can create a custom component to configure it. For more information about custom components and how to create them, see Use components to customize your Image Builder image.

When you create an image, the STIG hardening components log whether supported STIGs are applied or skipped. We recommend that you review the Image Builder logs for your images that use STIG hardening components. For more information about how to access and review Image Builder logs, see Troubleshoot pipeline builds.

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 hardening components

AWSTOE Windows STIG hardening components are designed for standalone servers and apply Local Group Policy. STIG-compliant hardening components install InstallRoot from the Department of Defense (DoD) on Windows infrastructure to download, install, and update the DoD certificates. They also remove unnecessary certificates to maintain STIG compliance. Currently, STIG baselines are supported for the following versions of Windows Server: 2012 R2, 2016, 2019, and 2022.

This section lists current settings for each of the Windows STIG hardening components, followed by a version history log.

STIG-Build-Windows-Low version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see STIG Viewing Tools.

- **Windows Server 2022 STIG Version 2 Release 4**

  V-254335, V-254336, V-254337, V-254338, V-254351, V-254357, V-254363, and V-254481
- **Windows Server 2019 STIG Version 3 Release 4**

  V-205691, V-205819, V-205858, V-205859, V-205860, V-205870, V-205871, and V-205923
- **Windows Server 2016 STIG Version 2 Release 10**

  V-224916, V-224917, V-224918, V-224919, V-224931, V-224942, and V-225060
- **Windows Server 2012 R2 MS STIG Version 3 Release 5**

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

- **Microsoft .NET Framework 4.0 STIG Version 2 Release 6**

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

- **Windows Firewall STIG Version 2 Release 2**

  V-241994, V-241995, V-241996, V-241999, V-242000, V-242001, V-242006, V-242007, and V-242008

- **Internet Explorer 11 STIG Version 2 Release 5**

  V-223016, V-223056, and V-223078

- **Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)**

  V-235727, V-235731, V-235751, V-235752, and V-235765

STIG-Build-Windows-Medium version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see STIG Viewing Tools.

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

- **Windows Server 2022 STIG Version 2 Release 4**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

  V-254247, V-254265, V-254269, V-254270, V-254271, V-254272, V-254273, V-254274, V-254276, V-254277, V-254278, V-254285, V-254286, V-254287, V-254288, V-254289, V-254290, V-254291, V-254292, V-254300, V-254301, V-254302, V-254303, V-254304, V-254305, V-254306, V-254307, V-254308, V-254309, V-254310, V-254311, V-254312, V-254313, V-254314, V-254315, V-254316, V-254317, V-254318, V-254319, V-254320, V-254321, V-254322, V-254323, V-254324, V-254325, V-254326, V-254327, V-254328, V-254329, V-254330, V-254331, V-254332, V-254333, V-254334, V-254339, V-254341, V-254342, V-254344, V-254345, V-254346, V-254347, V-254348, V-254349, V-254350, V-254355, V-254356, V-254356, V-254358, V-254359, V-254360, V-254361, V-254362, V-254364, V-254365, V-254366, V-254367, V-254368, V-254369, V-254370, V-254371, V-254372, V-254373, V-254375, V-254376, V-254377, V-254379, V-254380, V-254382, V-254383, V-254384, V-254431, V-254432, V-254433, V-254434, V-254435, V-254436, V-254438, V-254439, V-254442, V-254443, V-254444, V-254445, V-254449, V-254450, V-254451, V-254452, V-254453, V-254454, V-254455, V-254456, V-254459, V-254460, V-254461, V-254462, V-254463, V-254464, V-254468, V-254470, V-254471, V-254472, V-254473, V-254476, V-254477, V-254478, V-254479, V-254480, V-254482, V-254483, V-254484, V-254485, V-254486, V-254487, V-254488, V-254489, V-254490, V-254493, V-254494, V-254495, V-254497, V-254499, V-254501, V-254502, V-254503, V-254504, V-254505, V-254507, V-254508, V-254509, V-254510, V-254511, and V-254512
- **Windows Server 2019 STIG Version 3 Release 4**

  Includes all supported STIG settings that the hardening component 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-205832, V-205833, V-205834, V-205835, V-205836, V-205837, V-205838, V-205839, V-205840, V-205841, V-205842, V-205861, V-205863, V-205865, V-205866, V-205867, V-205868, V-205869, V-205872, V-205873, V-205874, V-205911, V-205912, V-205915, V-205916, V-205917, V-205918, V-205920, V-205921, V-205922, V-205924, V-205925, V-236001, and V-257503

- **Windows Server 2016 STIG Version 2 Release 10**

  Includes all supported STIG settings that the hardening component 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-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-225059, 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, V-225093, V-236000, and V-257502

- **Windows Server 2012 R2 MS STIG Version 3 Release 5**

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

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

- **Microsoft .NET Framework 4.0 STIG Version 2 Release 6**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus V-225238

- **Windows Firewall STIG Version 2 Release 2**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

V-241989, V-241990, V-241991, V-241993, V-241993, V-241998, and V-242003

- **Internet Explorer 11 STIG Version 2 Release 5**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

  V-223015, V-223017, V-223018, V-223019, V-223020, V-223021, V-223022, V-223023, V-223024, V-223025, V-223026, V-223027, V-223028, V-223029, V-223030, V-223031, V-223032, V-223033, V-223034, V-223035, V-223036, V-223037, V-223038, V-223039, V-223040, V-223041, V-223042, V-223043, V-223044, V-223045, V-223046, V-223048, V-223049, V-223050, V-223051, V-223052, V-223053, V-223054, V-223055, V-223057, V-223058, V-223059, V-223060, V-223061, V-223062, V-223063, V-223064, V-223065, V-223066, V-223067, V-223068, V-223069, V-223070, V-223071, V-223072, V-223073, V-223074, V-223075, V-223076, V-223077, V-223079, V-223080, V-223081, V-223082, V-223083, V-223084, V-223085, V-223086, V-223087, V-223088, V-223089, V-223090, V-223091, V-223092, V-223093, V-223094, V-223095, V-223096, V-223097, V-223098, V-223099, V-223100, V-223101, V-223102, V-223103, V-223104, V-223105, V-223106, V-223107, V-223108, V-223109, V-223110, V-223111, V-223112, V-223113, V-223114, V-223115, V-223116, V-223117, V-223118, V-223119, V-223120, V-223121, V-223122, V-223123, V-223124, V-223125, V-223126, V-223127, V-223128, V-223129, V-223130, V-223131, V-223132, V-223133, V-223134, V-223135, V-223136, V-223137, V-223138, V-223139, V-223140, V-223141, V-223142, V-223143, V-223144, V-223145, V-223146, V-223147, V-223148, V-223149, V-250540, and V-250541

- **Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

  V-235720, V-235721, V-235723, V-235724, V-235725, V-235726, V-235728, V-235729, V-235730, V-235732, V-235733, V-235734, V-235735, V-235736, V-235737, V-235738, V-235739, V-235740, V-235741, V-235742, V-235743, V-235744, V-235745, V-235746, V-235747, V-235748, V-235749, V-235750, V-235754, V-235756, V-235760, V-235761, V-235763, V-235764, V-235766, V-235767, V-235768, V-235769, V-235770, V-235771, V-235772, V-235773, V-235774, and V-246736

- **Microsoft Defender STIG Version 2 Release 4**

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:

V-213427, V-213429, V-213430, V-213431, V-213432, V-213433, V-213434, V-213435, V-213436, V-213437, V-213438, V-213439, V-213440, V-213441, V-213442, V-213443, V-213444, V-213445, V-213446, V-213447, V-213448, V-213449, V-213450, V-213451, V-213455, V-213464, V-213465, and V-213466

STIG-Build-Windows-High version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list of Windows STIGs, see the STIGs Document Library. For information about how to view the complete list, see STIG Viewing Tools.

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

- **Windows Server 2022 STIG Version 2 Release 4**

  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-254293, V-254352, V-254353, V-254354, V-254374, V-254378, V-254381, V-254446, V-254465, V-254466, V-254467, V-254469, V-254474, V-254475, and V-254500
- **Windows Server 2019 STIG Version 3 Release 4**

  Includes all supported STIG settings that the hardening component 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 10**

  Includes all supported STIG settings that the hardening component 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 MS STIG Version 3 Release 5**

  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-225274, V-225354, V-225364, V-225365, V-225366, V-225390, V-225396, V-225399, V-225444, V-225449, V-225491, V-225492, V-225493, V-225496, V-225497, V-225498, V-225505, V-225507, V-225547, V-225552, and V-225556

- **Microsoft .NET Framework 4.0 STIG Version 2 Release 6**

  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for the Microsoft .NET Framework. No additional STIG settings apply for Category I vulnerabilities.

- **Windows Firewall STIG Version 2 Release 2**

  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-241992, V-241997, and V-242002

- **Internet Explorer 11 STIG Version 2 Release 5**

  V-252910

- **Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)**

  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:

  V-235758 and V-235759

- **Microsoft Defender STIG Version 2 Release 4**

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:

V-213426, V-213452, and V-213453

STIG version history log for Windows

This section logs Windows hardening component version history for the quarterly STIG updates. To see the changes and published versions for a quarter, choose the title to expand the information.

2025 Q2 changes - 06/26/2025:

Updated STIG versions and applied STIGS for the 2025 Q2 release as follows:

STIG-Build-Windows-Low version 2025.2.x

- Windows Server 2022 STIG Version 2 Release 4
- Windows Server 2019 STIG Version 3 Release 4
- Windows Server 2016 STIG Version 2 Release 10
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 6
- Windows Firewall STIG Version 2 Release 2
- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)

STIG-Build-Windows-Medium version 2025.2.x

- Windows Server 2022 STIG Version 2 Release 4
- Windows Server 2019 STIG Version 3 Release 4
- Windows Server 2016 STIG Version 2 Release 10
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 6
- Windows Firewall STIG Version 2 Release 2
- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)

- Defender STIG Version 2 Release 4

STIG-Build-Windows-High version 2025.2.x

- Windows Server 2022 STIG Version 2 Release 4
- Windows Server 2019 STIG Version 3 Release 4
- Windows Server 2016 STIG Version 2 Release 10
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 6
- Windows Firewall STIG Version 2 Release 2
- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)
- Defender STIG Version 2 Release 4

2025 Q1 changes - 05/04/2025:

Updated STIGS for Internet Explorer 11 STIG Version 2 Release 5 for all STIG components for the 2025 first quarter release.

- STIG-Build-Windows-Low version 2025.1.x
- STIG-Build-Windows-Medium version 2025.1.x
- STIG-Build-Windows-High version 2025.1.x

2024 Q4 changes - 02/04/2025:

Updated STIG versions and applied STIGS for the 2024 Q4 release as follows:

STIG-Build-Windows-Low version 2024.4.0

- Windows Server 2022 STIG Version 2 Release 2
- Windows Server 2019 STIG Version 3 Release 2
- Windows Server 2016 STIG Version 2 Release 9
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 2

- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)

STIG-Build-Windows-Medium version 2024.4.0

- Windows Server 2022 STIG Version 2 Release 2
- Windows Server 2019 STIG Version 3 Release 2
- Windows Server 2016 STIG Version 2 Release 9
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 2
- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)
- Defender STIG Version 2 Release 4

STIG-Build-Windows-High version 2024.4.0

- Windows Server 2022 STIG Version 2 Release 2
- Windows Server 2019 STIG Version 3 Release 2
- Windows Server 2016 STIG Version 2 Release 9
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 2
- Internet Explorer 11 STIG Version 2 Release 5
- Microsoft Edge STIG Version 2 Release 2 (Windows Server 2022 only)
- Defender STIG Version 2 Release 4

2024 Q3 changes - 10/04/2023 (no changes):

There were no changes for Windows component STIGS for the 2024 third quarter release.

2024 Q2 changes - 05/10/2024 (no changes):

There were no changes for Windows component STIGS for the 2024 second quarter release.

2024 Q1 changes - 02/06/2024 (no changes):

There were no changes for Windows component STIGS for the 2024 first quarter release.

2023 Q4 changes - 12/04/2023 (no changes):

There were no changes for Windows component STIGS for the 2023 fourth quarter release.

2023 Q3 changes - 10/04/2023 (no changes):

There were no changes for Windows component STIGS for the 2023 third quarter release.

2023 Q2 changes - 05/03/2023 (no changes):

There were no changes for Windows component STIGS for the 2023 second quarter release.

2023 Q1 changes - 03/27/2023 (no changes):

There were no changes for Windows component STIGS for the 2023 first quarter release.

2022 Q4 changes - 02/01/2023:

Updated STIG versions and applied STIGS for the 2022 Q4 release as follows:

STIG-Build-Windows-Low version 2022.4.x

- Windows Server 2022 STIG Version 1 Release 1
- Windows Server 2019 STIG Version 2 Release 5
- Windows Server 2016 STIG Version 2 Release 5
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 2 Release 3
- Microsoft Edge STIG Version 1 Release 6 (Windows Server 2022 only)

STIG-Build-Windows-Medium version 2022.4.x

- Windows Server 2022 STIG Version 1 Release 1
- Windows Server 2019 STIG Version 2 Release 5
- Windows Server 2016 STIG Version 2 Release 5

- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 2 Release 3
- Microsoft Edge STIG Version 1 Release 6 (Windows Server 2022 only)
- Defender STIG Version 2 Release 4 (Windows Server 2022 only)

STIG-Build-Windows-High version 2022.4.x

- Windows Server 2022 STIG Version 1 Release 1
- Windows Server 2019 STIG Version 2 Release 5
- Windows Server 2016 STIG Version 2 Release 5
- Windows Server 2012 R2 MS STIG Version 3 Release 5
- Microsoft .NET Framework 4.0 STIG Version 2 Release 2
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 2 Release 3
- Microsoft Edge STIG Version 1 Release 6 (Windows Server 2022 only)
- Defender STIG Version 2 Release 4 (Windows Server 2022 only)

2022 Q3 changes - 09/30/2022 (no changes):

There were no changes for Windows component STIGS for the 2022 third quarter release.

2022 Q2 changes - 08/02/2022:

Updated STIG versions and applied STIGS for the 2022 Q2 release.

STIG-Build-Windows-Low version 1.5.x

- Windows Server 2019 STIG Version 2 Release 4
- Windows Server 2016 STIG Version 2 Release 4
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-Medium version 1.5.x

- Windows Server 2019 STIG Version 2 Release 4
- Windows Server 2016 STIG Version 2 Release 4
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-High version 1.5.x

- Windows Server 2019 STIG Version 2 Release 4
- Windows Server 2016 STIG Version 2 Release 4
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

2022 Q1 changes - 08/02/2022 (no changes):

There were no changes for Windows component STIGS for the 2022 first quarter release.

2021 Q4 changes - 12/20/2021:

Updated STIG versions and applied STIGS for the 2021 fourth quarter release.

STIG-Build-Windows-Low version 1.5.x

- Windows Server 2019 STIG Version 2 Release 3
- Windows Server 2016 STIG Version 2 Release 3
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-Medium version 1.5.x

- Windows Server 2019 STIG Version 2 Release 3
- Windows Server 2016 STIG Version 2 Release 3
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-High version 1.5.x

- Windows Server 2019 STIG Version 2 Release 3
- Windows Server 2016 STIG Version 2 Release 3
- Windows Server 2012 R2 MS STIG Version 3 Release 3
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 2 Release 1
- Internet Explorer 11 STIG Version 1 Release 19

2021 Q3 changes - 09/30/2021:

Updated STIG versions and applied STIGS for the 2021 third quarter release.

STIG-Build-Windows-Low version 1.4.x

- Windows Server 2019 STIG Version 2 Release 2
- Windows Server 2016 STIG Version 2 Release 2
- Windows Server 2012 R2 MS STIG Version 3 Release 2
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 1 Release 7
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-Medium version 1.4.x

- Windows Server 2019 STIG Version 2 Release 2

- Windows Server 2016 STIG Version 2 Release 2
- Windows Server 2012 R2 MS STIG Version 3 Release 2
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 1 Release 7
- Internet Explorer 11 STIG Version 1 Release 19

STIG-Build-Windows-High version 1.4.x

- Windows Server 2019 STIG Version 2 Release 2
- Windows Server 2016 STIG Version 2 Release 2
- Windows Server 2012 R2 MS STIG Version 3 Release 2
- Microsoft .NET Framework 4.0 STIG Version 2 Release 1
- Windows Firewall STIG Version 1 Release 7
- Internet Explorer 11 STIG Version 1 Release 19

Linux STIG hardening components

This section contains information about Linux STIG hardening components, followed by a version history log. If the Linux distribution doesn't have STIG settings of its own, the hardening component applies RHEL settings.

The Linux components have optional input parameters that help you customize the following behaviors for your Linux instance.

- **InstallPackages (string)** If the value is No, the component does not install any additional software packages. If the value is `Yes`, the component installs additional software packages that are required for maximum compliance. The default is No.
- **SetDoDConsentBanner (string)** If the value is No, the DoD consent banner is not shown when you attach to an instance that has one of the STIG Linux components installed. If the value is `Yes`, the DoD consent banner is shown before you log in when you attach to an instance that has one of the STIG Linux components installed. You must acknowledge the banner before you can log in. The default is No.

  For an example of the consent banner, see the Disclaimer Department of Defense Privacy and Consent Notice that appears when you access the DLA Document Services website.

The hardening component applies supported STIG settings to the infrastructure 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
- Amazon Linux 2023 (AL 2023)

RHEL 9 STIG settings

- RHEL 9
- CentOS Stream 9

STIG-Build-Linux-Low version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list, see the [STIGs Document Library](). For information about how to view the complete list, see [STIG Viewing Tools]().

RHEL 7 STIG Version 3 Release 15

- **RHEL 7/CentOS 7/AL2**

  V-204452, V-204576, and V-204605

RHEL 8 STIG Version 2 Release 3

- **RHEL 8/CentOS 8/AL 2023**

  V-230241, 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, V-230499, V-244527, and V-274141

RHEL 9 STIG Version 2 Release 4

- **RHEL 9/CentOS Stream 9**

  V-257782, V-257795, V-257796, V-257824, V-257880, V-257946, V-257947, V-258037, V-258067, V-258069, V-258076, V-258138, and V-258173

Ubuntu 18.04 STIG Version 2 Release 15

V-219163, V-219164, V-219165, V-219172, V-219173, V-219174, V-219175, V-219178, V-219179, V-219180, V-219210, V-219301, V-219322, V-219327, V-219332, and V-219333

Ubuntu 20.04 STIG Version 2 Release 2

V-238202, V-238203, V-238221, V-238222, V-238223, V-238224, V-238226, V-238234, V-238235, V-238237, V-238308, V-238323, V-238357, V-238362, and V-238373

Ubuntu 22.04 STIG Version 2 Release 4

V-260472, V-260476, V-260479, V-260480, V-260481, V-260520, V-260521, V-260549, V-260550, V-260551, V-260552, V-260581, and V-260596

Ubuntu 24.04 STIG Version 1 Release 1

V-270645, V-270646, V-270664, V-270677, V-270690, V-270695, V-270706, V-270710, V-270749, V-270752, V-270818, and V-270820

STIG-Build-Linux-Medium version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list, see the STIGs Document Library. For information about how to view the complete list, see STIG Viewing Tools.

Note

The STIG-Build-Linux-Medium hardening components include all listed STIG settings that AWSTOE applies for STIG-Build-Linux-Low hardening components, in addition to the STIG settings that are listed specifically for Category II vulnerabilities.

RHEL 7 STIG Version 3 Release 15

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 7/CentOS 7/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-204431, V-204434, V-204435, V-204437, V-204449, V-204450, V-204451, V-204457, V-204466, V-204490, V-204491, V-204503, V-204507, V-204508, V-204510, V-204511, V-204512, V-204514, V-204515, V-204516, V-204517, V-204521, V-204524, V-204531, 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-204562, V-204563, V-204564, V-204565, V-204566, V-204567, V-204568, V-204572, V-204579, V-204584, V-204585, V-204587, V-204588, V-204589, V-204590, V-204591, V-204592, V-204593, V-204596, V-204597, 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-204625, V-204630, V-204631, V-204633, V-233307, V-237634, V-237635, V-251703, V-255925, V-255927, V-255928, and V-256970

RHEL 8 STIG Version 2 Release 3

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 8/CentOS 8/AL 2023**

V-230228, V-230231, V-230233, V-230236, V-230237, V-230238, V-230239, V-230240, V-230244, V-230245, V-230246, V-230247, V-230248, V-230249, V-230250, V-230255, V-230257, V-230258, V-230259, V-230261, V-230262, V-230266, V-230267, V-230268, V-230273, V-230275, V-230277, V-230278, V-230279, V-230280, V-230282, V-230286, V-230287, V-230288, V-230290, V-230291, V-230296, V-230298, V-230310, V-230311, V-230312, V-230313, V-230314, V-230315, V-230316, V-230324, V-230325, V-230326, V-230327, V-230330, V-230332, V-230333, V-230335, V-230337, V-230339, V-230341, V-230343, V-230345, V-230348, V-230353, V-230354, V-230356, V-230357, V-230358, V-230359, V-230360, V-230361, V-230362, V-230363, V-230365, V-230366, V-230368, V-230369, V-230370, V-230373, V-230375, V-230376, V-230377, V-230378, V-230382, V-230383, V-230386, V-230387, V-230390, V-230392, V-230393, V-230394, V-230396, V-230397, V-230398, V-230399, V-230400, V-230401, 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-230418, V-230419, 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-230444, V-230446, V-230447, V-230448, V-230449, V-230455, V-230456, V-230462, V-230463, V-230464, V-230465, V-230466, V-230467, V-230471, V-230472, V-230473, V-230474, V-230478, V-230480, V-230482, V-230483, V-230488, V-230489, V-230502, V-230503, V-230505, V-230507, V-230523, V-230525, V-230526, V-230527, 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-230550, V-230555, V-230556, V-230557, V-230559, V-230560, V-230561, V-237640, V-237642, V-237643, V-244523, V-244524, V-244525, V-244526, V-244528, V-244533, V-244538, V-244542, V-244543, V-244544, V-244547, V-244550, V-244551, V-244552, V-244553, V-244554, V-250315, V-250316, V-250317, V-251710, V-251711, V-251713, V-251714, V-251715, V-251716, V-251717, V-251718, V-256974, and V-257258

RHEL 9 STIG Version 2 Release 4

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 9/CentOS Stream 9**

  V-257780, V-257781, V-257783, V-257786, V-257788, V-257790, V-257791, V-257792, V-257793, V-257794, V-257797, V-257798, V-257799, V-257800, V-257801, V-257802, V-257803, V-257804, V-257805, V-257806, V-257807, V-257808, V-257809, V-257810, V-257811, V-257812, V-257813, V-257814, V-257815, V-257816, V-257817, V-257818, V-257825, V-257827, V-257828, V-257829, V-257830, V-257831, V-257832, V-257833, V-257834, V-257836, V-257838, V-257839, V-257840, V-257841, V-257842, V-257849, V-257882, V-257883, V-257884, V-257885, V-257886, V-257887, V-257888, V-257889, V-257890, V-257891, V-257892, V-257893, V-257894, V-257895, V-257896, V-257897, V-257898, V-257899, V-257900, V-257901, V-257902, V-257903, V-257904, V-257905, V-257906, V-257907, V-257908, V-257909, V-257910, V-257911, V-257912, V-257913, V-257914, V-257915, V-257916, V-257917, V-257918, V-257919, V-257920, V-257921, V-257922, V-257923, V-257924, V-257925, V-257926, V-257927, V-257928, V-257929, V-257930, V-257933, V-257934, V-257935, V-257936, V-257939, V-257940, V-257942, V-257943, V-257944, V-257948, V-257949, V-257951, V-257952, V-257953, V-257954, V-257957, V-257958, V-257959, V-257960, V-257961, V-257962, V-257963, V-257964, V-257965, V-257966, V-257967, V-257968, V-257969, V-257970, V-257971, V-257972, V-257973, V-257974, V-257975, V-257976, V-257977, V-257978, V-257979, V-257980, V-257982, V-257983, V-257985, V-257987, V-257988, V-257992, V-257993, V-257994, V-257995, V-257996, V-257997, V-257998, V-257999, V-258000, V-258001, V-258002, V-258003, V-258004, V-258005, V-258006, V-258007, V-258008, V-258009, V-258010, V-258011, V-258028, V-258034, V-258035, V-258038, V-258039, V-258040, V-258041, V-258043, V-258046, V-258049, V-258052, V-258054, V-258055, V-258056, V-258057, V-258060, V-258063, V-258064, V-258065, V-258066, V-258068, V-258070, V-258071, V-258072, V-258073, V-258074, V-258075, V-258077, V-258079, V-258080, V-258081, V-258082, V-258083, V-258084, V-258085, V-258088, V-258089, V-258091, V-258092, V-258093, V-258095, V-258097, V-258098, V-258099, V-258100, V-258101, V-258102, V-258103, V-258104, V-258105, V-258107, V-258108, V-258109, V-258110, V-258111, V-258112, V-258113, V-258114, V-258115, V-258116, V-258117, V-258118, V-258119, V-258120, V-258122, V-258123, V-258124, V-258125, V-258126, V-258128, V-258129, V-258130, V-258133, V-258137, V-258140, V-258141, V-258142, V-258144, V-258145, V-258146, V-258147, V-258148, V-258150, V-258151, V-258152, V-258153, V-258154, V-258156, V-258157, V-258158, V-258159, V-258160, V-258161, V-258162, V-258163, V-258164, V-258165, V-258166, V-258167, V-258168, V-258169, V-258170, V-258171, V-258172, V-258175, V-258176, V-258177, V-258178, V-258179, V-258180, V-258181, V-258182, V-258183, V-258184, V-258185, V-258186, V-258187, V-258188, V-258189, V-258190, V-258191, V-258192, V-258193, V-258194, V-258195, V-258196, V-258197, V-258198, V-258199, V-258200, V-258201, V-258202, V-258203, V-258204, V-258205, V-258206, V-258207, V-258208, V-258209, V-258210, V-258211, V-258212, V-258213, V-258214, V-258215, V-258216, V-258217, V-258218, V-258219, V-258220, V-258221, V-258222, V-258223, V-258224, V-258225, V-258226, V-258227, V-258228, V-258229, V-258232, V-258233, V-258234, V-258237, V-258239, V-258240, and V-272488

Ubuntu 18.04 STIG Version 2 Release 15

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

V-219149, V-219155, V-219156, V-219160, V-219166, V-219168, V-219176, V-219181, V-219184, V-219186, V-219188, V-219189, V-219190, V-219191, V-219192, V-219193, V-219194, V-219195, V-219196, V-219197, V-219198, V-219199, V-219200, V-219201, V-219202, V-219203, V-219204, V-219205, V-219206, V-219207, V-219208, V-219209, V-219213, V-219214, V-219215, V-219216, V-219217, V-219218, V-219219, V-219220, V-219221, V-219222, V-219223, V-219224, V-219225, V-219226, V-219227, V-219228, V-219229, V-219230, V-219231, V-219232, V-219233, V-219234, V-219235, V-219236, V-219238, V-219239, V-219240, V-219241, V-219242, V-219243, V-219244, V-219250, V-219254, V-219257, V-219263, V-219264, V-219265, V-219266, V-219267, V-219268, V-219269, V-219270, V-219271, V-219272, V-219273, V-219274, V-219275, V-219276, V-219277, V-219279, V-219281, V-219287, V-219291, V-219297, V-219298, V-219299, V-219300, V-219303, V-219304, V-219306, V-219309, V-219310, V-219311, V-219315, V-219318, V-219319, V-219323, V-219326, V-219328, V-219330, V-219331, V-219335, V-219336, V-219337, V-219338, V-219339, V-219342, V-219344, V-233779, V-233780, and V-255906

Ubuntu 20.04 STIG Version 2 Release 2

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

V-238200, V-238205, V-238207, V-238209, V-238210, V-238211, V-238212, V-238213, V-238220, V-238225, V-238227, V-238228, V-238229, V-238230, V-238231, V-238232, V-238236, V-238238, V-238239, V-238240, V-238241, V-238242, V-238244, V-238245, V-238246, V-238247, V-238248, V-238249, V-238250, V-238251, V-238252, V-238253, V-238254, V-238255, V-238256, V-238257, V-238258, V-238264, V-238268, V-238271, V-238277, V-238278, V-238279, V-238280, V-238281, V-238282, V-238283, V-238284, V-238285, V-238286, V-238287, V-238288, V-238289, V-238290, V-238291, V-238292, V-238293, V-238294, V-238295, V-238297, V-238298, V-238299, V-238300, V-238301, V-238302, V-238303, V-238304, V-238309, V-238310, V-238315, V-238316, V-238317, V-238318, V-238319, V-238320, V-238324, V-238325, V-238329, V-238330, V-238333, V-238334, V-238337, V-238338, V-238339, V-238340, V-238341, V-238342, V-238343, V-238344, V-238345, V-238346, V-238347, V-238348, V-238349, V-238350, V-238351, V-238352, V-238353, V-238355, V-238356, V-238359, V-238360, V-238369, V-238370, V-238371, V-238376, V-238377, V-238378, V-251505, and V-255912

Ubuntu 22.04 STIG Version 2 Release 4

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

V-260471, V-260473, V-260474, V-260475, V-260477, V-260478, V-260485, V-260486, V-260487, V-260488, V-260489, V-260490, V-260491, V-260492, V-260493, V-260494, V-260495, V-260496, V-260497, V-260498, V-260499, V-260500, V-260505, V-260506, V-260507, V-260508, V-260509, V-260510, V-260511, V-260512, V-260513, V-260514, V-260522, V-260527, V-260528, V-260530, V-260533, V-260534, V-260535, V-260537, V-260538, V-260540, V-260542, V-260543, V-260545, V-260546, V-260547, V-260553, V-260554, V-260555, V-260556, V-260557, V-260560, V-260561, V-260562, V-260563, V-260564, V-260565, V-260566, V-260567, V-260569, V-260572, V-260573, V-260574, V-260575, V-260576, V-260582, V-260584, V-260585, V-260586, V-260588, V-260589, V-260590, V-260591, V-260594, V-260597, V-260598, V-260599, V-260600, V-260601, V-260602, V-260603, V-260604, V-260605, V-260606, V-260607, V-260608, V-260609, V-260610, V-260611, V-260612, V-260613, V-260614, V-260615, V-260616, V-260617, V-260618, V-260619, V-260620, V-260621, V-260622, V-260623, V-260624, V-260625, V-260626, V-260627, V-260628, V-260629, V-260630, V-260631, V-260632, V-260633, V-260634, V-260635, V-260636, V-260637, V-260638, V-260639, V-260640, V-260641, V-260642, V-260643, V-260644, V-260645, V-260646, V-260647, V-260648, and V-260649

Ubuntu 24.04 STIG Version 1 Release 1

Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:

V-270649, V-270651, V-270652, V-270653, V-270654, V-270656, V-270657, V-270659, V-270660, V-270661, V-270662, V-270663, V-270669, V-270672, V-270673, V-270674, V-270676, V-270678, V-270679, V-270680, V-270681, V-270683, V-270684, V-270685, V-270686, V-270687, V-270688, V-270689, V-270692, V-270693, V-270696, V-270697, V-270698, V-270699, V-270700, V-270701, V-270702, V-270703, V-270704, V-270705, V-270709, V-270715, V-270716, V-270718, V-270720, V-270721, V-270722, V-270723, V-270724, V-270725, V-270726, V-270727, V-270728, V-270729, V-270730, V-270731, V-270732, V-270733, V-270737, V-270739, V-270740, V-270741, V-270742, V-270743, V-270746, V-270750, V-270753, V-270755, V-270756, V-270757, V-270758, V-270759, V-270760, V-270765, V-270766, V-270767, V-270768, V-270769, V-270770, V-270771, V-270772, V-270773, V-270775, V-270776, V-270777, V-270778, V-270779, V-270780, V-270781, V-270782, V-270783, V-270784, V-270785, V-270786, V-270787, V-270788, V-270789, V-270790, V-270791, V-270792, V-270793, V-270794, V-270795, V-270796, V-270797, V-270798, V-270799, V-270800, V-270801, V-270802, V-270803, V-270804, V-270805, V-270806, V-270807, V-270808, V-270809, V-270810, V-270811, V-270812, V-270813, V-270814, V-270815, V-270821, V-270822, V-270823, V-270824, V-270825, V-270826, V-270827, V-270828, V-270829, V-270830, V-270831, and V-270832

STIG-Build-Linux-High version 2025.2.x

The following list contains STIG settings that the hardening component applies to your infrastructure. If a supported setting isn't applicable for your infrastructure, the hardening component skips that setting, and moves on. For example, some STIG settings might not apply to standalone servers. Organization-specific policies can also affect which settings the hardening component applies, such as a requirement for administrators to review document settings.

For a complete list, see the STIGs Document Library. For information about how to view the complete list, see STIG Viewing Tools.

> **Note**
>
> The STIG-Build-Linux-High hardening components include all listed STIG settings that AWSTOE applies for STIG-Build-Linux-Low and STIG-Build-Linux-Medium hardening components, in addition to the listed STIG settings that apply specifically for Category I vulnerabilities.

RHEL 7 STIG Version 3 Release 15

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 7/CentOS 7/AL2**

V-204424, V-204425, V-204442, V-204443, V-204447, V-204448, V-204455, V-204497, V-204502, V-204594, V-204620, and V-204621

RHEL 8 STIG Version 2 Release 3

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 8/CentOS 8/AL 2023**

  V-230223, V-230226, V-230264, V-230265, V-230487, V-230492, V-230529, V-230530, V-230531, V-230533, V-230558, and V-244540

RHEL 9 STIG Version 2 Release 4

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

- **RHEL 9/CentOS Stream 9**

  V-257784, V-257785, V-257820, V-257821, V-257826, V-257835, V-257955, V-257956, V-257984, V-257986, V-258059, V-258078, V-258094, V-258230, V-258235, and V-258238

Ubuntu 18.04 STIG Version 2 Release 15

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

V-219157, V-219158, V-219177, V-219212, V-219308, V-219314, V-219316, V-251507, and V-264388

Ubuntu 20.04 STIG Version 2 Release 2

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

V-238201, V-238218, V-238219, V-238326, V-238327, V-238380, and V-251504

Ubuntu 22.04 STIG Version 2 Release 4

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

V-260469, V-260482, V-260483, V-260523, V-260524, V-260526, V-260529, V-260539, V-260570, V-260571, and V-260579

Ubuntu 24.04 STIG Version 1 Release 1

Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities for this Linux distribution, plus:

V-270647, V-270648, V-270665, V-270666, V-270708, V-270711, V-270712, V-270713, V-270714, and V-270717

STIG version history log for Linux

This section logs Linux component version history. To see the changes and published versions for a quarter, choose the title to expand the information.

2025 Q2 changes - 06/26/2025:

Updated the following STIG versions, applied STIGS for the 2025 second quarter release:

STIG-Build-Linux-Low version 2025.2.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 3
- RHEL 9 STIG Version 2 Release 4
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 4
- Ubuntu 24.04 STIG Version 1 Release 1

STIG-Build-Linux-Medium version 2025.2.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 3
- RHEL 9 STIG Version 2 Release 4
- Ubuntu 18.04 STIG Version 2 Release 15

- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 4
- Ubuntu 24.04 STIG Version 1 Release 1

STIG-Build-Linux-High version 2025.2.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 3
- RHEL 9 STIG Version 2 Release 4
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 4
- Ubuntu 24.04 STIG Version 1 Release 1

2025 Q1 changes - 04/11/2025:

Updated the following STIG versions, applied STIGS for the 2025 first quarter release, and added support for Ubuntu 24.04:

STIG-Build-Linux-Low version 2025.1.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 2
- RHEL 9 STIG Version 2 Release 3
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 3
- Ubuntu 24.04 STIG Version 1 Release 1

STIG-Build-Linux-Medium version 2025.1.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 2
- RHEL 9 STIG Version 2 Release 3

- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 3
- Ubuntu 24.04 STIG Version 1 Release 1

STIG-Build-Linux-High version 2025.1.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 2
- RHEL 9 STIG Version 2 Release 3
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 2
- Ubuntu 22.04 STIG Version 2 Release 3
- Ubuntu 24.04 STIG Version 1 Release 1

2024 Q4 changes - 12/10/2024:

Updated the following STIG versions, applied STIGS for the 2024 fourth quarter release, and added information about two new input parameters for the Linux components:

STIG-Build-Linux-Low version 2024.4.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 1
- RHEL 9 STIG Version 2 Release 2
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 1
- Ubuntu 22.04 STIG Version 2 Release 2

STIG-Build-Linux-Medium version 2024.4.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 1
- RHEL 9 STIG Version 2 Release 2

- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 1
- Ubuntu 22.04 STIG Version 2 Release 2

STIG-Build-Linux-High version 2024.4.x

- RHEL 7 STIG Version 3 Release 15
- RHEL 8 STIG Version 2 Release 1
- RHEL 9 STIG Version 2 Release 2
- Ubuntu 18.04 STIG Version 2 Release 15
- Ubuntu 20.04 STIG Version 2 Release 1
- Ubuntu 22.04 STIG Version 2 Release 2

2024 Q3 changes - 10/04/2024 (no changes):

There were no changes for Linux component STIGS for the 2024 third quarter release.

2024 Q2 changes - 05/10/2024:

Updated STIG versions and applied STIGS for the 2024 second quarter release. Also added support for RHEL 9, CentOS Stream 9, and Ubuntu 22.04, as follows:

STIG-Build-Linux-Low version 2024.2.x

- RHEL 7 STIG Version 3 Release 14
- RHEL 8 STIG Version 1 Release 14
- RHEL 9 STIG Version 1 Release 3
- Ubuntu 18.04 STIG Version 2 Release 14
- Ubuntu 20.04 STIG Version 1 Release 12
- Ubuntu 22.04 STIG Version 1 Release 1

STIG-Build-Linux-Medium version 2024.2.x

- RHEL 7 STIG Version 3 Release 14
- RHEL 8 STIG Version 1 Release 14

- RHEL 9 STIG Version 1 Release 3
- Ubuntu 18.04 STIG Version 2 Release 14
- Ubuntu 20.04 STIG Version 1 Release 12
- Ubuntu 22.04 STIG Version 1 Release 1

STIG-Build-Linux-High version 2024.2.x

- RHEL 7 STIG Version 3 Release 14
- RHEL 8 STIG Version 1 Release 14
- RHEL 9 STIG Version 1 Release 3
- Ubuntu 18.04 STIG Version 2 Release 14
- Ubuntu 20.04 STIG Version 1 Release 12
- Ubuntu 22.04 STIG Version 1 Release 1

2024 Q1 changes - 02/06/2024:

Updated STIG versions and applied STIGS for the 2024 first quarter release as follows:

STIG-Build-Linux-Low version 2024.1.x

- RHEL 7 STIG Version 3 Release 14
- RHEL 8 STIG Version 1 Release 13
- Ubuntu 18.04 STIG Version 2 Release 13
- Ubuntu 20.04 STIG Version 1 Release 11

STIG-Build-Linux-Medium version 2024.1.x

- RHEL 7 STIG Version 3 Release 14
- RHEL 8 STIG Version 1 Release 13
- Ubuntu 18.04 STIG Version 2 Release 13
- Ubuntu 20.04 STIG Version 1 Release 11

STIG-Build-Linux-High version 2024.1.x

- RHEL 7 STIG Version 3 Release 14

- RHEL 8 STIG Version 1 Release 13
- Ubuntu 18.04 STIG Version 2 Release 13
- Ubuntu 20.04 STIG Version 1 Release 11

2023 Q4 changes - 12/07/2023:

Updated STIG versions and applied STIGS for the 2023 fourth quarter release as follows:

STIG-Build-Linux-Low version 2023.4.x

- RHEL 7 STIG Version 3 Release 13
- RHEL 8 STIG Version 1 Release 12
- Ubuntu 18.04 STIG Version 2 Release 12
- Ubuntu 20.04 STIG Version 1 Release 10

STIG-Build-Linux-Medium version 2023.4.x

- RHEL 7 STIG Version 3 Release 13
- RHEL 8 STIG Version 1 Release 12
- Ubuntu 18.04 STIG Version 2 Release 12
- Ubuntu 20.04 STIG Version 1 Release 10

STIG-Build-Linux-High version 2023.4.x

- RHEL 7 STIG Version 3 Release 13
- RHEL 8 STIG Version 1 Release 12
- Ubuntu 18.04 STIG Version 2 Release 12
- Ubuntu 20.04 STIG Version 1 Release 10

2023 Q3 changes - 10/04/2023:

Updated STIG versions and applied STIGS for the 2023 third quarter release as follows:

STIG-Build-Linux-Low version 2023.3.x

- RHEL 7 STIG Version 3 Release 12

- RHEL 8 STIG Version 1 Release 11
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 9

STIG-Build-Linux-Medium version 2023.3.x

- RHEL 7 STIG Version 3 Release 12
- RHEL 8 STIG Version 1 Release 11
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 9

STIG-Build-Linux-High version 2023.3.x

- RHEL 7 STIG Version 3 Release 12
- RHEL 8 STIG Version 1 Release 11
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 9

2023 Q2 changes - 05/03/2023:

Updated STIG versions and applied STIGS for the 2023 second quarter release as follows:

STIG-Build-Linux-Low version 2023.2.x

- RHEL 7 STIG Version 3 Release 11
- RHEL 8 STIG Version 1 Release 10
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 8

STIG-Build-Linux-Medium version 2023.2.x

- RHEL 7 STIG Version 3 Release 11
- RHEL 8 STIG Version 1 Release 10
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 8

STIG-Build-Linux-High version 2023.2.x

- RHEL 7 STIG Version 3 Release 11
- RHEL 8 STIG Version 1 Release 10
- Ubuntu 18.04 STIG Version 2 Release 11
- Ubuntu 20.04 STIG Version 1 Release 8

2023 Q1 changes - 03/27/2023:

Updated STIG versions and applied STIGS for the 2023 first quarter release as follows:

STIG-Build-Linux-Low version 2023.1.x

- RHEL 7 STIG Version 3 Release 10
- RHEL 8 STIG Version 1 Release 9
- Ubuntu 18.04 STIG Version 2 Release 10
- Ubuntu 20.04 STIG Version 1 Release 7

STIG-Build-Linux-Medium version 2023.1.x

- RHEL 7 STIG Version 3 Release 10
- RHEL 8 STIG Version 1 Release 9
- Ubuntu 18.04 STIG Version 2 Release 10
- Ubuntu 20.04 STIG Version 1 Release 7

STIG-Build-Linux-High version 2023.1.x

- RHEL 7 STIG Version 3 Release 10
- RHEL 8 STIG Version 1 Release 9
- Ubuntu 18.04 STIG Version 2 Release 10
- Ubuntu 20.04 STIG Version 1 Release 7

2022 Q4 changes - 02/01/2023:

Updated STIG versions and applied STIGS for the 2022 fourth quarter release as follows:

STIG-Build-Linux-Low version 2022.4.x

- RHEL 7 STIG Version 3 Release 9
- RHEL 8 STIG Version 1 Release 8
- Ubuntu 18.04 STIG Version 2 Release 9
- Ubuntu 20.04 STIG Version 1 Release 6

STIG-Build-Linux-Medium version 2022.4.x

- RHEL 7 STIG Version 3 Release 9
- RHEL 8 STIG Version 1 Release 8
- Ubuntu 18.04 STIG Version 2 Release 9
- Ubuntu 20.04 STIG Version 1 Release 6

STIG-Build-Linux-High version 2022.4.x

- RHEL 7 STIG Version 3 Release 9
- RHEL 8 STIG Version 1 Release 8
- Ubuntu 18.04 STIG Version 2 Release 9
- Ubuntu 20.04 STIG Version 1 Release 6

2022 Q3 changes - 09/30/2022 (no changes):

There were no changes for Linux component STIGS for the 2022 third quarter release.

2022 Q2 changes - 08/02/2022:

Introduced Ubuntu support, updated STIG versions and applied STIGS for the 2022 second quarter release as follows:

STIG-Build-Linux-Low version 2022.2.x

- RHEL 7 STIG Version 3 Release 7
- RHEL 8 STIG Version 1 Release 6
- Ubuntu 18.04 STIG Version 2 Release 6 (new)

- Ubuntu 20.04 STIG Version 1 Release 4 (new)

STIG-Build-Linux-Medium version 2022.2.x

- RHEL 7 STIG Version 3 Release 7
- RHEL 8 STIG Version 1 Release 6
- Ubuntu 18.04 STIG Version 2 Release 6 (new)
- Ubuntu 20.04 STIG Version 1 Release 4 (new)

STIG-Build-Linux-High version 2022.2.x

- RHEL 7 STIG Version 3 Release 7
- RHEL 8 STIG Version 1 Release 6
- Ubuntu 18.04 STIG Version 2 Release 6 (new)
- Ubuntu 20.04 STIG Version 1 Release 4 (new)

2022 Q1 changes - 04/26/2022:

Refactored to include better support for containers. Combined the previous AL2 script with RHEL 7. Updated STIG versions and applied STIGS for the 2022 first quarter release as follows:

STIG-Build-Linux-Low version 3.6.x

- RHEL 7 STIG Version 3 Release 6
- RHEL 8 STIG Version 1 Release 5

STIG-Build-Linux-Medium version 3.6.x

- RHEL 7 STIG Version 3 Release 6
- RHEL 8 STIG Version 1 Release 5

STIG-Build-Linux-High version 3.6.x

- RHEL 7 STIG Version 3 Release 6
- RHEL 8 STIG Version 1 Release 5

2021 Q4 changes - 12/20/2021:

Updated STIG versions, and applied STIGS for the 2021 fourth quarter release as follows:

STIG-Build-Linux-Low version 3.5.x

- RHEL 7 STIG Version 3 Release 5
- RHEL 8 STIG Version 1 Release 4

STIG-Build-Linux-Medium version 3.5.x

- RHEL 7 STIG Version 3 Release 5
- RHEL 8 STIG Version 1 Release 4

STIG-Build-Linux-High version 3.5.x

- RHEL 7 STIG Version 3 Release 5
- RHEL 8 STIG Version 1 Release 4

2021 Q3 changes - 09/30/2021:

Updated STIG versions, and applied STIGS for the 2021 third quarter release as follows:

STIG-Build-Linux-Low version 3.4.x

- RHEL 7 STIG Version 3 Release 4
- RHEL 8 STIG Version 1 Release 3

STIG-Build-Linux-Medium version 3.4.x

- RHEL 7 STIG Version 3 Release 4
- RHEL 8 STIG Version 1 Release 3

STIG-Build-Linux-High version 3.4.x

- RHEL 7 STIG Version 3 Release 4
- RHEL 8 STIG Version 1 Release 3

SCAP compliance validator component

The Security Content Automation Protocol (SCAP) is a set of standards that IT professionals can use to identify application security vulnerabilities for compliance. The SCAP Compliance Checker (SCC) is a SCAP-validated scanning tool, released by the Naval Information Warfare Center (NIWC) Atlantic. For more information, see Security Content Automation Protocol (SCAP) Compliance Checker (SCC) on the *NIWC Atlantic* website.

The AWSTOE `scap-compliance-checker-windows` and `scap-compliance-checker-linux` components download and install the SCC scanner on the pipeline build and test instances. When the scanner runs, it performs authenticated configuration scans using DISA SCAP Benchmarks, and provides a report that includes the following information. AWSTOE also writes the information to your application logs.

- STIG settings that are applied to the instance.
- An overall compliance score for the instance.

We recommend that you run SCAP validation as the final step in your build process, to ensure that you report accurate compliance validation results.

> **Note**
>
> You can review the reports with one of the STIG Viewing Tools. These tools are available online via the DoD Cyber Exchange.

The following sections describe the benchmarks that the SCAP validation components include.

scap-compliance-checker-windows version 2024.03.0

The `scap-compliance-checker-windows` component runs on the EC2 instances that Image Builder creates to build and test the image. AWSTOE logs both the report and the score that the SCC application produces.

The component performs the following workflow steps:

1. Downloads and installs the SCC application.
2. Imports the compliance benchmarks.
3. Runs validation using the SCC application.

4. Saves the compliance report and score locally on the build instance desktop.
5. Logs the compliance score from the local report to the AWSTOE application log files.

> **Note**
>
> AWSTOE currently supports SCAP compliance validation for Windows Server 2012 R2 MS, 2016, 2019, and 2022.

The SCAP compliance checker component for Windows includes the following benchmarks:

SCC Version: 5.10

2023 Q4 Benchmarks:

- U_MS_Defender_Antivirus_V2R5_STIG_SCAP_1-2_Benchmark
- U_MS_DotNet_Framework_4-0_V2R2_STIG_SCAP_1-2_Benchmark
- U_MS_IE11_V2R6_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_2012_and_2012_R2_DC_V3R5_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_2012_and_2012_R2_MS_V3R5_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Defender_Firewall_V2R3_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2016_V2R7_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2019_V3R2_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2022_V2R2_STIG_SCAP_1-2_Benchmark
- U_CAN_Ubuntu_20-04_LTS_V1R10_STIG_SCAP_1-2_Benchmark
- U_RHEL_7_V3R15_STIG_SCAP_1-3_Benchmark
- U_RHEL_8_V1R13_STIG_SCAP_1-3_Benchmark
- U_RHEL_9_V2R1_STIG_SCAP_1-3_Benchmark

scap-compliance-checker-linux version 2021.04.0

The `scap-compliance-checker-linux` component runs on the EC2 instances that Image Builder creates to build and test the image. AWSTOE logs both the report and the score that the SCC application produces.

The component performs the following workflow steps:

1. Downloads and installs the SCC application.
2. Imports the compliance benchmarks.
3. Runs validation using the SCC application.
4. Saves the compliance report and score locally, in the following location on the build instance: `/opt/scc/SCCResults`.
5. Logs the compliance score from the local report to the AWSTOE application log files.

> **Note**
>
> AWSTOE currently supports SCAP compliance validation for RHEL 7/8 and Ubuntu 18.04/20.04. The SCC application currently supports the x86 architecture for validation.

The SCAP compliance checker component for Linux includes the following benchmarks:

SCC Version: 5.10

2023 Q4 Benchmarks:

- U_CAN_Ubuntu_20-04_LTS_V1R10_STIG_SCAP_1-2_Benchmark
- U_RHEL_7_V3R15_STIG_SCAP_1-3_Benchmark
- U_RHEL_8_V1R13_STIG_SCAP_1-3_Benchmark
- U_RHEL_9_V2R1_STIG_SCAP_1-3_Benchmark
- U_MS_Defender_Antivirus_V2R5_STIG_SCAP_1-2_Benchmark
- U_MS_DotNet_Framework_4-0_V2R2_STIG_SCAP_1-2_Benchmark
- U_MS_IE11_V2R6_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_2012_and_2012_R2_DC_V3R5_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_2012_and_2012_R2_MS_V3R5_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Defender_Firewall_V2R3_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2016_V2R7_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2019_V3R2_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2022_V2R2_STIG_SCAP_1-2_Benchmark

SCAP version history

The following table describes important changes to the SCAP environment and settings described in this document.

Change	Description	Date
2025 Q1 SCAP Updates	• scap-compliance-checker-windows version 2024.03.0 (SCC Version: 5.10) • scap-compliance-checker-windows version 2025.01.0 (SCC Version: 5.10	April 11, 2025
2023 Q4 SCAP Updates	• scap-compliance-checker-windows version 2023.04.0 (SCC Version: 5.8) • scap-compliance-checker-windows version 2023.04.0 (SCC Version: 5.8)	December 20, 2021
2023 Q3 SCAP Updates	• scap-compliance-checker-windows version 2023.03.0 (SCC Version: 5.7.2) • scap-compliance-checker-linux version 2023.03.0 (SCC Version: 5.7.2	November 13, 2023
Added SCAP components	**Introduced the following SCAP components:** • Created scap-compliance-checker-linux version 2021.04.0 (SCC Version: 5.4.2) • Created scap-compliance-checker-linux version 2021.04.0 (SCC Version: 5.4.2)	December 20, 2021

Develop custom components for your Image Builder image

You can create your own components to customize your Image Builder images according to your exact specifications. Use the following steps to develop custom components for your Image Builder image or container recipes.

1. If you want to develop your component document and validate it locally, you can install the AWS Task Orchestrator and Executor (AWSTOE) application and set it up on your local machine. For more information, see Manual set up to develop custom components with AWSTOE.
2. Create a component document that uses the AWSTOE component document framework. For more information about the document framework, see Use the AWSTOE component document framework for custom components.
3. Specify your component document when you create a custom component. For more information, see Create a custom component with Image Builder.

Topics

Create a YAML component document for custom components in Image Builder

To build a component, you must provide a YAML or JSON application component document. The document contains the code that runs during the phases and steps that you define to provide customization for your image.

Some of the examples in this section create a build component that calls the `UpdateOS` action module in the AWSTOE component management application. The module updates the operating system. For more information about the `UpdateOS` action module, see UpdateOS.

The macOS operating system example uses the `ExecuteBash` action module to install and verify the `wget` utility. The `UpdateOS` action module doesn't support macOS. For more information about the `ExecuteBash` action module, see ExecuteBash. For more information about the phases, steps, and syntax for AWSTOE application component documents, see Use documents in AWSTOE.

> **Note**
>
> Image Builder determines the component type from the phases that are defined in the component document 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 image *Build stage*. If this build component has a `test` phase defined, that phase runs during the *Test stage*.
> - **Test** – To qualify 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 don't use a standalone test component. Rather, use the `test` phase in the associated build component.
>
> For more information about how Image Builder uses stages and phases to manage component workflow in its build process, see [Use components to customize your Image Builder image](#).

To create a YAML application component document for a sample application, follow the steps on the tab that matches your image operating system.

Linux

Create a YAML component file

Use a file editing tool to create your component document. Documentation examples use a file named *update-linux-os.yaml*, with 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 a file editing tool to create your component document. Documentation examples use a file named *update-windows-os.yaml*, with 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.

macOS

Create a YAML component file

Use a file editing tool to create your component document. Documentation examples use a file named *wget-macos.yaml*, with the following content:

```
name: WgetInstallDocument
description: This is wget installation document.
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: WgetBuildStep
        action: ExecuteBash
        inputs:
          commands:
            - |
```

```
            PATH=/usr/local/bin:$PATH
            sudo -u ec2-user brew install wget


  - name: validate
    steps:
      - name: WgetValidateStep
        action: ExecuteBash
        inputs:
          commands:
            - |
              function error_exit {
                echo $1
                echo "{\"failureMessage\":\"$2\"}"
                exit 1
              }

              type wget
              if [ $? -ne 0 ]; then
                error_exit "$stderr" "Wget installation failed!"
              fi

  - name: test
    steps:
      - name: WgetTestStep
        action: ExecuteBash
        inputs:
          commands:
            - wget -h
```

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.

Create a custom component with Image Builder

After you've completed your component document, you can use it to create a custom component that your Image Builder recipes can use. You can create a custom component from the Image Builder console, from the API or SDKs, or from the command line. For more information about how to create a custom component with input parameters and use it in your recipes, see Tutorial: Create a custom component with input parameters.

The following sections show you how to create components from the console or from the AWS CLI.

Contents

Create a custom component from the console

To create an AWSTOE application component from 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. In the **Definition document** section, the default option is **Define document content**. The component document defines the actions that Image Builder performs on the build and test instances to create your image.

   In the **Content** box, enter your YAML component document content. To start with a *Hello World* example for Linux, choose the **Use example** option. To learn more about how to create a YAML component document, or to copy and paste the *UpdateOS* example from that page, see Create a YAML component document for custom components in Image Builder.
5. After you enter 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**.

Update a 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 in Image Builder.

Create a custom component from the AWS CLI

In this section, you'll learn how to set up and use Image Builder commands in the AWS CLI to create an AWSTOE application component, as follows.

- Upload your YAML component document to an S3 bucket that you can reference from the command line.
- Create the AWSTOE application component with the **create-component** command.
- List component versions with the **list-components** command and a name filter to see what versions already exist. You can use the output to determine what the next version should be for updates.

To create an AWSTOE application component from an input YAML document, follow the steps that match your image operating system platform.

Linux

Store your application component document in Amazon S3

You can use an S3 bucket as a repository for your AWSTOE application component source document. To store your component document, follow these steps:

- **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://amzn-s3-demo-destination-bucket/my-path/update-linux-os.yaml
  ```

Create a component from the YAML document

To streamline the **create-component** command that you use in the AWS CLI, create a JSON file that contains all of the component parameters that you want to pass into the command. Include the location of the *update-linux-os.yaml* document that you created earlier. The `uri` key-value pair contains the file reference.

> **Note**
>
> The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API command request parameters, see the CreateComponent command in the *EC2 Image Builder API Reference*.
> To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*.

1. **Create a CLI input JSON file**

   Use a file editing tool to create a file named *create-update-linux-os-component.json*. Include 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://amzn-s3-demo-destination-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"
 }
}
```

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, while Linux and macOS use the forward slash (/).

Windows

Store your application component document in Amazon S3

You can use an S3 bucket as a repository for your AWSTOE application component source document. To store your component document, follow these steps:

- **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://amzn-s3-demo-destination-bucket/my-path/update-windows-os.yaml
  ```

Create a component from the YAML document

To streamline the **create-component** command that you use in the AWS CLI, create a JSON file that contains all of the component parameters that you want to pass into the command. Include the location of the `update-windows-os.yaml` document that you created earlier. The `uri` key-value pair contains the file reference.

> **Note**
>
> The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API command request parameters, see the CreateComponent command in the *EC2 Image Builder API Reference*.
> To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*..

1. **Create a CLI input JSON file**

   Use a file editing tool to create a file named `create-update-windows-os-component.json`. Include 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://amzn-s3-demo-destination-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, while Linux and macOS use 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, while Linux and macOS use the forward slash (/).

macOS

Store your application component document in Amazon S3

You can use an S3 bucket as a repository for your AWSTOE application component source document. To store your component document, follow these steps:

- **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 wget-macos.yaml s3://amzn-s3-demo-destination-bucket/my-path/wget-
  macos.yaml
  ```

Create a component from the YAML document

To streamline the **create-component** command that you use in the AWS CLI, create a JSON file that contains all of the component parameters that you want to pass into the command. Include the location of the *wget-macos.yaml* document that you created earlier. The `uri` key-value pair contains the file reference.

> **Note**
>
> The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API command request parameters, see the CreateComponent command in the *EC2 Image Builder API Reference*.
>
> To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*.

1. **Create a CLI input JSON file**

   Use a file editing tool to create a file named *install-wget-macos-component.json*. Include the following content:

   ```
   {
    "name": "install install-wget-macos-component",
   ```

```
 "semanticVersion": "1.1.2",
 "description": "An example component that installs and verifies the wget
 utility on macOS.",
 "changeDescription": "Initial version.",
 "platform": "macOS",
 "uri": "s3://amzn-s3-demo-destination-bucket/my-path/wget-macos.yaml",
 "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/98765432-
b123-456b-7f89-0123456f789c",
 "tags": {
  "MyTagKey-purpose": "install-software"
 }
}
```

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://install-wget-macos-
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, while Linux and macOS use the forward slash (/).

AWSTOE component versioning for updates from the 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 in Image Builder.

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. Use the **list-components** command with the AWS CLI, and filter on the name.

In this example, you filter on the name of the component that you created in the prior Linux examples. To list the component that you created, use the value of the `name` parameter from the JSON file that you used in the **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.

Import a script to create a component from the 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 *amzn-s3-demo-source-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://amzn-s3-demo-source-bucket/AdminConfig.ps1",
"kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/60763706-
b131-418b-8f85-3420912f020c"
}
```

To create the component from an imported script, run the following command.

```
aws imagebuilder import-component --cli-input-json file://import-component.json
```

Note

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 outdated or unused Image Builder resources.

How Image Builder uses the AWS Task Orchestrator and Executor application to manage components

EC2 Image Builder uses the AWS Task Orchestrator and Executor (AWSTOE) application to orchestrate complex workflows, modify system configurations, and test your images without the need for additional devops scripts or code. This application manages and runs components that implement its declarative document schema.

AWSTOE is a standalone application that Image Builder installs on its build and test instances when you create an image. You can also install it manually on EC2 instances to create your own custom components. It doesn't require any additional setup, and can also run on premises.

Contents

AWSTOE downloads

To install AWSTOE, choose the download link for your architecture and platform. If you attach to a VPC endpoint for your service (Image Builder, for example), it must have a custom endpoint policy attached that includes access to the S3 bucket for AWSTOE downloads. Otherwise, your build and test instances will not be able to download the bootstrap script (`bootstrap.sh`) and install the AWSTOE application. For more information see Create a VPC endpoint policy for Image Builder.

> **Important**
>
> AWS is phasing out support for TLS versions 1.0 and 1.1. To access the S3 bucket for AWSTOE downloads, your client software must use TLS version 1.2 or later. For more information, see this AWS Security Blog post.

Architecture	Platform	Download link	Example
386	AL 2 and 2023 RHEL 7, 8, and 9 Ubuntu 16.04, 18.04, 20.04, 22.04, and 24.04 CentOS 7 and 8	https://awstoe- **<region>**.s3 s.com/latest/linux/386/awstoe	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/386/awstoe

Architecture	Platform	Download link	Example
	SUSE 12 and 15		
AMD64	AL 2 and 2023 RHEL 7, 8, and 9 Ubuntu 16.04, 18.04, 20.04, 22.04, and 24.04 CentOS 7 and 8 CentOS Stream 8 SUSE 12 and 15	`https://awstoe- `**`<region>`**`.s3 s.com/latest/linux/amd64/awstoe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/amd64/awstoe
AMD64	macOS 10.14.x (Mojave), 10.15.x (Catalina), 11.x (Big Sur), 12.x (Monterey)	`https://awstoe- `*`region`*`.s3.r s.com/latest/darwin/amd64/awstoe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/darwin/amd64/awstoe
AMD64	Windows Server 2012 R2, 2016, 2019, and 2022	`https://awstoe- `**`<region>`**`.s3 s.com/latest/windows/amd64/awstoe.exe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/windows/amd64/awstoe.exe

Architecture	Platform	Download link	Example
ARM64	AL 2 and 2023 RHEL 7, 8, and 9 Ubuntu 16.04, 18.04, 20.04, 22.04, and 24.04 CentOS 7 and 8 CentOS Stream 8 SUSE 12 and 15	`https://awstoe- **<region>**.s3.amazonaws.com/latest/linux/arm64/awstoe`	https://awstoe-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/arm64/awstoe

Supported Regions

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

AWS Region name	AWS 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 (Osaka)	ap-northeast-3

AWS Region name	AWS Region
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Stockholm)	eu-north-1
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Israel (Tel Aviv)	il-central-1
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1

AWS Region name	AWS Region
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1

AWSTOE command reference

AWSTOE is a command line component management application that runs on Amazon EC2 instances. When Image Builder launches an EC2 build or test instance, it installs AWSTOE on the instance. Then it runs AWSTOE commands in the AWS CLI to install or validate the components that are specified in the image or container recipe.

> **Note**
>
> Some AWSTOE action modules require elevated permissions to run on a Linux server. To use elevated permissions, prefix the command syntax with **sudo**, or run the **sudo su** command one time when you log in before running the commands linked below. For more information about AWSTOE action modules, see Action modules supported by AWSTOE component manager.

run

Use the **run** command to run the YAML document scripts for one or more component documents.

validate

Run the **validate** command to validate the YAML document syntax for one or more component documents.

awstoe run command

This command runs the YAML component document scripts in the order in which they are included in the configuration file specified by the `--config` parameter, or the list of component documents specified by the `--documents` parameter.

> **Note**
>
> You must specify exactly one of the following parameters, never both:
>
> --config
>
> --documents

Syntax

```
awstoe run [--config <file path>] [--cw-ignore-failures <?>]
      [--cw-log-group <?>] [--cw-log-region us-west-2] [--cw-log-stream <?>]
      [--document-s3-bucket-owner <owner>] [--documents <file path,file path,...>]
      [--execution-id <?>] [--log-directory <file path>]
      [--log-s3-bucket-name <name>] [--log-s3-bucket-owner <owner>]
      [--log-s3-key-prefix <?>] [--parameters name1=value1,name2=value2...]
      [--phases <phase name>] [--state-directory <directory path>] [--version <?>]
      [--help] [--trace]
```

Parameters and options

Parameters

--config *./config-example.json*

Short form: -c *./config-example.json*

The configuration file *(conditional)*. This parameter contains the file location for the JSON file that contains configuration settings for the components this command is running. If you specify **run** command settings in a configuration file, you must not specify the `--documents` parameter. For more information about input configuration, see Configure input for the AWSTOE run command.

Valid locations include:

- A local file path (*./config-example.json*)
- An S3 URI (`s3://`*bucket/key*)

--cw-ignore-failures

Short form: N/A

Ignore logging failures from the CloudWatch Logs.

--cw-log-group

Short form: N/A

The `LogGroup` name for the CloudWatch Logs.

--cw-log-region

Short form: N/A

The AWS Region that applies to the CloudWatch Logs.

--cw-log-stream

Short form: N/A

The `LogStream` name for the CloudWatch Logs, that directs AWSTOE where to stream the `console.log` file.

--document-s3-bucket-owner

Short form: N/A

The account ID of the bucket owner for S3 URI-based documents.

--documents *./doc-1.yaml,./doc-n.yaml*

Short form: -d *./doc-1.yaml*,*./doc-n*

The component documents *(conditional)*. This parameter contains a comma-separated list of file locations for the YAML component documents to run. If you specify YAML documents for the **run** command using the `--documents` parameter, you must not specify the `--config` parameter.

Valid locations include:

- local file paths (*./component-doc-example.yaml*).
- S3 URIs (`s3://`*bucket/key*).
- Image Builder component build version ARNs (arn:aws:imagebuilder:us-west-*2:123456789012*:component/*my-example-component*/2021.12.02/1).

> **Note**
>
> There are no spaces between items in the list, only commas.

--execution-id

Short form: -i

This is the unique ID that applies to the execution of the current **run** command. This ID is included in output and log file names, to uniquely identify those files, and link them to the current command execution. If this setting is left out, AWSTOE generates a GUID.

--log-directory

Short form: -l

The destination directory where AWSTOE stores all of the log files from this command execution. By default, this directory is located inside of the following parent directory: `TOE_<DATETIME>_<EXECUTIONID>`. If you do not specify the log directory, AWSTOE uses the current working directory (`.`).

--log-s3-bucket-name

Short form: -b

If component logs are stored in Amazon S3 (recommended), AWSTOE uploads the component application logs to the S3 bucket named in this parameter.

--log-s3-bucket-owner

Short form: N/A

If component logs are stored in Amazon S3 (recommended), this is the owner account ID for the bucket where AWSTOE writes the log files.

--log-s3-key-prefix

Short form: -k

If component logs are stored in Amazon S3 (recommended), this is the S3 object key prefix for the log location in the bucket.

--parameters *`name1`*=*`value1`*,*`name2`*=*`value2`*...

Short form: N/A

Parameters are mutable variables that are defined in the component document, with settings that the calling application can provide at runtime.

--phases

Short form: -p

A comma-separated list that specifies which phases to run from the YAML component documents. If a component document includes additional phases, those will not run.

--state-directory

Short form: -s

The file path where state tracking files are stored.

--version

Short form: -v

Specifies the component application version.

Options

--help

Short form: -h

Displays a help manual for using the component management application options.

--trace

Short form: -t

Enables verbose logging to the console.

awstoe validate command

When you run this command, it validates the YAML document syntax for each of the component documents specified by the `--documents` parameter.

Syntax

```
awstoe validate [--document-s3-bucket-owner <owner>]
```

```
    --documents <file path,file path,...> [--help] [--trace]
```

Parameters and options

Parameters

--document-s3-bucket-owner

Short form: N/A

Source account ID of S3 URI-based documents provided.

--documents ***./doc-1.yaml,./doc-n.yaml***

Short form: -d *./doc-1.yaml*,*./doc-n*

The component documents *(required)*. This parameter contains a comma-separated list of file locations for the YAML component documents to run. Valid locations include:

- local file paths (*./component-doc-example.yaml*)
- S3 URIs (s3://*bucket/key*)
- Image Builder component build version ARNs (arn:aws:imagebuilder:us-west-*2*:*123456789012*:component/*my-example-component*/2021.12.02/1)

> **Note**
> There are no spaces between items in the list, only commas.

Options

--help

Short form: -h

Displays a help manual for using the component management application options.

--trace

Short form: -t

Enables verbose logging to the console.

Manual set up to develop custom components 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 manually install the AWSTOE application and use it as a stand-alone application to develop custom components. Image Builder takes care of these steps for you, if you use the Image Builder console or AWS CLI commands to create custom components. For more information, see Create custom components with Image Builder.

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, macOS and Windows based operating systems.

Topics

Verify the signature of the AWSTOE installation download on Linux or macOS

This topic describes the recommended process for verifying the validity of the installation download for the AWSTOE on Linux-based or macOS 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 application is altered or corrupted, do not run the installation file. Instead, contact Support For more information about your support options, see Support.

AWSTOE files for Linux-based and macOS 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
- Authenticating and importing the public key
- Verify the signature of the package

Installing the GPG tools

If your operating system is Linux, Unix, or macOS, 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, select the operating system that matches your instance.

Debian-based Linux

From a terminal, run the following command:

```
apt-get install gnupg
```

Red Hat–based Linux

From a terminal, run the following command:

```
yum install gnupg
```

macOS

From a terminal, run the following command:

```
brew 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 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 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 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:

BA 81 25 EE AC 64 2E A9 F3 C5 93 CA 6D 3E B7 93 7D 68 75 74

> **Note**
>
> During the roll-out window for a new binary, your signer certificate might not match the new thumbprint. If your signer certificate doesn't match, verify that the thumbprint value is:
> **F8 83 11 EE F0 4A A2 91 E3 79 21 BA 6B FC AF F8 19 92 12 D7**

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:

      BA 81 25 EE AC 64 2E A9 F3 C5 93 CA 6D 3E B7 93 7D 68 75 74

      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
- Step 2: Set AWS credentials
- Step 3: Develop component documents locally
- Step 4: Validate AWSTOE components
- Step 5: Run AWSTOE components

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

> **Important**
>
> AWS is phasing out support for TLS versions 1.0 and 1.1. To access the S3 bucket for AWSTOE downloads, your client software must use TLS version 1.2 or later. For more information, see this AWS Security Blog post.

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 or macOS. To verify your download on a Windows server, see Verify the signature on Windows.

> **Note**
>
> AWSTOE is invoked directly from its download location. There is no need for a separate install step. This also means that AWSTOE can make changes to the local environment. To ensure that you isolate changes during component development, we recommend that you use an EC2 instance to develop and test AWSTOE components.

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`.

```
$env:AWS_ACCESS_KEY_ID=your_access_key_id
```

```
$env:AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

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

```
set AWS_ACCESS_KEY_ID=your_access_key_id
```

```
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Step 3: Develop component documents locally

Components are authored with plaintext YAML documents. For more information about document syntax, see Use the AWSTOE component document framework for custom components.

The following are example *Hello World* component documents to help you get started.

Linux

Some of the Linux component examples in this guide refer to a component document file named `hello-world-linux.yml`. You can use the following document to get started with those examples.

```
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.'
```

Windows

Some of the Windows component examples in this guide refer to a component document file named `hello-world-windows.yml`. You can use the following document to get started with those examples.

```
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.'
```

macOS

Some of the macOS component examples in this guide refer to a component document file named `hello-world-macos.yml`. You can use the following document to get started with those examples.

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

Linux or macOS

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

Windows

```
awstoe.exe validate --documents C:\Users\user\Documents\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 Image Builder infrastructure configuration

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 amzn-s3-demo-destination-
bucket --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 build,test
```

Get help with AWSTOE

```
awstoe --help
```

Use the AWSTOE component document framework for custom components

To build a component using the AWS Task Orchestrator and Executor (AWSTOE) component framework, 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

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.

> **Tip**
>
> The service that uses your component to build an image might implement rules about what phases to use for their build process, and when those phases are allowed to run. 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` and `container-host-test` phases 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** – for Image Builder, this phase is generally used during the *build stage*.
  - **validate** – for Image Builder, this phase is generally used during the *build stage*.
  - **test** – for Image Builder, this phase is 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.

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 and macOS**: `/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 timeout. 0 and other negative values are not allowed.	Integer	No	7,200 sec (120 mins)
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,	String	No	Abort

Field	Description	Type	Required	Default value
	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			

Field	Description	Type	Required	Default value
	document to `SuccessWithIgnoredFailure` .			
maxAttempts	Maximum number of attempts allowed before failing the step.	Integer	No	1
inputs	Contains parameters required by the action module to run the step.	Dict	Yes	

Document examples

The following examples show AWSTOE component documents that perform tasks for the target operating system.

Linux

Example 1: Run a custom binary file

The following is an example document that downloads and runs a custom binary file on a Linux instance.

```
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://<replaceable>amzn-s3-demo-source-bucket</replaceable>/
<replaceable>myapplication</replaceable>
            destination: /tmp/<replaceable>myapplication</replaceable>
      - 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 }}'
```

Windows

Example 1: Install Windows updates

The following is an example document that installs all available Windows updates, runs a configuration script, validates the changes before the AMI is created, and tests 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}}'
```

Example 2: Install the AWS CLI on a Windows instance

The following is an example document that installs the AWS CLI on a Windows instance, using the setup file.

```
name: InstallCLISetUp
description: Install &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 }}'
```

Example 3: Install the AWS CLI with the MSI installer

The following is an example document that installs the AWS CLI with the MSI installer.

```
name: InstallCLIMSI
description: Install &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 }}'
```

macOS

Example 1: Run a custom macOS binary file

The following is an example document that downloads and runs a custom binary file on a macOS instance.

```
name: macOSBin
description: Download and run a binary file on macOS.
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: Download
        action: S3Download
        inputs:
          - source: s3://<replaceable>amzn-s3-demo-source-bucket</replaceable>/
<replaceable>myapplication</replaceable>
            destination: /tmp/<replaceable>myapplication</replaceable>
      - 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 }}'
```

Use variables in your custom component document

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.

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 }}
```

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.

> **Important**
>
> Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets Manager, see What is Secrets Manager? in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

Syntax

```
--parameters name1=value1,name2=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
```

Use Systems Manager Parameter Store parameters

You can reference AWS Systems Manager Parameter Store parameters (SSM parameters) in your component documents by prefixing variables with `aws:ssm`. For example,

`{{ aws:ssm:/my/param }}` resolves to the value of the SSM parameter `/my/param`.

This feature supports the following SSM parameter types:

- String – Maps to the AWSTOE string type.
- StringList – Maps to the AWSTOE `stringList` type.
- SecureString – Maps to the AWSTOE string type.

For more information about the Parameter Store see [AWS Systems Manager Parameter Store] in the *AWS Systems Manager User Guide*.

You can also reference AWS Secrets Manager secrets using an SSM parameter `SecureString`. For example: `{{ aws:ssm:/aws/reference/secretsmanager/test/test-secret }}`. For more information, see [Referencing AWS Secrets Manager secrets from Parameter Store parameters].

> **Important**
>
> Image Builder excludes `SecureString` parameter resolution from its logs. However, you are also responsible for ensuring that sensitive information is not logged through commands issued in the component document. For example, if you use the `echo` command with a secure string, the command writes a plaintext value to the log.

Required IAM permissions

To use Systems Manager parameters in your components, your instance role must have the `ssm:GetParameter` permission for the parameter resource ARN. For example:

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "ssm:GetParameter",
   "Resource": "arn:aws:ssm:*:111122223333:parameter/ImageBuilder-*"
  }
 ]
}
```

To access encrypted values, you'll also need the following permissions:

- Add `kms:Decrypt` for `SecureString` parameters or AWS Secrets Manager values that are encrypted with a customer managed AWS KMS key.
- Add `secretsmanager:GetSecretValue` if you reference a Secrets Manager secret.

Reference an SSM parameter in a component document

The following example shows how to reference an Systems Manager Parameter Store parameter of Systems Manager parameters in a component:

```
name: UseSSMParameterVariable
description: This is a sample component document that prints out the value of an SSM
 Parameter. Never do this for a SecureString parameter.
schemaVersion: 1.0

phases:
  - name: verify
    steps:
      - name: EchoParameterValue
        action: ExecuteBash
        inputs:
```

```
          commands:
            - echo "Log SSM parameter name: /my/test/param, value {{ aws:ssm:/my/test/
param }}."
```

Dynamic runtime variable resolution for SSM parameters

AWSTOE provides the following built-in function that you can use within variable references to manipulate or transform values at runtime.

resolve function

The `resolve` function resolves a variable reference inside of another variable reference, allowing for dynamic variable name referencing. This is useful when working with SSM parameters where part of the parameter path may be variable and passed in as a document parameter.

The `resolve` function only supports dynamic resolution of the name portion of an SSM parameter.

Syntax

The `dynamic_variable` in the following example represents the name of an SSM parameter, and must be one of the following:

- An SSM parameter reference (for example, `aws:ssm:/my/param`)
- A component document parameter reference (for example, `parameter-name`)

```
{{ aws:ssm:resolve(dynamic_variable) }}
```

Example: Resolve an SSM parameter at runtime

The following example shows how to use the `resolve` function in a YAML component document:

```
name: SsmParameterTest
description: This component verifies an SSM parameter variable reference with the echo
 command.
schemaVersion: 1.0

parameters:
  - parameter-name:
      type: string
      description: "test"
```

```
phases:
  - name: validate
    steps:
      - name: PrintDynamicVariable
        action: ExecuteBash
        inputs:
          commands:
            - echo "{{ aws:ssm:resolve(parameter-name) }}"
```

Use conditional constructs in AWSTOE

Conditional constructs perform different actions in your component document based on whether the specified conditional expression evaluates to `true` or `false`. You can use the `if` construct to control the flow of execution in your component document.

if Construct

You can use the `if` construct to evaluate whether a step should run or not. By default, when the `if` conditional expression evaluates to `true`, AWSTOE runs the step, and when the condition evaluates to `false`, AWSTOE skips the step. If a step is skipped, it's treated as a successful step when AWSTOE evaluates whether the phase and document ran successfully.

> **Note**
>
> An `if` statement is only evaluated one time, even if the step triggers a restart. If a step restarts, it recognizes that the `if` statement has already been evaluated, and continues where it left off.

Syntax

```
if:
  - <conditional expression>:
      [then: <step action>]
      [else: <step action>]
```

Key name	Required	Description
conditional expression	Yes	

Key name	Required	Description
		The conditional expression can contain exactly one of the following types of operators at the top level. • **Comparison operator** – For a list of comparison operators, and information about how they work in AWSTOE component documents, see Comparison operators. • **Logical operator** – Logical operators include `and`, `or`, and `not`, and operate on one or more comparison operators. For more information about how logical operators work in AWSTOE component documents, see Logical operators. If your expression must satisfy multiple conditions, use a logical operator to specify your conditions.
then	No	Defines the action to take if the conditional expression evaluates to `true`.

Key name	Required	Description
else	No	Defines the action to take if the conditional expression evaluates to `false`.
step action	Conditional	When you use `then` or `else`, you must specify one of the following step actions: • **Abort** – AWSTOE marks the step as failed. • **Execute** – AWSTOE runs the step. • **Skip** – AWSTOE skips the step.

Example 1: Install package

The following example steps from an AWSTOE component document use logical operators to test a parameter value and run the appropriate package manager commands to install an application if the package is unzipped.

```
- name: InstallUnzipAptGet
    action: ExecuteBash
    if:
      and:
          - binaryExists: 'apt-get'
          - not:
              binaryExists: 'unzip'
    inputs:
      commands:
          - sudo apt-get update
          - sudo apt-get install -y unzip
```

```
  - name: InstallUnzipYum
    action: ExecuteBash
    if:
      and:
          - binaryExists: 'yum'
          - not:
              binaryExists: 'unzip'
    inputs:
      commands:
          - sudo yum install -y unzip

  - name: InstallUnzipZypper
    action: ExecuteBash
    if:
      and:
          - binaryExists: 'zypper'
          - not:
              binaryExists: 'unzip'
    inputs:
      commands:
          - sudo zypper refresh
          - sudo zypper install -y unzip
```

Example 2: Skip a step

The following example shows two ways to skip a step. One uses a logical operator, and one uses a comparison operator with the `Skip` step action.

```
# Creates a file if it does not exist using not
- name: CreateMyConfigFile-1
  action: ExecuteBash
  if:
    not:
      fileExists: '/etc/my_config'
  inputs:
    commands:
      - echo "Hello world" > '/etc/my_config'

# Creates a file if it does not exist using then and else
- name: CreateMyConfigFile-2
  action: ExecuteBash
  if:
    fileExists: '/etc/my_config'
```

```
      then: Skip
      else: Execute
    inputs:
      commands:
        - echo "Hello world" > '/etc/my_config'
```

Use comparison operators in AWSTOE component documents

You can use the following comparison operators with the **Assert** action module and with conditional expressions that use the if Construct. A comparison operator can operate on a single value, for example `stringIsEmpty`, or it can compare a baseline value to a second value (variable value) to determine whether the conditional expression evaluates to `true` or `false`.

If the comparison operates on two values, the second value can be a chaining variable.

When comparing values of a different type, the following value conversions can occur prior to the comparison:

- For numeric comparisons, if the variable value is a string, AWSTOE converts the string to a number prior to the evaluation. If the conversion is not possible, the comparison returns `false`. For example, if the variable value is `"1.0"`, the conversion works, but if the variable value is `"a10"` the conversion fails.
- For string comparisons, if the variable value is a number, AWSTOE converts it to a string prior to the evaluation.

Compare strings

The following comparison operators work with strings to compare values, to test for spaces or an empty string, or to compare an input value to a regex pattern. String comparisons are not case sensitive, and they don't trim spaces from the beginning or the end of the string inputs.

String comparison operators

- stringIsEmpty
- stringIsWhitespace
- stringEquals
- stringLessThan
- stringLessThanEquals

- stringGreaterThan
- stringGreaterThanEquals
- patternMatches

stringIsEmpty

The `stringIsEmpty` operator returns `true` if the specified string doesn't contain any characters. For example:

```
# Evaluates to true
stringIsEmpty: ""

# Evaluates to false
stringIsEmpty: " "

# Evaluates to false
stringIsEmpty: "Hello."
```

stringIsWhitespace

Tests if the string specified for `stringIsWhitespace` contains only spaces. For example:

```
# Evaluates to true
stringIsWhitespace: "   "

# Evaluates to false
stringIsWhitespace: ""

# Evaluates to false
stringIsWhitespace: " Hello?"
```

stringEquals

Tests if the string specified for `stringEquals` is an exact match for the string specified in the `value` parameter. For example:

```
# Evaluates to true
stringEquals: 'Testing, testing...'
value: 'Testing, testing...'
```

```
# Evaluates to false
stringEquals: 'Testing, testing...'
value: 'Hello again.'

# Evaluates to false
stringEquals: 'Testing, testing...'
value: 'TESTING, TESTING....'

# Evaluates to false
stringEquals: 'Testing, testing...'
value: '   Testing, testing...'

# Evaluates to false
stringEquals: 'Testing, testing...'
value: 'Testing, testing...   '
```

stringLessThan

Tests if the string specified for `stringLessThan` is less than the string specified in the `value` parameter. For example:

```
# Evaluates to true
# This comparison operator isn't case sensitive
stringlessThan: 'A'
value: 'a'

# Evaluates to true - 'a' is less than 'b'
stringlessThan: 'b'
value: 'a'

# Evaluates to true
# Numeric strings compare as less than alphabetic strings
stringlessThan: 'a'
value: '0'

# Evaluates to false
stringlessThan: '0'
value: 'a'
```

stringLessThanEquals

Tests if the string specified for `stringLessThanEquals` is less than or equal to the string specified in the `value` parameter. For example:

```
# Evaluates to true - 'a' is equal to 'a'
stringLessThanEquals: 'a'
value: 'a'

# Evaluates to true - since the comparison isn't case sensitive, 'a' is equal to 'A'
stringLessThanEquals: 'A'
value: 'a'

# Evaluates to true - 'a' is less than 'b'
stringLessThanEquals: 'b'
value: 'a'

# Evaluates to true - '0' is less than 'a'
stringLessThanEquals: 'a'
value: '0'

# Evaluates to false - 'a' is greater than '0'
stringLessThanEquals: '0'
value: 'a'
```

stringGreaterThan

Tests if the string specified for `stringGreaterThan` is greater than the string specified in the `value` parameter. For example:

```
# Evaluates to false - since the comparison isn't case sensitive, 'A' is equal to
 'a'
stringGreaterThan: 'a'
value: 'A'

# Evaluates to true - 'b' is greater than 'a'
stringGreaterThan: 'a'
value: 'b'

# Evaluates to true - 'a' is greater than '0'
stringGreaterThan: '0'
value: 'a'

# Evaluates to false - '0' is less than 'a'
stringGreaterThan: 'a'
value: '0'
```

stringGreaterThanEquals

Tests if the string specified for `stringGreaterThanEquals` is greater than or equal to the string specified in the `value` parameter. For example:

```
# Evaluates to true - 'a' is equal to 'A'
stringGreaterThanEquals: 'A'
value: 'a'

# Evaluates to true - 'b' is greater than 'a'
stringGreaterThanEquals: 'a'
value: 'b'

# Evaluates to true - 'a' is greater than '0'
stringGreaterThanEquals: '0'
value: 'a'

# Evaluates to false - '0' is less than 'a'
stringGreaterThanEquals: 'a'
value: '0'
```

patternMatches

Tests if the string specified in the `value` parameter matches the regex pattern specified for `patternMatches`. The comparison uses to the Golang regexp package, which conforms to the RE2 syntax. For more information about RE2 rules, see the google / re2 repository in *GitHub*.

The following example shows a pattern match that returns `true`:

```
patternMatches: '^[a-z]+$'
value: 'ThisIsValue'
```

Compare numbers

The following comparison operators work with numbers. The values provided for these operators must be one of the following types, according to the YAML specification. Support for numeric comparisons uses the golang big package comparison operator, for example: func (*Float) Cmp.

- Integer
- Float (based on float64, which supports numbers from -1.7e+308 to +1.7e+308)
- A string that matches the following regex pattern: `^[-+]?([0-9]+[.])?[0-9]+$`

Number comparison operators

- numberEquals
- numberLessThan
- numberLessThanEquals
- numberGreaterThan
- numberGreaterThanEquals

numberEquals

Tests if the number specified for `numberEquals` is equal to the number specified in the `value` parameter. All of the following example comparisons return `true`:

```
# Values provided as a positive number
numberEquals: 1
value: 1

# Comparison value provided as a string
numberEquals: '1'
value: 1

# Value provided as a string
numberEquals: 1
value: '1'

# Values provided as floats
numberEquals: 5.0
value: 5.0

# Values provided as a negative number
numberEquals: -1
value: -1
```

numberLessThan

Tests if the number specified for `numberLessThan` is less than the number specified in the `value` parameter. For example:

```
# Evaluates to true
numberLessThan: 2
```

```
value: 1

# Evaluates to true
numberLessThan: 2
value: 1.9

# Evaluates to false
numberLessThan: 2
value: '2'
```

numberLessThanEquals

Tests if the number specified for `numberLessThanEquals` is less than or equal to the number specified in the `value` parameter. For example:

```
# Evaluates to true
numberLessThanEquals: 2
value: 1

# Evaluates to true
numberLessThanEquals: 2
value: 1.9

# Evaluates to true
numberLessThanEquals: 2
value: '2'

# Evaluates to false
numberLessThanEquals: 2
value: 2.1
```

numberGreaterThan

Tests if the number specified for `numberGreaterThan` is greater than the number specified in the `value` parameter. For example:

```
# Evaluates to true
numberGreaterThan: 1
value: 2

# Evaluates to true
numberGreaterThan: 1
value: 1.1
```

```
# Evaluates to false
numberGreaterThan: 1
value: '1'
```

numberGreaterThanEquals

Tests if the number specified for `numberGreaterThanEquals` is greater than or equal to the number specified in the `value` parameter. For example:

```
# Evaluates to true
numberGreaterThanEquals: 1
value: 2

# Evaluates to true
numberGreaterThanEquals: 1
value: 1.1

# Evaluates to true
numberGreaterThanEquals: 1
value: '1'

# Evaluates to false
numberGreaterThanEquals: 1
value: 0.8
```

Check files

The following comparison operators check the file hash or check if a file or folder exists.

File and folder operators

- binaryExists
- fileExists
- folderExists
- fileMD5Equals
- fileSHA1Equals
- fileSHA256Equals
- fileSHA512Equals

binaryExists

Tests whether an application is available in the current path. For example:

```
binaryExists: 'foo'
```

Note

On Linux and macOS systems, for an application named *foo*, this works the same as the following bash command: **type *foo* >/dev/null 2>&1**, where **$? == 0** indicates a successful comparison.

On Windows systems, for an application named *foo*, this works the same as the PowerShell command **& C:\Windows\System32\where.exe /Q *foo*** where **$LASTEXITCODE = 0** indicates a successful comparison.

fileExists

Tests whether a file exists at the specified path. You can provide an absolute or relative path. If the location you specify exists and is a file, the comparison evaluates to `true`. For example:

```
fileExists: '/path/to/file'
```

Note

On Linux and macOS systems, this works the same as the following bash command: **-d */path/to/file***, where **$? == 0** indicates a successful comparison.

On Windows systems, this works the same as the PowerShell command **Test-Path -Path '*C:\path\to\file*' -PathType 'Leaf'**.

folderExists

Tests whether a folder exists at the specified path. You can provide an absolute or relative path. If the location you specify exists and is a folder, the comparison evaluates to `true`. For example:

```
folderExists: '/path/to/folder'
```

> **Note**
>
> On Linux and macOS systems, this works the same as the following bash command: **-d *path/to/folder***, where **$? == 0** indicates a successful comparison.
> On Windows systems, this works the same as the PowerShell command **Test-Path -Path '*C:\path\to\folder*' -PathType 'Container'**.

fileMD5Equals

Tests whether a file's MD5 hash equals a specified value. For example:

```
fileMD5Equals: '<MD5Hash>'
path: '/path/to/file'
```

fileSHA1Equals

Tests whether a file's SHA1 hash equals a specified value. For example:

```
fileSHA1Equals: '<SHA1Hash>'
path: '/path/to/file'
```

fileSHA256Equals

Tests whether a file's SHA256 hash equals a specified value. For example:

```
fileSHA256Equals: '<SHA256Hash>'
path: '/path/to/file'
```

fileSHA512Equals

Tests whether a file's SHA512 hash equals a specified value. For example:

```
fileSHA512Equals: '<SHA512Hash>'
path: '/path/to/file'
```

Use logical operators in AWSTOE component documents

You can use the following logical operators to add or modify conditional expressions in your component document. AWSTOE evaluates conditional expressions in the order that the conditions are specified. For more information about comparison operators for component documents, see Use comparison operators in AWSTOE component documents.

and

With the and operator, you can evaluate two or more comparisons as a single expression. The expression evaluates to `true` when all of the conditions in the list are true. Otherwise, the expression evaluates to `false`.

Examples:

The following example performs two comparisons – a string and a number. Both comparisons are true, so the expression evaluates to true.

```
and:
  - stringEquals: 'test_string'
    value: 'test_string'
  - numberEquals: 1
    value: 1
```

The following example also performs two comparisons. The first comparison is false, at which point evaluation stops and the second comparison is skipped. The expression evaluates to `false`.

```
and:
  - stringEquals: 'test_string'
    value: 'Hello world!'
  - numberEquals: 1
    value: 1
```

or

With the `or` operator, you can evaluate two or more comparisons as a single expression. The expression evaluates to `true` when one of the specified comparisons is true. If none of the specified comparisons evaluate to `true`, the expression evaluates to `false`.

Examples:

The following example performs two comparisons – a string and a number. The first comparison is true, so the expression evaluates to `true` and the second comparison is skipped.

```
or:
  - stringEquals: 'test_string'
    value: 'test_string'
  - numberEquals: 1
    value: 3
```

The following example also performs two comparisons. The first comparison is false, and evaluation continues. The second comparison is true, so the expression evaluates to `true`.

```
or:
  - stringEquals: 'test_string'
    value: 'Hello world!'
  - numberEquals: 1
    value: 1
```

In the final example, both comparisons are false, so the expression evaluates to `false`.

```
or:
  - stringEquals: 'test_string'
    value: 'Hello world!'
  - numberEquals: 1
    value: 3
```

not

With the `not` operator, you can negate a single comparison. The expression evaluates to `true` if the comparison is false. If the comparison is true, then the expression evaluates to `false`.

Examples:

The following example performs a string comparison. The comparison is false, so the expression evaluates to `true`.

```
not:
  - stringEquals: 'test_string'
    value: 'Hello world!'
```

The following example also performs a string comparison. The comparison is true, so the expression evaluates to `false`.

```
not:
```

```
  - stringEquals: 'test_string'
    value: 'test_string'
```

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
- Types of looping constructs
- Step fields
- Step and iteration outputs

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
- forEach loop with input list
- forEach loop with delimited list

`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	String	No	""

Field	Description	Type	Required	Default
	compared to other loop names in the same phase.			
`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	String	No	""

Field	Description	Type	Required	Default
	other loop names in the same phase.			
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 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	String	Yes	n/a

Field	Description	Type	Required	Default
	interpretation by the YAML compiler.			

Field	Description	Type	Required	Default
`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: `"_"`	String	No	Comma: `","`

Field	Description	Type	Required	Default
	Chaining expressions cannot be used.			

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

This example uses the following chaining expression pattern to refer to another step's output: `<phase_name>.<step_name>.[inputs | outputs].<var_name>`.

```
- 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

Image building services, such as EC2 Image Builder, use AWSTOE action modules to help configure the EC2 instances that are used for building and testing customized machine images. This section describes the features of commonly used AWSTOE action modules, and how to configure them, including examples.

Components are authored with plaintext YAML documents. For more information about document syntax, see Use the AWSTOE component document framework for custom components.

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

The following cross-reference categorizes action modules by the type of actions that they perform.

General execution

- Assert (Linux, Windows, macOS)
- ExecuteBash (Linux, macOS)
- ExecuteBinary (Linux, Windows, macOS)
- ExecuteDocument (Linux, Windows, macOS)
- ExecutePowerShell (Windows)

File download and upload

- S3Download (Linux, Windows, macOS)

- S3Upload (Linux, Windows, macOS)
- WebDownload (Linux, Windows, macOS)

File system operations

- AppendFile (Linux, Windows, macOS)
- CopyFile (Linux, Windows, macOS)
- CopyFolder (Linux, Windows, macOS)
- CreateFile (Linux, Windows, macOS)
- CreateFolder (Linux, Windows, macOS)
- CreateSymlink (Linux, Windows, macOS)
- DeleteFile (Linux, Windows, macOS)
- DeleteFolder (Linux, Windows, macOS)
- ListFiles (Linux, Windows, macOS)
- MoveFile (Linux, Windows, macOS)
- MoveFolder (Linux, Windows, macOS)
- ReadFile (Linux, Windows, macOS)
- SetFileEncoding (Linux, Windows, macOS)
- SetFileOwner (Linux, Windows, macOS)
- SetFolderOwner (Linux, Windows, macOS)
- SetFilePermissions (Linux, Windows, macOS)
- SetFolderPermissions (Linux, Windows, macOS)

Software installation actions

- InstallMSI (Windows)
- UninstallMSI (Windows)

System actions

- Reboot (Linux, Windows)
- SetRegistry (Windows)
- UpdateOS (Linux, Windows)

General execution modules

The following section contains details for action modules that run commands and control execution workflow.

General execution action modules

- Assert (Linux, Windows, macOS)
- ExecuteBash (Linux, macOS)
- ExecuteBinary (Linux, Windows, macOS)
- ExecuteDocument (Linux, Windows, macOS)
- ExecutePowerShell (Windows)

Assert (Linux, Windows, macOS)

The **Assert** action module performs value comparisons using Comparison operators or Logical operators as input. The result of the operator expression (true or false) indicates the overall success or failure status for the step.

If the comparison or logical operator expression evaluates to `true`, the step is marked as `Success`. Otherwise, the step is marked as `Failed`. If the step fails, the `onFailure` parameter decides the outcome of the step.

Input

Key name	Description	Type	Required
`input`	Contains a single comparison or logical operator. Note, logical operators can contain more	This is variable, depending on the operator	Yes

Key name	Description	Type	Required
	than one comparison operator.		

Input example: A simple comparison using the `stringEquals` comparison operator

This example evaluates to `true`.

```
- name: StringComparison
  action: Assert
  inputs:
    stringEquals: '2.1.1'
    value: '{{ validate.ApplicationVersion.outputs.stdout }}'
```

Input example: Regex comparisons using the `patternMatches` comparison operator

These examples all evaluate to `true`.

```
- name: Letters only
  action: Assert
  inputs:
    patternMatches: '^[a-zA-Z]+$'
    value: 'ThisIsOnlyLetters'

- name: Letters and spaces only
  action: Assert
  inputs:
    patternMatches: '^[a-zA-Z\s]+$'
    value: 'This text contains spaces'

- name: Numbers only
  action: Assert
  inputs:
    patternMatches: '^[0-9]+$'
    value: '1234567890'
```

Input example: Nested comparisons with logical operators and chained variables

The following example demonstrates nested comparisons with logical operators that use comparisons with chained variables. The `Assert` evaluates to `true` if either of the following are true:

- The `ApplicationVersion` is greater than `2.0` and the `CPUArchitecture` equals `arm64`.
- The `CPUArchitecture` equals `x86_64`.

```
- name: NestedComparisons
  action: Assert
  inputs:
    or: # <- first level deep
      - and: # <- second level deep
          - numberGreaterThan: 2.0 # <- third level deep
            value: '{{ validate.ApplicationVersion.outputs.stdout }}'
          - stringEquals: 'arm64'
            value: '{{ validate.CPUArchitecture.outputs.stdout }}'
      - stringEquals: 'x86_64'
        value: '{{ validate.CPUArchitecture.outputs.stdout }}'
```

Output:

The output of an `Assert` is success or failure of the step.

ExecuteBash (Linux, macOS)

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

Key name	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: Before and after a reboot

```
name: ExitCode194Example
description: This shows how the exit code can be used to restart a system with
 ExecuteBash
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: RestartTrigger
        action: ExecuteBash
        inputs:
          commands:
            - |
              REBOOT_INDICATOR=/var/tmp/reboot-indicator
              if [ -f "${REBOOT_INDICATOR}" ]; then
                echo 'The reboot file exists. Deleting it and exiting with success.'
                rm "${REBOOT_INDICATOR}"
                exit 0
              fi
              echo 'The reboot file does not exist. Creating it and triggering a
 restart.'
              touch "${REBOOT_INDICATOR}"
              exit 194
```

Output

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

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.

Output example: Before reboot (first time through document)

```
{
 “stdout”: “The reboot file does not exist. Creating it and triggering a restart."
}
```

Output example: After reboot, (second time through document)

```
{
 “stdout”: “The reboot file exists. Deleting it and exiting with success."
}
```

ExecuteBinary (Linux, Windows, macOS)

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

Key name	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"
```

```
}
```

ExecuteDocument (Linux, Windows, macOS)

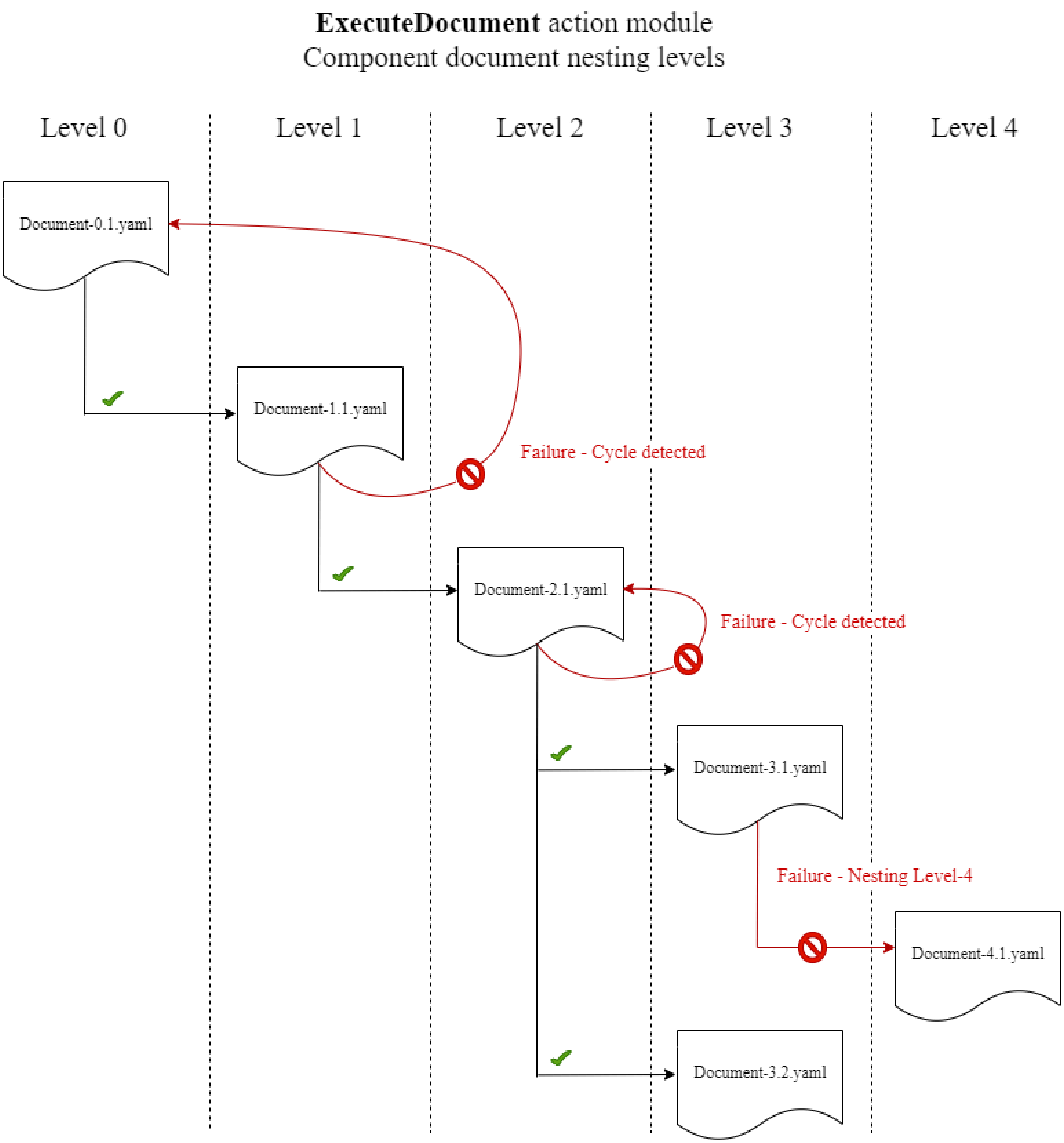
The **ExecuteDocument** action module adds support for nested component documents, running multiple component documents from one document. AWSTOE validates the document that is passed in the input parameter at run time.

Restrictions

- This action module runs one time, with no retries allowed, and no option to set timeout limits. **ExecuteDocument** sets the following default values, and returns an error if you try to change them.
  - `timeoutSeconds`: -1
  - `maxAttempts`: 1

  > **Note**
  >
  > You can leave these values blank, and AWSTOE uses the default values.

- Document nesting is allowed, up to three levels deep, but no more than that. Three levels of nesting translates to four document levels, as the top level isn't nested. In this scenario, the lowest level document must not call any other documents.
- Cyclic execution of component documents is not allowed. Any document that calls itself outside of a looping construct, or that calls another document higher up in the current chain of execution, initiates a cycle that can result in an endless loop. When AWSTOE detects a cyclic execution, it stops the execution and records the failure.



If a component document tries to run itself, or to run any of the component documents that are higher up in the current chain of execution, the execution fails.

Input

Key name	Description	Type	Required
`document`	Path of component document. Valid options include: • Local file paths • S3 URIs • EC2 Image Builder component build version ARNs	String	Yes
`document-s3-bucket-owner`	The account ID of the S3 bucket owner for the S3 bucket where component documents are stored. *(Recommended if you are using S3 URIs in your component document.)*	String	No
`phases`	Phases to run in the component document, expressed as a comma-separated list. If no phases are specified, then all phases run.	String	No
`parameters`		Parameter Map List	No

Key name	Description	Type	Required
	Input parameters that are passed in to the component document at runtime as key value pairs.		

Parameter map input

Key name	Description	Type	Required
`name`	The name of the input parameter to pass to the component document that the **ExecuteDocument** action module is running.	String	Yes
`value`	The value of the input parameter.	String	Yes

Input examples

The following examples show variations of the inputs for your component document, depending on your installation path.

Input example: Local document path

```
# main.yaml
schemaVersion: 1.0

phases:
  - name: build
    steps:
```

```
      - name: ExecuteNestedDocument
        action: ExecuteDocument
        inputs:
          document: Sample-1.yaml
          phases: build
          parameters:
            - name: parameter-1
              value: value-1
            - name: parameter-2
              value: value-2
```

Input example: S3 URI as a document path

```
# main.yaml
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: ExecuteNestedDocument
        action: ExecuteDocument
        inputs:
          document: s3://my-bucket/Sample-1.yaml
          document-s3-bucket-owner: 123456789012
          phases: build,validate
          parameters:
            - name: parameter-1
              value: value-1
            - name: parameter-2
              value: value-2
```

Input example: EC2 Image Builder component ARN as a document path

```
# main.yaml
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: ExecuteNestedDocument
        action: ExecuteDocument
        inputs:
          document: arn:aws:imagebuilder:us-west-2:aws:component/Sample-Test/1.0.0
```

```
          phases: test
          parameters:
            - name: parameter-1
              value: value-1
            - name: parameter-2
              value: value-2
```

Using a ForEach loop to run documents

```
# main.yaml
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: ExecuteNestedDocument
        action: ExecuteDocument
        loop:
          name: 'myForEachLoop'
          forEach:
            - Sample-1.yaml
            - Sample-2.yaml
        inputs:
          document: "{{myForEachLoop.value}}"
          phases: test
          parameters:
            - name: parameter-1
              value: value-1
            - name: parameter-2
              value: value-2
```

Using a For loop to run documents

```
# main.yaml
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: ExecuteNestedDocument
        action: ExecuteDocument
        loop:
          name: 'myForLoop'
```

```
        for:
          start: 1
          end: 2
          updateBy: 1
      inputs:
        document: "Sample-{{myForLoop.value}}.yaml"
        phases: test
        parameters:
          - name: parameter-1
            value: value-1
          - name: parameter-2
            value: value-2
```

Output

AWSTOE creates an output file called `detailedoutput.json` every time it runs. The file contains details about every phase and step of every component document that is invoked while it's running. For the **ExecuteDocument** action module, you can find a brief runtime summary in the `outputs` field, and details about the phases, steps, and documents that it runs in the `detailedOutput`.

```
{
 \"executedStepCount\":1,\"executionId\":\"97054e22-06cc-11ec-9b14-acde48001122\",
\"failedStepCount\":0,\"failureMessage\":\"\",\"ignoredFailedStepCount\":0,\"logUrl\":
\"\",\"status\":\"success\"
}",
```

Each component document's output summary object contains the following details, as shown here, with sample values:

- executedStepCount":1
- "executionId":"12345a67-89bc-01de-2f34-abcd56789012"
- "failedStepCount":0
- "failureMessage":""
- "ignoredFailedStepCount":0
- "logUrl":""
- "status":"success"

Output example

The following example shows output from the **ExecuteDocument** action module when a nested execution occurs. In this example, the `main.yaml` component document successfully runs the `Sample-1.yaml` component document.


```
{
    "executionId": "12345a67-89bc-01de-2f34-abcd56789012",
    "status": "success",
    "startTime": "2021-08-26T17:20:31-07:00",
    "endTime": "2021-08-26T17:20:31-07:00",
    "failureMessage": "",
    "documents": [
        {
            "name": "",
            "filePath": "main.yaml",
            "status": "success",
            "description": "",
            "startTime": "2021-08-26T17:20:31-07:00",
            "endTime": "2021-08-26T17:20:31-07:00",
            "failureMessage": "",
            "phases": [
                {
                    "name": "build",
                    "status": "success",
                    "startTime": "2021-08-26T17:20:31-07:00",
                    "endTime": "2021-08-26T17:20:31-07:00",
                    "failureMessage": "",
                    "steps": [
                        {
                            "name": "ExecuteNestedDocument",
                            "status": "success",
                            "failureMessage": "",
                            "timeoutSeconds": -1,
                            "onFailure": "Abort",
                            "maxAttempts": 1,
                            "action": "ExecuteDocument",
                            "startTime": "2021-08-26T17:20:31-07:00",
                            "endTime": "2021-08-26T17:20:31-07:00",
                            "inputs": "[{\"document\":\"Sample-1.yaml\",\"document-s3-
bucket-owner\":\"\",\"phases\":\"\",\"parameters\":null}]",
                            "outputs": "[{\"executedStepCount\":1,\"executionId\":
\"98765f43-21ed-09cb-8a76-fedc54321098\",\"failedStepCount\":0,\"failureMessage\":\"\",
\"ignoredFailedStepCount\":0,\"logUrl\":\"\",\"status\":\"success\"}]",
                            "loop": null,
```

```
                              "detailedOutput": [
                                  {
                                      "executionId": "98765f43-21ed-09cb-8a76-
fedc54321098",
                                      "status": "success",
                                      "startTime": "2021-08-26T17:20:31-07:00",
                                      "endTime": "2021-08-26T17:20:31-07:00",
                                      "failureMessage": "",
                                      "documents": [
                                          {
                                              "name": "",
                                              "filePath": "Sample-1.yaml",
                                              "status": "success",
                                              "description": "",
                                              "startTime": "2021-08-26T17:20:31-07:00",
                                              "endTime": "2021-08-26T17:20:31-07:00",
                                              "failureMessage": "",
                                              "phases": [
                                                  {
                                                      "name": "build",
                                                      "status": "success",
                                                      "startTime":
 "2021-08-26T17:20:31-07:00",
                                                      "endTime":
 "2021-08-26T17:20:31-07:00",
                                                      "failureMessage": "",
                                                      "steps": [
                                                          {
                                                              "name": "ExecuteBashStep",
                                                              "status": "success",
                                                              "failureMessage": "",
                                                              "timeoutSeconds": 7200,
                                                              "onFailure": "Abort",
                                                              "maxAttempts": 1,
                                                              "action": "ExecuteBash",
                                                              "startTime":
 "2021-08-26T17:20:31-07:00",
                                                              "endTime":
 "2021-08-26T17:20:31-07:00",
                                                              "inputs": "[{\"commands\":
[\"echo \\\"Hello World!\\\"\"]}]",
                                                              "outputs": "[{\"stdout\":
\"Hello World!\"}]",
                                                              "loop": null,
```

```
                                                                "detailedOutput": null
                                                        }]
                                                }]
                                        }]
                                }]
                        }]

                }]
        }]
}
```

ExecutePowerShell (Windows)

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

Key name	Description	Type	Required
`commands`	Contains a list of instructions or commands to run	String List	Yes. Must specify `commands` or `file`, not both.

Key name	Description	Type	Required
	as per PowerShell syntax. Multi-line YAML is allowed.		
`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: Before and after a reboot

```
name: ExitCode3010Example
description: This shows how the exit code can be used to restart a system with
 ExecutePowerShell
schemaVersion: 1.0
phases:
  - name: build
    steps:
      - name: RestartTrigger
        action: ExecutePowerShell
        inputs:
          commands:
            - |
              $rebootIndicator = Join-Path -Path $env:SystemDrive -ChildPath 'reboot-
indicator'
              if (Test-Path -Path $rebootIndicator) {
                Write-Host 'The reboot file exists. Deleting it and exiting with
 success.'
                Remove-Item -Path $rebootIndicator -Force | Out-Null
                [System.Environment]::Exit(0)
              }
              Write-Host 'The reboot file does not exist. Creating it and triggering a
 restart.'
              New-Item -Path $rebootIndicator -ItemType File | Out-Null
```

```
                [System.Environment]::Exit(3010)
```

Output

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

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.

Output example: Before reboot (first time through document)

```
{
 “stdout”: “The reboot file does not exist. Creating it and triggering a restart."
}
```

Output example: After reboot, (second time through document)

```
{
 “stdout”: “The reboot file exists. Deleting it and exiting with success."
}
```

File download and upload modules

The following section contains details for action modules that upload or download files.

Download and upload action modules

- S3Download (Linux, Windows, macOS)
- S3Upload (Linux, Windows, macOS)
- WebDownload (Linux, Windows, macOS)

S3Download (Linux, Windows, macOS)

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/*`*.

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

Key	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 asterisk wildcard (`/*`), to download a set of objects that match the key prefix.	String	Yes	N/A
`destination`	The local path where the Amazon S3 objects are downloaded. To download a single file, you must specify the file name as part of the path. For example,	String	Yes	N/A

Key	Description	Type	Required	Default
	/myfolder/package.zip .			
`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

Key	Description	Type	Required	Default
`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`	Boolean	No	true

Key	Description	Type	Required	Default
	false. Cannot download.			

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://amzn-s3-demo-source-bucket/path/to/package.zip
      destination: C:\myfolder\package.zip
      expectedBucketOwner: 123456789022
      overwrite: false
    - source: s3://amzn-s3-demo-source-bucket/path/to/package.zip
      destination: C:\myfolder\package.zip
      expectedBucketOwner: 123456789022
      overwrite: true
    - source: s3://amzn-s3-demo-source-bucket/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://amzn-s3-demo-source-bucket/path/to/*
        destination: C:\myfolder\
        expectedBucketOwner: 123456789022
        overwrite: false
      - source: s3://amzn-s3-demo-source-bucket/path/to/*
        destination: C:\myfolder\
        expectedBucketOwner: 123456789022
        overwrite: true
      - source: s3://amzn-s3-demo-source-bucket/path/to/*
        destination: C:\myfolder\
        expectedBucketOwner: 123456789022
```

Output

None.

S3Upload (Linux, Windows, macOS)

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

Key	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 on 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	String	No	N/A

Key	Description	Type	Required	Default
	ownership of the Amazon S3 bucket specified in the destinati on.			

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://amzn-s3-demo-destination-bucket/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://amzn-s3-demo-destination-bucket/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://amzn-s3-demo-destination-bucket/path/to/
        recurse: true
        expectedBucketOwner: 123456789022
```

Output

None.

WebDownload (Linux, Windows, macOS)

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

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

Key name	Description	Type	Required	Default
`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

Key name	Description	Type	Required	Default
`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`

Key name	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

Key name	Description	Type	Required	Default
`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

Key name	Description	Type
`destination`	Newline character-delimited string that specifies the destinati	String

Key name	Description	Type				
	on path where the downloaded files or resources are stored.					

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 operations modules

The following section contains details for action modules that perform file system operations.

File system operation action modules

- AppendFile (Linux, Windows, macOS)
- CopyFile (Linux, Windows, macOS)
- CopyFolder (Linux, Windows, macOS)
- CreateFile (Linux, Windows, macOS)
- CreateFolder (Linux, Windows, macOS)
- CreateSymlink (Linux, Windows, macOS)
- DeleteFile (Linux, Windows, macOS)
- DeleteFolder (Linux, Windows, macOS)
- ListFiles (Linux, Windows, macOS)
- MoveFile (Linux, Windows, macOS)
- MoveFolder (Linux, Windows, macOS)
- ReadFile (Linux, Windows, macOS)
- SetFileEncoding (Linux, Windows, macOS)
- SetFileOwner (Linux, Windows, macOS)
- SetFolderOwner (Linux, Windows, macOS)
- SetFilePermissions (Linux, Windows, macOS)
- SetFolderPermissions (Linux, Windows, macOS)

AppendFile (Linux, Windows, macOS)

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

Key name	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-1` `utf16-LE`, `utf-16-LE` `utf16-BE`, `utf-16-BE` , `utf32`, `utf-32`, `utf32-LE`,`utf-32-LE` , `utf32-BE`, and `utf-32-`	Yes

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

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 (Linux, Windows, macOS)

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

Key name	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	Boolean	No	`true`	N/A	Yes

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

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 (Linux, Windows, macOS)

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

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`source`	The source folder path.	String	Yes	N/A	N/A	Yes

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`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 (Linux, Windows, macOS)

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

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`path`	The file path.	String	Yes	N/A	N/A	Yes
`content`	The text content of the file.	String	No	N/A	N/A	Yes
`encoding`	The encoding standard.	String	No	`utf8`	`utf8`, `utf-8`, `utf16`,`utf-1` `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	Yes

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
					encoding option is case insensitive.	
`owner`	The user name or ID.	String	No	N/A	N/A	Not supported on Windows.
`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.

Key name	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 file without overwriting (Linux)

```
- name: CreatingFileWithoutOverwriteLinux
  action: CreateFile
  inputs:
    - path: /home/UserName/Sample.txt
      content: The text content of the sample file.
      overwrite: false
```

Input example: create a file without overwriting (Windows)

```
- name: CreatingFileWithoutOverwriteWindows
  action: CreateFile
  inputs:
    - path: C:\Temp\Sample.txt
      content: The text content of the sample file.
```

```
        overwrite: false
```

Input example: create a file with file properties

```
  - name: CreatingFileWithFileProperties
    action: CreateFile
    inputs:
      - path: SampleFolder/Sample.txt
        content: The text content of the sample file.
        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
```

Input example: create an empty file to skip a section in the Linux clean up script

```
  - name: CreateSkipCleanupfile
    action: CreateFile
    inputs:
      - path: <skip section file name>
```

For more information, see Override the Linux clean up script

Output

None.

CreateFolder (Linux, Windows, macOS)

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

Key name	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	N/A	Not supported

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
				current user.		on Windows.
`permissions`	The folder permissions.	String	No	`0777`	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 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 (Linux, Windows, macOS)

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

Key name	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 (Linux, Windows, macOS)

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

Key name	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 (Linux, Windows, macOS)

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

Key name	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 (Linux, Windows, macOS)

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

Key name	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
`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

Key name	Description	Type				
`files`	The list of files.	String				

Output example

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

MoveFile (Linux, Windows, macOS)

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

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

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`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: 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 (Linux, Windows, macOS)

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

Key name	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	Boolean	No	`true`	N/A	Yes

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

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 (Linux, Windows, macOS)

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

Key name	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-1 utf16-LE, utf-16-LE utf16-BE, utf-16-BE , utf32, utf-32, utf32-	Yes

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
					LE,`utf-32-LE` , `utf32-BE`, and `utf-32-BE` . The value of the encoding option is case insensitive.	
`printFileContent`	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 (Linux, Windows, macOS)

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

Key name	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

Input example: set file encoding property

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

Output

None.

SetFileOwner (Linux, Windows, macOS)

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

Key name	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

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
						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 (Linux, Windows, macOS)

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

Key name	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.
`recursive`	Overrides the default behavior of modifying ownership	Boolean	No	`true`	N/A	Not supported on Windows.

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
	for all of the contents of a folder when set to `false`.					

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 (Linux, Windows, macOS)

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

Key name	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 (Linux, Windows, macOS)

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

Key name	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.

Key name	Description	Type	Required	Default value	Acceptable values	Supported on all platforms
`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.

Software installation actions

The following section describes action modules that install or uninstall software.

IAM requirements

If your installation download path is an S3 URI, then the IAM role that you associate with your instance profile must have permission to run the `S3Download` action module. To grant the required permission, attach the `S3:GetObject` IAM policy to the IAM role that is associated with your instance profile, and specify the path for your bucket. For example, `arn:aws:s3:::BucketName/*`).

Complex MSI Inputs

If your input strings contain double quote characters ("), you must use one of the following methods to ensure that they are interpreted correctly:

- You can use single quotes (') on the outside of your string, to contain it, and double quotes (") inside of your string, as shown in the following example.

  ```
  properties:
    COMPANYNAME: '"Acme ""Widgets"" and ""Gizmos."""'
  ```

  In this case, if you need to use an apostrophe inside of your string, you must escape it. This means using another single quote (') before the apostrophe.

- You can use double quotes (") on the outside of your string, to contain it. And you can escape any double quotes inside of your string, using the backslash character (\), as shown in the following example.

  ```
  properties:
    COMPANYNAME: "\"Acme \"\"Widgets\"\" and \"\"Gizmos.\"\"\""
  ```

Both of these methods pass the value `COMPANYNAME="Acme ""Widgets"" and ""Gizmos."""` to the **msiexec** command.

Software installation action modules

- InstallMSI (Windows)
- UninstallMSI (Windows)

InstallMSI (Windows)

The `InstallMSI` action module installs a Windows application using an MSI file. You can specify the MSI file using a local path, an S3 object URI, or a web URL. The reboot option configures the reboot behavior of the system.

AWSTOE generates the **msiexec** command based on the input parameters for the action module. Values for the `path` (MSI file location) and `logFile` (log file location) input parameters must be enclosed in quotation marks (").

The following MSI exit codes are considered successful:

- 0 (Success)
- 1614 (ERROR_PRODUCT_UNINSTALLED)
- 1641 (Reboot Initiated)
- 3010 (Reboot Required)

Input

Key name	Description	Type	Required	Default value	Acceptable values
`path`	Specify the MSI file location using one of the following : • The local file path. The path can be absolute or relative •	String	Yes	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
	A valid S3 object URI. • A valid web HTTP/HTTPS URL (HTTPS is recommended) that follows the RFC 3986 standard. Chaining expressions are allowed.				

Key name	Description	Type	Required	Default value	Acceptable values
`reboot`	Configure the system reboot behavior that follows a successful run of the action module. **Settings:** • `Force` – Initiates a system reboot after the **msiexec** command runs successfully. • `Allow` – Initiates a system reboot if the **msiexec** command returns an exit code that	String	No	`Allow`	`Allow`, `Force`, `Skip`

Key name	Description	Type	Required	Default value	Acceptable values
	indicates a reboot is required. • `Skip` – Logs an informational message to the `console.log` file indicating that a reboot was skipped. This option prevents a reboot, even if the **msiexec** command returns an exit code that indicates a reboot is required.				

Key name	Description	Type	Required	Default value	Acceptable values
`logOptions`	Specify the options to use for MSI installation logging. Specified flags are passed to the MSI installer, along with the `/L` command line parameter to enable logging. If no flags are specified, AWSTOE uses the default value. For more information about log options for MSI, see [Command Line Options](#) in the Microsoft *Windows Installer*	String	No	`*VX`	`i,w,e,a,r,u,c,m,o,p,v,x,+,!,*`

Key name	Description	Type	Required	Default value	Acceptable values
	product documentation.				
`logFile`	An absolute or relative path to the log file location. If the log file path does not exist, it is created. If the log file path is not provided, AWSTOE does not store the MSI installation log.	String	No	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
`properties`	MSI logging property key-value pairs , for example: `TARGETDIR: "C:\target\location"` Note: Modification of the following properties is not allowed: • `REBOOT="ReallySuppress"` • `REINSTALLMODE="ecmus"` • `REINSTALL="ALL"`	Map[String]String	No	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
`ignoreAuthenticodeSignatureErrors`	Flag to ignore authenticode signature validation errors for the installer specified in path. The **Get-AuthenticodeSignature** command is used to validate installers. **Settings:** • `true` – Validation errors are ignored and the installer runs. • `false` – Validation errors are not ignored. The instal	Boolean	No	`false`	`true`, `false`

Key name	Description	Type	Required	Default value	Acceptable values
	ler runs only when validation is successful. This is the default behavior.				

Key name	Description	Type	Required	Default value	Acceptable values
`allowUnsignedInstaller`	Flag to allow running the unsigned installer specified in the path. The **Get-AuthenticodeSignature** command is used to validate installers. **Settings:** • `true` – Ignores the `NotSigned` status returned by the **Get-AuthenticodeSignature** command and runs the installer. • `false` – Requires	Boolean	No	`false`	`true`, `false`

Key name	Description	Type	Required	Default value	Acceptable values
	the instal ler to be signed. Unsigned installers will not run. This is the default behavior.				

Examples

The following examples show variations of the input section for your component document, depending on your installation path.

Input example: local document path installation

```
- name: local-path-install
  steps:
    - name: LocalPathInstaller
      action: InstallMSI
      inputs:
        path: C:\sample.msi
        logFile: C:\msilogs\local-path-install.log
        logOptions: '*VX'
        reboot: Allow
        properties:
          COMPANYNAME: '"Amazon Web Services"'
        ignoreAuthenticodeSignatureErrors: true
        allowUnsignedInstaller: true
```

Input example: Amazon S3 path installation

```
- name: s3-path-install
  steps:
    - name: S3PathInstaller
      action: InstallMSI
```

```
      inputs:
        path: s3://<bucket-name>/sample.msi
        logFile: s3-path-install.log
        reboot: Force
        ignoreAuthenticodeSignatureErrors: false
        allowUnsignedInstaller: true
```

Input example: web path installation

```
- name: web-path-install
  steps:
    - name: WebPathInstaller
      action: InstallMSI
      inputs:
        path: https://<some-path>/sample.msi
        logFile: web-path-install.log
        reboot: Skip
        ignoreAuthenticodeSignatureErrors: true
        allowUnsignedInstaller: false
```

Output

The following is an example of the output from the `InstallMSI` action module.

```
{
 "logFile": "web-path-install.log",
 "msiExitCode": 0,
 "stdout": ""
}
```

UninstallMSI (Windows)

The `UninstallMSI` action module allows you to remove a Windows application using an MSI file. You can specify the MSI file location using a local file path, an S3 object URI, or a web URL. The reboot option configures the reboot behavior of the system.

AWSTOE generates the **msiexec** command based on the input parameters for the action module. The MSI file location (`path`) and log file location (`logFile`) are explicitly enclosed in double quotes (") while generating the **msiexec** command.

The following MSI exit codes are considered successful:

- 0 (Success)
- 1605 (ERROR_UNKNOWN_PRODUCT)
- 1614 (ERROR_PRODUCT_UNINSTALLED)
- 1641 (Reboot Initiated)
- 3010 (Reboot Required)

Input

Key name	Description	Type	Required	Default value	Acceptable values
path	Specify the MSI file location using one of the following : • The local file path. The path can be absolute or relative. • A valid S3 object URI. • A valid web HTTP/ HTTPS URL (HTTPS is recommended) that follows the	String	Yes	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
	RFC 3986 standard. Chaining expressions are allowed.				

Key name	Description	Type	Required	Default value	Acceptable values
`reboot`	Configures the system reboot behavior that follows a successful run of the action module. **Settings:** • `Force` – Initiates a system reboot after the **msiexec** command runs successfully. • `Allow` – Initiates a system reboot if the **msiexec** command returns an exit code that	String	No	`Allow`	`Allow`, `Force`, `Skip`

Key name	Description	Type	Required	Default value	Acceptable values
	indicates a reboot is required. • `Skip` – Logs an informational message to the `console.log` file indicating that a reboot was skipped. This option prevents a reboot, even if the **msiexec** command returns an exit code that indicates a reboot is required.				

Key name	Description	Type	Required	Default value	Acceptable values
`logOptions`	Specify the options to use for MSI installation logging. Specified flags are passed to the MSI installer, along with the `/L` command line parameter to enable logging. If no flags are specified, AWSTOE uses the default value. For more information about log options for MSI, see [Command Line Options](#) in the Microsoft *Windows Installer*	String	No	`*VX`	`i,w,e,a,r,u,c,m,o,p,v,x,+,!,*`

Key name	Description	Type	Required	Default value	Acceptable values
	product documentation.				
`logFile`	An absolute or relative path to the log file location. If the log file path does not exist, it is created. If the log file path is not provided, AWSTOE does not store the MSI installation log.	String	No	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
`properties`	MSI logging property key-value pairs , for example: `TARGETDIR: "C:\target\location"` Note: Modification of the following properties is not allowed: • `REBOOT="ReallySuppress"` • `REINSTALLMODE="ecmus"` • `REINSTALL="ALL"`	Map[String]String	No	N/A	N/A

Key name	Description	Type	Required	Default value	Acceptable values
`ignoreAuthenticodeSignatureErrors`	Flag to ignore authenticode signature validation errors for the installer specified in path. The **Get-AuthenticodeSignature** command is used to validate installers. **Settings:** • `true` – Validation errors are ignored and the installer runs. • `false` – Validation errors are not ignored. The instal	Boolean	No	`false`	`true`, `false`

Key name	Description	Type	Required	Default value	Acceptable values
	ler runs only when validation is successful. This is the default behavior.				

Key name	Description	Type	Required	Default value	Acceptable values
`allowUnsignedInstaller`	Flag to allow running the unsigned installer specified in the path. The **Get-AuthenticodeSignature** command is used to validate installers. **Settings:** • `true` – Ignores the `NotSigned` status returned by the **Get-AuthenticodeSignature** command and runs the installer. • `false` – Requires	Boolean	No	`false`	`true`, `false`

Key name	Description	Type	Required	Default value	Acceptable values
	the installer to be signed. Unsigned installers will not run. This is the default behavior.				

Examples

The following examples show variations of the input section for your component document, depending on your installation path.

Input example: remove local document path installation

```
- name: local-path-uninstall
  steps:
    - name: LocalPathUninstaller
      action: UninstallMSI
      inputs:
        path: C:\sample.msi
        logFile: C:\msilogs\local-path-uninstall.log
        logOptions: '*VX'
        reboot: Allow
        properties:
          COMPANYNAME: '"Amazon Web Services"'
        ignoreAuthenticodeSignatureErrors: true
        allowUnsignedInstaller: true
```

Input example: remove Amazon S3 path installation

```
- name: s3-path-uninstall
  steps:
    - name: S3PathUninstaller
      action: UninstallMSI
```

```
      inputs:
        path: s3://<bucket-name>/sample.msi
        logFile: s3-path-uninstall.log
        reboot: Force
        ignoreAuthenticodeSignatureErrors: false
        allowUnsignedInstaller: true
```

Input example: remove web path installation

```
- name: web-path-uninstall
  steps:
    - name: WebPathUninstaller
      action: UninstallMSI
      inputs:
        path: https://<some-path>/sample.msi
        logFile: web-path-uninstall.log
        reboot: Skip
        ignoreAuthenticodeSignatureErrors: true
        allowUnsignedInstaller: false
```

Output

The following is an example of the output from the `UninstallMSI` action module.

```
{
 "logFile": "web-path-uninstall.log",
 "msiExitCode": 0,
 "stdout": ""
}
```

System action modules

The following section describes action modules that perform system actions or update system settings.

System action modules

- Reboot (Linux, Windows)
- SetRegistry (Windows)
- UpdateOS (Linux, Windows)

Reboot (Linux, Windows)

The **Reboot** action module reboots the instance. It has a configurable option to delay the start of the reboot. By default, `delaySeconds` is set to `0`, which means that there is no delay. Step timeout is not supported for the Reboot action module, as it does not apply when the instance is rebooted.

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 rerun the application after the reboot, and then reboots the system.

Post-reboot auto-run trigger:

- **Windows**. AWSTOE creates a Windows Task Scheduler entry with a trigger that runs automatically at `SystemStartup`
- **Linux**. AWSTOE adds a job in crontab that runs automatically after the system reboots.

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

This trigger is cleaned up when the application starts.

> **Retries**
>
> By default, the maximum number of retries is set to the Systems Manager `CommandRetryLimit`. If the number of reboots exceeds the retry limit, the automation fails. You can change the limit by editing the Systems Manager agent config file (`Mds.CommandRetryLimit`). See [Runtime Configuration] in the Systems Manager agent open source.

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

Key name	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 (Windows)

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

Key name	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

Key name	Description	Type	Required
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 (Linux, Windows)

The **UpdateOS** action module adds support for installing Windows and Linux updates. It installs all available updates by default. Alternatively, you can configure a list of one or more specific updates for the action module to install. You can also specify updates to exclude from the 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.

> **Note**
>
> **UpdateOS** doesn't support Amazon Linux 2023 (AL2023). We recommend that you update your base AMI to the new version that comes with every release. For other alternatives, see Control the updates received from major and minor releases in the *Amazon Linux 2023 User Guide*.

- **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

Key name	Description	Type	Required
`include`	For Windows, you can specify the following: • One or more Microsoft	String List	No

Key name	Description	Type	Required
	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.		

Key name	Description	Type	Required
`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`. • 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.	String List	No

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.

Configure input for the AWSTOE run command

To streamline command line input for your AWSTOE **run** command, you can include settings for command parameters and options in a JSON format input configuration file with a `.json` file extension. AWSTOE can read your file from one of the following locations:

- A local file path (*./config.json*).
- An S3 bucket (*s3://<bucket-path>/<bucket-name>/config.json*).

When you enter the **run** command, you can specify the input configuration file using the **--config** parameter. For example:

```
awstoe run --config <file-path>/config.json
```

Input configuration file

The input configuration JSON file includes key-value pairs for all of the settings that you can provide directly through **run** command parameters and options. If you specify a setting in both the input configuration file and the **run** command, as a parameter or option, the following rules of precedence apply:

Rules of precedence

1. A setting that is supplied directly to the **run** command in the AWS CLI, via a parameter or option, overrides any value that is defined in the input configuration file for the same setting.
2. A setting in the input configuration file overrides a component default value.
3. If no other settings are passed into the component document, it can apply a default value, if one exists.

There are two exceptions to this rule – documents and parameters. These settings work differently in the input configuration and as command parameters. If you use the input configuration file, you must not specify these parameters directly to the **run** command. Doing so will generate an error.

Component settings

The input configuration file contains the following settings. To streamline your file, you can leave out any optional settings that aren't needed. All settings are optional unless otherwise noted.

- **cwIgnoreFailures** (Boolean) – Ignore logging failures from the CloudWatch Logs.
- **cwLogGroup** (String) – The `LogGroup` name for the CloudWatch Logs.
- **cwLogRegion** (String) – The AWS Region that applies to the CloudWatch Logs.
- **cwLogStream** (String) – The `LogStream` name for the CloudWatch Logs, that directs AWSTOE where to stream the `console.log` file.
- **documentS3BucketOwner** (String) – The account ID of the bucket owner for S3 URI-based documents.
- **documents** (array of objects, required) – An array of JSON objects representing the YAML component documents that the AWSTOE **run** command is running. At least one component document must be specified.

  Each object consists of the following fields:

  - **path** (String, required) – The file location of the YAML component document. This must be one of the following:
    - A local file path (*./component-doc-example.yaml*).
    - An S3 URI (`s3://`*bucket/key*).

  - An Image Builder component build version ARN (arn:aws:imagebuilder:us-west-*2*:*123456789012*:component/*my-example-component*/2021.12.02/1).

- **parameters** (array of objects) – An array of key-value pair objects, each representing a component-specific parameter that the **run** command passes in when it runs the component document. Parameters are optional for components. The component document may or may not have parameters defined.

  Each object consists of the following fields:

  - **name** (String, required) – The name of the component parameter.
  - **value** (String, required) – The value to pass in to the component document for the named parameter.

  To learn more about component parameters, see the **Parameters** section in the Use variables in your custom component document page.

- **executonId** (String) – This is the unique ID that applies to the execution of the current **run** command. This ID is included in output and log file names, to uniquely identify those files, and link them to the current command execution. If this setting is left out, AWSTOE generates a GUID.
- **logDirectory** (String) – The destination directory where AWSTOE stores all of the log files from this command execution. By default, this directory is located inside of the following parent directory: `TOE_<DATETIME>_<EXECUTIONID>`. If you do not specify the log directory, AWSTOE uses the current working directory (`.`).
- **logS3BucketName** (String) – If component logs are stored in Amazon S3 (recommended), AWSTOE uploads the component application logs to the S3 bucket named in this parameter.
- **logS3BucketOwner** (String) – If component logs are stored in Amazon S3 (recommended), this is the owner account ID for the bucket where AWSTOE writes the log files.
- **logS3KeyPrefix** (String) – If component logs are stored in Amazon S3 (recommended), this is the S3 object key prefix for the log location in the bucket.
- **parameters** (array of objects) – An array of key-value pair objects that represent parameters that apply globally to all of the components that are included in the current **run** command execution.
  - **name** (String, required) – The name of the global parameter.
  - **value** (String, required) – The value to pass in to all of the component documents for the named parameter.

- **phases** (String) – A comma-separated list that specifies which phases to run from the YAML component documents. If a component document includes additional phases, those will not run.
- **stateDirectory** (String) – The file path where state tracking files are stored.
- **trace** (Boolean) – Enables verbose logging to the console.

Examples

The following example shows an input configuration file that runs the `build` and `test` phases for two component documents: `sampledoc.yaml`, and `conversation-intro.yaml`. Each component document has a parameter that applies only to itself, and both use one shared parameter. The `project` parameter applies to both component documents.

```
{
   "documents": [
     {
       "path": "<file path>/awstoe/sampledoc.yaml>",
       "parameters": [
         {
           "name": "dayofweek",
           "value": "Monday"
         }
       ]
     },
     {
       "path": "<file path>/awstoe/conversation-intro.yaml>",
       "parameters": [
         {
           "name": "greeting",
           "value": "Hello, HAL."
         }
       ]
     }
   ],
   "phases": "build,test",
   "parameters": [
     {
       "name": "project",
       "value": "examples"
     }
   ],
   "cwLogGroup": "<log_group_name>",
```

```
  "cwLogStream": "<log_stream_name>",
  "documentS3BucketOwner": "<owner_aws_account_number>",
  "executionId": "<id_number>",
  "logDirectory": "<local_directory_path>",
  "logS3BucketName": "<bucket_name_for_log_files>",
  "logS3KeyPrefix": "<key_prefix_for_log_files>",
  "logS3BucketOwner": "<owner_aws_account_number>"
}
```

Image Builder output image resources

After you have created image resources for 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 section of this guide.

This section covers how to list, view, and create images. For information about image workflows and how to manage them, see Manage build and test workflows for Image Builder images.

Contents

List images and build versions

On the **Images** page in the Image Builder console, you can see lists of all of the Image Builder image resources that you own, that are shared with you, and that you have access to. The list results include some key details about those resources.

You can also see all of the images in your account that have pending workflow actions.

Contents

- List images
- List images waiting for action
- List image build versions

List images

This section describes the different ways that you can list information about your images.

You can use one of the following methods to list Image Builder image resources that you have access to. For the API action, see ListImages in the *EC2 Image Builder API Reference*. For the associated SDK request, refer to the See Also link on the same page.

Contents

- List images in the console
- List images with AWS CLI commands

List images in the console

To open the **Images** list page in the console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Images** from the navigation pane.

The **Images** page in the console is divided into tabs, based on image ownership or workflow actions that are pending. This section covers the first three tabs that show images that you own or have access to.

Console tab: Owned by me

In the **Owned by me** tab, you can use the following filters to streamline the image list results.

- You can search for all or part of the name in the search bar.
- You can filter images based on their operating system platform (Windows, Linux, or macOS).
- You can filter images based on the type of output they produce (AMI or container image).
- You can use **Filter source** to find images that were imported from a virtual machine (VMIE), or from an ISO disk image.

Following the filter controls, the **Owned by me** tab shows a list of Image Builder images that you created, with the following details for the listed resources:

Name / Version

Image Builder image resource names start with the recipe name and version that they're built from. Select the link to see all of the related image build versions.

Type

The type of output image that Image Builder creates for this image resource (an AMI or a container image).

Platform

The operating system platform of the image resource, for example, "Linux", "Windows", or "macOS".

Image source

The origin of the base image that Image Builder used to build this image resource. This is primarily used to filter results for images that were imported from a virtual machine (**VMIE**).

Creation time

The date and time when Image Builder created the current version of the image resource.

ARN

The Amazon Resource Name (ARN) of the current version of the image resource.

Console tab: Shared with me

In the **Shared with me** tab, you can use the following filters to streamline the image list results.

- You can search for all or part of the name in the search bar.
- You can filter images based on their operating system platform (Windows, Linux, or macOS).
- You can filter images based on the type of output they produce (AMI or container image).
- You can use **Filter source** to find images that were imported from a virtual machine (VMIE), or from an ISO disk image.

Following the filter controls, the **Shared with me** tab shows a list of Image Builder images that were shared with you, with the following details for the listed resources:

Image name

The name of the image resource that was shared with you. To use a shared image in a recipe, you select the **Select managed images** option, and change the **Image origin** to **Images shared with me**.

Type

The type of output image that Image Builder creates for this image resource (an AMI or a container image).

Version

The version of the operating system platform for the image resource, usually a numeric field in the following format: <major>.<minor>.<patch>.

Image source

The origin of the base image that Image Builder used to build this image resource, if applicable. This is primarily used to filter results for images that were imported from a virtual machine (**VMIE**).

Platform

The operating system platform of the image resource, for example, "Linux", "Windows", or "macOS".

Creation time

The date and time when Image Builder created the version of the image resource that was shared with you.

Owner

The owner of the shared image resource.

ARN

The Amazon Resource Name (ARN) of the image resource version that was shared with you.

Console tab: Managed by Amazon

In the **Managed by Amazon** tab, you can use the following filters to streamline the image list results.

- You can search for all or part of the name in the search bar.

- You can filter images based on their operating system platform (Windows, Linux, or macOS).
- You can filter images based on the type of output they produce (AMI or container image).
- You can use **Filter source** to find images that were imported from a virtual machine (VMIE), or from an ISO disk image.

Following the filter controls, the **Managed by Amazon** tab shows a list of Amazon managed Image Builder images that you can use as base images for your recipes. Image Builder displays the following details for listed resources:

Image name

The name of the managed image. When you create a recipe, the default for your base image is **Quick start (Amazon managed)**. The images that are listed in this tab populate the **Image name** list associated with the operating system platform you choose for your base image when you create a recipe.

Type

The type of output image that Image Builder creates for this image resource (an AMI or a container image).

Version

The version of the operating system platform for the image resource, usually a numeric field in the following format: <major>.<minor>.<patch>.

Platform

The operating system platform of the image resource, for example, "Linux", "Windows", or "macOS".

Creation time

The date and time when Image Builder created the version of the image resource that was shared with you.

Owner

Amazon owns the managed images.

ARN

The Amazon Resource Name (ARN) of the image resource version that was shared with you.

List images with AWS CLI commands

When you run the **list-images** command in the AWS CLI, you can get a list of images that you own or have access to.

The following command example shows how to use the **list-images** command without filters to list all of the Image Builder image resources that you own.

Example: list all images

```
aws imagebuilder list-images
```

Output:


```
{
 "requestId": "1abcd234-e567-8fa9-0123-4567b890cd12",
 "imageVersionList": [
  {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-name/1.0.0",
    "name": "image-recipe-name",
    "type": "AMI",
    "version": "1.0.0",
    "platform": "Linux",
    "owner": "123456789012",
    "dateCreated": "2022-04-28T01:38:23.286Z"
  },
  {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-win/1.0.1",
    "name": "image-recipe-win",
    "type": "AMI",
    "version": "1.0.1",
    "platform": "Windows",
    "owner": "123456789012",
    "dateCreated": "2022-04-28T01:38:23.286Z"
  },
  {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-
macos/1.1.1",
    "name": "image-recipe-macos",
    "type": "AMI",
    "version": "1.1.1",
    "platform": "macOS",
    "owner": "123456789012",
```

```
      "dateCreated": "2022-04-28T01:38:23.286Z"
    }
  ]
}
```

When you run the **list-images** command, you can apply filters to streamline the results, as the following example shows. For more information about how to filter your results, see the list-images command in the *AWS CLI Command Reference*.

Example: filter for Linux images

```
aws imagebuilder list-images --filters name="platform",values="Linux"
```

Output:

```
{
 "requestId": "1abcd234-e567-8fa9-0123-4567b890cd12",
 "imageVersionList": [
  {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-name/1.0.0",
    "name": "image-recipe-name",
    "type": "AMI",
    "version": "1.0.0",
    "platform": "Linux",
    "owner": "123456789012",
    "dateCreated": "2022-04-28T01:38:23.286Z"
  }
 ]
}
```

List images waiting for action

When you use the `WaitForAction` step action in your image workflow, it pauses the workflow until you send it a signal to resume processing or fail the workflow. You can use this step action if you have an external process that needs to run before you continue. You can then use the `SendWorkflowStepAction` to send a signal to the paused step to `RESUME` or `STOP`. You can also stop or resume your workflow from the console.

The following tabs show how to get a list of all of the image resources in your account with workflow steps that are currently paused to wait for a signal to resume or stop. The tabs cover console steps and the AWS CLI command.

You can also use the API or an SDK to get a list of workflow steps that are waiting for action. For the API action, see ListWaitingWorkflowSteps in the *EC2 Image Builder API Reference*. For the associated SDK request, refer to the See Also link on the same page.

Console

To get to the **Waiting for action** tab in the console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Images** from the navigation pane. This opens the **Images** list page.
3. Select the **Waiting for action** tab from the list page.
4. (optional) To stop or resume a step, select the check box next to the name, and then choose **Stop step** or **Resume step**. You can select more than one check box to perform the same action for all selected steps.

Pending workflow step details

Workflow details for the pending step include the following:

- **Image name –** The name of the image resource that has the pending step. You can select the name link to display the detail page for that image.
- **Pending step name –** The name of the workflow step that's waiting for action.
- **Step execution Id –** Uniquely identifies the runtime instance of the workflow step. You can select the linked ID to display runtime details for the step.
- **Step start –** The timestamp when the runtime instance of the workflow step started.
- **Workflow ARN –** The Amazon Resource Name (ARN) of the workflow with the pending step.
- **Actions –** The step action that's in a wait state.

AWS CLI

When you run the **list-waiting-workflow-steps** command in the AWS CLI, you'll get a list of all of the images in your account that have workflow steps that are waiting for action before completing the image creation process.

The following command example shows how to use the **list-waiting-workflow-steps** command to list all of the images in your account with workflow steps that are waiting for action.

Example: list images in your account with waiting workflow steps

```
aws imagebuilder list-waiting-workflow-steps
```

Output:

The output for this example shows one image in the account with a step that's waiting for action.

```
{
    "steps": [
        {
            "imageBuildVersionArn": "arn:aws:imagebuilder:us-
west-2:111122223333:image/example-image/1.0.0/8",
            "name": "WaitForAction",
            "workflowExecutionId": "wf-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "stepExecutionId": "step-a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "workflowBuildVersionArn": "arn:aws:imagebuilder:us-
west-2:111122223333:workflow/test/wait-for-action/1.0.0/1",
            "startTime": "2023-11-21T23:21:23.609Z",
            "action": "WaitForAction"
        }
    ]
}
```

List image build versions

On the **Image build versions** page of the Image Builder console, you can see a list of build versions and additional details for an image resource that you own. You can also use commands or actions with the Image Builder API, SDKs, or AWS CLI to list image build versions.

You can use one of the following methods to list image build versions for image resources that you own. For the API action, see ListImageBuildVersions in the *EC2 Image Builder API Reference*. For the associated SDK request, refer to the See Also link on the same page.

Console

Version details

Details on the **Image build versions** page in the Image Builder console include the following:

- **Version –** The image resource build version. In the Image Builder console, the version links to an image detail page.
- **Type –** The type of output that Image Builder distributed when it created this image resource (an AMI or a container image).
- **Date created –** The date and time when Image Builder created the image build version.
- **Image status –** The current status of the image build version. Status can relate to the image build or disposition. For example, during the build process, you might see a status of `Building` or `Distributing`. For disposition of the image, you might see a status of `Deprecated` or `Deleted`.
- **Reason for failure –** The reason for the image status. The Image Builder console only displays the reason when the build fails (**Image status** equals `Failed`).
- **Security findings –** The aggregated image scan findings for the referenced image build version.
- **ARN –** The Amazon Resource Name (ARN) for the referenced version of the image resource.
- **Log stream –** A link to the log stream detail for the referenced image build version.

List versions

To list image build versions in the Image Builder console, perform the following steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Images** from the navigation pane. By default, the image list shows the current version of each of the images that you own.
3. To see a list of all the versions for an image, choose the current version link. The link opens the **Image build versions** page that lists all of the build versions for a specific image.

AWS CLI

When you run the **list-image-build-versions** command in the AWS CLI, you'll get a full list of build versions for the specified image resource. You must own the image to run this command.

The following command example shows how to use the **list-image-build-versions** command to list all of the build versions for the specified image.

Example: list build versions for a specific image

```
aws imagebuilder list-image-build-versions --image-version-arn
 arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-name/1.0.0
```

Output:

The output for this example includes two build versions for the specified image recipe.


```
{
  "requestId": "12f3e45d-67cb-8901-af23-45ed678c9b01",
  "imageSummaryList": [
    {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-
name/1.0.0/2",
    "name": "image-recipe-name",
    "type": "AMI",
    "version": "1.0.0/2",
    "platform": "Linux",
    "osVersion": "Amazon Linux 2",
    "state": {
      "status": "AVAILABLE"
    },
    "owner": "123456789012",
    "dateCreated": "2023-03-10T01:04:40.609Z",
    "outputResources": {
      "amis": [
        {
        "region": "us-west-2",
        "image": "ami-012b3456789012c3d",
        "name": "image-recipe-name 2023-03-10T01-05-12.541Z",
        "description": "First verison of image-recipe-name",
        "accountId": "123456789012"
        }
      ]
    },
    "tags": {}
    },
    {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/image-recipe-
name/1.0.0/1",
    "name": "image-recipe-name",
    "type": "AMI",
    "version": "1.0.0/1",
    "platform": "Linux",
```

```
    "osVersion": "Amazon Linux 2",
    "state": {
      "status": "AVAILABLE"
    },
    "owner": "123456789012",
    "dateCreated": "2023-03-10T00:07:16.384Z",
    "outputResources": {
      "amis": [
        {
        "region": "us-west-2",
        "image": "ami-0d1e23456789f0a12",
        "name": "image-recipe-name 2023-03-10T00-07-18.146132Z",
        "description": "First verison of image-recipe-name",
        "accountId": "123456789012"
        }
      ]
    },
    "tags": {}
    }
  ]
}
```


> **Note**
>
> Output from the **list-image-build-versions** command doesn't include security findings or log streams at this time.

View image resource details

On the image details page in the Image Builder console, you can view details for a specific image resource that you own. You can also use commands or actions with the Image Builder API, SDKs, or AWS CLI to get image details.

For more information about resources that another AWS account shared with you through a AWS Resource Access Manager (AWS RAM) resource share, see Access AWS resources shared with you in the *AWS RAM User Guide*.

Contents

- Get image policy details from the AWS CLI

View image details in the Image Builder console

The image detail page in the Image Builder console includes a summary section, with additional information grouped into tabs. The page heading is the name and build version of the recipe that created the image. If a tab doesn't apply to your image, the tab is inactive and doesn't display data.

Console detail sections and tabs

- Summary section
- Output resources tab
- Infrastructure configuration tab
- Distribution settings tab
- Workflow tab
- Security findings tab
- Tags tab

Summary section

The summary section spans the width of the page and includes the following details. These details are always displayed.

Recipe

The recipe name and version that doesn't include the build version. For example, if the build version is `sample-linux-recipe | 1.0.1/2`, the recipe is `sample-linux-recipe | 1.0.1`, and the build version is 2.

Date created

The date and time when Image Builder created the image build version.

Image status

The current status of the image build version. Status can relate to the image build or disposition. For example, during the build process, you might see a status of `Building` or `Distributing`. For disposition of the image, you might see a status of `Deprecated` or `Deleted`.

Reason for failure

The reason for the image status. The Image Builder console only displays the reason when the build fails (**Image status** equals `Failed`).

Output resources tab

The **Output resources** tab lists output and distribution details for the image resource that is currently displayed. The information that Image Builder displays depends on the type of recipe that the pipeline used to create the image, as follows.

Image recipe

- **Region** – The distribution Region for the output Amazon Machine Image (AMI) that is specified in the **Image** column.
- **Image** – The ID of the AMI that Image Builder distributed to the destination. This ID is linked to the **Amazon Machine Images (AMIs)** page in the Amazon EC2 console.

> **Note**
> Image Builder creates the AMI after it creates the output image resource, and before it distributes the AMI to the destination.

- **Name** – The name of the AMI that Image Builder distributed to the destination.
- **Description** – The optional description from the image recipe that the pipeline used to create the output image resource.
- **Account** – The AWS account that owns the currently displayed Image Builder image resource.

Container recipe

Image Builder displays the following details for output created from a container recipe.

- **Region** – The distribution Region for the container image that is specified in the **Image URI** column.
- **Image URI** – The URI of the output container image that Image Builder distributed to the ECR repository in the destination Region.

> **Note**
>
> Image Builder displays one row per destination. The output image always has at least one entry for distribution to the account that created the image. Additional destinations can include distributions across Regions, AWS accounts, or AWS Organizations. For more information, see Manage Image Builder distribution settings.

Infrastructure configuration tab

The **Infrastructure configuration** tab displays the Amazon EC2 infrastructure settings that Image Builder used to build and test the image that is currently displayed. Image Builder always displays the name of the infrastructure configuration resource (**Configuration name**) and its Amazon Resource Name (**ARN**). If your infrastructure configuration sets the values, additional infrastructure details can include the following

- Instance types
- An instance profile
- Network infrastructure
- Security group settings
- An Amazon S3 location where Image Builder stores application logs
- An Amazon EC2 key pair for troubleshooting
- An Amazon SNS topic for event notifications

For more information, see Manage Image Builder infrastructure configuration.

Distribution settings tab

The **Distribution settings** tab displays settings that Image Builder used to distribute your output images. Image Builder always displays the name of the distribution configuration resource (**Configuration name**) and its Amazon Resource Name (**ARN**). Additional distribution details depend on the type of recipe that the Image Builder pipeline used to create the image, as follows:

Image recipe

If your distribution configuration resource sets the values, additional distribution details can include the following,:

- **Region** – The distribution Region for the output Amazon Machine Image (AMI).
- **Output AMI name** – The name of the AMI that Image Builder distributed to the destination.
- **Encryption (KMS key)** – If configured, the AWS KMS key that Image Builder uses to encrypt the image for distribution to the target Region.
- **Target accounts for distribution** – If you configured cross-account distribution, this column displays a comma-separated list of AWS accounts to share the output image with in the target Region.
- **Principals with shared permission** – A comma-separated list of the AWS principals that have permission to launch your image, for example, AWS accounts or groups, AWS Organizations or organizational units (OUs).

> **Note**
>
> When you grant permission for other principals to launch your image, you still own the image. AWS bills your account for all of the instances that Amazon EC2 launches from your image.

- **Target accounts for faster launch configuration** – The AWS accounts where EC2 Fast Launch distributes pre-provisioned snapshots for launch.
- **Associated license configurations** – The License Manager license configuration ARNs that are associated with the AMI in the specified Region.
- **Launch template configuration** – Identifies an Amazon EC2 launch template to use for a specific account.
- **Set launch template default version** – Sets the specified Amazon EC2 launch template as the default launch template for the specified AWS account.

Container recipe

Container distributions always include the following details:

- **Region** – The distribution Region for the container image specified in the **Image URI** column.
- **Image URI** – The URI of the output container image that Image Builder distributed to the Amazon ECR repository in the destination Region.

> **Note**
>
> Image Builder displays one row per destination. The output image always has at least one entry for distribution to the account that created the image. Additional destinations can include distributions across Regions, AWS accounts, or AWS Organizations. For more information, see Manage Image Builder distribution settings.

Workflow tab

Workflows define the sequence of steps that Image Builder performs when it creates a new image. All images have build and test workflows. Containers have an additional workflow for distribution. The **Workflow** tab displays the applicable workflows that Image Builder ran for your image.

Filter workflow types

Image Builder initially displays the build or import workflow summary and workflow steps by default. However, the **Workflow** filter shows all of the workflows that are in progress or completed for your image. To view a different workflow, select from the list.

Image workflows that produce AMI output can have build, import, or test workflows. Container workflows that produce container output can have build, test, or distribution workflows.

> **Note**
>
> If the workflow hasn't started yet, it doesn't appear in the list. For example, if your image build that has both build and test workflows configured has just started, the build workflow is the only workflow type that appears in the list. When the test workflow begins, Image Builder adds it to the list.

Following the **Workflow** filter, the selected workflow shows a runtime summary that includes the following details for every workflow type:

Workflow status

The current runtime status for this workflow. Values can include the following:

- Pending
- Skipped

- Running
- Completed
- Failed
- Rollback-in-progress
- Rollback-completed

Execution ID

A unique identifier that Image Builder assigns to keep track of runtime resources each time it runs a workflow.

Start

The timestamp when the runtime instance of this workflow started.

End

The timestamp when this runtime instance of the workflow finished.

Total steps

The total number of steps in the workflow. This should equal the sum of the step counts for steps that succeeded, were skipped, and failed.

Steps succeeded

A runtime count for the number of steps in the workflow that ran successfully.

Steps failed

A runtime count for the number of steps in the workflow that failed.

Steps skipped

A runtime count for the number of steps in the workflow that were skipped.

The details in the following list report the current status for all of the steps in this runtime instance of the workflow. Image Builder displays the same details for all image types.

Step #

A number that represents the order in which Image Builder runs the workflow steps.

Step ID

A unique identifier for the workflow step, assigned at runtime.

Step status

The current runtime status of the specified workflow step.

Rollback status

The current rollback status if this runtime instance of the workflow failed.

Step name

The name of the specified workflow step.

Start

The timestamp when the specified step for this runtime instance of the workflow started.

End

The timestamp when the specified step for this runtime instance of the workflow finished.

Security findings tab

If you've activated scanning, the **Security findings** tab displays Common Vulnerabilities and Exposures (CVE) findings. Amazon Inspector identified these findings on the test instance that Image Builder launched to create your new image. To ensure that Image Builder captures findings for your image, you must configure scanning as follows:

1. Activate Amazon Inspector scans for your account. For more information, see Getting started with Amazon Inspector in the *Amazon Inspector User Guide*.
2. Activate security findings for the pipeline that creates this image. When you activate security findings for your pipeline, Image Builder saves a snapshot of the findings before it terminates the test instance. For more information, see Configure security scans for Image Builder images in the AWS Management Console

The **Security findings** tab includes the following details for each vulnerability that Amazon Inspector identified for your image.

Severity

The severity level of the CVE finding. Values are as follows:

- Untriaged
- Informational

- Low
- Medium
- High
- Critical

Finding ID

The unique identifier for the CVE finding that Amazon Inspector detected for your image when it scanned the test instance. The ID is linked to the **Security findings > By vulnerability** page. For more information, see Manage security findings for Image Builder images in the AWS Management Console.

Source

The source of the vulnerability information for the CVE finding.

Age

The number of days since the finding was first observed for your image.

Inspector score

The score that Amazon Inspector assigned for the CVE finding.

Tags tab

The **Tags** tab displays any tags that you have defined for your image.

Get image policy details from the AWS CLI

The following example shows how to get the details of an image policy with its Amazon Resource Name (ARN).

```
aws imagebuilder get-image-policy --image-arn arn:aws:imagebuilder:us-
west-2:123456789012:image/example-image/2019.12.02
```

Create custom images with Image Builder

There are several different ways that you can create a new Image Builder image. For example, you can use one of the following methods to create an image with the AWS Management Console or AWS CLI. You can also use the CreateImage API action or run a build pipeline to create the image.

For the SDK request associated with the API action, you can refer to the See Also link for that command in the *EC2 Image Builder API Reference*.

AWS Management Console

To create a new image from an existing pipeline, you can manually run the pipeline, as follows. You can also use the pipeline wizard to create a new image from scratch. See Pipeline wizard: Create AMI or Pipeline wizard: Create container image, depending on the type of image you want to create.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image pipelines** from the navigation pane.
3. Select the check box next to the **Pipeline name** that you want to run.
4. To create the image, select **Run pipeline** from the **Actions** menu. This starts the pipeline.

You can also specify a schedule to run your pipeline, or use Amazon EventBridge to run your pipeline based on rules that you configure.

AWS CLI

Before you run the **create-image** command in the AWS CLI, you must create the following resources if they don't already exist:

Required resources

- **Recipe** – You must specify exactly one recipe for your image, as follows:

  Image recipe

    Specify the Amazon Resource Name (ARN) for your image recipe resource with the `--image-recipe-arn` parameter.

  Container recipe

    Specify the ARN for your container recipe resource with the `--container-recipe-arn` parameter.
- **Infrastructure configuration** – Specify the ARN for your infrastructure configuration resource with the `--infrastructure-configuration-arn` parameter.

You can also specify any of the following resources that your image requires:

Optional resources and configuration

- **Distribution configuration** – By default, Image Builder distributes the output image resource to your account in the Region where you run the **create-image** command. To provide additional destinations or configuration for your distribution, specify the ARN for your distribution configuration resource with the `--distribution-configuration-arn` parameter.
- **Image scanning** – To configure snapshots for Amazon Inspector findings on your image or container test instance, use the `--image-scanning-configuration` parameter. For container images you also specify the ECR repository that Amazon Inspector uses for its scans.
- **Image tests** – To suppress the Image Builder test stage, use the `--image-tests-configuration` parameter. Alternatively, you can set a timeout for how long it can run.
- **Image tags** – Use the `--tags` parameter to add tags to your output image resource.
- **Image workflows** – If you don't specify any build or test workflows, Image Builder creates your image with its default image workflow. To specify workflows that you've created, use the `--workflows` parameter.

> **Note**
>
> If you specify image workflows, you must also provide the name or ARN of the IAM role that Image Builder uses to run your workflow actions in the `--execution-role` parameter.

The following example shows how to create an image with the create-image AWS CLI command. For more information, see the *AWS CLI Command Reference*.

Example: Create a basic image with default distribution

```
aws imagebuilder create-image --image-recipe-arn arn:aws:imagebuilder:us-west-2:123456789012:image-recipe/simple-recipe-linux/1.0.0 --infrastructure-configuration-arn arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/simple-infra-config-linux
```

Output:

```
{
```

```
"requestId": "1abcd234-e567-8fa9-0123-4567b890cd12",
"imageVersionList": [
    {
        "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/simple-recipe-
linux/1.0.0",
        "name": "simple-recipe-linux",
        ...
    }
]
}
```

Cancel image creation from the AWS CLI

To cancel an in-progress image build, use the **cancel-image-creation** command, as follows:

```
aws imagebuilder cancel-image-creation --image-build-version-arn
 arn:aws:imagebuilder:us-west-2:123456789012:image/my-example-recipe/2019.12.03/1
```

Import and export virtual machine images with Image Builder

When you export a VM from its virtualization environment, that process creates a set of one or more disk container files that act as snapshots of your VM's environment, settings, and data. You can use these files to import your VM, and use it as the base image for your image recipes. To export, you can create VM disk files as output from your custom image build, and distribute the files.

Image Builder supports the following file formats for your VM disk containers:

- Open Virtualization Archive (OVA)
- Virtual Machine Disk (VMDK)
- Virtual Hard Disk (VHD/VHDX)
- Raw

The import uses the disks to create an Amazon Machine Image (AMI) and an Image Builder image resource, either of which can serve as the base image for your custom image recipe. The VM disks must be stored in S3 buckets for the import. Alternatively, you can import from an existing EBS snapshot.

In the Image Builder console, you can import the image directly, and then use the output image or AMI in your recipes, or you can specify import parameters when you are creating your recipe or recipe version. For more information about importing as part of your image recipe, see VM import configuration.

Import a VM into Image Builder

Image Builder integrates with the Amazon EC2 VM Import/Export API to enable the import process to run asynchronously in the background. Image Builder references the task ID from the VM import to track its progress, and creates an Image Builder image resource as output. This allows you to reference the Image Builder image resource in your recipes before the VM import finishes.

Console

To import a VM with the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Images** from the navigation pane.
3. To open the import dialog, choose **Import image**.
4. Enter the following **General** information:
   - Specify a unique **Name** for your image.
   - Specify a **Version** for the base image. Use the following format: *major.minor.patch*.
5. Choose the import type: **VM import**.
6. Provide details for each of the following sections on the **Import image** page. Then choose **Import image** when you're done.

Base image operating system

1. Select the **Image Operating System (OS)** option that matches your VM OS platform.
2. Select the **OS version** that matches the version for your VM from the list.

VM import configuration

1. When you export your VM from its virtualization environment, that process creates a set of one or more disk container files. These act as snapshots of your VM's environment, settings, and data. You can use these files to import your VM as the base image for your image

recipe. For more information about importing VMs in Image Builder, see Import and export VM images.

To specify the location of your import source, follow these steps:

Import source

Specify the source for the first VM image disk container or snapshot to import in the **Disk container 1** section.

a. **Source** – This can be either an S3 bucket, or an EBS snapshot.
b. **Select S3 location of disk** – Enter the location in Amazon S3 where your disk images are stored. To browse for the location, choose **Browse S3**.
c. To add a disk container, choose **Add disk container**.

2. **IAM role**

   To associate an IAM role with your VM import configuration, select the role from the **IAM role** dropdown list, or choose **Create new role** to create a new one. If you create a new role, the IAM Roles console page opens in a separate tab.

3. **Advanced settings – *optional***

   The following settings are optional. With these settings, you can configure encryption, licensing, tags, and more for the base image that the import creates.

   Base image architecture

   To specify the architecture of your VM import source, select a value from the **Architecture** list.

   Encryption

If your VM disk images are encrypted, you must provide a key to use for the import process. To specify a KMS key for the import, select a value from the **Encryption (KMS key)** list. The list contains KMS keys that your account has access to in the current Region.

License management

When you import a VM, the import process automatically detects the VM OS and applies the appropriate license to the base image. Depending on your OS platform, the license types are as follows:

- **License included** – An appropriate AWS license for your platform is applied to your base image.
- **Bring your own license (BYOL)** – Retains the license from your VM, if applicable.

To attach license configurations created with AWS License Manager to your base image, select from the **License configuration name** list. For more information about License Manager, see [Working with AWS License Manager]

> **Note**
>
> - License configurations contain licensing rules based on the terms of your enterprise agreements.
> - Linux only supports BYOL licenses.

Tags (base image)

Tags use key-value pairs to assign searchable text to your Image Builder resource. To specify tags for the imported base image, enter key-value pairs using the **Key** and **Value** boxes.

To add a tag, choose **Add tag**. To remove a tag, choose **Remove tag**.

AWS CLI

To import a VM from disks into an AMI and create an Image Builder image resource that you can reference right away, follow these steps from the AWS CLI:

1. Initiate a VM import, with the Amazon EC2 VM Import/Export **import-image** command in the AWS CLI. Make note of the task ID that is returned in the command response. You'll need it for the next step. For more information, see [Importing a VM as an image using VM Import/Export](#) in the *VM Import/Export User Guide*.

2. **Create a CLI input JSON file**

   To streamline the Image Builder **import-vm-image** command that is used in the AWS CLI, we create a JSON file that contains all of the import configuration that we want to pass into the command.

   > **Note**
   >
   > The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API operation request parameters, see the [ImportVmImage](#) operation in the *EC2 Image Builder API Reference*.
   > To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*. to the Image Builder **import-vm-image** command as options.

   Here is a summary of the parameters that we specify in this example:

   - **name** (string, required) – The name for the Image Builder image resource to create as output from the import.
   - **semanticVersion** (string, required) – The semantic version for the output image that 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 in Image Builder](#).
   - **description** (string) – The description of the image recipe.
   - **platform** (string, required) – The operating system platform for the imported VM.
   - **vmImportTaskId** (string, required) – The `ImportTaskId` (AWS CLI) from the Amazon EC2 VM import process. Image Builder monitors the import process to pull in the AMI

that it creates and build an Image Builder image resource that can be used in recipes right away.

- **tags** (string map) – Tags are key-value pairs that are attached to the import resources. Up to 50 key-value pairs are allowed.

Save the file as `import-vm-image.json`, to use in the Image Builder **import-vm-image** command.

```
{
    "name": "example-request",
    "semanticVersion": "1.0.0",
    "description": "vm-import-test",
    "platform": "Linux",
    "vmImportTaskId": "import-ami-01ab234567890cd1e",
    "tags": {
     "Usage": "VMIE"
    }
}
```

3. **Import the image**

Run the [import-vm-image](#) command, with the file that you created as input:

```
aws imagebuilder import-vm-image --cli-input-json file://import-vm-image.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, while Linux and macOS use the forward slash (/).

Distribute VM disks from your image build from the AWS CLI

You can set up distribution of supported VM disk format files to S3 buckets in target Regions as part of your regular image build process, using Image Builder distribution configurations in the

AWS CLI. For more information, see Example: Create distribution settings for output VM disks from the AWS CLI.

Import verified Windows ISO disk images with Image Builder

A Windows operating system ISO file is a disk image file that contains the complete installation package for a specific version of the Windows operating system. Microsoft provides official Windows operating system ISO files for download, either directly from their website or through authorized resellers. It's important to ensure that you obtain the ISO files from a trusted and legitimate source to avoid potential malware or unauthorized versions.

EC2 Image Builder uses the `build-image-from-iso` import workflow to import the ISO disk file and create a secondary volume from it. After configuration is complete, Image Builder takes a snapshot of the volume it created from the import and uses it to create an Amazon Machine Image (AMI).

Supported operating systems for ISO disk image import

Image Builder supports the following Windows operating system ISO disk images:

- Windows 11 Enterprise version 24H2 (x64)
- Windows 11 Enterprise version 23H2 (x64)
- Windows 11 Enterprise version 22H2 (x64)

Image Builder does not support the following Windows operating system ISO disk images:

- Long-Term Servicing Channel (LTSC) images
- ISO disk images created from the Windows Media Creation Tool
- Evaluation images

Prerequisites to import an ISO disk image

To import an ISO disk image, you must first meet the following prerequisites:

- The operating system of the disk image must be one that Image Builder supports. For a list of supported operating systems, see Supported operating systems for ISO disk image import.
- To ensure that you can import your ISO image, download it from the Microsoft 365 admin center.

- Before you can run the import process, you must upload your ISO disk file to Amazon S3 in the same AWS account and AWS Region where the import runs.
- The file extension is case sensitive for the import process, and must be `.ISO`. If your file extension is lowercase, you can run one of the following commands to rename it:

  Command

  ```
  aws s3 cp s3://amzn-s3-demo-bucket/Win11_24H2_English.iso s3://amzn-s3-demo-bucket/Win11_24H2_English.ISO
  ```

  PowerShell

  ```
  Copy-S3Object -BucketName amzn-s3-demo-bucket -Key Win11_24H2_English.iso -DestinationKey Win11_24H2_English.ISO
  ```
- Microsoft licensing is not automatically included with the import. You must bring your own license (BYOL). For more information about licensing for Microsoft software, see Licensing on the *Amazon Web Services and Microsoft Frequently Asked Questions* page.
- The import process uses two separate IAM roles, as follows:

  Execution role

  This role grants permission for Image Builder to call AWS services on your behalf. You can specify the AWSServiceRoleForImageBuilder service-linked role, which includes the permissions needed for the execution role, or you can create your own role.

  Instance profile role

  This role grants permission for the actions that the service performs on the EC2 instance. You can specify an instance profile role in your infrastructure configuration resource. Attach the following managed policies to your instance profile role to ensure that you have all of the permissions needed for the import process.

  - EC2InstanceProfileForImageBuilder
  - AmazonSSMManagedInstanceCore

  For more information, see Manage Image Builder infrastructure configuration.

Import an ISO disk image into Image Builder

Before you start the import process, make sure that you've met all of the Prerequisites.

The import process additionally installs the following software and drivers on your image:

- EC2Launch v2
- AWS Systems Manager agent
- AWS NVMe driver
- AWS ENA network driver
- AWS PCI Serial Driver
- EC2 Windows utilities

Console

To import an ISO disk image with the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Images** from the navigation pane.
3. To open the import dialog, choose **Import image**.
4. Enter the following **General** information:
   - Specify a unique **Name** for your image.
   - Specify a **Version** for the base image. Use the following format: *major.minor.patch*.
5. Choose the import type: **ISO import**.
6. Enter the following **ISO import configuration** details. Then choose **Import image** when you're done.
   - **S3 URI –** Enter the location where your ISO disk file is stored. To browse for the file, choose **Browse S3**.
   - **IAM role –** To associate an IAM role with your import configuration, select the role from the **IAM role** dropdown list, or choose **Create new role** to create a new one. If you create a new role, the IAM Roles console page opens in a separate tab.

     You can specify the AWSServiceRoleForImageBuilder service-linked role, or you can specify your own custom role for service access.
7. You can optionally add tags to your Image Builder image resource. This does not add the tags to your AMI.

8. The **ISO infrastructure configuration** defines settings for the instance that Image Builder launches to host the import process. You can use an infrastructure configuration that Image Builder creates, based on service defaults, or you can use an existing infrastructure configuration. For more information, see Manage Image Builder infrastructure configuration.

   To create a new infrastructure configuration , choose **Create infrastructure configuration**. This opens in a separate tab. When you're done creating the new resource, you can return to the import configuration, and choose **Use existing infrastructure configuration**.

9. To start the import process, choose **Import image**.

After the import is complete, your image appears in the list of images that you own. For more details, see List images.

AWS CLI

This example shows how to import an image from an ISO disk file and create an AMI from it with the AWS CLI.

Here is a summary of the parameters that we specify in this example:

- **name** (string, required) – The name for the Image Builder image resource to create as output from the import.
- **semanticVersion** (string, required) – The semantic version for the output image that 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 in Image Builder.
- **description** (string) – The description of the image recipe.
- **executionRole** (string) – The name or Amazon Resource Name (ARN) for the IAM role that grants Image Builder access to perform workflow actions to import an image from a Microsoft ISO file. You can specify the AWSServiceRoleForImageBuilder service-linked role, or you can specify your own custom role for service access.
- **platform** (string, required) – The operating system platform for the ISO disk image. Valid values include `Windows`.
- **osVersion** (string, required) – The operating system version for the ISO disk image. Valid values include `Microsoft Windows 11`.

- **infrastructureConfigurationArn** (string, required) – The Amazon Resource Name (ARN) of the infrastructure configuration resource that's used for launching the EC2 instance on which the ISO image is built.
- **uri** (string, required) – The URI of the ISO disk file that's stored in Amazon S3.

```
aws imagebuilder import-disk-image \
    --name "example-iso-disk-import" \
    --semantic-version "1.0.0" \
    --description "Import an ISO disk image" \
    --execution-role "AWSServiceRoleForImageBuilder" \
    --platform "Windows" \
    --os-version "Microsoft Windows 11" \
    --infrastructure-configuration-arn "arn:aws:imagebuilder:us-
east-1:111122223333:infrastructure-configuration/example-infrastructure-
configuration-123456789abc",
    --uri: "s3://amzn-s3-demo-source-bucket/examplefile.iso"
```

After the import is complete, your image appears in the list of images that you own. For more details, see List images.

PowerShell

This example shows how to import an image from an ISO disk file and create an AMI from it with PowerShell.

Here is a summary of the parameters that we specify in this example:

- **name** (string, required) – The name for the Image Builder image resource to create as output from the import.
- **semanticVersion** (string, required) – The semantic version for the output image that 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 in Image Builder.
- **description** (string) – The description of the image recipe.
- **executionRole** (string) – The name or Amazon Resource Name (ARN) for the IAM role that grants Image Builder access to perform workflow actions to import an image from a Microsoft ISO file. You can specify the AWSServiceRoleForImageBuilder service-linked role, or you can specify your own custom role for service access.

- **platform** (string, required) – The operating system platform for the ISO disk image. Valid values include `Windows`.
- **osVersion** (string, required) – The operating system version for the ISO disk image. Valid values include `Microsoft Windows 11`.
- **infrastructureConfigurationArn** (string, required) – The Amazon Resource Name (ARN) of the infrastructure configuration resource that's used for launching the EC2 instance on which the ISO image is built.
- **uri** (string, required) – The URI of the ISO disk file that's stored in Amazon S3.

```
Import-EC2IBDiskImage `
    -Name "example-iso-disk-import" `
    -SemanticVersion "1.0.0" `
    -Description "Import an ISO disk image" `
    -ExecutionRole "AWSServiceRoleForImageBuilder" `
    -Platform "Windows" `
    -OsVersion "Microsoft Windows 11" `
    -InfrastructureConfigurationArn "arn:aws:imagebuilder:us-
east-1:111122223333:infrastructure-configuration/example-infrastructure-
configuration-123456789abc" `
    -Uri "s3://amzn-s3-demo-source-bucket/examplefile.ISO"
```

After the import is complete, your image appears in the list of images that you own. For more details, see List images.

Launch an instance from the output AMI

When you launch an instance from an AMI that the import process creates, the Windows operating system runs Sysprep Specialize, which automatically downloads and installs EC2Launch v2 and the Systems Manager Agent from public S3 endpoints. These endpoints require public internet access. If you launch an instance from a private subnet, the Sysprep Specialize process is unable to access the S3 endpoints, and the launch fails.

Manage security findings for Image Builder images

When you activate security scanning with Amazon Inspector, it continuously scans machine images and running instances in your account for operating system and programming language vulnerabilities. If activated, security scanning is automatic, and Image Builder can save a snapshot of the findings from your test instance when you create a new image. Amazon Inspector is a paid service.

When Amazon Inspector discovers vulnerabilities in your software or network settings, it takes the following actions:

- Notifies you that there was a finding.
- Rates the severity of the finding. The severity rating categorizes vulnerabilities to help you prioritize your findings, and includes the following values:
  - Untriaged
  - Informational
  - Low
  - Medium
  - High
  - Critical
- Provides information about the finding, and links to additional resources for more detail.
- Offers remediation guidance to help you resolve the issues that generated the finding.

Configure security scans for Image Builder images in the AWS Management Console

If you've activated Amazon Inspector for your account, Amazon Inspector automatically scans the EC2 instances that Image Builder launches to build and test a new image. Those instances have a short lifespan during the build and test process, and their findings would normally expire as soon as those instances shut down. To help you investigate and remediate findings for your new image, Image Builder can optionally save any findings that Amazon Inspector identified on your test instance during the build process as a snapshot.

Step 1: Activate Amazon Inspector security scans for your account

To activate Amazon Inspector security scans for your account from the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Security scanning settings** from the navigation pane. This opens the **Security scanning** dialog box.

The dialog box displays the scanning status for your account. If Amazon Inspector is already activated for your account, the status shows **Enabled**.

3. Follow steps 1 and 2 of the instructions to activate Amazon Inspector scanning.

> **Note**
>
> Amazon Inspector incurs charges. For more information, see Amazon Inspector pricing.

If you've activated scanning for your pipeline, Image Builder takes a snapshot of the findings for your build instance when you create a new image. This way, you can access the findings after Image Builder terminates the build instance.

Step 2: Configure your pipeline to save snapshots for vulnerability findings

To configure vulnerability finding snapshots for your pipeline, perform the following steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image pipelines** from the navigation pane.
3. Pick one of the following methods to specify pipeline details:

   Create a new pipeline

   1. From the **Image pipelines** page, choose **Create image pipeline**. This opens the **Specify pipeline details** page in the pipeline wizard.

   Update an existing pipeline

   1. From the **Image pipelines** page, choose the **Pipeline name** link for the pipeline that you want to update. This opens the pipeline detail page.

      > **Note**
      >
      > Alternatively, you can select the check box next to the name of the pipeline that you want to update, and then choose **View details**.

   2. From the pipeline details page, select **Edit pipeline** from the **Action** menu. This takes you to the **Edit pipeline** page.

4. In the **General** section from the pipeline wizard or the **Edit pipeline** page, select the **Enable security scan** check box.

> **Note**
>
> If you want to turn off the snapshots later, you can edit your pipeline to clear the check box. This doesn't deactivate Amazon Inspector scanning for your account. To deactivate Amazon Inspector scanning, see Deactivating Amazon Inspector in the *Amazon Inspector User Guide*.

Manage security findings for Image Builder images in the AWS Management Console

The **Security findings** list pages display high-level information about the findings for your resources, with views based on several different filters that you can apply. Each view includes the following options at the top to change your view:

- **All security findings** – This is the default view if you choose **Security findings** page from the navigation pane in the Image Builder console.
- **By vulnerability** – This view shows a high-level list of all of the image resources in your account that have findings. The **Finding ID** is linked to more detailed information about the finding. This information appears on a panel that opens on the right side of the page. The panel includes the following information:
  - A detailed description of the finding.
  - A **Finding details** tab. This tab includes a finding overview, affected packages, summary remediation advice, vulnerability details, and related vulnerabilities. The **Vulnerability ID** links to detailed vulnerability information in the National Vulnerability Database.
  - A **Score breakdown** tab. This tab includes a side-by-side comparison of the CVSS and Amazon Inspector scores so that you can see where Amazon Inspector has modified a score, if applicable.
- **By image pipeline** – This view shows the number of findings for each image pipeline in your account. Image Builder displays counts for medium severity and higher findings, plus a total for all findings. All data in the list is linked, as follows:
  - The **Image pipeline name** column links to the detail page for the specified image pipeline.

  - The severity level column links open the **All security findings** view, filtered by the associated image pipeline name and severity level.

  You can also use search criteria to refine your results.

- **By image** – This view shows the number of findings for each image build in your account. Image Builder displays counts for medium severity and higher findings, plus a total for all findings. All data in the list is linked, as follows:
  - The **Image name** column links to the image detail page for the specified image build. For more information, see View image resource details.
  - The severity level column links open the **All security findings** view, filtered by the associated image build name and severity level.

  You can also use search criteria to refine your results.

Image Builder shows the following details in the **Findings** list section of the default **All security findings** view.

Severity

The severity level of the CVE finding. Values are as follows:

- Untriaged
- Informational
- Low
- Medium
- High
- Critical

Finding ID

The unique identifier for the CVE finding that Amazon Inspector detected for your image when it scanned the build instance. The ID is linked to the **Security findings > By vulnerability** page.

Image ARN

The Amazon Resource Name (ARN) for the image with the finding specified in the **Finding ID** column.

Pipeline

The pipeline that built the image specified in the **Image ARN** column.

Description

A short description of the finding.

Inspector score

The score that Amazon Inspector assigned for the CVE finding.

Remediation

Links to details about the recommended course of action to remediate the finding.

Published date

The date and time when this vulnerability was first added to the vendor's database.

Clean up Image Builder 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 outdated or unused Image Builder resources.

Manage lifecycle policies for Image Builder images

When you create custom images, it's important that you have a plan to retire those images before they become obsolete. Image Builder pipelines can apply updates and security patches automatically. However, each build creates a new version of the image and all of the associated resources that it distributes. Earlier versions remain in your account until you manually delete them, or create a script to perform the task.

With Image Builder lifecycle management policies, you can automate the process of deprecating, disabling, and deleting outdated images and their associated resources. Associated resources can include output images that you've distributed to other AWS accounts, organizations, and organizational units (OUs) across AWS Regions. You define the rules for how and when to take each step in the lifecycle process, and which steps to include in your policy.

Benefits of automated lifecycle management

Overall benefits of automated lifecycle management include the following:

- Simplifies lifecycle management for your custom images with an automated way to retire images and associated resources.
- Helps to prevent compliance risks that come from using outdated images to launch new instances.
- Keeps image inventories fresh by removing outdated images.
- Can reduce storage and data transfer costs by optionally removing associated resources for images that are deleted.

Realize cost savings

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. When you remove unused or outdated images and their associated resources from your AWS account, you can realize time and cost savings in the following ways:

- Reduce the time it takes to patch existing images when you're not also patching unused or outdated images.
- For AMI image resources that you delete, you can choose to also remove distributed AMIs and their associated snapshots. This approach can save on the cost of storing snapshots.

- For container image resources that you delete, you can choose to delete underlying resources. This approach can save on Amazon ECR storage costs and data transfer rates for your Docker images that are stored in ECR repositories.

> **Note**
>
> Image Builder can't evaluate the potential impact for all possible downstream dependencies, such as Auto Scaling groups or launch templates. You must consider downstream dependencies for your images when you configure policy actions.

Contents

Lifecycle management prerequisites for Image Builder images

Before you can define EC2 Image Builder lifecycle management policies and rules for your image resources, you must meet the following prerequisites.

- Create an IAM role that grants permission for Image Builder to run lifecycle policies. To create the role, see Create an IAM role for Image Builder lifecycle management.
- Create an IAM role in the destination account for associated resources that were distributed across accounts. The role grants permission for Image Builder to perform lifecycle actions in the destination account for associated resources. To create the role, see Create an IAM role for Image Builder cross-account lifecycle management.

  > **Note**
  >
  > This prerequisite doesn't apply if you've granted launch permissions for an output AMI. With launch permissions, the account you shared with owns the instances that are launched from the shared AMI, but all of the AMI resources remain in your account.

- For container images, you must add the following tag to your ECR repositories to grant access for Image Builder to run lifecycle actions on the container images stored in the repository: `LifecycleExecutionAccess: EC2 Image Builder`.

Create an IAM role for Image Builder lifecycle management

To grant permission for Image Builder to run lifecycle policies, you must first create the IAM role that it uses to perform lifecycle actions. Follow these steps to create the service role that grants permission.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** from the navigation pane.
3. Choose **Create role**. This opens to the first step in the process **Select trusted entity** to create your role.
4. Select the **Custom trust policy** option for the **Trusted entity type**.
5. Copy the following JSON trust policy and paste it into the **Custom trust policy** text area, replacing the sample text. This trust policy allows Image Builder to assume the role that you create to run lifecycle actions.

   JSON

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

6. Select the following managed policy from the list: **EC2ImageBuilderLifecycleExecutionPolicy**, then choose **Next**. This opens the **Name, review, and create** page.

> **Tip**
>
> Filter on `image` to streamline results.

7. Enter a **Role name**.
8. After you've reviewed your settings, choose **Create role**.

Create an IAM role for Image Builder cross-account lifecycle management

To grant permission for Image Builder to perform lifecycle actions in destination accounts for associated resources, you must first create the IAM role that it uses to perform lifecycle actions in those accounts. You must create the role in the destination account.

Follow these steps to create the service role that grants permission *in the destination account*.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** from the navigation pane.
3. Choose **Create role**. This opens to the first step in the process **Select trusted entity** to create your role.
4. Select the **Custom trust policy** option for the **Trusted entity type**.
5. Copy the following JSON trust policy and paste it into the **Custom trust policy** text area, replacing the sample text. This trust policy allows Image Builder to assume the role that you create to run lifecycle actions.

> **Note**
>
> When Image Builder uses this role in the destination account to act on associated resources that were distributed across accounts, it's acting on behalf of the destination account owner. The AWS account that you configure as the `aws:SourceAccount` in the trust policy is the account where Image Builder distributed those resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "imagebuilder.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "444455556666"
                },
                "StringLike": {
                    "aws:SourceArn": "arn:*:imagebuilder:*:*:image/*/*/*"
                }
            }
        }
    ]
}
```

6. Select the following managed policy from the list: **EC2ImageBuilderLifecycleExecutionPolicy**, then choose **Next**. This opens the **Name, review, and create** page.

> **Tip**
>
> Filter on `image` to streamline results.

7. Enter `Ec2ImageBuilderCrossAccountLifecycleAccess` as the **Role name**.

> **Important**
>
> `Ec2ImageBuilderCrossAccountLifecycleAccess` must be the name of this role.

8. After you've reviewed your settings, choose **Create role**.

List lifecycle management policies for Image Builder image resources

You can get a list of your image lifecycle management policies that includes key detail columns on the **Lifecycle policies** list page in the AWS Management Console, or with commands or actions in the Image Builder API, SDKs, or AWS CLI.

You can use one of the following methods to list Image Builder image lifecycle policy resources in your AWS account. For the API action, see ListLifecyclePolicies in the *EC2 Image Builder API Reference*. For the associated SDK request, refer to the See Also link on the same page.

AWS Management Console

The following details are shown in the console for your existing policies. You can select any column to change the sort order for your results. The policy list is initially sorted by **Policy name**. The column name for the current sort order is bold.

If you have more than one page of results, the paging arrows in the top right corner of the panel become active. You can filter results by policy name, policy status, output image type and image resource ARN with the search bar.

- **Policy name –** The name of the policy.
- **Policy status –** Whether the policy is active or inactive.
- **Type –** The type of output image that Image Builder distributes when you create a new image version (AMI or container image).
- **Last execution date –** The last time the lifecycle policy ran.
- **Date created –** The timestamp from the creation of the lifecycle policy.
- **ARN –** The Amazon Resource Name (ARN) of the lifecycle policy resource.

To list lifecycle policies in the AWS Management Console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Select **Lifecycle policies** from the navigation pane. This shows a list of image lifecycle policies in your account.

Available actions

You can also perform the following actions for your lifecycle policy from the **Lifecycle policies** list page.

To create a new image lifecycle policy, choose **Create lifecycle policy**. For more information about how to create a policy, see Create lifecycle policies.

For all of the following actions, you must select the policy first. To select a policy, you can select the check box next to the **Policy name**.

- To turn the policy off or on, select **Disable policy** or **Enable policy** from the **Actions** menu.
- To change the policy, select **Edit policy** from the **Actions** menu.
- To delete a policy, select **Delete policy** from the **Actions** menu.
- To create a new policy that uses your selected policy for baseline settings, select **Clone policy** from the **Actions** menu.

AWS CLI

The following command example shows how to use the AWS CLI to list image lifecycle policies for a specific AWS Region. For more information about the parameters and options that you can use with this command, see the list-lifecycle-policies command in the AWS CLI Command Reference.

Example:

```
aws imagebuilder list-lifecycle-policies \
--region us-west-1
```

Output:

```
{
    "lifecyclePolicySummaryList": [
        {
            "arn": "arn:aws:imagebuilder:us-west-2:111122223333:lifecycle-policy/
sample-lifecycle-policy1",
            "name": "sample-lifecycle-policy1",
            "status": "DISABLED",
            "executionRole": "arn:aws:iam::111122223333:role/sample-lifecycle-role",
            "resourceType": "AMI_IMAGE",
            "dateCreated": "2023-11-07T14:57:01.603000-08:00",
```

```
            "tags": {}
        },
        {
            "arn": "arn:aws:imagebuilder:us-west-2:111122223333:lifecycle-policy/
sample-lifecycle-policy2",
            "name": "sample-lifecycle-policy2",
            "status": "ENABLED",
            "executionRole": "arn:aws:iam::111122223333:role/sample-lifecycle-role",
            "resourceType": "AMI_IMAGE",
            "dateCreated": "2023-09-06T10:43:21.436000-07:00",
            "dateLastRun": "2023-11-13T04:43:46.106000-08:00",
            "tags": {}
        },
        {
            "arn": "arn:aws:imagebuilder:us-west-2:111122223333:lifecycle-policy/
sample-lifecycle-policy3",
            "name": "sample-lifecycle-policy3",
            "status": "ENABLED",
            "executionRole": "arn:aws:iam::111122223333:role/sample-lifecycle-role",
            "resourceType": "AMI_IMAGE",
            "dateCreated": "2023-10-19T15:16:40.046000-07:00",
            "dateUpdated": "2023-10-21T20:07:15.958000-07:00",
            "dateLastRun": "2023-11-12T09:27:45.830000-08:00"
}]}
```

Note

To use your default AWS Region, run this command without the `--region` parameter.

View lifecycle policy details

The lifecycle policy detail page in the Image Builder console includes a summary section, with additional information grouped into tabs. The page heading is the name of the policy.

On the lifecycle policy details page in the Image Builder console, you can view details for a specific lifecycle policy. You can also use commands or actions with the Image Builder API, SDKs, or AWS CLI to get policy details.

Contents

View lifecycle policy details in the Image Builder console

The image detail page in the Image Builder console includes a summary section, with additional information grouped into tabs. The page heading is the name and build version of the recipe that created the image.

Console detail sections and tabs

- Summary section
- Rules tab
- Scope tab
- RunLog tab

Summary section

The summary section spans the width of the page and includes the following details. These details are always displayed.

Policy status

Whether the policy is active or inactive.

Type

The type of output image that Image Builder distributes when you create a new image version (AMI or container image).

Date created

The timestamp from the creation of the lifecycle policy.

Date modified

The last time the lifecycle policy was updated.

Last run date

The last time the lifecycle policy ran.

IAM role

The IAM role that Image Builder uses to perform lifecycle actions.

ARN

The Amazon Resource Name (ARN) of the lifecycle policy resource.

Description

The description for the lifecycle policy, if entered.

Rules tab

The **Rules** tab displays the lifecycle rules that you configured for the policy you're viewing. The tab includes the following details:

- **Name** – The name of the rule. These names are static, based on policy actions you can configure.
  - `Deprecation rule`
  - `Disable rule`
  - `Deletion rule`
- **Rule** – A short description of the action that's configured for the rule.
- **Rule conditions** – Lists configuration for associated resource handling, exceptions to the rule, and retention settings, if applicable.

For more information about rule configuration, see How lifecycle rules work.

Scope tab

The **Scope** tab displays the resource selection criteria that are configured for the policy you're viewing. The tab includes the following details:

- **Filter: *`type of filter`*** – The filter type you used to define the scope. The filter type can be one of the following:
  - `recipes` – The recipes that were used to create the images that the lifecycle policy applies to.
  - `tags` – A set of tags that Image Builder uses to select image resources that the lifecycle policy applies to.
- A search bar – You can filter the list by **Name** to streamline results that display in the tab.
- **Name** – Each row contains a name or tag that you've configured for the filter criteria.
- **Version** – If you've configured a recipe filter, Image Builder displays the recipe version.

RunLog tab

Each time you run the policy for your configured resources, Image Builder saves runtime details. Each row in the table represents a single runtime instance. The tab includes the following details:

- **Execution ID** – Identifies the lifecycle policy runtime instance.
- **Execution status** – Runtime status that reports if the policy action is currently running, ran successfully, failed, or was canceled.
- **Resource impacted** – Indicates whether the runtime instance identified any image resources for lifecycle actions.
- **Start date** – The timestamp when the runtime instance started.
- **End date** – The timestamp when the runtime instance ended.

Create lifecycle policies

When you create a new EC2 Image Builder lifecycle policy, the configuration depends on what kind of image the policy is for. The API action to create a lifecycle policy for AMI image resources and container image resources is the same (CreateLifecyclePolicy). However, the configuration for the image resources and associated resources is different. This section shows you how to create lifecycle management policies for both.

> **Note**
>
> Before you create a lifecycle policy, make sure that you've met all Prerequisites.

Create lifecycle management policies for Image Builder AMI image resources

You can use one of the following methods to create an AMI image lifecycle policy with the AWS Management Console or AWS CLI. You can also use the CreateLifecyclePolicy API action. For the associated SDK request, you can refer to the See Also link for that command in the *EC2 Image Builder API Reference*.

AWS Management Console

To create a lifecycle policy for AMI image resources in the AWS Management Console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Lifecycle policies** from the navigation pane.
3. Choose **Create lifecycle policy**.
4. Configure policy settings described in the following procedures.
5. To create the lifecycle policy after you've configured settings, choose **Create policy**.

Configure **General** settings for your policy.

1. Select the **AMI** option from **Policy type**.
2. Enter the **Policy name**.
3. Optionally enter a **Description** for your lifecycle policy.
4. By default, **Activate** is turned on. The default setting activates the lifecycle policy and adds it to the schedule right away. To create a policy that's initially deactivated, you can turn **Activate** off.
5. Select the **IAM role** that you created for lifecycle policy permissions. If you haven't created this role yet, see Prerequisites for more information.

Configure the **Rule scope** for your policy.

This section configures the resource selection for your lifecycle policy, based on the type of filter that you use.

1. **Filter type: Recipes –** To apply lifecycle rules to image resources based on the recipe that created them, select up to 50 recipe versions for the policy.
2. **Filter type: Tags –** To apply lifecycle rules to image resources based on resource tags, enter a list of up to 50 key value pairs for the policy to match on.

Turn on one or more of the following **Lifecycle rules** to apply to the resources that the lifecycle policy selects. If a resource matches on more than one lifecycle rule when the policy runs, Image Builder performs rule actions in the following order: 1) Deprecate, 2) Disable, 3) Delete.

Deprecate rule

Sets the Image Builder image resource status to `Deprecated`. Image Builder pipelines still run for deprecated images. You can optionally set the deprecation time for associated AMIs without affecting your ability to launch new instances.

- **Unit count –** Specify the integer value for the period of time that must pass after an image resource is created before it's marked as `Deprecated`.
- **Unit –** Select the time range to use. The range can be `Days`, `Weeks`, `Months`, or `Years`.
- **Deprecate AMIs –** Select the checkbox to mark associated Amazon EC2 AMIs with a deprecation date. The AMIs remain available, and you can still launch new instances from them.

Disable rule

Sets the Image Builder image resource status to `Disabled`. This prevents Image Builder pipelines from running for this image. You can optionally disable the associated AMI to prevent new instance launches.

- **Unit count –** Specify the integer value for the period of time that must pass after an image resource is created before it's marked as `Disabled`.
- **Unit –** Select the time range to use. The range can be `Days`, `Weeks`, `Months`, or `Years`.
- **Disable AMIs –** Select the checkbox to disable associated Amazon EC2 AMIs. You can no longer use the AMIs or launch new instances from them.

Delete rule

Deletes the image resources by age or by count. You define the threshold that meets your needs. When an Image Builder image resource passes the threshold, it's removed. You can optionally deregister associated AMIs or delete the snapshots for those AMIs. You can also specify tags for resources that you want to retain past the threshold.

When you configure the **Delete rule** by age, Image Builder deletes the image resource after a period of time that you configure. For example, delete image resources after 6 months. When you configure by count, Image Builder retains the most recent number of images that you specify, or as close to that number as possible, and deletes earlier versions.

- **By age**
  - **Unit count –** Specify the integer value for the period of time that must pass after an image resource is created before it's deleted.
  - **Unit –** Select the time range to use. The range can be `Days`, `Weeks`, `Months`, or `Years`.
  - **Retain at least one image per recipe –** Select the check box to keep the latest available image resource for each recipe version that this rule affects.

  By count
  - **Image count –** Specify the integer value for the number of recent image resources to keep for each recipe version.
- **Deregister AMIs –** Select the check box to deregister associated Amazon EC2 AMIs. You can no longer use the AMIs or launch new instances from them.
- **Retain images, AMIs, and snapshots with associated tags –** Select the checkbox to enter a list of tags for image resources that you want to keep. Tags apply to image resources and Amazon EC2 AMIs. You can enter up to 50 key value pairs.

Tags (optional)

Add tags to your lifecycle policy.

AWS CLI

To create a new Image Builder lifecle policy, you can use the **create-lifecycle-policy** command in the AWS CLI.

Create lifecycle management policies for Image Builder container image resources

You can use one of the following methods to create a container image lifecycle policy with the AWS Management Console or AWS CLI. You can also use the CreateLifecyclePolicy API action. For the associated SDK request, you can refer to the See Also link for that command in the *EC2 Image Builder API Reference*.

AWS Management Console

To create a lifecycle policy for container image resources in the AWS Management Console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Lifecycle policies** from the navigation pane.
3. Choose **Create lifecycle policy**.
4. Configure policy settings described in the following procedures.
5. To create the lifecycle policy after you've configured settings, choose **Create policy**.

Policy configuration: General settings

Configure **General** settings for your policy.

1. Select the **AMI** option from **Policy type**.
2. Enter the **Policy name**.
3. Optionally enter a **Description** for your lifecycle policy.
4. By default, **Activate** is turned on. The default setting activates the lifecycle policy and adds it to the schedule right away. To create a policy that's initially deactivated, you can turn **Activate** off.
5. Select the **IAM role** that you created for lifecycle policy permissions. If you haven't created this role yet, see Prerequisites for more information.

Configure the **Rule scope** for your policy.

This section configures the resource selection for your lifecycle policy, based on the type of filter that you use.

1. **Filter type: Recipes –** To apply lifecycle rules to image resources based on the recipe that created them, select up to 50 recipe versions for the policy.
2. **Filter type: Tags –** To apply lifecycle rules to image resources based on resource tags, enter a list of up to 50 key value pairs for the policy to match on.

Delete rule

For container images, this rule deletes the Image Builder container image resource. You can optionally remove Docker images that were distributed to ECR repositories to prevent them from being used to run new containers.

When you configure the **Delete rule** by age, Image Builder deletes the image resource after a period of time that you configure. For example, delete image resources after 6 months. When you configure by count, Image Builder retains the most recent number of images that you specify, or as close to that number as possible, and deletes earlier versions.

- **By age**
  - **Unit count** – Specify the integer value for the period of time that must pass after an image resource is created before it's deleted.
  - **Unit** – Select the time range to use. The range can be `Days`, `Weeks`, `Months`, or `Years`.
  - **Retain at least one image** – Select the checkbox to keep only the latest available image resource for each recipe version that this rule affects.

  By count
  - **Image count** – Specify the integer value for the number of recent image resources to keep for each recipe version.
- **Delete ECR container images** – Select the check box to delete associated container images stored in an ECR repository. You can no longer use the container image as a base to create new images, or to run new containers.
- **Retain images with associated tags** – Select the checkbox to enter a list of tags for image resources that you want to keep.

Tags (optional)

Add tags to your lifecycle policy.

AWS CLI

To create a new Image Builder lifecle policy, you can use the **create-lifecycle-policy** command in the AWS CLI.

How lifecycle management rules work for Image Builder image resources

Image lifecycle policies use the lifecycle rules that you define to implement your overall resource management strategy. The rules that you define help ensure the freshness of your available images and minimize costs for underlying infrastructure such as snapshot storage for output AMIs, or ECR repository storage and data transfer rates for container images.

You can configure the following types of rules for your policies.

Deprecate rule

Sets the Image Builder image resource status to `Deprecated`. Image Builder pipelines still run for deprecated images. You can optionally set the deprecation time for associated AMIs without affecting your ability to launch new instances.

When an AMI is deprecated, it's ignored by general searches. For example, if you run the Amazon EC2 **describe-images** command in the AWS CLI, it would not return deprecated AMIs in the result set. However, you can still find deprecated AMIs with their AMI ID.

This rule is not available for container images.

Disable rule

Sets the Image Builder image resource status to `Disabled`. This prevents Image Builder pipelines from running for this image. You can optionally disable the associated AMI to prevent new instance launches.

When an AMI is disabled, it becomes private and can't be used to launch new instances. If you shared the AMI with any accounts, organizations, or organizational units, they lose access to your AMI when it becomes private.

This rule is not available for container images.

Delete rule

Deletes the image resources by age or by count. You define the threshold that meets your needs. When an Image Builder image resource passes the threshold, it's removed. You can optionally deregister associated AMIs or delete the snapshots for those AMIs. You can also specify tags for resources that you want to retain past the threshold.

For container images, this rule deletes the Image Builder container image resource. You can optionally remove container images that were distributed to ECR repositories to prevent them from being used to run new containers.

Contents

- View lifecycle management rule details for a policy

AMI lifecycle exclusion rules

The following exclusion rules define exceptions to the lifecycle rules for AMIs. AMIs that meet the criteria specified by the exclusion rules are excluded from lifecycle actions. Exclusion rules are not available in the AWS Management Console.

The following terms use API notation from the `LifecyclePolicyDetailExclusionRules` data type.

Exclusion rules

amis

Contains the settings in `LifecyclePolicyDetailExclusionRulesAmis` shown in the list that follows.

tagMap

You can provide a list of up to 50 tags that skip lifecycle actions for any type of resource.

The following terms use API notation from the `LifecyclePolicyDetailExclusionRulesAmis` data type.

AMI exclusion rules

isPublic

Configures whether public AMIs are excluded from the lifecycle action.

lastLaunched

Specifies configuration details for Image Builder to exclude the most recent resources from lifecycle actions.

regions

Configures AWS Regions that are excluded from the lifecycle action.

sharedAccounts

Specifies AWS accounts whose resources are excluded from the lifecycle action.

tagMap

Lists tags that should be excluded from lifecycle actions for the AMIs that have them.

View lifecycle management rule details for a policy

Rules are defined within the lifecycle management policies that you create for your Image Builder image resources. In the console, the lifecycle policy details page has a Rules tab that shows the details of the rules that you configured for the policy.

To get policy details in the AWS CLI, you can run the get-lifecycle-policy command. The policy details in the response contain a list of the actions (rules) that you defined for the policy, that include all of your configured settings.

Configure custom images with Image Builder

Configuration 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 section of this guide.

Contents

Manage recipes in Image Builder

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 works as expected. Image Builder provides automatic version choices for each component. By default, you can apply up to 20 components to a recipe. This includes both build and test components.

After you create a recipe, you can't modify or replace it. To update components after you create a recipe, you must create a new recipe or recipe version. You can always apply tags to your existing recipes. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources section of this guide.

> **Tip**
>
> You can use Amazon managed components in your recipes, or you can develop your own custom components. For more information, see Develop custom components for your Image Builder image. For image recipes that create output AMIs, you can also use AWS Marketplace image products and components. For more information about integration with AWS Marketplace products, see AWS Marketplace integration in Image Builder.

This section covers how to list, view, and create recipes.

Contents

List and view image recipe details

This section describes the various ways that you can find information and view details for your EC2 Image Builder image recipes.

Image recipe details

List image recipes from the 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 from the 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 from the 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 from the console.

On the recipe detail page, you can:

- Delete the recipe. For more information about deleting resources in Image Builder, see Delete outdated or unused Image Builder resources.
- 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 Tutorial: Create an image pipeline with output AMI from the Image Builder console wizard

> **Note**
>
> When you create a pipeline from an existing recipe, the option to create a new recipe is not available.

Get image recipe details from the 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 from the 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 ways that you can find information and view details for your EC2 Image Builder container recipes.

Container recipe details

- List container recipes in the console
- List container recipes with the AWS CLI
- View container recipe details in the console
- Get container recipe details with the AWS CLI
- Get container recipe policy details with the AWS CLI

List container recipes in the console

To see a list of the container recipes that have been 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**, and then choose **View details**.

List container recipes with the 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 in the console

To view details for a specific container recipe with the Image Builder console, select the container recipe to review, and use the steps described in List container recipes in the console.

On the recipe detail page, you can do the following:

- Delete the recipe. For more information on how to delete resources in Image Builder, see Delete outdated or unused Image Builder resources.
- Create a new version.
- Create a pipeline from the recipe. After ou choose **Create pipeline from this recipe**, you are taken to the pipeline wizard. For more information on how to create an Image Builder pipeline using the pipeline wizard, see Tutorial: Create an image pipeline with output AMI from the Image Builder console wizard

> **Note**
>
> When you create a pipeline from an existing recipe, the option to create a new recipe isn't available.

Get container recipe details with the 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 with the 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 a new version of an image recipe

This section describes how to create a new version of an image recipe.

Contents

Create a new image recipe version from the console

When you create a new recipe version, it's 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. This base detail isn't pre-filled with the current version or any kind of a sequence. Enter the version number that you want to create in the format *<major>.<minor>.<patch>*. If the version already exists, you 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 that you chose.

  To see details that are associated with your base image selection, choose the tab that matches your selection.

  Managed image

  - **Image Operating System (OS)** – *Not editable*.
  - **Image name** – Pre-selected, based on the combination of base image choices that 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 image option defaults to the **Use selected OS version** option.

    > **Important**
    >
    > If you're using semantic versioning to kick off pipeline builds, make sure that you change this value to **Use latest available OS version**. To learn more about semantic versioning for Image Builder resources, see Semantic versioning in Image Builder.

  AWS Marketplace image

  - **Subscriptions** – This tab should be open, and the subscribed image from AWS Marketplace should be pre-selected to match your base recipe. If you change the image that your recipe uses as its base image, you might lose other details that depend on the original image that you chose.

    For more information about AWS Marketplace products, see Buying products in the *AWS Marketplace Buyer Guide*.

  Custom AMI

    AMI source (Required) - Enter the AMI ID or an AWS Systems Manager (SSM) Parameter Store parameter that contains an AMI ID to use as the base image. The SSM Agent must be pre-installed in the selected AMI.

  - **AMI ID** – This setting is not pre-filled with your original entry. Enter the AMI ID for your base image. Example: *`ami-1234567890abcdef1`*.

    - **SSM parameter** – Enter the name or ARN of the SSM Parameter Store parameter that contains the AMI ID for your base image. Example: `/ib/test/param` or `arn:aws:ssm:us-east-1:111122223333:parameter/ib/test/param`.

- **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, select the check box so that the agent isn't included in your AMI.
    - **User data** – You can use this area to provide commands, or a command script to run, when you launch your build instance. However, this value replaces any commands that Image Builder might have added to ensure that Systems Manager is installed. These commands include the clean-up script that Image Builder normally runs for Linux images prior to creating the new image.

      When Image Builder launches an instance, user data scripts run during the cloud-init phase, before component execution begins. This step is logged to the following file on the instance: `var/log/cloud-init.log`.

      Note

      - 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.
      - For Linux images, ensure that clean-up steps run by including a command to create an empty file named `perform_cleanup` in your user data script. Image Builder detects this file, and runs the clean-up script prior to creating the new image. For more information and a sample script, see Security best practices for Image Builder.

- **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.

  CIS hardening components don't follow the standard component ordering rules in Image Builder recipes. The CIS hardening components always run last to ensure that the benchmark tests run against your output image.

> **Note**
>
> Build and test component lists display available components based on the component owner type. To add a component, choose **Add build components**, and select the ownership filter that applies. For example, to add a build component that's associated with an AWS Marketplace product, select `AWS Marketplace`. This opens a selection panel on the right side of the console interface that lists AWS Marketplace components. For the CIS component, select `Third party managed`.

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 input parameter, the default value appears in the **Value** box with greyed-out text. If no other value is entered, Image Builder uses the default value.

  If an input parameter is required, but doesn't have a default value defined in the component, you must provide a value. Image Builder won't create the recipe version if there are any required parameters that are missing and don't have a default value defined.

  > **Important**
  >
  > Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets Manager, see [What is Secrets Manager?](#) in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see [AWS Systems Manager Parameter Store](#) in the *AWS Systems Manager User Guide*.

To expand settings for **Versioning options** or **Input parameters**, you can choose the arrow next to the name of the setting. To expand all of the settings for all selected components, you can toggle the **Expand all** switch off and on.

- **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, and encrypt new or existing volumes.

  To encrypt volumes for the images that Image Builder creates under your account in the source Region (where the build runs), you must use the storage volume encryption in the image recipe. Encryption that runs during the distribution phase of the build is only for images that are distributed to other accounts or Regions.

  > **Note**
  >
  > If you use encryption for your volumes, you must select the key for each volume separately, even if the key is the same one that's used for the root volume.

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, and then choose **Create recipe**.

   Your final image can contain up to four product codes from AWS Marketplace image products and components. If your selected base image and components contain more than four product codes, Image Builder returns an error when you try to create the recipe.

For more information on how to create an image recipe when you create an image pipeline, see Step 2: Choose recipe in the **Get started** section of this guide.

Create an image recipe with the AWS CLI

To create an image recipe with the Image Builder `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 from the AWS CLI, you must create the components that the recipe uses. The image recipe example in the following step refers to example components that are created in the Create a custom component from the AWS CLI section of this guide.

After you create your components, or if you are using existing components, note the ARNs that you want to include in the recipe.

1. **Create a CLI input JSON file**

   You can provide all of the input for the **create-image-recipe** command with inline command parameters. However, the resulting command can be quite long. To streamline the command, you can instead provide a JSON file that contains all of the recipe settings.

   > **Note**
   >
   > The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API operation request parameters, see the CreateImageRecipe command in the *EC2 Image Builder API Reference*.
   > To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*.

   Here is a summary of the parameters that these examples specify:

   - **name** (string, required) – The name of the image recipe.
   - **description** (string) – The description of the image recipe.
   - **parentImage** (string, required) – The image that the image recipe uses as a base for your customized image. You can specify the parent image using one of the following options:
     - AMI ID
     - Image Builder image resource ARN
     - AWS Systems Manager (SSM) Parameter Store parameter, prefixed by `ssm:`, followed by the parameter name or ARN.
     - AWS Marketplace product ID

> **Note**
>
> The Linux and macOS examples use an Image Builder AMI, and the Windows example uses an ARN.

- **semanticVersion** (string, required) – The semantic version of the image recipe, expressed in the following format, with numeric values in each position to indicate a specific version: *<major>.<minor>.<patch>*. For example a value might be `1.0.0`. To learn more about semantic versioning for Image Builder resources, see Semantic versioning in Image Builder.
- **components** (array, required) – Contains an array of `ComponentConfiguration` objects. At least one build component must be specified:

> **Note**
>
> Image Builder installs components in the order that you specified them in the recipe. However, CIS hardening components always run last to ensure that the benchmark tests run against your output image.

  - **componentARN** (string, required) – The component ARN.

    > **Tip**
    >
    > 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.

  - **parameters** (array of objects) – Contains an array of `ComponentParameter` objects. If an input parameter is required, but doesn't have a default value defined in the component, you must provide a value. Image Builder won't create the recipe version if there are any required parameters that are missing and don't have a default value defined.

    > **Important**
    >
    > Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets

Manager, see What is Secrets Manager? in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

- **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 option is set to `true`, then the agent is removed from the final image. If the option is 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 isn't 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 isn't available in the AMI:

    - The `userDataOverride` is empty or has been omitted from the JSON file.
    - Image Builder automatically installed the Systems Manager agent on the build instance for an operating system that didn't have the agent pre-installed on the base 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 `IyEvYmluL2Jhc2gKbWtkaXIgLXAgL3Zhci9iYi8KdG91Y2ggL3Zhcg==`:

```
#!/bin/bash
mkdir -p /var/bb/
touch /var
```

The Linux example uses this encoded value.

Linux

The base image (`parentImage` property) in the following example is an AMI. When you use an AMI, you must have access to the AMI, and the AMI must be in the source Region (the same Region where Image Builder runs the command). Save the file as `create-image-recipe.json`, and use it in the **create-image-recipe** command.

```
{
"name": "BB Ubuntu Image recipe",
"description": "Hello World image recipe for Linux.",
"parentImage": "ami-1234567890abcdef1",
"semanticVersion": "1.0.0",
"components": [
 {
  "componentArn": "arn:aws:imagebuilder:us-west-2:111122223333:component/bb$"
 }
],
"additionalInstanceConfiguration": {
 "systemsManagerAgent": {
   "uninstallAfterBuild": true
 },
 "userDataOverride": "IyEvYmluL2Jhc2gKbWtkaXIgLXAgL3Zhci9iYi8KdG91Y2ggL3Zhcg=="
}
}
```

Windows

The following example refers to the latest version of the Windows Server 2016 English Full Base image. The ARN in this example references the latest image based on the semantic version filters that you've 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-2:111122223333:component/my-
example-component/2019.12.02/1"
 },
 {
  "componentArn": "arn:aws:imagebuilder:us-west-2:111122223333:component/my-
imported-component/1.0.0/1"
 }
]
}
```

Note

To learn more about semantic versioning for Image Builder resources, see [Semantic versioning in Image Builder](#).

macOS

The base image (`parentImage` property) in the following example is an AMI. When you use an AMI, you must have access to the AMI, and the AMI must be in the source Region (the same Region where Image Builder runs the command). Save the file as `create-image-recipe.json`, and use it in the **create-image-recipe** command.

```
{
"name": "macOS Catalina Image recipe",
"description": "Hello World image recipe for macOS.",
"parentImage": "ami-1234567890abcdef1",
"semanticVersion": "1.0.0",
"components": [
 {
  "componentArn": "arn:aws:imagebuilder:us-
west-2:111122223333:component/catalina$"
```

```
    }
  ],
  "additionalInstanceConfiguration": {
    "systemsManagerAgent": {
      "uninstallAfterBuild": true
    },
    "userDataOverride": "IyEvYmluL2Jhc2gKbWtkaXIgLXAgL3Zhci9iYi8KdG91Y2ggL3Zhcg=="
  }
}
```

2. **Create the recipe**

   Use the following command to create the recipe. Provide the name of the JSON file that you created in the prior step in the `--cli-input-json` parameter:

   ```
   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, while Linux and macOS use the forward slash (/).

   Your final image can contain up to four product codes from AWS Marketplace image products and components. If your selected base image and components contain more than four product codes, Image Builder returns an error when run the `create-image-recipe` command.

Import a VM as your base image in the console

In this section, we focus on how to import a virtual machine (VM) as the base image for your image recipe. We don't cover other steps involved with creating a recipe or recipe version here. For additional steps to create a new image recipe with the pipeline creation wizard in the Image Builder console, see Pipeline wizard: Create AMI. For additional steps to create a new image recipe or recipe version, see Create a new version of an image recipe.

To import a VM as the base image for your image recipe in the Image Builder console, follow these steps, along with any other required steps, to create your recipe or recipe version.

1. In the **Select image** section for the base image, select the **Import base image** option.
2. Choose the **Image Operating System (OS)** and **OS version** as you normally would.

VM import configuration

When you export your VM from its virtualization environment, that process creates a set of one or more disk container files that act as snapshots of your VM environment, settings, and data. You can use these files to import your VM as the base image for your image recipe. For more information about importing VMs in Image Builder, see Import and export VM images

To specify the location of your import source, follow these steps:

Import source

Specify the source for the first VM image disk container or snapshot to import in the **Disk container 1** section.

1. **Source** – This can be either an S3 bucket or an EBS snapshot.
2. **Select S3 location of disk** – Enter the location in Amazon S3 where your disk images are stored. To browse for the location, choose **Browse S3**.
3. To add a disk container, choose **Add disk container**.

IAM role

To associate an IAM role with your VM import configuration, select the role from the **IAM role** dropdown list, or choose **Create new role** to create a new one. If you create a new role, the IAM Roles console page opens in a separate tab.

Advanced settings – *optional*

The following settings are optional. With these settings, you can configure encryption, licensing, tags, and more for the base image that the import creates.

General

1. Specify a unique **Name** for the base image. If you do not enter a value, the base image inherits the recipe name.

2. Specify a **Version** for the base image. Use the following format: *<major>.<minor>.<patch>*. If you do not enter a value, the base image inherits the recipe version.

3. You can also enter a **Description** for the base image.

Base image architecture

To specify the architecture of your VM import source, select a value from the **Architecture** list.

Encryption

If your VM disk images are encrypted, you must provide a key to use for the import process. To specify an AWS KMS key for the import, select a value from the **Encryption (KMS key)** list. The list contains KMS keys that your account has access to in the current Region.

License management

When you import a VM, the import process automatically detects the VM OS and applies the appropriate license to the base image. Depending on your OS platform, the license types are as follows:

- **License included** – An appropriate AWS license for your platform is applied to your base image.
- **Bring your own license (BYOL)** – Retains the license from your VM, if applicable.

To attach license configurations created with AWS License Manager to your base image, select from the **License configuration name** list. For more information about License Manager, see [Working with AWS License Manager](#)

> **Note**
>
> - License configurations contain licensing rules based on the terms of your enterprise agreements.
> - Linux only supports BYOL licenses.

Tags (base image)

Tags use key-value pairs to assign searchable text to your Image Builder resource. To specify tags for the imported base image, enter key-value pairs with the **Key** and **Value** boxes.

To add a tag, choose **Add tag**. To remove a tag, choose **Remove tag**.

Create a new version of a container recipe

This section shows you how to create a new version of a container recipe.

Contents

Create a new container recipe version with the console

Creating a new version of a container recipe 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. This detail isn't pre-filled with the current version or any kind of a sequence. Enter the version number that you want to create in the format *major.minor.patch*. If the version already exists, you encounter an error.

Base image

- **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 that you chose.

  For Docker container images, you can choose from public images hosted on DockerHub, existing container images in Amazon ECR, or Amazon-managed container images. 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 that 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. Auto-versioning options defaults to the **Use selected OS version** option.

> **Important**
>
> If you're using semantic versioning to kick off pipeline builds, make sure that you change this value to **Use latest available OS version**. To learn more about semantic versioning for Image Builder resources, see Semantic versioning in Image Builder.

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 source** (Required) – Identify a custom AMI to use as the base image for your container build and test instance. This can be an AMI ID or an AWS Systems Manager (SSM) Parameter Store parameter that contains an AMI ID.
  - **AMI ID** – This setting is not pre-filled with your original entry. Enter the AMI ID for your base image. Example: `ami-1234567890abcdef1`.
  - **SSM parameter** – Enter the name or ARN of the SSM Parameter Store parameter that contains the AMI ID for your base image. Example: `/ib/test/param` or `arn:aws:ssm:us-east-1:111122223333:parameter/ib/test/param`.
- **Storage (volumes)**

EBS volume 1 (AMI root) – Pre-filled. You can't edit the root volume **Device name**, **Snapshot**, or **IOPS** selections. However, you can change all of the remaining settings, such as the **Size**. You can also add new volumes.

> **Note**
>
> If you specified a base AMI that was shared with you from another account, the snapshots for any secondary volumes that are specified must also be shared with your account.

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.

  CIS hardening components don't follow the standard component ordering rules in Image Builder recipes. The CIS hardening components always run last to ensure that the benchmark tests run against your output image.

  > **Note**
  >
  > Build and test component lists display available components based on the component owner type. To add a component, choose **Add build components**, and select the ownership filter that applies. For example, to add a build component that's associated with an AWS Marketplace product, select `AWS Marketplace`. This opens a selection panel on the right side of the console interface that lists AWS Marketplace components. For the CIS component, select `Third party managed`.

  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 input parameter, the default value appears in the **Value** box with greyed-out text. If no other value is entered, Image Builder uses the default value.

  If an input parameter is required, but doesn't have a default value defined in the component, you must provide a value. Image Builder won't create the recipe version if there are any required parameters that are missing and don't have a default value defined.

  > **Important**
  >
  > Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets Manager, see [What is Secrets Manager?](#) in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see [AWS Systems Manager Parameter Store](#) in the *AWS Systems Manager User Guide*.

To expand settings for **Versioning options** or **Input parameters**, you can choose the arrow next to the name of the setting. To expand all of the settings for all selected components, you can toggle the **Expand all** switch off and on.

Dockerfile template

- **Dockerfile template –** Pre-filled, but editable. You can specify any of the following contextual variables that Image Builder replaces with build information at runtime.

parentImage (required)

At build time, this variable resolves to the base image for your recipe.

Example:

```
FROM
{{{ imagebuilder:parentImage }}}
```

environments (required if components are specified)

This variable will resolves to a script that runs components.

Example:

```
{{{ imagebuilder:environments }}}
```

components (optional)

Image Builder resolves build and test component scripts for the components that the container recipe includes. This variable can be placed anywhere in the Dockerfile, after the environments variable.

Example:

```
{{{ imagebuilder:components }}}
```

Target repository

- **Target repository name** – The Amazon ECR repository where your output image is stored if there is no other repository specified in your pipeline's distribution configuration for the Region where the pipeline runs (Region 1).

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, and then choose **Create recipe**.

For more information on how to create a container recipe when you create an image pipeline, see Step 2: Choose recipe in the **Get started** section of this guide.

Create a container recipe with the AWS CLI

To create an Image Builder container recipe with 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 with the AWS CLI, you must create 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 custom component from the AWS CLI section of this guide.

After you create your components, or if you are using existing components, note the ARNs that you want to include in the recipe.

1. **Create a CLI input JSON file**

   You can provide all of the input for the **create-container-recipe** command with inline command parameters. However, the resulting command can be quite long. To streamline the command, you can instead provide a JSON file that contains all of the container recipe settings

   > **Note**
   >
   > The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API operation request parameters, see the CreateContainerRecipe command in the *EC2 Image Builder API Reference*.
   >
   > To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*.

   Here is a summary of the parameters in this example:

   - **components** (array of objects, required) – Contains an array of `ComponentConfiguration` objects. At least one build component must be specified:

> **Note**
>
> Image Builder installs components in the order that you specified them in the recipe. However, CIS hardening components always run last to ensure that the benchmark tests run against your output image.

- **componentARN** (string, required) – The component ARN.

  > **Tip**
  >
  > To use the example to create your own container recipe, replace the example ARNs with the ARNs for the components that you are using for your recipe,. These include the AWS Region, name, and the version number for each.

- **parameters** (array of objects) – Contains an array of `ComponentParameter` objects. If an input parameter is required, but doesn't have a default value defined in the component, you must provide a value. Image Builder won't create the recipe version if there are any required parameters that are missing and don't have a default value defined.

  > **Important**
  >
  > Component parameters are plain text values, and are logged in AWS CloudTrail. We recommend that you use AWS Secrets Manager or the AWS Systems Manager Parameter Store to store your secrets. For more information about Secrets Manager, see What is Secrets Manager? in the *AWS Secrets Manager User Guide*. For more information about AWS Systems Manager Parameter Store, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

  - **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, expressed 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 Docker container image to use in the container recipe as a baseline for your customized image.
  - Public images hosted on DockerHub
  - Existing container images in Amazon ECR
  - Amazon-managed container images
- **platformOverride** (string) – Specifies the operating system platform when you use a custom base image.
- **semanticVersion** (string, required) – The semantic version of the container recipe specified in the following format, with numeric values in each position to indicate a specific version: *<major>.<minor>.<patch>*. An example would be `1.0.0`. To learn more about semantic versioning for Image Builder resources, see Semantic versioning in Image Builder.
- **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 base image for a container build and test instance. This can contain an AMI ID or it can specify an AWS Systems Manager (SSM) Parameter Store parameter, prefixed by `ssm:`, followed by the parameter name or ARN. If you use an SSM parameter, the parameter value must contain an AMI ID. If you don't specify a base image, Image Builder uses 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 that these mappings apply to.
    - **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 if there is no other repository specified in your pipeline's distribution configuration for the Region where the pipeline runs (Region 1).
  - **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-west-2:111122223333:component/
helloworldal2/x.x.x"
   }
 ],
 "containerType": "DOCKER",
 "description": "My Linux Docker container image",
 "dockerfileTemplateData": "FROM
 {{{ imagebuilder:parentImage }}}\n{{{ imagebuilder:environments }}}\n{{{ imagebuilder:comp
 "name": "amazonlinux-container-recipe",
 "parentImage": "amazonlinux:latest",
 "platformOverride": "Linux",
 "semanticVersion": "1.0.2",
 "tags": {
   "sometag" : "Tag detail"
 },
 "instanceConfiguration": {
   "image": "ami-1234567890abcdef1",
   "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. Provide the name of the JSON file that you created in the prior step in the `--cli-input-json` parameter:

   ```
   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, while Linux and macOS use 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 outdated or unused Image Builder resources.

Manage Image Builder infrastructure configuration

You can use infrastructure configurations to specify the Amazon EC2 infrastructure that Image Builder uses to build and test your EC2 Image Builder image. Infrastructure settings include:

- Instance types for your build and test infrastructure. We recommend that you specify more than one instance type because this allows Image Builder to launch an instance from a pool with sufficient capacity. This can reduce your transient build failures.

For Mac images, you might want to choose instance types that natively support the macOS operating system. For more information, see Amazon EC2 Mac instances in the *Amazon EC2 User Guide*.

- Instance placement settings where you can specify the host, host placement group, or Availability Zone where the instances that launch from your image should go.
- An instance profile that provides your build and test instances 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. The instance profile also requires a minimal set of permissions for EC2 Image Builder to successfully communicate with the instance. For more information, see Get set up to build custom images with Image Builder.
- The VPC, subnet, and security groups for your pipeline's build and test instances.
- The Amazon S3 location where Image Builder stores application logs from your build and testing. 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`**`/*`).
- An Amazon EC2 key pair that allows you to log on to your instance to troubleshoot if your build fails and you set `terminateInstanceOnFailure` to `false`.
- An SNS topic where Image Builder sends event notifications. For more information about how Image Builder integrates with Amazon SNS, see Amazon SNS integration in Image Builder.

> **Note**
>
> If your SNS topic is encrypted, the key that encrypts this topic must reside in the account where the Image Builder service runs. Image Builder can't send notifications to SNS topics that are encrypted with keys from other accounts.

You can create and manage infrastructure configurations using the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI.

Contents

- Image Builder and AWS PrivateLink interface VPC endpoints

> **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 section of this guide.

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 from the AWS CLI
- Get infrastructure configuration details from the AWS CLI

List infrastructure configurations from the AWS CLI

The following example shows how to list of all of your infrastructure configurations, using the **list-infrastructure-configurations** command in the AWS CLI.

```
aws imagebuilder list-infrastructure-configurations
```

Get infrastructure configuration details from the AWS CLI

The following example shows how use the **get-infrastructure-configuration** 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 an infrastructure configuration

This section describes how you can use the Image Builder console or **imagebuilder** commands in the AWS CLI to create an infrastructure configuration,

Console

To create an infrastructure configuration resource from the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. From the navigation pane, choose **Infrastructure configuration**.
3. Choose **Create infrastructure configuration**.
4. In the **General** section, enter the following required information:
   - Enter the **Name** of your infrastructure configuration resource.
   - Select an **IAM role** that you want to associate with the instance profile for component permissions on your build and test instances. Image Builder uses these permissions to download and run your components, upload logs to CloudWatch, and perform any additional actions that the components in your recipe specify.
5. In the **AWS infrastructure** panel, you can configure remaining infrastructure settings that are available. Enter the following required information:
   - **Instance type** – You can specify one or more instance types to use for this build. The service will pick one of these instance types based on availability.

     > **Note**
     >
     > Mac instances run on `.metal` instance types on a Dedicated Host. Your instance type must match one of the types that are defined for the host that it runs on. For more information about Mac instances and a list of instance types that natively support the macOS operating system, see Amazon EC2 Mac instances in the *Amazon EC2 User Guide*.

   - **SNS topic (optional)** – Select an SNS topic to receive notifications and alerts from EC2 Image Builder.

If you do not supply values for the following settings, they use service-specific defaults, where applicable.

- **VPC, subnet, and security groups** – Image Builder uses your default VPC and subnet. For more information about configuring VPC interface endpoints, see [Image Builder and AWS PrivateLink interface VPC endpoints](#).
- In the **Troubleshooting settings** section, you can configure the following values:
  - By default, the **Terminate instance on failure** check box is selected. However, when a build fails, you can log on to the EC2 instance to troubleshoot. If you want your instance to continue to run after a build failure, clear the check box.
  - **Key pair** – If your EC2 instance continues to run after a build failure, you can create a key pair or use an existing key pair to log on to the instance and troubleshoot.
  - **Logs** – You can specify an S3 bucket where Image Builder can write application logs to help troubleshoot your build and tests. If you don't specify an S3 bucket, Image Builder writes the application logs to the instance.
- In the **Instance metadata settings** section, you can configure the following values to apply to the EC2 instances that Image Builder uses to build and test your image:
  - Select the **Metadata version** to determine if EC2 requires a signed token header for instance metadata retrieval requests.
    - **V1 and V2 (token optional)** – Default value if you don't select anything.
    - **V2 (token required)**

    > **Note**
    >
    > We recommend that you configure all EC2 instances that Image Builder launches from a pipeline build to use IMDSv2 so that instance metadata retrieval requests require a signed token header.
  - **Metadata token response hop limit** – The number of network hops that the metadata token can travel. Minimum hops: 1, maximum hops: 64, with a default of one hop.
- In the **Instance placement settings** section, you can configure the following values to apply to the EC2 instances that Image Builder uses to build and test your image:
  - You can select the **Availability Zone** where Image Builder launches instances during image creation.

- Optionally select **Tenancy** for the servers that run the instances that you launch. By default, EC2 instances run on shared tenancy hardware. This means that multiple AWS accounts might share the same physical hardware. An instance with `dedicated` tenancy runs on single-tenant hardware. An instance with `host` tenancy runs on a Dedicated Host.

  Mac instances require a Dedicated Host that's created as a prerequisite before you build a custom image. Select `host` for your macOS image. You can then select a target host or host resource group to launch instances, but it's not required if your Dedicated Host has auto-placement enabled. For more information, see Auto-placement in the *Amazon EC2 User Guide*.

  - **Tenancy host ID** – The ID of the Dedicated Host on which the instances run.
  - **Tenancy host resource group** – The Amazon Resource Name (ARN) of the host resource group in which to launch the instances.

6. In the **Infrastructure tags** section (optional), you can assign metadata tags to the Amazon EC2 instance that Image Builder launches during the build process. Tags are entered as key value pairs.
7. In the **Tags** section (optional), you can assign metadata tags to the infrastructure configuration resource that Image Builder creates as output. Tags are entered as key value pairs.

AWS CLI

The following procedure shows how to configure the infrastructure for your image with the Image Builder **create-infrastructure-configuration** command in the AWS CLI. The command in step 2 takes in the file that you create in step 1. For these examples, the file from step 1 is called `create-infrastructure-configuration.json`.

1. **Create a CLI input JSON file**

   The following examples show variations of the JSON file that you might create for infrastructure configuration. Use a file editing tool to create a JSON file of your own.

   Example 1: Configuration to retain an instance from a failed build

This example specifies two instance types, `m5.large` and `m5.xlarge`. We recommend that you specify more than one instance type because this allows Image Builder to launch an instance from a pool with sufficient capacity. This can reduce your transient build failures.

The `instanceProfileName` specifies the instance profile that provides the instance with the permissions that the profile requires 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. The instance profile also requires a minimal set of permissions for EC2 Image Builder to successfully communicate with the instance. For more information, see Get set up to build custom images with Image Builder.

```
{
    "name": "ExampleInfraConfigDontTerminate",
    "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"
}
```

Example 2: macOS configuration with auto-placement

This example specifies instance types and placement for a Mac instance where the Dedicated Host has auto-placement enabled.

```
{
   "name": "macOSInfraConfigAutoPlacement",
```

```
    "description": "An example infrastructure configuration for macOS.",
    "instanceProfileName": "EC2InstanceProfileForImageBuilder",
    "instanceTypes": ["mac1.metal, mac2.metal"],
    "terminateInstanceOnFailure": false,
    "placement": {
       "tenancy": "host"
    }
}
```

Example 3: macOS configuration with Host ID specified

This example specifies instance type and placement for a Mac instance that targets a specific Dedicated Host.

```
{
    "name": "macOSInfraConfigHostPlacement",
    "description": "An example infrastructure configuration for macOS.",
    "instanceProfileName": "EC2InstanceProfileForImageBuilder",
    "instanceTypes": ["mac2-m1ultra.metal"],
    "terminateInstanceOnFailure": false,
    "placement": {
       "tenancy": "host",
       "hostId" : "h-1234567890abcdef0"
    }
}
```

2. **Use the file you created as input when you run the following command.**

```
aws imagebuilder create-infrastructure-configuration --cli-input-json
 file://create-infrastructure-configuration.json
```

Update an infrastructure configuration

This section covers how you can use the Image Builder console or **imagebuilder** commands in the AWS CLI to update an infrastructure configuration resource. To track your resources, you can apply tags as follows. Tags are entered as key value pairs.

- *Resource tags* assign metadata tags to the Amazon EC2 instance that Image Builder launches during the build process.

- *Tags* assign metadata tags to the infrastructure configuration resource that Image Builder creates as output.

Console

You can edit the following infrastructure configuration details from the Image Builder console:

- The **Description** for your infrastructure 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 update an infrastructure configuration resource from the Image Builder console, follow these steps:

Choose an existing Image Builder infrastructure configuration

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. To see a list of the infrastructure configuration resources under your account, choose **Infrastructure configuration** from the navigation pane.
3. To view details or edit an infrastructure configuration, choose the **Configuration name** link. This opens the detail view for the infrastructure configuration.

   > **Note**
   >
   > You can also select the check box next to the **Configuration name**, then choose **View detail**.

4. From the upper right corner of the **Infrastructure details** panel, choose **Edit** .
5. When you're ready to save updates you've made to your infrastructure configuration, choose **Save changes**.

AWS CLI

The following example shows how to update the infrastructure configuration for your image with the Image Builder update-infrastructure-configuration command in the AWS CLI.

1. **Create a CLI input JSON file**

   This infrastructure configuration example uses the same settings as the create example, except that we've updated the `terminateInstanceOnFailure` setting to `false`. After we run the **update-infrastructure-configuration** command, pipelines that use this infrastructure configuration terminate the build and test instances when the build fails.

   Use a 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-infrastructure-configuration.json`:

```
{
"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"
}
```

2. **Use the file you created as input when you run the following command.**

```
aws imagebuilder update-infrastructure-configuration --cli-input-json
 file://update-infrastructure-configuration.json
```

Image Builder and AWS PrivateLink interface VPC endpoints

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. When you create a new image, you can specify the VPC subnet-id in your infrastructure configuration.

> **Note**
>
> Each service that you access from within a VPC has its own interface endpoint, with its own endpoint policy. Image Builder downloads the AWSTOE component manager application and accesses managed resources from S3 buckets to create custom images. To grant access to those buckets, you must update the S3 endpoint policy to allow it. For more information, see Custom policies for S3 bucket access.

For more information about VPC endpoints, 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.

Create an interface VPC endpoint for Image Builder

To create a VPC endpoint for the Image Builder service, you can use 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*.

Create 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.

If you are using Amazon-managed components in your recipe, the VPC endpoint for Image Builder must allow access to the following serviced-owned component library:

`arn:aws:imagebuilder:region:aws:component/*`

> **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*.

Custom policies for S3 bucket access

Image Builder uses a publicly available S3 bucket to store and access managed resources, such as components. It also downloads the AWSTOE component management application from a separate S3 bucket. If you use a VPC endpoint for Amazon S3 in your environment, you'll need to ensure that your S3 VPC endpoint policy allows Image Builder to access the following S3 buckets. The bucket names are unique per AWS Region (*region*) and the application environment (*environment*). Image Builder and AWSTOE support the following application environments: `prod`, `preprod`, and `beta`.

- The AWSTOE component manager bucket:

  ```
  s3://ec2imagebuilder-toe-region-environment
  ```

  Example: s3://ec2imagebuilder-toe-us-west-2-prod/*
- The Image Builder managed resources bucket:

  ```
  s3://ec2imagebuilder-managed-resources-region-environment/components
  ```

  Example: s3://ec2imagebuilder-managed-resources-us-west-2-prod/components/*

VPC endpoint policy examples

This section includes examples of custom VPC endpoint policies.

General VPC endpoint policy for Image Builder actions

The following example endpoint policy for Image Builder denies permission to delete Image Builder images and components. The example policy also grants permission to perform all other EC2 Image Builder actions.

JSON

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

```
    "Statement": [
        {
            "Action": "imagebuilder:*",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "imagebuilder:DeleteImage",
            "Effect": "Deny",
            "Resource": "*"
        },
        {
            "Action": "imagebuilder:DeleteComponent",
            "Effect": "Deny",
            "Resource": "*"
        }
    ]
}
```

Restrict access by organization, allow managed component access

The following example endpoint policy shows how to restrict access to identities and resources that belong to your organization and provide access to the Amazon-managed Image Builder components. Replace *region*, *principal-org-id*, and *resource-org-id* with your organization's values.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:PrincipalOrgID": "principal-org-id",
                    "aws:ResourceOrgID": "resource-org-id"
                }
            }
        },
        {
            "Sid": "AllowAccessToEC2ImageBuilderComponents",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "imagebuilder:GetComponent"
            ],
            "Resource": [
                "arn:aws:imagebuilder:us-east-1:aws:component/*"
            ]
        }
    ]
}
```

VPC endpoint policy for Amazon S3 bucket access

The following S3 endpoint policy example shows how to provide access to the S3 buckets that Image Builder uses to build custom images. Replace *region* and *environment* with your organization's values. Add any other required permissions to the policy based on your application requirements.

> **Note**
>
> For Linux images, if you don't specify user data in your image recipe, Image Builder adds a script to download and install the Systems Manager agent on the build and test instances for your image. To download the agent, Image Builder accesses the S3 bucket for your build Region.
> To ensure that Image Builder can bootstrap the build and test instances, add the following additional resource to your S3 endpoint policy:
> `"arn:aws:s3:::amazon-ssm-region/*"`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowImageBuilderAccessToAppAndComponentBuckets",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::ec2imagebuilder-toe-region-environment/*",
        "arn:aws:s3:::ec2imagebuilder-managed-resources-region-environment/
components/*"
      ]
    }
  ]
}
```

Manage Image Builder distribution settings

Before you configure distribution settings for your output images, we recommend that you verify availability for any underlying infrastructure or other requirements for instances that are launched from your output image in the distribution target Regions. For example, not all Regions support EC2 Mac Dedicated Hosts, which are required to launch instances from a macOS image. For more information about instance types and pricing for Dedicated Hosts, see Amazon EC2 Dedicated Hosts Pricing.

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. With distribution settings, you can perform the following actions:

AMI distribution

- Specify the name and description of your output AMI.

- Authorize other AWS accounts, organizations, and OUs to launch the AMI from the owner's account. The owner account is billed for charges that are associated with the AMI.

> **Note**
>
> To make an AMI public, set the launch permission authorized accounts to `all`. For information and examples, see **ModifyImageAttribute** in the *Amazon EC2 API Reference*.

- Create a copy of the output AMI for each of the specified target accounts, organizations, and OUs in the destination Region. The target accounts, organizations, and OUs own their AMI copies, and are billed for any associated charges. For more information about distributing your AMI to AWS Organizations and OUs, see Share an AMI with organizations or OUs.
- Copy the AMI to the owner's account in other AWS Regions.
- Export VM image disks to Amazon Simple Storage Service (Amazon S3). For more information, see Example: Create distribution settings for output VM disks from the AWS CLI.

Container image distribution

- Specify the ECR repository where Image Builder stores the output image in the distribution Region.

You can use your distribution settings in the following ways to deliver images to target Regions, accounts, AWS Organizations and organizational units (OUs) one time, or with every pipeline build:

- To automatically deliver updated images to specified Regions, accounts, Organizations, and OUs, 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, accounts, Organizations, and OUs, use distribution settings with an Image Builder pipeline that you run one time from the Image Builder console, using **Run pipeline** from the **Actions** menu.
- To create a new image and deliver it to the specified Regions, accounts, Organizations, and OUs, use distribution settings with the following API action or Image Builder command in the AWS CLI:
  - The **CreateImage** action in the Image Builder API.
  - The **create-image** command in the AWS CLI.
- To export virtual machine (VM) image disks to S3 buckets in target Regions as part of your regular image build process.

> **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 section of this guide.

Contents

List and view distribution configuration detail

This section describes the various ways that you can find information and view details for your EC2 Image Builder distribution configuration.

Distribution configuration detail

List distribution configurations from the console

To see a list of the distribution configurations 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 **Distribution settings** from the navigation pane. This shows a list of the distribution configurations that are created under your account.

3. To view details or create new distribution configuration, choose the **Configuration name** link. This opens the detail view for the distribution settings.

> **Note**
>
> You can also select the check box next to the **Configuration name**, then choose **View details**.

View distribution configuration details from the console

To view details for a specific distribution configuration using the Image Builder console, select the configuration to review, using the steps described in List distribution configurations from the console.

On the distribution detail page, you can:

- **Delete** the distribution configuration. For more information about deleting resources in Image Builder, see Delete outdated or unused Image Builder resources.
- **Edit** distribution details.

List distributions from the AWS CLI

The following example shows how to use the **list-distribution-configurations** command in the AWS CLI to list all of your distributions.

```
aws imagebuilder list-distribution-configurations
```

Get distribution configuration detail from the AWS CLI

The following example shows how to use the **get-distribution-configuration** command in the AWS CLI to get the details of a distribution 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 AMI distribution configurations

This section covers creating and updating distribution configurations for an Image Builder AMI.

Contents

AMI distribution prerequisites

Some distribution settings have prerequisites, as follows:

Topics

Prerequisites for SSM output parameters

Before you create a new AMI distribution configuration that sets an AWS Systems Manager Parameter Store parameter (SSM parameter), ensure that you've met the following prerequisites.

Execution role

When you create a pipeline or use the create-image command in the AWS CLI, you can only specify one Image Builder execution role. If you have defined an Image Builder workflow execution role, you would add any additional feature permissions to that role. Otherwise, you would create a new custom role that includes the required permissions.

- To store the output AMI ID in an SSM parameter during distribution, you must specify the `ssm:PutParameter` action in your Image Builder execution role, with the parameter listed as a resource.
- When you set the parameter data type to `AWS EC2 Image` to signal Systems Manager to validate the parameter value as an AMI ID, you must also add the `ec2:DescribeImages` action.

Prerequisites for EC2 Fast Launch

Before you create a new distribution configuration for EC2 Fast Launch for Windows AMIs, ensure that you've met the following prerequisites.

- If you provide a custom launch template when you configure EC2 Fast Launch, the service uses the VPC and other configuration settings that you've defined in the launch template. For more information, see Use a launch template when you set up EC2 Fast Launch.
- If you don't use a custom launch template to configure your settings, you must attach the EC2FastLaunchFullAccess policy to the IAM role that Image Builder uses to create your image. When you create a pipeline or use the create-image command in the AWS CLI, you can only specify one Image Builder execution role. If you have defined an Image Builder workflow execution role, you would add any additional feature permissions to that role. Otherwise, you would create a new custom role that includes the required permissions.

  Then, when Image Builder copies your image, EC2 Fast Launch automatically creates an AWS CloudFormation stack with the following resources in your AWS account.
  - A virtual private cloud (VPC)
  - Private subnets across multiple Availability Zones
  - A launch template configured with Instance Metadata Service Version 2 (IMDSv2)
  - A security group with no inbound or outbound rules

> **Note**
>
> Image Builder doesn't support cross-account distribution for AMIs with EC2 Fast Launch pre-enabled. EC2 Fast Launch must be enabled from the destination account.

Create an AMI distribution configuration

Distribution configurations include the output AMI name, specific Region settings for encryption, launch permissions, and AWS accounts, organizations, and organizational units (OUs) that can launch the output AMI, and license configurations.

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), or configure EC2 Fast Launch for output Windows AMIs. To make an AMI public, set the launch permission authorized accounts to `all`. See the examples for making an AMI public at EC2 **ModifyImageAttribute**.

Console

Follow these steps to create a new AMI distribution configuration in the AWS Management 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 configurations that are created under your account.
3. Choose **Create distribution settings** near the top of the **Distribution settings** panel.
4. In the **Image type** section, choose the **Amazon Machine Image (AMI)** output type.
5. In the **General** section, enter a **Name** for your distribution configuration, and optional description.
6. In the **Region settings** section, enter the following details for each Region where you are distributing your AMI:
   a. The AMI is distributed to the current Region (**Region 1**), by default. **Region 1** is the source for the distribution. Some settings for **Region 1** are not open for editing. For any Regions that you add, you can choose a Region from the **Region** dropdown list.

      The **Kms key** identifies the AWS KMS key that's used to encrypt the EBS volumes for your image in the target Region. It's important to note that this doesn't apply for the original AMI that the build creates under your account in the source Region (**Region 1**). Encryption that runs during the distribution phase of the build is only for images that are distributed to other accounts or Regions.

      To encrypt the EBS volumes for the AMI that's created in the source Region for your account, you must set the KMS key in the image recipe block device mapping (**Storage (volumes)** in the console).

      Image Builder copies the AMI to the **Target accounts** that you specify for the Region.

> **Prerequisite**
>
> To copy an image across accounts, you must create the `EC2ImageBuilderDistributionCrossAccountRole` role in all of the distribution target accounts, and attach the Ec2ImageBuilderCrossAccountDistributionAccess policy managed policy to the role.

The **Output AMI name** is optional. If you provide a name, the final output AMI name includes an appended timestamp of when the AMI is built. If you do not specify a name, Image Builder appends the build timestamp to the recipe name. This ensures unique AMI names for each build.

i. With AMI sharing, you can grant access for specified AWS Principals to launch instances from your AMI. If you expand the **AMI sharing** section, you can enter the following details:

   - **Launch permissions** – Select **Private** if you want to keep your AMI private, and allow access for specific AWS Principals to launch an instance from your private AMI. Select **Public** if you want to make your AMI public. Any AWS Principal can launch an instance from your public AMI.
   - **Principals** – You can grant access for the following types of AWS Principals to launch instances:
     - **AWS account** – Grant access to a specific AWS account
     - **Organizational unit (OU)** – Grant access to an OU, and all of its child entities. Child entities include OUs and AWS accounts.
     - **Organization** – Grant access to your AWS Organizations, and all of its child entities. Child entities include OUs and AWS accounts.

       First, select the Principal type. Then enter the ID for the AWS Principal to which you want to grant access in the box to the right of the drop-down list. You can enter multiple IDs of different types.

ii. You can expand the **License configuration** section to attach license configurations created with AWS License Manager to your Image Builder images. License configurations contain licensing rules based on the terms of your enterprise agreements. Image Builder automatically includes license configurations that were associated with your base AMI.

iii. You can expand the **Launch template configuration** section to specify an EC2 launch template to use for launching instances from the AMI you create.

If you are using an EC2 launch template, you can instruct Image Builder to create a new version of your launch template that includes the latest AMI ID after the build completes. To update the launch template, configure the settings as follows:

- **Launch template name –** Select the name of the launch template that you want Image Builder to update.
- **Set the default version –** Select this check box to update the launch template default version to the new version.

To add another launch template configuration, choose **Add launch template configuration**. You can have up to five launch template configurations per Region.

iv. You can expand the **SSM parameter configurations** section to configure an SSM parameter that will store the output AMI ID for the image that's distributed to the destination Region. You can optionally specify a distribution account in the Region.

Parameter name – Enter the name for your parameter. For example `/output/image/param`.

Data type – Keep the default value (`AWS EC2 Image`). This tells Systems Manager to validate the parameter value to ensure that it's a valid AMI ID.

b. To add distribution settings for another Region, choose **Add Region**.

7. Choose **Create settings** when you are done.

AWS CLI

The following example shows how to use the **create-distribution-configuration** command to create a new distribution configuration for your AMI, using the AWS CLI.

1. **Create a CLI input JSON file**

   Use a file-editing tool to create a JSON file with keys shown in one of the following examples, and values that are valid for your environment. These examples define which

AWS accounts, AWS Organizations or organizational units (OUs) have permission to launch the AMI you distribute to the specified Regions. Name the file `create-ami-distribution-configuration.json`, for use in the next step:

Example 1: Distribute to AWS accounts

This example distributes an AMI to two Regions, and specifies AWS accounts that have launch permissions in each Region.

```
{
 "name": "MyExampleAccountDistribution",
 "description": "Copies AMI to eu-west-1, and specifies accounts that can launch
 instances in each Region.",
 "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"
     ]
    }
   }
  }
```

```
  ]
}
```

Example 2: Distribute to Organizations and OUs

This example distributes an AMI to the source Region, and specifies organization and OU launch permissions.

```
{
 "name": "MyExampleAWSOrganizationDistribution",
 "description": "Shares AMI with the Organization and OU",
 "distributions": [
  {
   "region": "us-west-2",
   "amiDistributionConfiguration": {
    "name": "Name {{ imagebuilder:buildDate }}",
    "launchPermission": {
     "organizationArns": [
      "arn:aws:organizations::123456789012:organization/o-myorganization123"
     ],
     "organizationalUnitArns": [
      "arn:aws:organizations::123456789012:ou/o-123example/ou-1234-
myorganizationalunit"
     ]
    }
   }
  }
 ]
}
```

Example 3: Store the output AMI ID in an SSM parameter

This example stores the output AMI ID in an AWS Systems Manager Parameter Store parameter in the distribution Region.

```
{
 "name": "SSMParameterOutputAMI",
 "description": "Updates an SSM parameter with the output AMI ID for the
 distribution.",
 "distributions": [
  {
   "region": "us-west-2",
```

```
    "amiDistributionConfiguration": {
     "name": "Name {{ imagebuilder:buildDate }}"
    },
    "ssmParameterConfigurations": [
     {
      "amiAccountId": "111122223333",
      "parameterName": "/output/image/param",
      "dataType": "aws:ec2:image"
     }
    ]
   }
  ]
 }
```

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, while Linux and macOS use the forward slash (/).

For more detailed information, see **create-distribution-configuration** in the *AWS CLI Command Reference*.

Update an AMI distribution configuration

You can change your AMI distribution configuration. However, the changes you make do not apply to any resources that Image Builder has already distributed. For example, if you have distributed an AMI to a Region that you later remove from your distribution, the AMI that was already distributed remains in that Region until you remove it manually.

AWS Management Console

Follow these steps to an AMI distribution configuration in the AWS Management 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 configurations that are created under your account.
3. To view details or update a distribution configuration, choose the **Configuration name** link. This opens the detail view for the distribution settings.

> **Note**
>
> You can also select the check box next to the **Configuration name**, then choose **View details**.

4. To edit distribution configuration, choose **Edit** from the upper right corner of the **Distribution details** section. Some fields are locked, such as the **Name** of the distribution configuration, and the default **Region** that is displayed as **Region 1**. For more information about the distribution configuration settings, see Create an AMI distribution configuration.
5. Choose **Save changes** when you are done.

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 a file-editing tool to create a JSON file with the keys shown in the following example, and 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 specifies accounts that can launch
 instances in each Region.",
 "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, while Linux and macOS use 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 section.

Example: Enable EC2 Fast Launch with a launch template for output AMIs

The following example shows how to use the **create-distribution-configuration** command with a launch template to create distribution settings that have EC2 Fast Launch configured for your AMI, from the AWS CLI.

To configure EC2 Fast Launch settings without a launch template, ensure that you've met all of the EC2 Fast Launch prerequisites before you create your distribution configuration.

1. **Create a CLI input JSON file**

   Use a file editing tool to create a JSON file with keys as shown in the following example, plus values that are valid for your environment.

   This example launches instances for all of its target resources simultaneously, because the maximum number of parallel launches is greater than the target resource count. This file is named `ami-dist-config-win-fast-launch.json` in the command example shown in the next step.

```
{
"name": "WinFastLaunchDistribution",
"description": "An example of Windows AMI EC2 Fast Launch settings in the
 distribution configuration.",
"distributions": [
 {
  "region": "us-west-2",
  "amiDistributionConfiguration": {
   "name": "Name {{imagebuilder:buildDate}}",
   "description": "Includes Windows AMI EC2 Fast Launch settings.",
   "amiTags": {
    "KeyName": "Some Value"
   }
```

```
  },
  "fastLaunchConfigurations": [{
   "enabled": true,
   "snapshotConfiguration": {
    "targetResourceCount": 5
   },
   "maxParallelLaunches": 6,
   "launchTemplate": {
    "launchTemplateId": "lt-0ab1234c56d789012",
    "launchTemplateVersion": "1"
    }
  }],
  "launchTemplateConfigurations": [{
       "launchTemplateId": "lt-0ab1234c56d789012",
       "setDefaultVersion": true
    }]
 }]
}
```

> **Note**
>
> You can specify the `launchTemplateName` instead of the `launchTemplateId` in the `launchTemplate` section, but you can't specify both the name and Id.

2. **Run the following command, using the file you created as input.**

```
aws imagebuilder create-distribution-configuration --cli-input-json file://ami-dist-config-win-fast-launch.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, while Linux and macOS use the forward slash (/).

For more detailed information, see **create-distribution-configuration** in the *AWS CLI Command Reference*.

Example: Create distribution settings for output VM disks from the AWS CLI

The following example shows how to use the **create-distribution-configuration** command to create distribution settings that will export VM image disks to Amazon S3 with every image build.

1. **Create a CLI input JSON file**

   You can streamline the **create-distribution-configuration** command that you use in the AWS CLI. To do this, create a JSON file that contains all of the export configuration that you want to pass into the command.

   > **Note**
   >
   > The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API operation request parameters. To review the API operation request parameters, see the **CreateDistributionConfiguration** command in the *EC2 Image Builder API Reference*.
   > To provide the data values as command line parameters, refer to the parameter names specified in the *AWS CLI Command Reference*. to the **create-distribution-configuration** command as options.

   Here is a summary of the parameters that we specify in the `s3ExportConfiguration` JSON object for this example:

   - **roleName** (string, required) – The name of the role that grants VM Import/Export permission to export images to your S3 bucket.
   - **diskImageFormat** (string, required) – Export the updated disk image to one of the following supported formats:
     - **Virtual Hard Disk (VHD)** – Compatible with Citrix Xen and Microsoft Hyper-V virtualization products.
     - **Stream-optimized ESX Virtual Machine Disk (VMDK)** – Compatible with VMware ESX and VMware vSphere versions 4, 5, and 6.

    - **Raw** – Raw format.
- **s3Bucket** (string, required) – The S3 bucket in which to store the output disk images for your VM.

Save the file as `export-vm-disks.json`. Use the file name in the **create-distribution-configuration** command.

```
{
 "name": "example-distribution-configuration-with-vm-export",
 "description": "example",
 "distributions": [
  {
   "region": "us-west-2",
   "amiDistributionConfiguration": {
    "description": "example-with-vm-export"

   },
   "s3ExportConfiguration": {
    "roleName": "vmimport",
    "diskImageFormat": "RAW",
    "s3Bucket": "vm-bucket-export"
   }
  }],
 "clientToken": "abc123def4567ab"
}
```

2. **Run the following command, using the file you created as input.**

```
aws imagebuilder create-distribution-configuration --cli-input-json file://export-vm-disks.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, while Linux and macOS use the forward slash (/).

For more detailed information, see **[create-distribution-configuration]** in the *AWS CLI Command Reference*.

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 from the 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, while Linux and macOS use the forward slash (/).

For more detailed information, see create-distribution-configuration in the *AWS CLI Command Reference*.

Update distribution settings for your container image from the 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, while Linux and macOS use 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 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 with the AWS CLI. To create the JSON file for an infrastructure configuration, see Create an infrastructure configuration.

Prerequisites for cross-account AMI distribution

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 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 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 key, 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 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. The following example shows a trust policy in the destination account that specifies the account ID from the source account.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::444455556666:root"
        },
        "Action": "sts:AssumeRole"
    }]
}
```

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.

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowRoleToPerformKMSOperationsOnBehalfOfTheDestinationAccount",
   "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"
                }
            }
        }
    ]
}
```

6. If you use an AWS Systems Manager Parameter Store parameter to store the AMI ID of the output AMI for the distribution account and Region, you must add the following policy to your `EC2ImageBuilderDistributionCrossAccountRole` in each destination account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:*:111122223333:parameter/ImageBuilder-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages"
            ],
            "Resource": "*"
        }
    ]
}
```

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 don't 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 from the 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 for cross-account AMI distribution 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 from the 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 for cross-account AMI distribution 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, while Linux and macOS use the forward slash (/).

You can also provide JSON directly in the command, using the `--distributions` parameter.

Configure AMI distribution with an 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 distribution 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

> **Note**
>
> When you include a launch template to enable Windows Fast Launch in your image, the launch template must include the following tag so that Image Builder can enable Windows Fast Launch on your behalf.

```
   CreatedBy: EC2 Image Builder
```

Add an EC2 launch template to AMI distribution settings from the 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 EC2 launch template to AMI distribution settings from the 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, while Linux and macOS use the forward slash (/).

You can also provide JSON directly in the command, using the `--distributions` parameter.

Share Image Builder resources with AWS RAM

EC2 Image Builder integrates with AWS Resource Access Manager (AWS RAM) so that you can share the following types of Image Builder resources with any AWS account or through AWS Organizations.

- Components
- Images
- Recipes

To share resources through AWS RAM, you must create 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. The following list includes the types of accounts and organizations that you can share with.

- Specific AWS accounts inside or outside of its organization in AWS Organizations.
- An organizational unit (OU) inside of its organization in AWS Organizations.
- Its entire organization in AWS Organizations.
- AWS Organizations or OUs outside of its organization.

In this model, the AWS account that owns the resource (owner) shares it with other AWS accounts or through AWS Organizations (consumers) within the same Region. When a shared resource is updated, consumers get those updates automatically.

> **Note**
>
> 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 are shared with them.

Topics

- Unshare an Image Builder resource from AWS RAM

Resource owners

Image Builder resources can only be shared in the AWS Region where they are created. When you share these resources, they will not replicate across Regions.

To get a list of the Image Builder resources that you own and can share, specify the ownership filter in the console or when you run the command in the AWS CLI.

- List Image Builder components
- List images
- List and view image recipe details
- List and view container recipe details

For more information about AWS RAM, see the *AWS RAM User Guide*.

Prerequisites for sharing Image Builder resources

To share an Image Builder resource, such as a component, image, or recipe:

- Your AWS account must own the Image Builder resource that you want to share. 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, organizations, or OUs.
- In order to share your Image Builder resources with AWS Organizations and OUs using AWS RAM, you must enable sharing. For more information, see Enable Sharing with AWS Organizations in the *AWS RAM User Guide*.
- If you distribute an image encrypted with AWS KMS across accounts in different Regions, you must create a 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.

The following resources that Image Builder creates from your pipeline build are not considered Image Builder resources – rather, they are external resources that Image Builder distributes in your account, and to the AWS Regions, accounts, and organizations or organizational units (OUs) that you specify in your distribution configuration.

- Amazon Machine Images (AMIs)
- Container images that reside in Amazon ECR

For more information about distribution settings for your AMI, see Create and update AMI distribution configurations. For more information about distribution settings for your container image in Amazon ECR, see Create and update distribution settings for container images.

For more information about sharing your AMI with AWS Organizations and OUs, see Share an AMI with organizations or OUs.

Resource consumers

Consumers can use a shared resource, but cannot modify it in any way. When they create Image Builder recipes, they can specify a shared image as the base image, and they can add shared components. They can also specify a shared recipe when they create an Image Builder image pipeline, or when they use the **create-image** command in the AWS CLI.

If you belong to an organization in AWS Organizations, and sharing within your organization is enabled, consumers in your organization are automatically granted access to the shared resource. Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared resource after accepting the invitation.

Create an AWS RAM resource share for your Image Builder resources

To share an Image Builder component, image, or recipe, you must add it to an AWS Resource Access Manager resource share. The resource share specifies the resources to share and the consumers with whom they are shared.

The following options are available for sharing your resources.

Option 1: Create a RAM resource share

When you create a RAM resource share, you can share a component, image, or recipe that you own in a single step. Use one of the following methods to create your resource share:

- **Console**

To create your resource share using the AWS RAM console, see Share AWS resources owned by you in the *AWS RAM User Guide*.

- **AWS CLI**

  To create your resource share using the AWS RAM command line interface, run the **create-resource-share** command in the AWS CLI.

Option 2: Apply a resource policy and promote to an existing resource share

The second option for sharing your resources involves two steps, running commands in the AWS CLI for both. The first step uses Image Builder commands in the AWS CLI to apply resource-based policies to the shared resource. The second step promotes the resource to a RAM resource share using the **promote-resource-share-created-from-policy** AWS RAM command in the AWS CLI to ensure that the resource is visible to all principals with whom you've shared it.

1. **Apply the resource policy**

   To successfully apply the resource policy, you must ensure that the account with which you are sharing has permission to access any underlying resources.

   Choose the tab that matches your resource type for the applicable command.

   Image

   You can apply a resource policy to an image, to allow others to use it as the base image in their recipes.

   Run the **put-image-policy** Image Builder command in the AWS CLI, to identify the AWS principals to share the image with.

```
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" ] } ] }'
```

Component

You can apply a resource policy to a build or test component to enable cross-account sharing. This command gives other accounts permission to use your component in their recipes. To successfully apply the resource policy, you must ensure that the account with which you are sharing has permission to access any resources referenced by the shared component, such as files hosted on private repositories.

Run the **put-component-policy** Image Builder command in the AWS CLI, to identify the AWS principals to share the component with.

```
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" ] } ] }'
```

Image recipe

You can apply a resource policy to an image recipe to enable cross-account sharing. This command gives other accounts permission to use your recipe to create images in their accounts. To successfully apply the resource policy, you must ensure that the account with which you are sharing has permission to access any resources that the recipe references, such as the base image, or selected components.

Run the **put-image-recipe-policy** Image Builder command in the AWS CLI, to identify the AWS principals to share the image with.

```
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" ] } ] }'
```

Container recipe

You can apply a resource policy to a container recipe to enable cross-account sharing. This command gives other accounts permission to use your recipe to create images in their accounts. To successfully apply the resource policy, you must ensure that the account with which you are sharing has permission to access any resources that the recipe references, such as the base image, or selected components.

Run the **put-container-recipe-policy** Image Builder command in the AWS CLI, to identify the AWS principals to share the image with.

```
aws imagebuilder put-container-recipe-policy --container-recipe-arn
 arn:aws:imagebuilder:us-west-2:123456789012:container-recipe/my-example-
container-recipe/2021.12.03 --policy '{ "Version": "2012-10-17", "Statement":
 [ { "Effect": "Allow", "Principal": { "AWS": [ "123456789012" ] }, "Action":
 [ "imagebuilder:GetContainerRecipe", "imagebuilder:ListContainerRecipes" ],
 "Resource": [ "arn:aws:imagebuilder:us-west-2:123456789012:container-recipe/my-
example-container-recipe/2021.12.03" ] } ] }'
```

> **Note**
>
> To set the correct policies for sharing and unsharing a resource, the resource owner must have `imagebuilder:put*` permissions.

2. **Promote as a RAM resource share**

   To ensure that the resource is visible to all principals with whom you've shared it, run the **promote-resource-share-created-from-policy** AWS RAM command in the AWS CLI.

Unshare an Image Builder resource from AWS RAM

To unshare an Image Builder resource that you own, such as a shared component, image, or recipe, you must remove it from the AWS Resource Access Manager resource share. You can do this using the AWS RAM console or the AWS CLI.

> **Note**
>
> Owners cannot delete a shared resource until it is no longer shared. An owner cannot unshare these resources until none of the consumers depend on 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 RAM User Guide*.

To unshare a shared component, image, or recipe that you own using the AWS CLI

Use the **disassociate-resource-share** command to stop sharing the resource.

Tag Image Builder output 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

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 Amazon Resource Name (ARN.) For example, `"arn:aws:imagebuilder:us-west-2:123456789012:component/myexample-component/2019.12.02/1"` shows the build version as 1.

To help you keep track of Amazon Machine Images (AMIs) that you've distributed, Image Builder automatically adds the following tags to your output AMIs.

- `"CreatedBy":"EC2 Image Builder"`
- `"Ec2ImageBuilderArn":"arn:aws:imagebuilder:`*`us-west-2`*`:`*`123456789012`*`:image/`*`simple-recipe-linux`*`/`*`1.0.0/10`*`"`. This tag contains the ARN of the Image Builder image resource that was used to create the AMI.

Contents

Tag a resource from the 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 from the 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 from the 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 outdated or unused 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.

Deleting resources does not delete any Amazon EC2 AMIs or Amazon ECR container images that are created during the image build process. You must clean those up separately, using the appropriate Amazon EC2 or Amazon ECR console actions, or API or AWS CLI commands.

> **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 from 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 from 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 from the 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 from the 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 build and test workflows for Image Builder images

An image workflow defines the sequence of steps that EC2 Image Builder performs during the build and test stages of the image creation process. This is part of the overall Image Builder workflow framework.

Image workflow benefits

- With image workflows, you have more flexibility, visibility, and control over the image creation process.
- You can add customized workflow steps when you define your workflow document, or you can choose to use the Image Builder default workflow.
- You can exclude workflow steps that are included in default image workflows.
- You can create test-only workflows that skip the build process entirely. You can do the same to create build-only workflows.

> **Note**
>
> You can't modify an existing workflow, but you can clone it or create a new version.

Workflow topics

- Workflow framework: Stages
- Service access
- Use managed workflows for your images
- List image workflows
- Create an image workflow
- Create a YAML workflow document

Workflow framework: Stages

To customize image workflows, it's important to understand the workflow stages that make up the image creation workflow framework.

The image creation workflow framework includes the following 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 recipe can include components that install an application or modify the operating system firewall settings.

   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, there are some differences between images that create AMIs and container images. For AMI workflows, Image Builder launches an EC2 instance from the snapshot that it 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. For container workflows, the tests run on the same instance that was used for building.

The workflow framework also includes a distribution stage. However, Image Builder handles the workflows for that stage.

Service access

To run image workflows, Image Builder needs permission to perform workflow actions. You can specify the AWSServiceRoleForImageBuilder service-linked role, or you can specify your own custom role for service access, as follows.

- **Console** – In the pipeline wizard **Step 3 Define image creation process**, select the service-linked role or your own custom role from the **IAM role** list in the **Service access** panel.
- **Image Builder API** – In the CreateImage action request, specify the service-linked role or your own custom role as the value for the `executionRole` parameter.

To learn more about how to create a service role, see Creating a role to delegate permissions to an AWS service in the *AWS Identity and Access Management User Guide*.

Use managed workflows for your images

Managed workflows are created and maintained by AWS. When you use managed workflows in your image pipelines or for one-off image creation, you can select the Amazon Resource Name

(ARN) of the managed workflow that you want to use. Amazon provides the latest versions that have patches and other updates applied. To get a list of managed workflows, see List image workflows, and filter on **Owner = Amazon** (console).

List image workflows

On the **Image workflows** list page in the Image Builder console, you can get a list of the image workflow resources that you own or have access to, along with some key details about these resources. You can also use commands or actions with the Image Builder API, SDKs, or AWS CLI to list image workflows in your account.

You can use one of the following methods to list image workflow resources that you own or have access to. For the API action, see ListWorkflows in the *EC2 Image Builder API Reference*. For the associated SDK request, refer to the See Also link on the same page.

You can filter on the following details to streamline the list of results. For example, if you filter on **Owner = Amazon** in the console, the list displays only AWS managed workflows.

- Workflow
- Version
- Type
- Owner

Console

Workflow details

Details on the **Image workflows** list page in the Image Builder console include the following:

- **Workflow** – The name of the most recent version of the image workflow resource. In the Image Builder console, the **Workflow** column links to the workflow detail page.
- **Version** – The most recent version of the image workflow resource.
- **Type** – The workflow type: `BUILD` or `TEST`.
- **Owner** – The owner of the workflow resource.
- **Creation time** – The date and time when Image Builder created the most recent version of the image workflow resource.

- **ARN –** The Amazon Resource Name (ARN) of the current version of the image workflow resource.

List image workflows

To list image workflow resources in the Image Builder console, perform the following steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image workflows** from the navigation pane.

Filter results

On the **Image workflows** list page, you can search for specific image workflows to filter your results. The following filters are available for image workflows:

`Workflow`

You can enter all or part of a workflow name to streamline results. The default is to show all workflows in the list.

`Version`

You can enter all or part of a version number to streamline results. The default is to show all versions in the list.

`Type`

You can filter by the workflow type or view all types. The default is to show all workflow types in the list.

- *BUILD*
- *TEST*

`Owner`

When you select the owner filter from the search bar, Image Builder shows a list of the owners for the image workflows in your account. You can select an owner from the list to streamline results. The default is to show all owners in the list.

- *AWS account* – The account that owns the workflow resource.

- *Amazon* – Workflow resources that Amazon owns and manages.

AWS CLI

When you run the list-workflows command in the AWS CLI, you can get a list of image workflows that you own or have access to.

The following command example shows how to use the **list-workflows** command without filters to list all of the Image Builder image workflow resources that you own or have access to.

Example: list all image workflows

```
aws imagebuilder list-workflows
```

Output:

```
{
 "workflowVersionList": [
  {
    "name": "example-test-workflow",
    "dateCreated": "2023-11-21T22:53:14.347Z",
    "version": "1.0.0",
    "owner": "111122223333",
    "type": "TEST",
    "arn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/test/example-test-
workflow/1.0.0"
  },
  {
    "name": "example-build-workflow",
    "dateCreated": "2023-11-20T12:26:10.425Z",
    "version": "1.0.0",
    "owner": "111122223333",
    "type": "BUILD",
    "arn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/build/example-
build-workflow/1.0.0"
  }
 ]
}
```

When you run the **list-workflows** command, you can apply filters to streamline the results, as the following example shows. For more information about how to filter your results, see the list-workflows command in the *AWS CLI Command Reference*.

Example: filter for build workflows

```
aws imagebuilder list-workflows --filters name="type",values="BUILD"
```

Output:


```
{
 "workflowVersionList": [
  {
    "name": "example-build-workflow",
    "dateCreated": "2023-11-20T12:26:10.425Z",
    "version": "1.0.0",
    "owner": "111122223333",
    "type": "BUILD",
    "arn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/build/example-build-workflow/1.0.0"
  }
 ]
}
```


Create an image workflow

When you create an image workflow, you have more control over your image creation process. You can specify what workflow runs when Image Builder builds your image, and what workflows run when it tests the image. You can also specify a customer managed key to encrypt your workflow resources. To learn more about encryption for your workflow resources, see Encryption and key management in Image Builder.

For image creation, you can specify one build stage workflow, and one or more test stage workflows. You can even skip the build or test stage entirely, depending on your needs. You configure the actions that your workflow takes in the YAML definition document that your workflow uses. For more information about syntax for your YAML document, see Create a YAML workflow document.

For steps to create a new build or test workflow select the tab that matches the environment you'll use.

AWS Management Console

You can use the following process to create a new workflow in the Image Builder console.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. Choose **Image workflows** from the navigation pane. This displays a list of image workflows that your account owns or has access to.

   > **Note**
   >
   > You'll always see the Amazon managed workflow resources that Image Builder uses for its default workflows in your list. To view details for those workflows, you can select the **Workflow** link.

3. To create a new workflow, choose **Create image workflow**. This displays the **Create image workflow** page.
4. Configure the details for your new workflow. To create a build workflow, select the **Build** option near the top of the form. To create a test workflow, select the **Test** option near the top of the form. Image Builder populates the **Templates** list based on this option. All other steps are the same for build and test workflows.

   General

   The general section includes settings that apply to your workflow resource, such as name and description. The general settings include the following:

   - **Image workflow name** (required) – The name for your image workflow. The name must be unique in your account. The name can be up to 128 characters in length. Valid characters include letters, numbers, spaces, `-`, and `_`.
   - **Version** (required) – The semantic version for the workflow resource to create (*major.minor.patch*).
   - **Description** (optional) – Optionally add a description for your workflow.
   - **KMS key** (optional) – You can encrypt your workflow resources with an customer managed key. For more information, see Encrypt image workflows with a customer managed key.

   Definition document

   The YAML workflow document contains all of the configuration for your workflow.

Get started

- To start with an Image Builder default template as a baseline for your workflow, select the **Start from templates** option. This option is selected by default. After you choose what template to use from the **Templates** list, this copies the default configuration from the template you selected into the **Content** for your new workflow document, where you can make changes.
- To define your workflow document from scratch, select the **Start from scratch** option. This populates the **Content** with a short outline of some important parts of the document format to help you get started.

The **Content** panel includes a status bar at the bottom that shows warnings or errors for your YAML document. For more information about how to create a YAML workflow document, see Create a YAML workflow document.

5. When you've completed your workflow, or if you want to save progress and come back to it later, choose **Create workflow**.

AWS CLI

Before you run the **create-workflow** command in the AWS CLI, you must create the YAML document that contains all of the configuration for your workflow. For more information, see Create a YAML workflow document.

The following example shows how to create a build workflow with the create-workflow AWS CLI command. The `--data` parameter refers to a YAML document that contains the build configuration for the workflow you create.

Example: Create workflow

```
aws imagebuilder create-workflow --name example-build-workflow --semantic-
version 1.0.0 --type BUILD --data file://example-build-workflow.yml
```

Output:

```
{
"workflowBuildVersionArn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/
build/example-build-workflow/1.0.0/1",
```

```
"clientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

The following example shows how to create a test workflow with the create-workflow AWS CLI command. The `--data` parameter refers to a YAML document that contains the build configuration for the workflow you create.

Example: Create test workflow

```
aws imagebuilder create-workflow --name example-test-workflow --semantic-
version 1.0.0 --type TEST --data file://example-test-workflow.yml
```

Output:

```
{
"workflowBuildVersionArn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/
test/example-test-workflow/1.0.0/1",
"clientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

Create a YAML workflow document

The YAML format definition document configures input, output, and workflow steps for the build and test stages of the image build process. You can start from templates that include standardized steps, or you can start from scratch to define your own workflow. Whether you use a template or start from scratch, you can customize the workflow to fit your needs.

Structure of a YAML workflow document

The YAML workflow document that Image Builder uses to perform image build and test actions is structured as follows.

- Workflow document identification
- Workflow document input parameters
- Workflow document steps
- Workflow document outputs

Workflow document identification

Uniquely identifies the workflow. This section can include the following attributes.

Field	Description	Type	Required
name	The name of the workflow document.	String	No
description	The document description.	String	No
schemaVersion	The document schema version, currently 1.0.	String	Yes

Example

```
---
name: sample-test-image
description: Workflow for a sample image, with extra configuration options exposed
 through workflow parameters.
schemaVersion: 1.0
```

Workflow document input parameters

This part of the workflow document defines input parameters that the caller can specify. If you don't have any parameters, you can leave this section out. If you do specify parameters, each parameter can include the following attributes.

Field	Description	Type	Required	Constraints
name	The name of the parameter.	String	Yes	

Field	Description	Type	Required	Constraints
description	The parameter description.	String	No	
default	The default value of the parameter, if no value is provided. If you don't include a default value in the parameter definition, the parameter value is required at runtime.	Matches the parameter data type.	No	
type	The data type of the parameter. If you don't include the data type in the parameter definition, the parameter type defaults to a string value required at runtime.	String	Yes	The data type of the parameter must be one of the following: • `string` • `integer` • `boolean` • `stringList`

Example

Specify the parameter in the workflow document.

```
parameters:
  - name: waitForActionAtEnd
    type: boolean
    default: true
    description: "Wait for an external action at the end of the workflow"
```

Use the parameter value in the workflow document.

```
$.parameters.waitForActionAtEnd
```

Workflow document steps

Specifies up to 15 step actions for the workflow. Steps run in the order that they're defined within the workflow document. In case of failure, a rollback runs in reverse order, starting with the step that failed, and working backward through prior steps.

Each step can refer to the output of any prior step actions. This is known as *chaining, or referencing*. To refer to output from a prior step action, you can use a JSONPath selector. For example:

```
$.stepOutputs.step-name.output-name
```

For more information, see Use dynamic variables in your workflow document.

> **Note**
>
> Even though the step itself doesn't have an output attribute, any output from a step action is included in `stepOutput` for the step.

Each step can include the following attributes.

Field	Description	Type	Required	Default value	Constraints
action	The workflow action that	String	Yes		Must be a supported step action

Field	Description	Type	Required	Default value	Constraints
	this step performs.				for Image Builder workflow documents.

Field	Description	Type	Required	Default value	Constraints
`if`, followed by a set of conditional statements that modify the `if` operator.	Conditional statements add flow of control decision points to the body of your workflow steps.	Dict	No		Image Builder supports the following conditional statements as modifiers to the `if` operator: • Branching conditions and modifiers: `if`, `and`, `or`, `not`. Branching conditions are specified on a line by themselves. • Comparison operators: `booleanEquals` , `numberEquals` ,

Field	Description	Type	Required	Default value	Constraints
					`numberGreaterThan` , `numberGreaterThanEquals` , `numberLessThan` , `numberLessThanEquals` , `stringEquals` .
description	The step description.	String	No		Empty strings are not allowed. If included, length must be 1-1024 characters.

Field	Description	Type	Required	Default value	Constraints
inputs	Contains parameters that the step action needs to run. You can specify key values as static values, or with a JSONPath variable that resolves to the correct data type.	Dict	Yes		
name	The name of the step. This name must be unique within the workflow document.	String	Yes		Length must be between 3-128 characters. Can include alphanumeric characters and _. No spaces.

Field	Description	Type	Required	Default value	Constraints
onFailure	Configures the action to take if the step fails, as follows. **Behavior** • `Abort` – Fails the step, fails the workflow, and doesn't run any remaining steps after the step that failed. If rollback is enabled, the rollback begins with the step that failed, and continues until all steps that allow it	String	No	`Abort`	`Abort` | `Continue`

Field	Description	Type	Required	Default value	Constraints
	are rolled back. • `Continue` – Fails the step, but continues to run remaining steps after the step that failed. In this case, there is no rollback.				
rollbackEnabled	Configures whether the step will be rolled back if a failure occurs. You can use a static Boolean value or a dynamic JSONPath variable that resolves to a Boolean value.	Boolean	No	`true`	`true` | `false` | or a JSONPath variable that resolves to true or false.

Field	Description	Type	Required	Default value	Constraints
timeoutSeconds	The maximum time, in seconds, that the step runs before failing and retrying, if retries apply.	Integer	No	Depends on the default defined for the step action, if applicable.	Between 1-86400 seconds (24 hrs maximum)

Example

```
steps:
  - name: LaunchTestInstance
    action: LaunchInstance
    onFailure: Abort
    inputs:
      waitFor: "ssmAgent"

  - name: ApplyTestComponents
    action: ExecuteComponents
    onFailure: Abort
    inputs:
      instanceId.$: "$.stepOutputs.LaunchTestInstance.instanceId"

  - name: TerminateTestInstance
    action: TerminateInstance
    onFailure: Continue
    inputs:
      instanceId.$: "$.stepOutputs.LaunchTestInstance.instanceId"

  - name: WaitForActionAtEnd
    action: WaitForAction
    if:
      booleanEquals: true
```

```
      value: "$.parameters.waitForActionAtEnd"
```

Workflow document outputs

Defines outputs for the workflow. Each output is a key value pair that specificies the name of the output and the value. You can use outputs to export data at runtime that subsequent workflows can use. This section is optional.

Each output that you define includes the following attributes.

Field	Description	Type	Required
name	The name of the output. The name must be unique across the workflows that you include in your pipeline.	String	Yes
value	The value for the output. The value of the string can be a dyanmic variable, such as an output file from a step action. For more information, see Use dynamic variables in your workflow document.	String	Yes

Example

Create an output image ID for the workflow document with step output from the `createProdImage` step.

```
outputs:
```

```
  - name: 'outputImageId'
    value: '$.stepOutputs.createProdImage.imageId'
```

Refer to the workflow output in the next workflow.

```
$.workflowOutputs.outputImageId
```

Supported step actions for your workflow document

This section includes details for the step actions that Image Builder supports.

Terms used in this section

AMI

Amazon Machine Image

ARN

Amazon Resource Name

Supported actions

- BootstrapInstanceForContainer
- CollectImageMetadata
- CollectImageScanFindings
- CreateImage
- ExecuteComponents
- LaunchInstance
- RunCommand
- RunSysPrep
- SanitizeInstance
- TerminateInstance
- WaitForAction

BootstrapInstanceForContainer

This step action runs a service script to bootstrap the instance with minimum requirements to run container workflows. Image Builder uses the **sendCommand** in the Systems Manager API to run this script. For more information, see AWS Systems Manager Run Command.

> **Note**
>
> The bootstrap script installs the AWS CLI and Docker packages that are prerequisites for Image Builder to successfully build Docker containers. If you don't include this step action, the image build could fail.

Default Timeout: 60 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID of the instance to bootstrap.	String	Yes		This must be the output instance ID from the workflow step that launched the instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that	String

Output name	Description	Type
	ran the bootstrap script on the instance.	
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	String

Example

Specify the step action in the workflow document.

```
- name: ContainerBootstrapStep
  action: BootstrapInstanceForContainer
  onFailure: Abort
  inputs:
      instanceId.$: $.stepOutputs.LaunchStep.instanceId
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.ContainerBootstrapStep.status
```

CollectImageMetadata

This step action is only valid for build workflows.

EC2 Image Builder runs [AWS Systems Manager (Systems Manager) Agent] on the EC2 instances it launches to build and test your image. Image Builder collects additional information about the instance used during the build phase with [Systems Manager Inventory]. This information includes the operating system (OS) name and version, as well as the list of packages and their respective versions as reported by your operating system.

> **Note**
>
> This step action only works for images that create AMIs.

Default Timeout: 30 minutes

Rollback: Image Builder rolls back any Systems Manager resources that were created during this step.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The build instance to apply the metadata settings to.	String	Yes		This must be the output instance ID from the workflow step that launched the build instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
osVersion	The operating system name and version collected from the build instance.	String
associationId	The Systems Manager association ID used for inventory collection.	String

Example

Specify the step action in the workflow document.

```
- name: CollectMetadataStep
  action: CollectImageMetadata
  onFailure: Abort
  inputs:
      instanceId: $.stepOutputs.LaunchStep.instanceId
```

Use output from the step action in the workflow document.

```
$.stepOutputs.CollectMetadataStep.osVersion
```

CollectImageScanFindings

If Amazon Inspector is enabled for your account and image scanning is enabled for your pipeline, this step action collects image scan findings reported by Amazon Inspector for your test instance. This step action is not available for build workflows.

Default Timeout: 120 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID for the instance that scanning ran on.	String	Yes		This must be the output instance ID from the workflow step that launched the instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that ran the script to collect findings.	String
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	String

Example

Specify the step action in the workflow document.

```
- name: CollectFindingsStep
  action: CollectImageScanFindings
  onFailure: Abort
  inputs:
      instanceId.$: $.stepOutputs.LaunchStep.instanceId
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.CollectFindingsStep.status
```

CreateImage

This step action creates an image from a running instance with the Amazon EC2 `CreateImage` API. During the creation process, the step action waits as necessary to verify that the resources have reached the correct state before it continues.

Default Timeout: 720 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The instance to create the new image from.	String	Yes		The instance for the provided instance ID must be in a `running` state when this step starts.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
imageId	The AMI ID of the image that's created.	String

Example

Specify the step action in the workflow document.

```
- name: CreateImageFromInstance
  action: CreateImage
  onFailure: Abort
  inputs:
      instanceId.$: "i-1234567890abcdef0"
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.CreateImageFromInstance.imageId
```

ExecuteComponents

This step action runs components that are specified in the recipe for the current image being built. Build workflows run build components on the build instance. Test workflows only run test components on the test instance.

Image Builder uses the **sendCommand** in the Systems Manager API to run components. For more information, see AWS Systems Manager Run Command.

Default Timeout: 720 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID for the instance that the component s should run on.	String	Yes		This must be the output instance ID from the workflow step that launched the instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that ran the components on the instance.	String

Output name	Description	Type
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	String

Example

Specify the step action in the workflow document.

```
- name: ExecComponentsStep
  action: ExecuteComponents
  onFailure: Abort
  inputs:
      instanceId: $.stepOutputs.LaunchStep.instanceId
```

Use output from the step action in the workflow document.

```
$.stepOutputs.ExecComponentsStep.status
```

LaunchInstance

This step action launches an instance in your AWS account and waits until the Systems Manager agent is running on the instance before moving on to the next step. The launch action uses settings from your recipe and infrastructure configuration resources that are associated with your image. For example, the instance type to launch comes from the infrastructure configuration. The output is the instance ID of the instance that it launched.

The `waitFor` input configures the condition that satisfies the step completion requirement.

Default Timeout: 60 minutes

Rollback: For build instances, rollback performs the action that you've configured in your infrastructure configuration resource. By default, build instances are terminated if image creation fails. However, there is a setting in the infrastructure configuration to keep the build instance for troubleshooting.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
imageIdOv erride	The image to use for launching the instance	String	No	Build stage: Image recipe base image Test stage: Output AMI from the build stage	Must be a valid AMI ID
instanceT ypesOverride	Image Builder tries each instance type in the list until it finds one that launches successfully	List of String	No	Instance types specified in your Infrastru cture Configura tion	Must be valid instance types
waitFor	The condition to wait for before completing the workflow step and moving on to the next step	String	Yes		Image Builder supports `ssmAgent`.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
instanceId	The instance ID of the instance that launched.	String

Example

Specify the step action in the workflow document.

```
- name: LaunchStep
  action: LaunchInstance
  onFailure: Abort
  inputs:
    waitFor: ssmAgent
```

Use output from the step action in the workflow document.

```
$.stepOutputs.LaunchStep.instanceId
```

RunCommand

This step action runs a command document for your workflow. Image Builder uses the **sendCommand** in the Systems Manager API to run it for you. For more information, see AWS Systems Manager Run Command.

Default Timeout: 12 hours

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID of the instance to run the command document on.	String	Yes		This must be the output instance ID from the workflow step that

Input name	Description	Type	Required	Default	Constraints
					launched the instance for this workflow.
documentName	The name of the Systems Manager command document to run.	String	Yes		
parameters	A list of key value pairs for any parameters that the command document requires.	dictionary<string, list<string>>	Conditional		
documentVersion	The command document version to run.	String	No	$DEFAULT	

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that ran the command document on the instance.	String

Output name	Description	Type
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	List of strings

Example

Specify the step action in the workflow document.

```
- name: RunCommandDoc
  action: RunCommand
  onFailure: Abort
  inputs:
    documentName: SampleDocument
    parameters:
        osPlatform:
          - "linux"
    instanceId.$: $.stepOutputs.LaunchStep.instanceId
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.RunCommandDoc.status
```

RunSysPrep

This step action uses the **sendCommand** in the Systems Manager API to run the `AWSEC2-RunSysprep` document for Windows instances before the build instance shuts down for the snapshot. These actions follow AWS best practices for hardening and cleaning the image.

Default Timeout: 60 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID of the instance to run the `AWSEC2-RunSysprep` document on.	String	Yes		This must be the output instance ID from the workflow step that launched the instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that ran the `AWSEC2-RunSysprep` document on the instance.	String
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	String

Example

Specify the step action in the workflow document.

```
- name: RunSysprep
```

```
  action: RunSysPrep
  onFailure: Abort
  inputs:
      instanceId.$: $.stepOutputs.LaunchStep.instanceId
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.RunSysprep.status
```

SanitizeInstance

This step action runs the recommended sanitize script for Linux instances before the build instance shuts down for the snapshot. The sanitize script helps ensure that the final image follows security best practices, and that build artifacts or settings that should not carry over to your snapshot are removed. For more information about the script, see Required post-build clean up. This step action does not apply to container images.

Image Builder uses the **sendCommand** in the Systems Manager API to run this script. For more information, see AWS Systems Manager Run Command.

Default Timeout: 60 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID of the instance to sanitize.	String	Yes		This must be the output instance ID from the workflow step that launched the instance for this workflow.

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
runCommandId	The ID of the Systems Manager **sendCommand** that ran the sanitize script on the instance.	String
status	The status returned from the Systems Manager **sendCommand**.	String
output	Output returned from the Systems Manager **sendCommand**.	String

Example

Specify the step action in the workflow document.

```
- name: SanitizeStep
  action: SanitizeInstance
  onFailure: Abort
  inputs:
      instanceId: $.stepOutputs.LaunchStep.instanceId
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.SanitizeStep.status
```

TerminateInstance

This step action terminate the instance with the instance id that's passed in as input.

Default Timeout: 30 minutes

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
instanceId	The ID of the instance to terminate.	String	Yes		

Outputs: There are no outputs for this step action.

Example

Specify the step action in the workflow document.

```
- name: TerminateInstance
  action: TerminateInstance
  onFailure: Continue
  inputs:
      instanceId.$: i-1234567890abcdef0
```

WaitForAction

This step action pauses the running workflow and waits to receive an external action from the Image Builder **SendWorkflowStepAction** API action. This step publishes an EventBridge event to your default EventBridge event bus with detail type `EC2 Image Builder Workflow Step Waiting`. The step can also send an SNS notification if you provide an SNS Topic ARN.

Default Timeout: 3 days

Rollback: There is no rollback for this step action.

Inputs: The following table includes supported inputs for this step action.

Input name	Description	Type	Required	Default	Constraints
snsTopicArn	An optional SNS topic ARN to send a notificat ion to when	String	No		

Input name	Description	Type	Required	Default	Constraints
	the workflow step is pending.				

Outputs: The following table includes outputs for this step action.

Output name	Description	Type
action	The action that the **SendWorkflowStepAction** API action returns.	String (RESUME or STOP)
reason	The reason for the returned action.	String

Example

Specify the step action in the workflow document.

```
- name: SendEventAndWait
  action: WaitForAction
  onFailure: Abort
  inputs:
    snsTopicArn: arn:aws:sns:us-west-2:111122223333:ExampleTopic
```

Use the output of the step action value in the workflow document.

```
$.stepOutputs.SendEventAndWait.reason
```

Use dynamic variables in your workflow document

You can use dynamic variables in your workflow documents to represent values that vary at runtime for your image creation process. Dynamic variable values are represented as JSONPath selectors with structural nodes that uniquely identify the target variable.

JSONPath dynamic workflow variable structure

```
$.<document structure>.[<step name>.]<variable name>
```

The first node after the root ($) refers to the workflow document structure, such as `stepOutputs`, or in the case of Image Builder system variables, `imageBuilder`. The following list contains supported JSONPath workflow document structure nodes.

Document structure nodes

- parameters - The workflow parameters
- stepOutputs - Outputs from a step in the same workflow doc
- workflowOutputs - Outputs from a workflow doc that already ran
- imagebuilder - Image Builder system variables

The `parameters` and `stepOutputs` document structure nodes include an optional node for the step name. This helps ensure unique variable names across all of the steps.

The final node in the JSONPath is the name of the target variable, such as `instanceId`.

Each step can refer to the output of any prior step actions with these JSONPath dynamic variables. This is also known as *chaining, or referencing*. To refer to output from a prior step action, you might use the following dynamic variable.

```
$.stepOutputs.step-name.output-name
```

When an input parameter refers to a dynamic variable, the chaining indicator (`.$`) must be attached to the end of the parameter name, as shown in the following example.

Example

```
  - name: ApplyTestComponents
    action: ExecuteComponents
    onFailure: Abort
    inputs:
      instanceId.$: "$.stepOutputs.LaunchTestInstance.instanceId"
```

Use Image Builder system variables

Image Builder provides the following system variables that you can use in your workflow document:

Variable name	Description	Type	Example value
cloudWatchLogGroup	The name of the CloudWatch Logs group for output logs. Format: `/aws/imagebuilder/` *`<recipe-name>`*	String	`/aws/imagebuilder/` *`sampleImageRecipe`*
cloudWatchLogStream	The name of the CloudWatch Logs stream for output logs.	String	*`1.0.0/1`*
collectImageMetadata	The setting that directs Image Builder whether to collect instance metadata.	Boolean	`true` | `false`
collectImageScanFindings	The current value of the setting that enables Image Builder to collect image scan findings.	Boolean	`true` | `false`
imageBuildNumber	The build version number of the image.	Integer	*`1`*
imageId	The AMI id of the base image.	String	*`ami-1234567890abcdef1`*

Variable name	Description	Type	Example value
imageName	The name of the image.	String	*sampleImage*
imageType	The image output type.	String	AMI | Docker
imageVersionNumber	The version number of the image.	String	*1.0.0*
instanceProfileName	The name of the instance profile role that Image Builder uses to launch build and test instances.	String	*SampleImageBuilderInstanceProfileRole*
platform	The operating system platform of the image that's built.	String	Linux | Windows | MacOS
s3Logs	A JSON object that contains configuration for the S3 logs that Image Builder writes.	JSON object	{'s3Logs': {'s3BucketName': '*sample-bucket* ', 's3KeyPrefix': '*ib-logs*'}}
securityGroups	The security group IDs that apply to build and test instances.	List [String]	*[sg-1234567890abcdef1, sg-11112222333344445]*

Variable name	Description	Type	Example value
sourceImageARN	The Amazon Resource Name (ARN) of the Image Builder image resource that the workflow uses for build and test stages.	String	arn:aws:imagebuilder:*us-east-1* :*111122223333* :image/*sampleImage* /*1.0.0/1*
subnetId	The ID of the subnet to launch the build and test instances into.	String	*subnet-1234567890abcdef1*
terminateInstanceOnFailure	The current value of the setting that directs Image Builder to terminate the instance on failure or keep it for troubleshooting.	Boolean	`true` | `false`
workflowPhase	The current stage that's running for the workflow execution.	String	`Build` | `Test`
workingDirectory	The path to the working directory.	String	`/tmp`

Use conditional statements in your workflow steps

Conditional statements begin with the `if` statement document attribute. The ultimate purpose of the `if` statement is to determine whether to run the step action or to skip it. If the `if` statement resolves to `true`, then the step action runs. If it resolves to `false`, Image Builder skips the step action and records a step status of `SKIPPED` in the log.

The `if` statement supports branching statements (`and`, `or`) and conditional modifiers (`not`). It also supports the following comparison operators that perform value comparisons (equal, less than, greater than) based on the data types it compares (string or number).

Supported comparison operators

- `booleanEquals`
- `numberEquals`
- `numberGreaterThan`
- `numberGreaterThanEquals`
- `numberLessThan`
- `numberLessThanEquals`
- `stringEquals`

Rules for branching statements and conditional modifiers

The following rules apply for branching statements (`and`, `or`) and conditional modifiers (`not`).

- Branching statements and conditional modifiers must appear on a line by themselves.
- Branching statements and conditional modifiers must follow level rules.
  - There can only be one statement at the parent level.
  - Each child branch or modifier starts a new level.

  For more information about levels, see Nested levels in conditional statements.
- Each branching statement must have at least one child conditional statement, but no more than ten.
- Conditional modifiers operate on only one child conditional statement.

Nested levels in conditional statements

Conditional statements operate at several levels in a section of their own. For example, the `if` statement attribute appears at the same level in your workflow document as the step name and action. This is the base of the conditional statement.

You can specify up to four levels of conditional statements, but only one statement can appear at the parent level. All other branching statements, conditional modifiers, or conditional operators are indented from there, one indent per level.

The following outline shows the maximum number of nested levels for a conditional statement.

```
base:
  parent:
    - child (level 2)
        - child (level 3)
            child (level 4)
```

`if` attribute

The `if` attribute specifies the conditional statement as a document attribute. This is level zero.

Parent level

This is the first level of nesting for conditional statements. There can be only one statement at this level. If you don't need branching or modifiers, this can be a conditional operator with no child statements. This level doesn't use dash notation, except for conditional operators.

Child levels

Levels two through four are considered child levels. Child statements can include branching statements, conditional modifiers, or conditional operators.

Example: Nested levels

The following example shows the maximum number of levels in a conditional statement.

```
if:
  and:                                  #first level
    - stringEquals: 'my_string'         #second level
      value: 'my_string'
    - and:                              #also second level
```

```
        - numberEquals: '1'        #third level
          value: 1
        - not:                     #also third level
            stringEquals: 'second_string'      #fourth level
            value: "diff_string"
```

Nesting rules

- Each branch or modifier at the child level starts a new level.
- Each level is indented.
- There can be a maximum of four levels, including one statement, modifier, or operator at the parent level, and up to three additional levels.

Conditional statement examples

This group of examples show various aspects of conditional statements.

Branching: and

The `and` branching statement operates on a list of expressions that are children of the branch, all of which must evaluate to `true`. Image Builder evaluates the expressions in the order that they appear in the list. If any expression evaluates to `false`, then processing stops and the branch is considered `false`.

The following example evaluates to `true`, because both expressions evaluate to `true`.

```
if:
  and:
    - stringEquals: 'test_string'
      value: 'test_string'
    - numberEquals: 1
      value: 1
```

Branching: or

The `or` branching statement operates on a list of expressions that are children of the branch, at least one of which must evaluate to `true`. Image Builder evaluates the expressions in the order that they appear in the list. If any expression evaluates to `true`, then processing stops and the branch is considered `true`.

The following example evaluates to `true`, even though the first expression is `false`.

```
if:
  or:
    - stringEquals: 'test_string'
      value: 'test_string_not_equal'
    - numberEquals: 1
      value: 1
```

Conditional modifier: not

The `not` conditional modifier negates the conditional statements that are children of the branch.

The following example evaluates to `true` when the `not` modifier negates the `stringEquals` conditional statement.

```
if:
  not:
    - stringEquals: 'test_string'
      value: 'test_string_not_equal'
```

Conditional statement: booleanEquals

The `booleanEquals` comparison operator compares Boolean values and returns true if the Boolean values exact match.

The following example determines if `collectImageScanFindings` is enabled.

```
if:
  - booleanEquals: true
    value: '$.imagebuilder.collectImageScanFindings'
```

Conditional statement: stringEquals

The `stringEquals` comparison operator compares two strings and returns true if the strings are an exact match. If either value isn't a string, Image Builder converts it to a string before it compares.

The following example compares the platform system variable to determine if the workflow is running on a Linux platform.

```
if:
  - stringEquals: 'Linux'
    value: '$.imagebuilder.Platform'
```

Conditional statement: numberEquals

The `numberEquals` comparison operator compares two numbers and returns true if the numbers are equal. The numbers to compare must be one of the following formats.

- Integer
- Float
- A string that matches the following regex pattern: `^-?[0-9]+(\.)?[0-9]+$`.

The following example comparisons all evaluate to `true`.

```
if:
  # Value provider as a number
  numberEquals: 1
  value: '1'

  # Comparison value provided as a string
  numberEquals: '1'
  value: 1

  # Value provided as a string
  numberEquals: 1
  value: '1'

  # Floats are supported
  numberEquals: 5.0
  value: 5.0

  # Negative values are supported
  numberEquals: -1
  value: -1
```

Manage custom image creation in Image Builder through a repeatable pipeline process

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 from the AWS CLI](#)
- [Get image pipeline details from the AWS CLI](#)

List image pipelines from the AWS CLI

The following example shows how to use the **list-image-pipelines** command in the AWS CLI to list all of your image pipelines.

```
aws imagebuilder list-image-pipelines
```

Get image pipeline details from the AWS CLI

The following example shows how to use the **get-image-pipeline** command in the AWS CLI to get the details about an image pipeline through 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 from the Image Builder console, through the Image Builder API, or with **imagebuilder** commands in the AWS CLI. You can use the **Create image pipeline** console wizard to guide you through the following steps:

- Specify pipeline details such as name, description, and resource tags.
- Select an AMI image recipe that includes a base image from quick-start Amazon managed images, images that you created or that were shared with you, or images that you subscribe to through the AWS Marketplace. The recipe also includes components that perform the following tasks on the EC2 instances that Image Builder uses to build your image:

  - Add and remove software
  - Customize settings and scripts
  - Run selected tests
- Specify workflows to configure image build and test steps that your pipeline runs.
- Define infrastructure configuration for your pipeline with default settings or settings that you configure yourself. Configuration includes the instance type and key pair to use for your image, security and network settings, log storage and troubleshooting settings, and SNS notifications.

  This is an *optional* step. Image Builder uses default settings for your infrastructure configuration if you don't define the configuration yourself.
- Define distribution settings to deliver your images to destination AWS Regions and accounts. You can specify a KMS key for encryption, configure AMI sharing or license configuration, or configure a launch template for the AMIs you distribute.

  This is an *optional* step. If you don't define the configuration yourself, Image Builder uses default naming for your output AMI, and distributes the AMI to the source Region. The source Region is the Region where you run the pipeline.

For more information and a step-by-step tutorial about using the **Create image pipeline** console wizard with default values where provided, see Tutorial: Create an image pipeline with output AMI from the Image Builder console wizard.

Contents

Create an AMI image pipeline from the AWS CLI

You can create an AMI image pipeline with a JSON file that contains configuration details as input to the **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 Image Builder.
- `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.

When you run the **create-image-pipeline** command in the AWS CLI, many of the configuration resources are optional. However, some of the resources have conditional requirements, depending on what type of image the pipeline creates. The following resources are required for AMI image pipelines:

- Image recipe ARN
- Infrastructure configuration ARN

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, while Linux and macOS use the forward slash (/).

2. Run the following command, using the file that you created as input.

```
aws imagebuilder create-image-pipeline --cli-input-json file://create-image-
pipeline.json
```

Update AMI image pipelines from the 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 from the AWS CLI 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 in Image Builder.
- **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.

Edit infrastructure configuration for your pipeline

Infrastructure configuration includes the following details that you can edit after creating the pipeline:

- The **Description** for your infrastructure 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 configuration.
- **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 Image Builder.

Update AMI image pipelines from the AWS CLI

You can update an AMI image pipeline using a JSON file as input to the **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

You can update an AMI image pipeline with the **update-image-pipeline** command in the AWS CLI as follows:

> **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, while Linux and macOS use 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
```

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 Tutorial: Create an image pipeline with output Docker container image from the Image Builder console wizard.

Contents

Create a container image pipeline from the AWS CLI

You can create a container image pipeline using a JSON file as input to the **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 Image Builder.
- `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.

When you run the **create-image-pipeline** command in the AWS CLI, many of the configuration resources are optional. However, some of the resources have conditional requirements, depending on what type of image the pipeline creates. The following resources are required for container image pipelines:

- Container recipe ARN
- Infrastructure configuration ARN

If you do not include a distribution configuration resource when you run the **create-image-pipeline** command, the output image is stored in the ECR repository that you specify as the target repository in your container recipe in the Region where you run the command. If you include a distribution configuration resource for your pipeline, the target repository that you have specified for the first Region in the distribution is used.

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, while Linux and macOS use 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 from the 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 from the AWS CLI 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 a new version of a container recipe.
- **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.

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 Image Builder.

Update container image pipelines from the AWS CLI

You can update a container image pipeline using a JSON file as input to the **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 (if included in the current pipeline)

> **Note**
>
> If the distribution settings resource is included, then the ECR repository that's specified as the target repository in the distribution settings for the Region where the command runs

(Region 1) takes precedence over the target repository that's specified in the container recipe.

Follow these steps to update a container image pipeline using the **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, while Linux and macOS use 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
```

Configure image pipeline workflows in Image Builder

With image workflows, you can customize the workflows that your pipeline runs to build and test images according to your needs. The workflows that you define run within the context of the Image Builder workflow framework. For more information about the stages that make up the workflow framework, see Manage build and test workflows for Image Builder images.

Build workflow

Build workflows run during the `Build` stage of the workflow framework. You can specify only one build workflow for your pipeline. Or you can skip the build entirely to configure a test-only pipeline.

Test workflow

Test workflows run during the `Test` stage of the workflow framework. You can specify up to ten test workflows for your pipeline. You can also skip tests entirely if you only want your pipeline to build.

Define test groups for test workflows

Test workflows are defined within test groups. You can run up to ten test workflows for your pipeline. You decide whether to run the test workflows in a specific order or to run as many as possible at the same time. How they run depends on how you define your test groups. The following scenarios demonstrate several ways that you can define your test workflows.

> **Note**
>
> If you use the console to create workflows, we recommend that you take time to plan how you want to run your test workflows before you define your test groups. In the console, you can add or remove test workflows and groups, but you can’t reorder them.

Scenario 1: Run one test workflow at a time

To run all of your test workflows one at a time, you can configure up to ten test groups, each with a single test workflow in it. Test groups run one at a time, in the order that you add them to your pipeline. This is one way to ensure that your test workflows run one at a time in a specific order.

Scenario 2: Run multiple test workflows at the same time

If the order doesn't matter, and you want to run as many test workflows as possible at the same time, you can configure a single test group and put the maximum number of test workflows in it. Image Builder starts up to five test workflows at the same time, and starts additional test workflows as others complete. If your goal is to run your test workflows as fast as possible, this is one way to do it.

Scenario 3: Mix and match

If you have a mixed scenario, with some test workflows that can run at the same time and some that should run one at a time, you can configure your test groups to accomplish this goal. The only limit to how you configure your test groups is the maximum number of test workflows that can run for your pipeline

Set workflow parameters in an Image Builder pipeline from the console

Workflow parameters function the same way for build workflows and test workflows. When you create or update a pipeline, you select build and test workflows that you want to include. If you defined parameters in the workflow document for a workflow that you selected, Image Builder displays them in the **Parameters** panel. The panel is hidden for workflows that don't have parameters defined.

Each parameter displays the following attributes that your workflow document defined:

- **Name** (*not editable*) – The name of the parameter.
- **Type** (*not editable*) – The data type for the parameter value.
- **Value** – The value for the parameter. You can edit the parameter value to set it for your pipeline.

Specify the IAM service role that Image Builder uses to run workflow actions

To run image workflows, Image Builder needs permission to perform workflow actions. You can specify the AWSServiceRoleForImageBuilder service-linked role, or you can specify your own custom role for service access, as follows.

- **Console** – In the pipeline wizard **Step 3 Define image creation process**, select the service-linked role or your own custom role from the **IAM role** list in the **Service access** panel.
- **Image Builder API** – In the CreateImage action request, specify the service-linked role or your own custom role as the value for the `executionRole` parameter.

To learn more about how to create a service role, see Creating a role to delegate permissions to an AWS service in the *AWS Identity and Access Management User Guide*.

Run your 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 **start-image-pipeline-execution** command in the AWS CLI to start an image pipeline manually. When you run this command, the pipeline builds and distributes a new image.

```
aws imagebuilder start-image-pipeline-execution --image-pipeline-arn
 arn:aws:imagebuilder:us-west-2:111122223333:image-pipeline/my-example-pipeline
```

To see what resources are created when the build pipeline runs, see Resources created.

Use cron expressions in 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 the Universal Coordinated Time (UTC) time zone by default, or you can specify the time zone. 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`	`, - * /`
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

Wildcard	Description
	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 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.

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 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 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.

For a summary of system generated events that Image Builder sends to EventBridge, see Event messages that Image Builder sends.

> **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*.

Integrate products and services in Image Builder

EC2 Image Builder integrates with AWS Marketplace and other AWS services and applications to help you create robust, secure custom machine images.

Products

Image Builder recipes can incorporate image products from AWS Marketplace and Image Builder managed components to provide specialized build and test functionality, as follows.

- **AWS Marketplace image products** – Use an image product from AWS Marketplace as the base image in your recipe to meet organizational standards, such as CIS Hardening. When you create a recipe from the Image Builder console, you can choose from your existing subscriptions, or search for a specific product from AWS Marketplace. When you create a recipe from the Image Builder API, CLI, or SDK, you can specify an image product Amazon Resource Name (ARN) to use as your base image.
- **Image Builder components** – Components that you specify in your recipes can perform build and test actions, for example, to install software or perform compliance validation. Some image products that you subscribe to from AWS Marketplace might include a companion component that you can use in your recipes. The CIS Hardened images include a matching AWSTOE component that you can use in your recipe to enforce CIS Benchmarks Level 1 guidelines for your configuration.

> **Note**
>
> For more information about compliance-related products, see Compliance products for your Image Builder images.

Services

Image Builder integrates with the following AWS services to provide detailed event metrics, logging, and monitoring. This information helps you track your activity, troubleshoot image build issues, and create automations based on event notifications.

- **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.

- **AWS CloudTrail** – Monitor Image Builder events that are sent to CloudTrail. For more information about CloudTrail integration with Image Builder, see Log Image Builder API calls using CloudTrail.

  To learn more about CloudTrail, including how to turn it on and find your log files, see the AWS CloudTrail User Guide.
- **Amazon CloudWatch Logs** – Monitor, store, and access your Image Builder log files with CloudWatch. Optionally, you can save your logs to an S3 bucket. To learn more about CloudWatch integration with Image Builder, see Monitor Image Builder logs with Amazon CloudWatch Logs.

  For more information about CloudWatch Logs, see What is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*.
- **Amazon Elastic Container Registry (Amazon ECR)** – Amazon ECR is a managed AWS container image registry service that is secure, scalable, and reliable. Container images that you create with Image Builder are stored in Amazon ECR in your source Region (where your build runs), and in any Regions where you distribute the container image. For more information about Amazon ECR, see the Amazon Elastic Container Registry 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*.
- **Amazon Inspector** – Discover vulnerabilities in your software and network settings with automatic scans for the EC2 test instance that Image Builder launches create a new image. Image Builder saves findings for your output image resource so that you can investigate and remediate after your test instance terminates. For more information about scans and pricing, see What is Amazon Inspector? in the *Amazon Inspector User Guide*.

  Amazon Inspector can also scan your ECR repositories if you configure enhanced scanning. For more information, see Scanning Amazon ECR container images in the *Amazon Inspector User Guide*.

> **Note**
>
> Amazon Inspector is a paid feature.

- **AWS License Manager** – You can attach a License Manager self-managed license to an output AMI during the distribution process. The license that you specify for the destination Region must already exist in that Region. For more information about self-managed licenses, see Self-managed licenses in License Manager.
- **AWS Marketplace** – See a list of your current AWS Marketplace product subscriptions, and search for image products directly from Image Builder. You can also use an image product that you've subscribed to as the base image for an Image Builder recipe. For more information about managing AWS Marketplace subscriptions, see Buying products in the *AWS Marketplace Buyer Guide*.
- **AWS Resource Access Manager (AWS RAM)** – With AWS RAM, you can share 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 Image Builder resources with AWS RAM.
- **Amazon Simple Notification Service (Amazon SNS)** – If configured, publish detailed messages about your image status to an SNS topic that you subscribe to. For more information about Amazon SNS, see What is Amazon SNS? in the *Amazon Simple Notification Service Developer Guide*.

Product and service integration topics

- Amazon EventBridge integration in Image Builder
- Amazon Inspector integration in Image Builder
- AWS Marketplace integration in Image Builder
- Amazon SNS integration in Image Builder
- Compliance products for your Image Builder images

Amazon EventBridge integration in Image Builder

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, and these events then run in parallel.

With EventBridge, you can 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 Amazon SQS 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 rules integrate with Image Builder image pipelines, see Use EventBridge rules with Image Builder pipelines.

Event messages that Image Builder sends

Image Builder sends event messages to EventBridge when there are significant changes in status for Image Builder resources. For example, when there's a state change for an image. The following examples show typical JSON event messages that Image Builder might send.

EC2 Image Builder Image State Change

Image Builder sends this event when the state changes for an image resource during image creation. For example, when the image status changes from one state to another, as follows:

- From `building` to `testing`
- From `testing` to `distribution`
- From `testing` to `failed`
- From `integrating` to `available`

```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "EC2 Image Builder Image State Change",
    "source": "aws.imagebuilder",
    "account": "111122223333",
    "time": "2024-01-18T17:50:56Z",
    "region": "us-west-2",
    "resources": ["arn:aws:imagebuilder:us-west-2:111122223333:image/
cmkencryptedworkflowtest-a1b2c3d4-5678-90ab-cdef-EXAMPLE22222/1.0.0/1"],
    "detail": {
        "previous-state": {
            "status": "TESTING"
        },
        "state": {
            "status": "AVAILABLE"
        }
    }
```

```
}
```

EC2 Image Builder CVE Detected

If you have CVE detection enabled for your image, Image Builder sends a message with the results whenever an image scan completes.


```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "EC2 Image Builder CVE Detected",
    "source": "aws.imagebuilder",
    "account": "111122223333",
    "time": "2023-03-01T16:59:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:imagebuilder:us-east-1:111122223333:image/test-image/1.0.0/1",
        "arn:aws:imagebuilder:us-east-1:111122223333:image-pipeline/test-pipeline"
    ],
    "detail": {
        "resource-id": "i-1234567890abcdef0",
        "finding-severity-counts": {
            "all": 0,
            "critical": 0,
            "high": 0,
            "medium": 0
        }
    }
}
```


EC2 Image Builder Workflow Step Waiting

Image Builder sends a message when a `WaitForAction` workflow step pauses to wait for an asynchronous action to complete.


```
{
    "version": "0",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "detail-type": "EC2 Image Builder Workflow Step Waiting",
    "source": "aws.imagebuilder",
    "account": "111122223333",
    "time": "2024-01-18T16:54:44Z",
```

```
    "region": "us-west-2",
    "resources": ["arn:aws:imagebuilder:us-west-2:111122223333:image/
workflowstepwaitforactionwithvalidsnstopictest-a1b2c3d4-5678-90ab-cdef-
EXAMPLE22222/1.0.0/1", "arn:aws:imagebuilder:us-west-2:111122223333:workflow/build/
build-workflow-a1b2c3d4-5678-90ab-cdef-EXAMPLE33333/1.0.0/1"],
    "detail": {
        "workflow-execution-id": "wf-a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "workflow-step-execution-id": "step-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "workflow-step-name": "TestAutoSNSStop"
    }
}
```


Amazon Inspector integration in Image Builder

When you activate security scanning with Amazon Inspector, it continuously scans machine images and running instances in your account for operating system and programming language vulnerabilities. If activated, security scanning is automatic, and Image Builder can save a snapshot of the findings from your test instance when you create a new image. Amazon Inspector is a paid service.

When Amazon Inspector discovers vulnerabilities in your software or network settings, it takes the following actions:

- Notifies you that there was a finding.
- Rates the severity of the finding. The severity rating categorizes vulnerabilities to help you prioritize your findings, and includes the following values:
  - Untriaged
  - Informational
  - Low
  - Medium
  - High
  - Critical
- Provides information about the finding, and links to additional resources for more detail.
- Offers remediation guidance to help you resolve the issues that generated the finding.

Configure security scans

If you've activated Amazon Inspector for your account, Amazon Inspector automatically scans the EC2 instances that Image Builder launches to build and test a new image. Those instances have a short lifespan during the build and test process, and their findings would normally expire as soon as those instances shut down. To help you investigate and remediate findings for your new image, Image Builder can optionally save any findings that Amazon Inspector identified on your test instance during the build process as a snapshot.

To configure security scans for your pipeline, see Configure security scans for Image Builder images in the AWS Management Console.

Review security findings

In the Image Builder console, you can view security findings for all of your Image Builder resources in one place. You can see all findings on the **Security findings** page in the **Security Overview** section, or you can group your findings by vulnerability, by image pipeline, or by image. The console defaults to display all security findings. The summary panel for the **All security findings** option shows the number of findings that you have for each severity level. For more information, see Manage security findings for Image Builder images in the AWS Management Console.

To learn more about Amazon Inspector vulnerability findings, see Understanding findings in Amazon Inspector in the *Amazon Inspector User Guide*.

AWS Marketplace integration in Image Builder

AWS Marketplace is a curated digital catalog where you can find and subscribe to third-party software, data, and services that help you build solutions to fit your business needs. AWS Marketplace brings authenticated buyers and registered sellers together with software listings from popular categories such as security, networking, storage, machine learning, and more.

An AWS Marketplace seller can be an independent software vendor (ISV), a reseller, or an individual who has something to offer that works with AWS products and services. When the seller submits a product in AWS Marketplace, they define the price of the product, and the terms and conditions of use. Buyers agree to the pricing, terms, and conditions set for the offer. To learn more about AWS Marketplace, see What is AWS Marketplace?

AWS Marketplace integration features

Image Builder integrates with AWS Marketplace to provide the following capabilities directly from the Image Builder console:

- Search for image products that are available in AWS Marketplace.
- Search for AWS Marketplace image products that deliver components.
- See a list of your current AWS Marketplace product subscriptions.
- Use an AWS Marketplace image product that you've subscribed to as the base image for an Image Builder recipe.
- Use AWS Marketplace components that you've subscribed to in an Image Builder recipe.

Image Builder integrates with AWS Marketplace to show image products and components that you've subscribed to. You can also search for AWS Marketplace image products and components from the **Discover products** page without leaving the Image Builder console.

The output AMI that Image Builder creates includes the product codes from AWS Marketplace image products and components. You can have up to four product codes for your final customized image.

AWS Marketplace subscriptions in Image Builder

The **Subscriptions** page in the AWS Marketplace section of the Image Builder console shows you a list of the AWS Marketplace products that you're currently subscribed to. Each subscribed product shows the following details:

- The product name. This is linked to the product detail page in AWS Marketplace. The product detail page for your subscribed product opens in a new tab in your browser.
- The **Publisher**. This is linked to the publisher detail page in AWS Marketplace. The publisher detail page opens in a new tab in your browser.
- The **Version** that you subscribed to.
- If there are any **Associated components** included with your subscribed product, Image Builder displays a link to the component detail.

At the top of the page, you can search for a specific product by name, or you can page through your results with the pagination controls. To use a subscribed image product in a new recipe, select a subscribed product and choose **Create new recipe**. Image Builder pre-selects the first product in your list by default.

> **Note**
>
> If you're looking for a product that you just subscribed to, and you don't see it in the list, use the refresh button at the top of the tab to refresh your results. It might take a few minutes for a new subscription to appear in the list.

Discover AWS Marketplace image products from the Image Builder console

This section focuses on AWS Marketplace image products to use as a base image in your recipe. For products that include associated software components, you can filter on the product owner in the console and in the API, SDK, and CLI. For more information, see List Image Builder components. For more information about finding, subscribing to, and using AWS Marketplace components, see Use AWS Marketplace components to customize your image.

Discover products

To find an AWS Marketplace image product from the Image Builder console, follow these steps:

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. From the navigation pane, choose **Discover products** in the **AWS Marketplace** section.
3. You can search for image products in the **Image products** tab on the **Discover products** page.

   Image Builder pre-filters products from AWS Marketplace to focus on machine images that you can use in your Image Builder recipes. For more information about AWS Marketplace integration with Image Builder, choose the tab that matches what you want to see.

   This tab contains two panels. On the left, the **Refine results** panel helps you filter your results to find the products that you want to subscribe to. On the right, the **Search products** panel shows the products that meet your filter criteria, and also gives you the option to search by product name.

   Refine results

   The following list shows just a few of the filters that you can apply to your product search:

   - Select one or more product categories, such as infrastructure software or machine learning.

- Choose the operating systems for your image product or choose all products for a specific operating system platform, for example **All Linux/Unix**.
- Choose one or more publishers to display their available products. Select the **Show All** link to display all of the publishers that have products that fit the filters that you've applied.

> **Note**
>
> Publisher names are not in alphabetical order. If you're looking for a specific publisher, like `Center for Internet Security`, you can enter part of the name in the search box at the top of the **All publishers** dialog. You should spell out the name, as an abbreviation, such as `CIS` might not produce the results that you're looking for.
> You can also browse the publisher names page by page.

Filter choices are dynamic. Each choice that you make affects your options for all of the other categories. There are thousands of products available in AWS Marketplace, so the more you can filter, the more likely you are to find what you want.

Search products

To find a specific product by name, you can enter part of the name in the search bar at the top of this panel. Each product result includes the following details:

- The product name and logo. Both of these are linked to the product detail page in AWS Marketplace. The detail page opens in a new tab in your browser. From there, you can subscribe to the image product if you want to use it in an Image Builder recipe. For more information, see Buying products in the *AWS Marketplace Buyer Guide*.

  If you subscribe to the image product in AWS Marketplace, switch back to the Image Builder tab in your browser, and refresh your list of subscribed image products to see it.

  > **Note**
  >
  > It might take a few minutes before your new subscription is available.

- The publisher name. This is linked to the publisher detail page in AWS Marketplace. The publisher detail page opens in a new tab in your browser.

- The product version.
- The product star rating, and direct links to the review section of the product detail page in AWS Marketplace. The detail page opens in a new tab in your browser.
- The first few lines of the product description.

Directly below the search bar, you can see how many results your search produced and what subset of those results is currently displayed. You can use additional controls on the right side of the panel to adjust your settings for the number of products to display at one time, and the sort order to apply to your results. You can also use the pagination control to page through your results.

Use an AWS Marketplace image product in Image Builder recipes

Open the **Create recipe** page and select an AWS Marketplace image product to use as your base image, as follows.

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. From the navigation pane, choose **Image recipes** in the **AWS Marketplace** section. This shows you a list of image recipes that you've created.
3. Choose **Create image recipe**. This opens the **Create recipe** page.
4. Enter your recipe **Name** and **Version** in the **Recipe details** section as usual.
5. In the **Base image** section, choose the **AWS Marketplace images** option. This shows you a list of the AWS Marketplace image products that you’ve subscribed to in the **Subscriptions** tab. You can choose your base image from the list.

   You can also search for other image products that are available in AWS Marketplace directly from the **AWS Marketplace** tab. Choose **Add products**, or open the **AWS Marketplace** tab directly. For more information about how to set filters and search in the AWS Marketplace, see [Discover AWS Marketplace image products from the Image Builder console](#).
6. Enter remaining details as usual. If any or your product subscriptions include build components, you can select them from the **Build components** list. Select `AWS Marketplace` from the component owner type list to see them, or select `Third party managed` for the CIS component.
7. Choose **Create recipe**.

Your final image can contain up to four product codes from AWS Marketplace image products and components. If your selected base image and components contain more than four product codes, Image Builder returns an error when you try to create the recipe.

Amazon SNS integration in Image Builder

Amazon Simple Notification Service (Amazon SNS) is a managed service that provides asynchronous message delivery from publishers to subscribers (also known as producers and consumers).

You can specify an SNS topic in your infrastructure configuration. When you create an image or run a pipeline, Image Builder can publish detailed messages about your image status to this topic. When the image status reaches one of the following states, Image Builder publishes a message:

- `AVAILABLE`
- `FAILED`

For an example SNS message from Image Builder, see SNS message format. If you want to create a new SNS topic, see Getting started with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

Encrypted SNS Topics

If your SNS topic is encrypted, you must grant permission in the AWS KMS key policy for the Image Builder service role to perform the following actions:

- `kms:Decrypt`
- `kms:GenerateDataKey`

> **Note**
>
> If your SNS topic is encrypted, the key that encrypts this topic must reside in the account where the Image Builder service runs. Image Builder can't send notifications to SNS topics that are encrypted with keys from other accounts.

Example KMS key policy addition

The following example shows the additional section that you add to the KMS key policy. Use the Amazon Resource Name (ARN) for the IAM service-linked role that Image Builder created under your account when you first created an Image Builder image. To learn more about the Image Builder service-linked role, see Use IAM service-linked roles for Image Builder.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:role/aws-service-role/
imagebuilder.amazonaws.com/AWSServiceRoleForImageBuilder"
    },
    "Action": [
      "kms:GenerateDataKey*",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }]
}
```

You can use one of the following methods to get the ARN.

AWS Management Console

To get the ARN for the service-linked role that Image Builder created under your account from the AWS Management Console, follow these steps:

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. Search for `ImageBuilder`, and choose the following **Role name** from the results: `AWSServiceRoleForImageBuilder`. This displays the role detail page.
4. To copy the ARN to your clipboard, choose the icon next to the ARN name.

AWS CLI

To get the ARN for the service-linked role that Image Builder created under your account from the AWS CLI, use the IAM get-role command, as follows.

```
aws iam get-role --role-name AWSServiceRoleForImageBuilder
```

Partial sample output:

```
{
    "Role": {
        "Path": "/aws-service-role/imagebuilder.amazonaws.com/",
        "RoleName": "AWSServiceRoleForImageBuilder",
        ...
        "Arn": "arn:aws:iam::123456789012:role/aws-service-role/
imagebuilder.amazonaws.com/AWSServiceRoleForImageBuilder",
        ...
}
```

SNS message format

After Image Builder publishes a message to your Amazon SNS topic, other services that subscribe to the topic can filter on the message format and determine if it meets criteria for further action. For example, a success message might initiate a task to update an AWS Systems Manager parameter store, or to launch an external compliance testing workflow for the output AMI.

The following example shows the JSON payload for a typical message that Image Builder publishes when a pipeline build runs to completion, and creates a Linux image.

```
{
  "versionlessArn": "arn:aws:imagebuilder:us-west-1:123456789012:image/example-linux-
image",
  "semver": 1237940039285380274899124227,
  "arn": "arn:aws:imagebuilder:us-west-1:123456789012:image/example-linux-
image/1.0.0/3",
  "name": "example-linux-image",
  "version": "1.0.0",
  "type": "AMI",
  "buildVersion": 3,
  "state": {
    "status": "AVAILABLE"
  },
  "platform": "Linux",
  "imageRecipe": {
    "arn": "arn:aws:imagebuilder:us-west-1:123456789012:image-recipe/example-linux-
image/1.0.0",
```

```
    "name": "amjule-barebones-linux",
    "version": "1.0.0",
    "components": [
      {
        "componentArn": "arn:aws:imagebuilder:us-west-1:123456789012:component/update-
linux/1.0.2/1"
      }
    ],
    "platform": "Linux",
    "parentImage": "arn:aws:imagebuilder:us-west-1:987654321098:image/amazon-linux-2-
x86/2022.6.14/1",
    "blockDeviceMappings": [
      {
        "deviceName": "/dev/xvda",
        "ebs": {
          "encrypted": false,
          "deleteOnTermination": true,
          "volumeSize": 8,
          "volumeType": "gp2"
        }
      }
    ],
    "dateCreated": "Feb 24, 2021 12:31:54 AM",
    "tags": {
      "internalId": "1a234567-8901-2345-bcd6-ef7890123456",
      "resourceArn": "arn:aws:imagebuilder:us-west-1:123456789012:image-recipe/example-
linux-image/1.0.0"
    },
    "workingDirectory": "/tmp",
    "accountId": "462045008730"
  },
  "sourcePipelineArn": "arn:aws:imagebuilder:us-west-1:123456789012:image-pipeline/
example-linux-pipeline",
  "infrastructureConfiguration": {
    "arn": "arn:aws:imagebuilder:us-west-1:123456789012:infrastructure-configuration/
example-linux-infra-config-uswest1",
    "name": "example-linux-infra-config-uswest1",
    "instanceProfileName": "example-linux-ib-baseline-admin",
    "tags": {
      "internalId": "234abc56-d789-0123-a4e5-6b789d012c34",
      "resourceArn": "arn:aws:imagebuilder:us-west-1:123456789012:infrastructure-
configuration/example-linux-infra-config-uswest1"
    },
    "logging": {
```

```
      "s3Logs": {
        "s3BucketName": "amzn-s3-demo-bucket"
      }
    },
    "keyPair": "example-linux-key-pair-uswest1",
    "terminateInstanceOnFailure": true,
    "snsTopicArn": "arn:aws:sns:us-west-1:123456789012:example-linux-ibnotices-
uswest1",
    "dateCreated": "Feb 24, 2021 12:31:55 AM",
    "accountId": "123456789012"
  },
  "imageTestsConfigurationDocument": {
    "imageTestsEnabled": true,
    "timeoutMinutes": 720
  },
  "distributionConfiguration": {
    "arn": "arn:aws:imagebuilder:us-west-1:123456789012:distribution-configuration/
example-linux-distribution",
    "name": "example-linux-distribution",
    "dateCreated": "Feb 24, 2021 12:31:56 AM",
    "distributions": [
      {
        "region": "us-west-1",
        "amiDistributionConfiguration": {}
      }
    ],
    "tags": {
      "internalId": "345abc67-8910-12d3-4ef5-67a8b90c12de",
      "resourceArn": "arn:aws:imagebuilder:us-west-1:123456789012:distribution-
configuration/example-linux-distribution"
    },
    "accountId": "123456789012"
  },
  "dateCreated": "Jul 28, 2022 1:13:45 AM",
  "outputResources": {
    "amis": [
      {
        "region": "us-west-1",
        "image": "ami-01a23bc4def5a6789",
        "name": "example-linux-image 2022-07-28T01-14-17.416Z",
        "accountId": "123456789012"
      }
    ]
  },
```

```
  "buildExecutionId": "ab0cd12e-34fa-5678-b901-2c3456d789e0",
  "testExecutionId": "6a7b8901-cdef-234a-56b7-8cd89ef01234",
  "distributionJobId": "1f234567-8abc-9d0e-1234-fa56b7c890de",
  "integrationJobId": "432109b8-afe7-6dc5-4321-0ba98f7654e3",
  "accountId": "123456789012",
  "osVersion": "Amazon Linux 2",
  "enhancedImageMetadataEnabled": true,
  "buildType": "USER_INITIATED",
  "tags": {
    "internalId": "901e234f-a567-89bc-0123-d4e567f89a01",
    "resourceArn": "arn:aws:imagebuilder:us-west-1:123456789012:image/example-linux-
image/1.0.0/3"
  }
}
```


The following example shows the JSON payload for a typical message that Image Builder publishes for a pipeline build failure for a Linux image.


```
{
  "versionlessArn": "arn:aws:imagebuilder:us-west-2:123456789012:image/my-example-
image",
  "semver": 1237940039285380274899124231,
  "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/my-example-image/1.0.0/7",
  "name": "My Example Image",
  "version": "1.0.0",
  "type": "AMI",
  "buildVersion": 7,
  "state": {
    "status": "FAILED",
    "reason": "Image Failure reason."
  },
  "platform": "Linux",
  "imageRecipe": {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image-recipe/my-example-
image/1.0.0",
    "name": "My Example Image",
    "version": "1.0.0",
    "description": "Testing Image recipe",
    "components": [
      {
        "componentArn": "arn:aws:imagebuilder:us-west-2:123456789012:component/my-
example-image-component/1.0.0/1"
      }
```

```
    ],
    "platform": "Linux",
    "parentImage": "ami-0cd12345db678d90f",
    "dateCreated": "Jun 21, 2022 11:36:14 PM",
    "tags": {
      "internalId": "1a234567-8901-2345-bcd6-ef7890123456",
      "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-recipe/my-
example-image/1.0.0"
    },
    "accountId": "123456789012"
  },
  "sourcePipelineArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-
example-image-pipeline",
  "infrastructureConfiguration": {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-configuration/
my-example-infra-config",
    "name": "SNS topic Infra config",
    "description": "An example that will retain instances of failed builds",
    "instanceTypes": [
      "t2.micro"
    ],
    "instanceProfileName": "EC2InstanceProfileForImageBuilder",
    "tags": {
      "internalId": "234abc56-d789-0123-a4e5-6b789d012c34",
      "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:infrastructure-
configuration/my-example-infra-config"
    },
    "terminateInstanceOnFailure": true,
    "snsTopicArn": "arn:aws:sns:us-west-2:123456789012:example-pipeline-notification-
topic",
    "dateCreated": "Jul 5, 2022 7:31:53 PM",
    "accountId": "123456789012"
  },
  "imageTestsConfigurationDocument": {
    "imageTestsEnabled": true,
    "timeoutMinutes": 720
  },
  "distributionConfiguration": {
    "arn": "arn:aws:imagebuilder:us-west-2:123456789012:distribution-configuration/my-
example-distribution-config",
    "name": "New distribution config",
    "dateCreated": "Dec 3, 2021 9:24:22 PM",
    "distributions": [
      {
```

```
        "region": "us-west-2",
        "amiDistributionConfiguration": {},
        "fastLaunchConfigurations": [
          {
            "enabled": true,
            "snapshotConfiguration": {
              "targetResourceCount": 2
            },
            "maxParallelLaunches": 2,
            "launchTemplate": {
              "launchTemplateId": "lt-01234567890"
            },
            "accountId": "123456789012"
          }
        ]
      }
    ],
    "tags": {
      "internalId": "1fecd23a-4f56-7f89-01e2-345678abbe90",
      "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:distribution-
configuration/my-example-distribution-config"
    },
    "accountId": "123456789012"
  },
  "dateCreated": "Jul 5, 2022 7:40:15 PM",
  "outputResources": {
    "amis": []
  },
  "accountId": "123456789012",
  "enhancedImageMetadataEnabled": true,
  "buildType": "SCHEDULED",
  "tags": {
    "internalId": "456c78b9-0e12-3f45-afb6-7e89b0f1a23b",
    "resourceArn": "arn:aws:imagebuilder:us-west-2:123456789012:image/my-example-
image/1.0.0/7"
  }
}
```


Compliance products for your Image Builder images

With constantly evolving security standards, it can be a challenge to maintain compliance and safeguard your organization from cyber threats. To help ensure that your custom images are compliant, and stay that way through automatic updates when publishers release new versions, Image Builder integrates with AWS Marketplace compliance products and Image Builder components.

Image Builder integrates with the following compliance products:

- **Center for Internet Security (CIS) Benchmarks hardening**

  You can use CIS Hardened Images and the related CIS hardening components to build custom images that comply with the latest CIS Benchmarks Level 1 guidelines. CIS Hardened Images are available in AWS Marketplace. To learn more about how to set up and use CIS Hardened Images and hardening components, see the Quick Start Guide: CIS Hardening Components for EC2 Image Builder in the CIS security wiki.

  > **Note**
  >
  > When you subscribe to a CIS Hardened Image, you also get access to the associated build component that runs a script to enforce CIS Benchmark Level 1 guidelines for your configuration. For more information, see CIS hardening components.

- **Security Technical Implementation Guides (STIG)**

  For STIG compliance, use can use Amazon-managed AWS Task Orchestrator and Executor (AWSTOE) STIG components in your Image Builder recipes. STIG components scan your build instance for misconfigurations and run a remediation script to correct issues that they find. We can't guarantee STIG compliance for the images that you build with Image Builder. You must work with your organization's compliance team to verify that your final image is compliant. For a complete list of AWSTOE STIG components that you can use in your Image Builder recipes, see Amazon managed STIG hardening components for Image Builder.

Monitor events and logs in Image Builder

To maintain the reliability, availability, and performance of your EC2 Image Builder pipelines, it's important to monitor events and logs. Events and logs help you see the big picture and dive down into the details when an API call fails. Image Builder integrate with services that can 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

- Log Image Builder API calls using CloudTrail
- Monitor Image Builder logs with Amazon CloudWatch Logs

Log Image Builder API calls using CloudTrail

EC2 Image Builder is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Image Builder. CloudTrail captures all API calls for 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 Amazon 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 Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Image Builder actions are logged by CloudTrail and 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, see the CloudTrail userIdentity element.

Monitor Image Builder logs with Amazon CloudWatch Logs

CloudWatch Logs support is turned on 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 (x.x.x/x):

```
ImageVersion/ImageBuildVersion
```

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 Image Builder issues.

Security in 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

Data protection and the AWS shared responsibility model in 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. You are also responsible for 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 users with AWS IAM Identity Center or 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 require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- 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 sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 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) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text 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 text 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 Image Builder

Image Builder encrypts data in transit and at rest by default with a service-owned KMS key, except for the following:

- **Custom components** – Image Builder encrypts custom components with your default KMS key, or a service-owned KMS key.
- **Image workflows** – Image Builder can encrypt your image workflows with a customer managed key if you specify the key during workflow creation. Image Builder handles encryption and decryption with your key to run the workflows that you've configured for your images.

You can manage your own keys through AWS KMS. However, you don't have permission to 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.

Encryption context

To provide an additional integrity and authenticity check on your encrypted data, you have the option of including an encryption context when you encrypt the data. When a resource is encrypted with an encryption context, AWS KMS cryptographically binds the context to the ciphertext. The resource can only be decrypted if the requester provides an exact, case-sensitive match for the context.

The policy examples in this section use an encryption context that resembles the Amazon Resource Name (ARN) of an Image Builder workflow resource.

Encrypt image workflows with a customer managed key

To add a layer of protection, you can encrypt your Image Builder workflow resources with your own customer managed key. If you use your customer managed key to encrypt the Image Builder workflows that you create, you must grant access in the key policy for Image Builder to use your key when it encrypts and decrypts workflow resources. You can revoke access at any time. However, Image Builder will not have access to any workflows that are already encrypted if you revoke access to the key.

The process to grant Image Builder access to use your customer managed key has two steps, as follows:

Step 1: Add key policy permissions for Image Builder workflows

To enable Image Builder to encrypt and decrypt workflow resources when it creates or uses those workflows, you must specify permissions in the KMS key policy.

This example key policy grants access for Image Builder pipelines to encrypt workflow resources during the creation process, and decrypt workflow resources to use them. The policy also grants access for key administrators. The encryption context and resource specification use a wildcard to cover all Regions where you have workflow resources.

As a prerequisite for using image workflows, you created an IAM workflow execution role that grants permission for Image Builder to run workflow actions. The principal for the first statement shown in the key policy example here must specify your IAM workflow execution role.

For more information about customer managed keys, see [Managing access to customer managed keys]() in the *AWS Key Management Service Developer Guide*.

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Allow access to build images with encrypted workflow",
   "Effect": "Allow",
   "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/YourImageBuilderExecutionRole"
   },
   "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
   ],
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "kms:EncryptionContext:aws:imagebuilder:arn":
 "arn:aws:imagebuilder:*:111122223333:workflow/*"
    }
   }
  },
  {
```

```
      "Sid": "Allow access for key administrators",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": [
        "kms:*"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*"
    }
  ]
}
```

Step 2: Grant key access to your workflow execution role

The IAM role that Image Builder assumes to run your workflows needs permission to use your customer managed key. Without access to your key, Image Builder won't be able to encrypt or decrypt your workflow resources with it.

Edit the policy for your workflow execution role to add the following policy statement.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToWorkflowKey",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-west-2:111122223333:key/key_ID",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:aws:imagebuilder:arn":
  "arn:aws:imagebuilder:*:111122223333:workflow/*"
        }
      }
    }
  ]
```

```
}
```

AWS CloudTrail events for image workflows

The following examples show typical AWS CloudTrail entries for encrypting and decrypting image workflows that are stored with a customer managed key.

Example: GenerateDataKey

This example shows what a CloudTrail event might look like when Image Builder invokes the AWS KMS **GenerateDataKey** API action from the Image Builder **CreateWorkflow** API action. Image Builder must encrypt a new workflow before it creates the workflow resource.


```
{
 "eventVersion": "1.08",
 "userIdentity": {
  "type": "AssumedRole",
  "principalId": "PRINCIPALID1234567890:workflow-role-name",
  "arn": "arn:aws:sts::111122223333:assumed-role/Admin/workflow-role-name",
  "accountId": "111122223333",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "sessionContext": {
   "sessionIssuer": {
    "type": "Role",
    "principalId": "PRINCIPALID1234567890",
    "arn": "arn:aws:iam::111122223333:role/Admin",
    "accountId": "111122223333",
    "userName": "Admin"
   },
   "webIdFederationData": {},
   "attributes": {
    "creationDate": "2023-11-21T20:29:31Z",
    "mfaAuthenticated": "false"
   }
  },
  "invokedBy": "imagebuilder.amazonaws.com"
 },
 "eventTime": "2023-11-21T20:31:03Z",
 "eventSource": "kms.amazonaws.com",
 "eventName": "GenerateDataKey",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "imagebuilder.amazonaws.com",
```

```
  "userAgent": "imagebuilder.amazonaws.com",
  "requestParameters": {
   "encryptionContext": {
    "aws:imagebuilder:arn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/build/
sample-encrypted-workflow/1.0.0/*",
    "aws-crypto-public-key": "key value"
   },
   "keyId": "arn:aws:kms:us-west-2:111122223333:alias/ExampleKMSKey",
   "numberOfBytes": 32
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "readOnly": true,
  "resources": [
   {
    "accountId": "111122223333",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLEzzzzz"
   }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
 }
```


Example: Decrypt

This example shows what a CloudTrail event might look like when Image Builder invokes the AWS KMS **Decrypt** API action from the Image Builder **GetWorkflow** API action. Image Builder pipelines need to decrypt a workflow resource before they can use it.


```
{
  "eventVersion": "1.08",
  "userIdentity": {
   "type": "AssumedRole",
   "principalId": "PRINCIPALID1234567890:workflow-role-name",
   "arn": "arn:aws:sts::111122223333:assumed-role/Admin/workflow-role-name",
   "accountId": "111122223333",
   "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
   "sessionContext": {
    "sessionIssuer": {
```

```
    "type": "Role",
    "principalId": "PRINCIPALID1234567890",
    "arn": "arn:aws:iam::111122223333:role/Admin",
    "accountId": "111122223333",
    "userName": "Admin"
   },
   "webIdFederationData": {},
   "attributes": {
    "creationDate": "2023-11-21T20:29:31Z",
    "mfaAuthenticated": "false"
   }
  },
  "invokedBy": "imagebuilder.amazonaws.com"
 },
 "eventTime": "2023-11-21T20:34:25Z",
 "eventSource": "kms.amazonaws.com",
 "eventName": "Decrypt",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "imagebuilder.amazonaws.com",
 "userAgent": "imagebuilder.amazonaws.com",
 "requestParameters": {
  "keyId": "arn:aws:kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLEzzzzz",
  "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
  "encryptionContext": {
   "aws:imagebuilder:arn": "arn:aws:imagebuilder:us-west-2:111122223333:workflow/build/
sample-encrypted-workflow/1.0.0/*",
   "aws-crypto-public-key": "ABC123def4567890abc12345678/90dE/F123abcDEF+4567890abc123D
+ef1=="
  }
 },
 "responseElements": null,
 "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
 "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
 "readOnly": true,
 "resources": [
  {
   "accountId": "111122223333",
   "type": "AWS::KMS::Key",
   "ARN": "arn:aws:kms:us-west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLEzzzzz"
  }
 ],
 "eventType": "AwsApiCall",
 "managementEvent": true,
```

```
 "recipientAccountId": "111122223333",
 "eventCategory": "Management"
}
```

Data storage in Image Builder

Image Builder doesn't 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.

Inter-network Traffic Privacy in 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 integration for Image Builder

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Image Builder.

Service user – If you use the Image Builder service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more 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 Image Builder, see Troubleshoot IAM issues in Image Builder.

Service administrator – If you're in charge of Image Builder resources at your company, you probably have full access to Image Builder. It's your job to determine which Image Builder features and resources your service users 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 Image Builder, see How Image Builder works with IAM policies and roles.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Image Builder. To view example Image Builder identity-based policies that you can use in IAM, see Image Builder identity-based policies.

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 Image Builder works with IAM policies and roles

Before you use IAM to manage access to Image Builder, learn 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 most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Image Builder

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Image Builder

To view examples of Image Builder identity-based policies, see Image Builder identity-based policies.

Resource-based policies within Image Builder

Supports resource-based policies: Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

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, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They 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 Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Image Builder

Supports policy actions: Yes

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 `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Image Builder actions, see Actions defined by EC2 Image Builder in the *Service Authorization Reference*.

Policy actions in Image Builder use the following prefix before the action:

```
imagebuilder
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
 "imagebuilder:action1",
 "imagebuilder:action2"
 ]
```

To view examples of Image Builder identity-based policies, see Image Builder identity-based policies.

Policy resources for Image Builder

Supports policy resources: Yes

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": "*"
```

To see a list of Image Builder resource types and their ARNs, see Resources defined by EC2 Image Builder in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by EC2 Image Builder.

To view examples of Image Builder identity-based policies, see Image Builder identity-based policies.

Policy condition keys for Image Builder

Supports service-specific policy condition keys: Yes

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 `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Image Builder condition keys, see Condition keys for EC2 Image Builder in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by EC2 Image Builder.

To view examples of Image Builder identity-based policies, see Image Builder identity-based policies.

ACLs in Image Builder

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Image Builder

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Image Builder

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Image Builder

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Image Builder

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Image Builder functionality. Edit service roles only when Image Builder provides guidance to do so.

Service-linked roles for Image Builder

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about the Image Builder service-linked role, see Use IAM service-linked roles for Image Builder.

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 ARN is made up of multiple nodes that help identify the resource and ensure that the name is unique. The last nodes in the name include several variations in formatting for the resource type, name, and ID. When Image Builder creates a resource it uses the following format:

`arn:aws:imagebuilder:`*`region`*`:`*`owner`*`:`*`resource-type/resource-name/version/build-version`*

> **Note**
>
> The build version is not always included in the resource ARN. However, some API operations, such as [GetComponent](), need the build version to uniquely identify a resource to retrieve.

For the resources that Image Builder uses in its recipes, such as the base image or components, the owner node can be one of the following:

- The account number of the resource owner
- For Amazon managed resources: `aws`

- For AWS Marketplace resources: `aws-marketplace`

The following example shows the ARN for a managed component to install the Amazon CloudWatch agent on Linux:

```
arn:aws:imagebuilder:us-east-1:aws:component/amazon-cloudwatch-agent-linux/1.0.1/1
```

This example shows the ARN for a fictitious managed component from the AWS Marketplace:

```
arn:aws:imagebuilder:us-east-1:aws-marketplace:component/example-linux-software-component/1.0.1
```

For more information about getting a list of components, including the use of an ownership filter, see List Image Builder components.

Example ARNs

The following are some examples of resource ARNs that you might specify in an IAM policy:

- Instance ARN

  ```
  "Resource": "arn:aws:imagebuilder:us-east-1:111122223333:instance/i-1234567890abcdef0"
  ```

- Wildcard (*) example to specify all instances for a given account

  ```
  "Resource": "arn:aws:imagebuilder:us-east-1:111122223333:instance/*"
  ```

- Wildcard (*) example to specify all versions of a managed image workflow

  ```
  "Resource": "arn:aws:imagebuilder:us-east-1:aws:workflow/build/build-image/*"
  ```

- Managed image ARN

  ```
  "Resource": "arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2-arm64/2024.12.17/1"
  ```

- Wildcard (*) example to specify all versions of a managed image

  ```
  "Resource": "arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2-arm64/x.x.x"
  ```

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 that Image Builder creates.

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 that are attached to the resource that Image Builder created. This allows you to manage Image Builder resources through defined tags.

Availability – This key is available to only the `CreateInfrastrucutreConfiguration` and `UpdateInfrastructureConfiguration` APIs.

imagebuilder:LifecyclePolicyResourceType

Works with [string operators](#).

Use this key to filter access by the Lifecycle resource type specified in the request.

The value for this key can be either `AMI_IMAGE` or `CONTAINER_IMAGE`.

Availability – This key is available to only the `CreateLifecyclePolicy` and `UpdateLifecyclePolicy` APIs.

imagebuilder:Ec2MetadataHttpTokens

Works with string operators.

Use this key to filter access by the EC2 Instance Metadata HTTP Token Requirement specified in the request.

This value for this key can be either `optional` or `required`.

Availability – This key is available to only the `CreateInfrastrucutreConfiguration` and `UpdateInfrastructureConfiguration` APIs.

imagebuilder:StatusTopicArn

Works with string operators.

Use this key to filter access by the SNS Topic ARN in the request to which terminal state notifications will be published.

Availability – This key is available to only the `CreateInfrastrucutreConfiguration` and `UpdateInfrastructureConfiguration` APIs.

Examples

To view examples of Image Builder identity-based policies, see Image Builder identity-based policies.

Image Builder resource-based policies

Resource-based policies 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.

For information about how to attach a resource-based policy to a component, image, or image recipe, see Share Image Builder resources with AWS RAM.

> **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 from the AWS CLI.

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. A user with administrative access 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 Use IAM service-linked roles for Image Builder.

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 user with administrative access can change the permissions for this role. However, doing so might break the functionality of the service.

Manage data perimeters for S3 bucket download access in Image Builder

EC2 Image Builder maintains two classes of AWS service-owned S3 buckets that contain downloadable resources needed to run Image Builder workloads in your account. If you use data perimeters to control access to Amazon S3 in your environment, you might need to explicitly allow access to these buckets. You can use the bucket ARN or bucket URL to allowlist these buckets, depending on how you control access to Amazon S3.

Component management bootstrapping scripts (Required)

This S3 bucket contains bootstrapping scripts to set up the AWSTOE application on the EC2 instances that are used to create images. Image Builder requires access to download the scripts to support build and testing for new images.

- **S3 bucket ARN:** `arn:`*`<AWS partition>`*`:s3:::ec2imagebuilder-managed-resources-`*`<AWS Region>`*`-prod`
- **S3 bucket URL:** `https://ec2imagebuilder-managed-resources-`*`<AWS Region>`*`.s3.`*`<AWS Region>`*`.`*`<AWS partition-specific domain name>`*

Managed components

This S3 bucket contains package payloads for Amazon managed components. Image Builder requires access to download any managed components that are configured in your recipes.

- **S3 bucket ARN:** `arn:`*`<AWS partition>`*`:s3:::ec2imagebuilder-toe-`*`<AWS Region>`*`-prod`
- **S3 bucket URL:** `https://ec2imagebuilder-toe-`*`<AWS Region>`*`.s3.`*`<AWS Region>`*`.`*`<AWS partition-specific domain name>`*

Image Builder identity-based policies

Topics

- Identity-based policy best practices
- Using the Image Builder console

Identity-based policy best practices

Identity-based policies determine whether someone can create, access, or delete 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM 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
- AWSImageBuilderFullAccess policy

For more information about Image Builder managed policies, see Use AWS managed policies for EC2 Image Builder.

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:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:*imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/*imagebuilder*"
        },
        {
            "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:ListBucket"
                ],
                "Resource": "arn:aws:s3::*:*imagebuilder*"
            }
        ]
    }
```

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.

Image Builder resource-based policies

For information about how to create a component, see Use components to customize your Image Builder image.

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*.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "IBPolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "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"}
      }
    }
  ]
}
```

Use AWS managed policies for EC2 Image Builder

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies 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".

To view the permissions for this policy, see AWSImageBuilderFullAccess in the *AWS Managed Policy Reference*.

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.

To view the permissions for this policy, see AWSImageBuilderReadOnlyAccess in the *AWS Managed Policy Reference*.

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. For more information about the Image Builder service-linked role, see Use IAM service-linked roles for Image Builder.

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 create, take snapshots of and register images (AMIs) that it creates and launch EC2 instances in your account. Image Builder uses related snapshots, volumes, network interfaces, subnets, security groups, license configuration and key pairs as required, as long as the image, instance, and volumes that are being created or used are tagged with `CreatedBy: EC2 Image Builder` or `CreatedBy: EC2 Fast Launch`.

  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, hosts, and tags on your Amazon EC2 resources.

  Image Builder can update image settings to enable or disable faster launching of Windows instances in your account, where the image is tagged with `CreatedBy: EC2 Image Builder`.

  Additionally, Image Builder can start, 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.

- **Amazon ECR** – Access is granted for Image Builder to create a repository if needed for container image vulnerability scans, and tag the resources it creates to limit the scope of its operations. Access is also granted for Image Builder to delete the container images that it created for the scans after it takes snapshots of the vulnerabilities.
- **EventBridge** – Access is granted for Image Builder to create and manage EventBridge rules.
- **IAM** – Access is granted for Image Builder to pass any role in your account to Amazon EC2, and to VM Import/Export.
- **Amazon Inspector** – Access is granted for Image Builder to determine when Amazon Inspector completes build instance scans, and to collect findings for images that are configured to allow it.
- **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 your account.
- **Systems Manager** – Access is granted for Image Builder to list Systems Manager commands and their invocations, inventory entries , describe instance information and automation execution statuses, describe hosts for instance placement support, and get command invocation details. Image Builder can also send automation signals, and stop automation executions for any resource in your account.

  Image Builder is able to issue run command invocations to any instance that is tagged `"CreatedBy": "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.

  Image Builder is able to read public Parameter Store Parameters, and read and update private Parameters prefixed with `/imagebuilder/` so that it can update the Parameter value with the output AMI IDs that Image Builder creates from a new build.

- **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.

To view the permissions for this policy, see AWSServiceRoleForImageBuilder in the *AWS Managed Policy Reference*.

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.

To view the permissions for this policy, see Ec2ImageBuilderCrossAccountDistributionAccess in the *AWS Managed Policy Reference*.

EC2ImageBuilderLifecycleExecutionPolicy policy

The **EC2ImageBuilderLifecycleExecutionPolicy** policy grants permissions for Image Builder to perform actions such as deprecate, disable, or delete Image Builder image resources and their underlying resources (AMIs, snapshots) to support automated rules for image lifecycle management tasks.

Permissions details

This policy includes the following permissions:

- **Amazon EC2** – Access is granted for Amazon EC2 to perform the following actions for Amazon Machine Images (AMIs) in the account that are tagged with `CreatedBy: EC2 Image Builder`.
  - Enable and disable an AMI.
  - Enable and disable image deprecation.

  - Describe and deregister an AMI.
  - Describe and modify AMI image attributes.
  - Delete volume snapshots that are associated with the AMI.
  - Retrieve tags for a resource.
  - Add or remove tags from an AMI for deprecation.
- **Amazon ECR** – Access is granted for Amazon ECR to perform the following batch actions on ECR repositories with the `LifecycleExecutionAccess: EC2 Image Builder` tag. Batch actions support automated container image lifecycle rules.
  - `ecr:BatchGetImage`
  - `ecr:BatchDeleteImage`

  Access is granted at the repository level for ECR repositories that are tagged with `LifecycleExecutionAccess: EC2 Image Builder`.
- **AWS Resource groups** – Access is granted for Image Builder to get resources based on tags.
- **EC2 Image Builder** – Access is granted for Image Builder to delete Image Builder image resources.

To view the permissions for this policy, see EC2ImageBuilderLifecycleExecutionPolicy in the *AWS Managed Policy Reference*.

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/`.
- **Amazon EC2** – Access is granted to describe volumes and snapshots, to create snapshots of volume or snapshot resources that Image Builder created, and to create tags for Image Builder resources.
- **Image Builder** – Access is granted to get any Image Builder or AWS Marketplace 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-`, or resources that have an ISO file extension.

To view the permissions for this policy, see EC2InstanceProfileForImageBuilder in the *AWS Managed Policy Reference*.

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-`.

To view the permissions for this policy, see EC2InstanceProfileForImageBuilderECRContainerBuilds in the *AWS Managed Policy Reference*.

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 page.

Change	Description	Date
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role to support the use of AWS Systems Manager (SSM) Parameter Store Parameters in recipes and during image distribution. • Added ssm:GetParameter to allow Image Builder to read public SSM Parameters and private SSM Parameters prefixed with `/imagebuilder/` so that they can be used in recipes. • Added ssm:PutParameter to allow Image Builder to update private SSM Parameters prefixed with `/imagebuilder/` with the output AMI IDs that Image Builder creates from a new build.	July 23, 2025
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role to support the import of Microsoft client OS ISO files as the base image. • Added ec2:RegisterImage to allow Image Builder	December 30, 2024

Change	Description	Date
	to create a snapshot and create and register an AMI whose baseline operating system was imported from verified ISO disk files.	
[EC2InstanceProfileForImageBuilder](#) – Update to an existing policy	Image Builder made the following changes to the instance profile policy to support image creation from disk image files. • Added the following ec2 actions: DescribeVolumes and DescribeSnapshots on all resources to retrieve details. Also added the following actions for resources that were created by Image Builder: CreateSnapshot for volume and snapshot resources, and CreateTags. Added s3:GetObject for resources with an "ISO" file extension.	December 30, 2024
[EC2InstanceProfileForImageBuilder](#) – Updated policy	Image Builder updated the `EC2InstanceProfileForImageBuilder` policy to allow Image Builder to get AWS Marketplace components.	December 2, 2024

Change	Description	Date
EC2ImageBuilderLifecycleExecutionPolicy – New policy	Image Builder added the new `EC2ImageBuilderLifecycleExecutionPolicy` policy that contains permissions for image lifecycle management.	November 17, 2023
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role to provide instance placement support. • Added ec2:DescribeHosts enables Image Builder to poll the hostId to determine when it's in a valid state to launch an instance. • Added ssm:GetCommandInvocation, API action to improve the method that Image Builder uses to get details of the command invocation.	October 19, 2023

Change	Description	Date
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role to provide instance placement support. • Added ec2:DescribeHosts enable Image Builder to poll the hostId to determine when it's in a valid state to launch an instance. • Added ssm:GetCommandInvocation, API action to improve the method that Image Builder uses to get details of the command invocation.	September 28, 2023

Change	Description	Date
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role to allow Image Builder workflows to collect vulnerability findings for both AMI and ECR container image builds. The new permissions support the CVE detection and reporting feature. • Added inspector2:ListCoverage and inspector2:ListFindings to allow Image Builder to determine when Amazon Inspector completes test instance scans, and to collect findings for images that are configured to allow it. • Added ecr:CreateRepository, with a requirement for Image Builder to tag the repository with `CreatedBy: EC2 Image Builder` (tag-on-create). Also added ecr:TagResource (required for tag-on-create) with the same CreatedBy tag constraint, and an additional constraint that requires the repository name to start with `image-builder-`	March 30, 2023

Change	Description	Date
	`*`. The name constraint prevents the escalation of privileges and prevents changes to repositories that Image Builder didn't create. • Added ecr:BatchDeleteImage for ECR repositories tagged with `CreatedBy : EC2 Image Builder`. This permission requires the repository name to start with `image-builder-*` . • Added event permissions for Image Builder to create and manage Amazon EventBridge managed rules that include `ImageBuilder-*` in the name.	
[AWSServiceRoleForImageBuilder]() – Update to an existing policy	Image Builder made the following changes to the service role: • Added License Manager licenses as a resource for the ec2:RunInstance call to allow customers to use base image AMIs that are associated with a license configuration.	March 22, 2022

Change	Description	Date
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role: - Added permissions for EC2 EnableFastLaunch API action, to enable and disable faster launching for Windows instances. - Tightened scope more for ec2:CreateTags action and resource tag conditions.	February 21, 2022
AWSServiceRoleForImageBuilder – Update to an existing policy	Image Builder made the following changes to the service role: - Added permissions to call the VMIE service to import a VM and create a base AMI from it. - Tightened scope for ec2:CreateTags action and resource tag conditions.	November 20, 2021
AWSServiceRoleForImageBuilder – 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

Change	Description	Date
AWSImageBuilderFullAccess – Update to an existing policy	Image Builder made the following changes to the full access role: • 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

Use IAM service-linked roles for 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 image in the AWS Management Console, the AWS CLI, or the AWS API, Image Builder creates the service-linked role for you.

The following actions create a new image:

- Run the pipeline wizard in the Image Builder console to create a custom image.
- Use one of the following API actions, or its corresponding AWS CLI command:
  - The **CreateImage** API action (**create-image** in the AWS CLI).
  - The **ImportVmImage** API action (**import-vm-image** in the AWS CLI).
  - The **StartImagePipelineExecution** API action (**start-image-pipeline-execution** in the AWS CLI).

> **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 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 outdated or unused Image Builder resources.

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 Role in the *IAM User Guide*.

Supported Regions for 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.

Troubleshoot IAM issues in Image Builder

Topics

- I am not authorized to perform an action in Image Builder

- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Image Builder resources

I am not authorized to perform an action in Image Builder

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `imagebuilder:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 imagebuilder:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `imagebuilder:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to 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 Image Builder. However, the action requires the service to have permissions that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

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 or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Image Builder supports these features, see How Image Builder works with IAM policies and roles.
- To learn 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*.
- To learn 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*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation resources for 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.

You can incorporate compliance products from AWS Marketplace or components from AWS Task Orchestrator and Executor (AWSTOE) into your Image Builder images to help ensure that your images are compliant. For more information, see Compliance products for your Image Builder images.

Data redundancy and resilience in 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 Image Builder

The AWS global network provides security capabilities and controls network access for services like EC2 Image Builder. For more information about the infrastructure security that AWS provides for its services, see the Infrastructure Security section in the *Introduction to AWS Security* whitepaper.

To send requests through the AWS global network for Image Builder API actions, your client software must comply with the following security guidelines:

- To send requests for Image Builder API actions, client software must use a supported version of Transport Layer Security (TLS).

> **Note**
>
> AWS is phasing out support for TLS versions 1.0 and 1.1. We strongly recommend that you update your client software to use TLS version 1.2 or later so that you can still connect. For more information, see this AWS Security Blog post.

- Client software must support cipher suites with perfect forward secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most current systems, such as Java 7 and later, support these modes.
- You must sign your API requests with an access key ID and a secret access key that is associated with an AWS Identity and Access Management (IAM) principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials for your requests.

Additionally, the EC2 instances that Image Builder uses to build and test images must have access to AWS Systems Manager.

Patch Management for Image Builder images

AWS provides updated managed AMIs each month that have the latest updates and security patches applied for the following operating systems. You can use these AMIs as the base image for your customizations. For more information, see Supported operating systems.

- Linux distributions including Amazon Linux 2, AL2023, Red Hat Enterprise Linux (RHEL), CentOS, Ubuntu, SUSE Linux Enterprise Server
- Windows Server 2016 and later

- macOS 10.14.x and later

After you create a custom image, you are responsible for Amazon EC2 system patching, per the shared responsibility model. If the EC2 instances in your application workload can be easily replaced, then it might be most efficient to update the base AMI and redeploy all compute nodes based on this image.

> **Note**
>
> For macOS patching, we recommend that you create a new version of your recipe that uses the latest managed AMI for the base image, and then build an updated custom image from the recipe and your other image build resources. If your Mac instances are not easily replaced, see the Update the operating system and software on Mac instances page in the *Amazon EC2 User Guide* for more information.

The following are two ways you can keep your Image Builder AMIs up to date.

- **AWS-provided patching components** – EC2 Image Builder provides the following build components that install all pending operating system updates:
  - `update-linux`
  - `update-windows`

  These components use the `UpdateOS` action module. For more information, see UpdateOS. 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.

  Component that uses the `UpdateOS` action module (Linux and Windows only. The `UpdateOS` action module is not supported for macOS.)

```
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 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.
- Periodically apply managed AMI updates to your macOS recipes and create new images to launch instances that have the latest security patches.

We strongly recommend 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.

IMDSv2 for Image Builder pipelines

When your Image Builder pipeline runs, it sends HTTP requests to launch EC2 instances that Image Builder uses to build and test your image. To configure the version of IMDS that your pipeline uses for the launch requests, set the `httpTokens` parameter in your Image Builder infrastructure configuration instance metadata settings.

Note

We recommend that you configure all EC2 instances that Image Builder launches from a pipeline build to use IMDSv2 so that instance metadata retrieval requests require a signed token header.

For more information about Image Builder infrastructure configuration, see Manage Image Builder infrastructure configuration. For more information about EC2 instance metadata options for Linux images, see Configure the instance metadata options in the Amazon EC2 User Guide. For Windows images, see Configure the instance metadata options in the Amazon EC2 User Guide.

Required post-build clean up

After Image Builder completes all of the build steps for your custom image, Image Builder prepares the build instance for testing and image creation. Before shutting down the build instance to create the snapshot, Image Builder performs the following clean up to ensure the security of your image:

Linux

The Image Builder pipeline runs a clean up script to help ensure that the final image follows security best practices, and to remove any build artifacts or settings that should not carry over to your snapshot. However, you can skip sections of the script, or override the user data entirely. Therefore, the images produced by Image Builder pipelines are not necessarily compliant with any specific regulatory criteria.

When the pipeline completes its build and test stages, Image Builder automatically runs the following clean-up script just before it creates the output image.

Important

If you override **User data** in your recipe, the script doesn't run. In that case, make sure that you include a command in your user data that creates an empty file named perform_cleanup. Image Builder detects this file and runs the clean-up script prior to creating the new image.

```
#!/bin/bash
if [[ ! -f {{workingDirectory}}/perform_cleanup ]]; then
    echo "Skipping cleanup"
    exit 0
else
    sudo rm -f {{workingDirectory}}/perform_cleanup
fi

function cleanup() {
    FILES=("$@")
    for FILE in "${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
};


# Clean up for cloud-init files
CLOUD_INIT_FILES=(
    "/etc/sudoers.d/90-cloud-init-users"
    "/etc/locale.conf"
    "/var/log/cloud-init.log"
    "/var/log/cloud-init-output.log"
)
if [[ -f {{workingDirectory}}/skip_cleanup_cloudinit_files ]]; then
    echo "Skipping cleanup of cloud init files"
else
    echo "Cleaning up cloud init files"
    cleanup "${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 ls /var/lib/cloud | sudo wc -l ) -gt 0 ]]; then
        echo "Deleting /var/lib/cloud/*"
        sudo rm -rf /var/lib/cloud/* || true
    fi;
fi;


# Clean up for temporary instance files
INSTANCE_FILES=(
    "/etc/.updated"
    "/etc/aliases.db"
    "/etc/hostname"
    "/var/lib/misc/postfix.aliasesdb-stamp"
    "/var/lib/postfix/master.lock"
    "/var/spool/postfix/pid/master.pid"
    "/var/.updated"
    "/var/cache/yum/x86_64/2/.gpgkeyschecked.yum"
)
if [[ -f {{workingDirectory}}/skip_cleanup_instance_files ]]; then
    echo "Skipping cleanup of instance files"
else
    echo "Cleaning up instance files"
    cleanup "${INSTANCE_FILES[@]}"
fi;


# Clean up for ssh files
SSH_FILES=(
    "/etc/ssh/ssh_host_rsa_key"
    "/etc/ssh/ssh_host_rsa_key.pub"
    "/etc/ssh/ssh_host_ecdsa_key"
    "/etc/ssh/ssh_host_ecdsa_key.pub"
    "/etc/ssh/ssh_host_ed25519_key"
    "/etc/ssh/ssh_host_ed25519_key.pub"
    "/root/.ssh/authorized_keys"
)
if [[ -f {{workingDirectory}}/skip_cleanup_ssh_files ]]; then
    echo "Skipping cleanup of ssh files"
else
    echo "Cleaning up ssh files"
    cleanup "${SSH_FILES[@]}"
    USERS=$(ls /home/)
    for user in $USERS; do
```

```
        echo Deleting /home/"$user"/.ssh/authorized_keys;
        sudo find /home/"$user"/.ssh/authorized_keys -type f -exec shred -zuf {} \;
    done
    for user in $USERS; do
        if [[ -f /home/"$user"/.ssh/authorized_keys ]]; then
            echo Failed to delete /home/"$user"/.ssh/authorized_keys;
            exit 1
        fi;
    done;
fi;


# Clean up for instance log files
INSTANCE_LOG_FILES=(
    "/var/log/audit/audit.log"
    "/var/log/boot.log"
    "/var/log/dmesg"
    "/var/log/cron"
)
if [[ -f {{workingDirectory}}/skip_cleanup_instance_log_files ]]; then
    echo "Skipping cleanup of instance log files"
else
    echo "Cleaning up instance log files"
    cleanup "${INSTANCE_LOG_FILES[@]}"
fi;

# Clean up for TOE files
if [[ -f {{workingDirectory}}/skip_cleanup_toe_files ]]; then
    echo "Skipping cleanup of TOE files"
else
    echo "Cleaning TOE files"
    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
fi

# Clean up for ssm log files
if [[ -f {{workingDirectory}}/skip_cleanup_ssm_log_files ]]; then
    echo "Skipping cleanup of ssm log files"
else
    echo "Cleaning up ssm log files"
    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
fi


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

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

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

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 [[ $( 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

sudo touch /etc/machine-id
```

Windows

After the Image Builder pipeline customizes Windows images, it runs the Microsoft Sysprep utility. These actions follow AWS best practices for hardening and cleaning the image.

macOS

The Image Builder pipeline runs a clean up script to help ensure that the final image follows security best practices, and to remove any build artifacts or settings that should not carry over to your snapshot. However, you can skip sections of the script, or override the user data entirely. Therefore, the images produced by Image Builder pipelines are not necessarily compliant with any specific regulatory criteria.

When the pipeline completes its build and test stages, Image Builder automatically runs the following clean-up script just before it creates the output image.

> **Important**
>
> If you override **User data** in your recipe, the script doesn't run. In that case, make sure that you include a command in your user data that creates an empty file named `perform_cleanup`. Image Builder detects this file and runs the clean-up script prior to creating the new image.

```
#!/bin/bash
if [[ ! -f {{workingDirectory}}/perform_cleanup ]]; then
  echo "Skipping cleanup"
  exit 0
else
  sudo rm -f {{workingDirectory}}/perform_cleanup
fi

function cleanup() {
  FILES=("$@")
  for FILE in "${FILES[@]}"; do
      if [[ -f "$FILE" ]]; then
          echo "Deleting $FILE";
          sudo rm -f $FILE;
      fi;
      if [[ -f $FILE ]]; then
          echo "Failed to delete '$FILE'. Failing."
          exit 1
      fi;
  done
};
```

```
# Clean up for cloud-init files
CLOUD_INIT_FILES=(
  "/etc/sudoers.d/90-cloud-init-users"
  "/etc/locale.conf"
  "/var/log/cloud-init.log"
  "/var/log/cloud-init-output.log"
)
if [[ -f {{workingDirectory}}/skip_cleanup_cloudinit_files ]]; then
  echo "Skipping cleanup of cloud init files"
else
  echo "Cleaning up cloud init files"
  cleanup "${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 rm -f {} \;
  fi;

  if [[ $( sudo ls /var/lib/cloud | sudo wc -l ) -gt 0 ]]; then
      echo "Deleting /var/lib/cloud/*"
      sudo rm -rf /var/lib/cloud/* || true
  fi;
fi;


# Clean up for temporary instance files
INSTANCE_FILES=(
  "/etc/.updated"
  "/etc/aliases.db"
  "/etc/hostname"
  "/var/lib/misc/postfix.aliasesdb-stamp"
  "/var/lib/postfix/master.lock"
  "/var/spool/postfix/pid/master.pid"
  "/var/.updated"
  "/var/cache/yum/x86_64/2/.gpgkeyschecked.yum"
)
if [[ -f {{workingDirectory}}/skip_cleanup_instance_files ]]; then
  echo "Skipping cleanup of instance files"
else
  echo "Cleaning up instance files"
  cleanup "${INSTANCE_FILES[@]}"
fi;


# Clean up for ssh files
```

```
SSH_FILES=(
  "/etc/ssh/ssh_host_rsa_key"
  "/etc/ssh/ssh_host_rsa_key.pub"
  "/etc/ssh/ssh_host_ecdsa_key"
  "/etc/ssh/ssh_host_ecdsa_key.pub"
  "/etc/ssh/ssh_host_ed25519_key"
  "/etc/ssh/ssh_host_ed25519_key.pub"
  "/root/.ssh/authorized_keys"
)
if [[ -f {{workingDirectory}}/skip_cleanup_ssh_files ]]; then
  echo "Skipping cleanup of ssh files"
else
  echo "Cleaning up ssh files"
  cleanup "${SSH_FILES[@]}"
  USERS=$(ls /home/)
  for user in $USERS; do
      echo Deleting /home/"$user"/.ssh/authorized_keys;
      sudo find /home/"$user"/.ssh/authorized_keys -type f -exec rm -f {} \;
  done
  for user in $USERS; do
      if [[ -f /home/"$user"/.ssh/authorized_keys ]]; then
          echo Failed to delete /home/"$user"/.ssh/authorized_keys;
          exit 1
      fi;
  done;
fi;


# Clean up for instance log files
INSTANCE_LOG_FILES=(
  "/var/log/audit/audit.log"
  "/var/log/boot.log"
  "/var/log/dmesg"
  "/var/log/cron"
  "/var/log/amazon/ec2/ec2-macos-init.log"
  "/var/log/amazon/ec2/ena-ethernet.log"
  "/var/log/amazon/ec2/system-monitoring.log"
)
if [[ -f {{workingDirectory}}/skip_cleanup_instance_log_files ]]; then
  echo "Skipping cleanup of instance log files"
else
  echo "Cleaning up instance log files"
  cleanup "${INSTANCE_LOG_FILES[@]}"
fi;
```

```
# Clean up for TOE files
if [[ -f {{workingDirectory}}/skip_cleanup_toe_files ]]; then
  echo "Skipping cleanup of TOE files"
else
  echo "Cleaning TOE files"
  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 rm -f {} \;
  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
fi

# Clean up for ssm log files
if [[ -f {{workingDirectory}}/skip_cleanup_ssm_log_files ]]; then
  echo "Skipping cleanup of ssm log files"
else
  echo "Cleaning up ssm log files"
  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 rm -f {} \;
  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
fi


if [[ $( sudo find /var/log/sa/sa* -type f | sudo wc -l ) -gt 0 ]]; then
    echo "Deleting /var/log/sa/sa*"
    sudo rm -f /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

if [[ $( sudo find /var/lib/dhclient/dhclient*.lease -type f | sudo wc -l ) -gt
 0 ]]; then
        echo "Deleting /var/lib/dhclient/dhclient*.lease"
        sudo rm -f /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

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 rm -f {} \;
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

if [[ -f "/var/lib/rsyslog/imjournal.state" ]]; then
        echo "Deleting /var/lib/rsyslog/imjournal.state"
        sudo rm -f /var/lib/rsyslog/imjournal.state
        sudo rm -f /var/lib/rsyslog/imjournal.state
fi
```

```
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 rm -f {} \;
      sudo rm -rf /var/log/journal/*
fi

sudo touch /etc/machine-id
```

Override the Linux clean up script

Image Builder creates images that are secure by default and follow our security best practices. However, some more advanced use-cases might require you to skip one or more sections of the built-in clean up script. If you do need to skip some of the clean up, we strongly recommend that you test your output AMI to ensure the security of your image.

> **Important**
>
> Skipping sections in the clean up script can result in sensitive information, such as owner account details or SSH keys being included in the final image, and in any instance launched from that image. You might also experience problems with launching in different Availability Zones, Regions, or accounts.

The following table outlines the sections of the clean up script, the files that are deleted in that section, and the file names that you can use to flag a section that Image Builder should skip. To skip a specific section of the clean up script, you can use the CreateFile component action module or a command in your user data (if overriding) to create an empty file with the name specified in the **Skip section file name** column.

> **Note**
>
> The files that you create to skip a section of the clean up script should not include a file extension. For example, if you want to skip the `CLOUD_INIT_FILES` section of the script, but you create a file named `skip_cleanup_cloudinit_files.txt`, Image Builder will not recognize the skip file.

Input

Clean up section	Files removed	Skip section file name
CLOUD_INIT_FILES	/etc/sudoers.d/90-cloud-init-users /etc/locale.conf /var/log/cloud-init.log /var/log/cloud-init-output.log	skip_cleanup_cloudinit_files
INSTANCE_FILES	/etc/.updated /etc/aliases.db /etc/hostname /var/lib/misc/postfix.aliasesdb-stamp /var/lib/postfix/master.lock /var/spool/postfix/pid/master.pid /var/.updated /var/cache/yum/x86_64/2/.gpgkeyschecked.yum	skip_cleanup_instance_files
SSH_FILES	/etc/ssh/ssh_host_rsa_key /etc/ssh/ssh_host_rsa_key.pub	skip_cleanup_ssh_files

Clean up section	Files removed	Skip section file name

	`/etc/ssh/ssh_host_ecdsa_key`
	`/etc/ssh/ssh_host_ecdsa_key.pub`
	`/etc/ssh/ssh_host_ed25519_key`
	`/etc/ssh/ssh_host_ed25519_key.pub`
	`/root/.ssh/authorized_keys`
	`/home/<all users>/.ssh/authorized_keys;`	

`INSTANCE_LOG_FILES`	`/var/log/audit/audit.log`
	`/var/log/boot.log`
	`/var/log/dmesg`
	`/var/log/cron`
		`skip_cleanup_instance_log_files`

`TOE_FILES`	`{{workingDirectory}}/TOE_*`	`skip_cleanup_toe_files`
`SSM_LOG_FILES`	`/var/log/amazon/ssm/*`	`skip_cleanup_ssm_log_files`

Troubleshoot Image Builder issues

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 `workflow execution ID` in the failure message, such as the one in the following example output:

```
Workflow Execution ID: wf-12345abc-6789-0123-abc4-567890123abc failed with reason: …
```

Image Builder arranges and directs image build actions through a series of steps that are defined for the runtime stages in its standard image creation process. The build and test stages of the process each have an associated workflow. When Image Builder runs a workflow to build or test a new image, it generates a workflow metadata resource that keeps track of runtime details.

Container images have an additional workflow that runs during distribution.

Research details for runtime instance failures for your workflow

To troubleshoot a runtime failure for your workflow, you can call the GetWorkflowExecution and ListWorkflowStepExecutions API actions with your `workflow execution ID`.

Review workflow runtime logs

- **Amazon CloudWatch Logs**

Image Builder publishes detailed workflow execution logs to the following Image Builder CloudWatch Logs group and stream:

LogGroup:

```
/aws/imagebuilder/ImageName
```

LogStream (x.x.x/x):

```
ImageVersion/ImageBuildVersion
```

With CloudWatch Logs, you can search log data with filter patterns. For more information, see Search log data using filter patterns in the *Amazon CloudWatch Logs User Guide*.

- **AWS CloudTrail**

  All build activity is also logged in CloudTrail if it's activated in your account. You can filter CloudTrail events by the source `imagebuilder.amazonaws.com`. Alternatively, you can search for the Amazon EC2 instance ID that is returned in the execution log to see more details about the pipeline execution.

- **Amazon Simple Storage Service (S3)**

  If you've specified an S3 bucket name and key prefix in your infrastructure configuration, the workflow step runtime log path follows this pattern:

```
S3://S3BucketName/KeyPrefix/ImageName/ImageVersion/ImageBuildVersion/
WorkflowExecutionId/StepName
```

  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 `application.log`, to diagnose the cause of the problem on the instance.

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.

You can also change the instance settings with the **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.

Troubleshooting scenarios

This section lists the following detailed troubleshooting scenarios:

- Access denied – status code 403
- Build times out while verifying the Systems Manager Agent availability on the build instance
- Windows secondary disk is offline at launch
- Build fails with CIS hardened base image
- AssertInventoryCollection fails (Systems Manager Automation)

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 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.

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 Access AWS services through 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 Use IAM service-linked roles for Image Builder.

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*.

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.

AssertInventoryCollection fails (Systems Manager Automation)

Description

Systems Manager Automation shows a failure in the `AssertInventoryCollection` automation step.

Cause

You or your organization might have created a Systems Manager State Manager association that collects inventory information for EC2 instances. If enhanced image metadata collection is enabled for your Image Builder pipeline (this is the default), Image Builder attempts to create a new inventory association for the build instance. However, Systems Manager does not allow multiple inventory associations for managed instances, and prevents a new association if one already exists. This causes the operation to fail, and results in a failed pipeline build.

Solution

To resolve this issue, turn off enhanced image metadata collection, using one of the following methods:

- Update your image pipeline in the console, to clear the **Enable enhanced metadata collection** check box. Save your changes and run a pipeline build.

  For more information about updating your AMI image pipeline using the EC2 Image Builder console, see Update AMI image pipelines from the console. For more information about updating your container image pipeline using the EC2 Image Builder console, see Update a container image pipeline from the console.
- You can also update your image pipeline with the **update-image-pipeline** command in the AWS CLI. To do this, include the `EnhancedImageMetadataEnabled` property in your JSON file, set to `false`. The following example shows the property set to `false`.

```
{
    "name": "MyWindows2019Pipeline",
    "description": "Builds Windows 2019 Images",
    "enhancedImageMetadataEnabled": false,
    "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"
}
```

To prevent this from happening for new pipelines, clear the **Enable enhanced metadata collection** check box when you create a new pipeline using the EC2 Image Builder console, or set the value of the `EnhancedImageMetadataEnabled` property in your JSON file to `false` when you create your pipeline using the AWS CLI.

Documentation history of changes to the 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: 2025-04-30**

Change	Description	Date
IAM policy update: service role policy	Updated the service-linked role policy to support the use of SSM Parameters in recipes and during image distribut ion. For more information, see the AWSServiceRoleForI mageBuilder policy.	July 23, 2025
STIG Q2 2025 updates	Updated Windows STIG versions and applied STIGS for 2025 second quarter release.	June 26, 2025
STIG Q1 2025 updates	Updated Windows STIG versions and applied STIGS and SCAP component versions for 2025 first quarter release.	May 5, 2025
Feature release: Support SSM Parameters	Image Builder now supports the use of AWS Systems Manager(SSM) Parameter Store Parameters in recipes and during image distribution.	April 30, 2025

STIG Q4 updates	Updated Windows STIG versions and applied STIGS for 2024 fourth quarter rel ease.	February 4, 2025
IAM policy update: service role policy	Updated the service-linked role policy to support image creation from imported offici al Microsoft client OS ISO files. For more information, see the AWSServiceRoleForI mageBuilder policy.	December 30, 2024
IAM policy update: instance profile policy	Updated the instance profile role policy to support image creation from disk image files. For more information, see the EC2InstanceProfileForImageB uilder policy.	December 30, 2024
Feature release: ISO to AMI	Image Builder can now create an image from official ISO disk files for Microsoft Windows 11 and later client operating systems.	December 30, 2024
STIG Q4 updates	Updated Linux STIG versions and applied STIGS for 2024 fourth quarter release. Add ed information about two new input parameters for the Linux components.	December 10, 2024

IAM policy update: instance profile policy	Updated the instance profile policy to grant access to get AWS Marketplace component s. For more information, see the EC2InstanceProfile ForImageBuilder policy.	December 2, 2024
Feature release: AWS Marketplace components	Added support for AWS Marketplace software components.	December 2, 2024
Feature release: macOS support	Added support for macOS images.	October 22, 2024
Document support for logical operators in AWS Task Orchestrator and Executor	Use logical operators to add or modify condition al expressions in your component document.	August 16, 2024
Document support for conditional constructs in AWS Task Orchestrator and Executor	Use conditional construct s like the "if" statement to direct the flow of control for component actions in your component document.	August 16, 2024
Document support for comparison operators in AWS Task Orchestrator and Executor	Use comparison operators to compare values in your component document.	August 16, 2024
Added Assert action module	Added documentation for the `ExecuteDocument` action module under `General execution`.	August 14, 2024
Operating system support	Added support for RHEL 9 and Ubuntu 24.04 LTS.	August 2, 2024

Document update: Reorganize content	Reorganized documentation to improve presentation and navigation.	June 21, 2024
STIG Q2 updates	Updated Linux STIG versions and applied STIGS for 2024 second quarter release. Added support for RHEL 9, CentOS Stream 9, and Ubuntu 22.04. There were no changes to Windows versions.	May 10, 2024
STIG Q1 updates	Updated Linux STIG versions and applied STIGS for 2024 first quarter release. There were no changes to Windows versions.	February 23, 2024
STIG Q1 updates	Updated Linux STIG versions and applied STIGS for 2024 first quarter release. There were no changes to Windows versions.	February 23, 2024
Feature release: Image workflow management	With image workflows, you have more flexibility, visibilit y, and control over the image creation process. You can customize build and test steps for your workflows, or you can use the Image Builder default workflow.	December 12, 2023

STIG Q4 updates	Updated Linux STIG versions and applied STIGS for 2023 fourth quarter release. There were no changes to Windows versions. Also updated Linux and Windows SCAP for new component, software, and benchmark numbers.	December 7, 2023
Feature release: Image lifecycle management	With image lifecycle management policies and rules, you can define your resource management strategy to ensure that outdated images and their associated resources go through a process of tagging and removal.	November 17, 2023
IAM policy update: service role	Updated the service-linked role policy for instance placement support. For more information, see the AWSServiceRoleForImageBuilder policy.	October 19, 2023
STIG Q3 updates	Updated STIG versions and applied STIGS for 2023 third quarter release. Additionally updated messaging to clarify that third-party packages are not automatically installed, with very few exceptions. All skipped STIGs are logged.	October 5, 2023

New STIG Versions	Updated STIG versions and applied STIGS for 2023 second quarter release.	May 3, 2023
New STIG Versions	Updated STIG versions and applied STIGS for 2023 first quarter release. Added support for AL2023.	April 14, 2023
Update supported Regions for AWSTOE	Added AWSTOE support for the following AWS Regions: Asia Pacific (Hyderabad), Asia Pacific (Jakarta), Europe (Zurich), Europe (Spain), and Middle East (UAE).	April 13, 2023
AWSTOE application download updates	Updated the signature for the AWSTOE installation download on Windows. Also updated TLS note that application downloads from S3 buckets now require TLS version 1.2 or later.	March 31, 2023
Feature release: Enhanced build workflows	Added runtime details for image builds in the new workflow tab in the image build version details. Improved information for troubleshooting builds.	March 30, 2023

Feature release: CVE detection and reporting	For accounts that have activated Amazon Inspector scans, Image Builder can capture the common vulnerability and exposures (CVE) findings from Amazon Inspector during the test stage of the build process for new images, including container images stored in Amazon ECR. Image Builder creates a snapshot of the findings to support detail analysis. Image Builder also reports on findings counts that can be filtered by account, by pipeline, or by image, with the ability to drill down on details.	March 30, 2023
Added version history	Added version history to the Windows and Linux sections.	February 17, 2023
New STIG Versions	Updated STIG versions and applied STIGS for 2022 fourth quarter release.	February 1, 2023
Feature release: AWS Marketplace integration and CIS hardening	Added AWS Marketplace integration to easily find and use a subscribed image as the baseline for a new custom image, including CIS Hardened Images and a new CIS Hardening component from the Center for Internet Security.	January 13, 2023

CIS hardening components	Added CIS hardening components that are owned and maintained by CIS.	January 13, 2023
New STIG Versions	Introduced Ubuntu support, updated STIG versions, and applied STIGS for 2022 second quarter release.	July 20, 2022
Document update: Navigation for Create YAML component document page	Moved the Create YAML component document content to its own page, and updated other pages to reference it.	June 7, 2022
New STIG Versions	Updated STIG versions and applied STIGS for 2022 first quarter release.	April 25, 2022
Added ExecuteDocument action module	Added documentation for the `ExecuteDocument` action module under `General execution`.	March 28, 2022
Feature release: Support for faster launching Windows AMI	Added distribution configuration settings to support faster launching for Windows AMIs.	February 21, 2022
Maintenance release: Update AWSTOE binary thumbprint	Updated binary thumbprint for AWSTOE signer certificate.	February 18, 2022
Feature release: Configure input for AWSTOE	Added support for using a JSON configuration file as input for the AWSTOE **run** command.	February 3, 2022

New STIG Versions	Updated STIG versions and applied STIGS for 2021 fourth quarter release. Also add ed a section for new SCAP Compliance Checker (SCC) components.	December 22, 2021
Feature release: VM Import/ Export (VMIE) integration	Added support for VM import via all channels (console, API/ CLI, etc.), and for VM export via API/CLI. VM export is not currently available from the Image Builder console.	December 20, 2021
Feature release: AMI sharing for AWS Organizations and OUs	Updated distribution configuration to add support for sharing output AMIs with AWS Organizations and OUs.	November 24, 2021
Document update: Update component stages and phases	Expanded content for component stages in Image Builder, and how those intera ct with AWSTOE component phases.	September 22, 2021
Document update: Add CloudTrail integration content	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	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
Document update: Reorder AWSTOE pages	Rearranged AWSTOE pages for clarity.	August 11, 2021
Feature release: Parameter ized components and additional instance configura tion	Added support for specifyin g 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	Improved messaging around resource tagging.	June 25, 2021
Feature release: Launch template integration	Added support for using Amazon EC2 launch templates for AMI distribution in the Distribution settings.	April 7, 2021

Feature release: Container build enhancements	Added support for configuri ng block device mappings and specifying AMIs to use as the base image for container builds.	April 7, 2021
New STIG versions	Updated STIG versions and applied STIGS.	March 5, 2021
Update cron expressions	Image Builder cron processin g 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	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 rearrange d to reflect new functiona lity 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	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	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	Create looping constructs to define a repeated sequence of instructions in the AWSTOE application.	July 29, 2020
Support for local development of AWSTOE components	Develop and test image components locally with the AWSTOE application.	July 28, 2020
Encrypted AMIs	EC2 Image Builder adds support for encrypted AMI distribution.	July 1, 2020

AutoScaling deprecation	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 PrivateLi nk, 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	Updated STIG versions and applied STIGS.	January 23, 2020
Troubleshooting	Added general troublesh ooting scenarios.	January 22, 2020
STIG Components	You can create STIG-comp liant images with AWSTOE STIG components.	January 22, 2020