# Table of Contents

What is EC2 Image Builder ........................................................................................................... 1  
Features of EC2 Image Builder ....................................................................................................... 1  
Supported operating systems .......................................................................................................... 2  
Supported image formats .................................................................................................................. 2  
Concepts ........................................................................................................................................ 3  
Pricing ........................................................................................................................................... 5  
Related AWS services ..................................................................................................................... 5  
How EC2 Image Builder works ....................................................................................................... 7  
AMI elements ................................................................................................................................ 7  
Default quotas ................................................................................................................................. 8  
AWS Regions and Endpoints .......................................................................................................... 8  
Component management ................................................................................................................. 8  
Image testing ................................................................................................................................ 8  
Semantic versioning ........................................................................................................................ 9  
Resources created ............................................................................................................................ 9  
Distribution .................................................................................................................................... 10  
Sharing Resources .......................................................................................................................... 10  
Compliance ....................................................................................................................................... 10  
Get started ..................................................................................................................................... 11  
Prerequisites ................................................................................................................................... 11  
EC2 Image Builder service-linked role ........................................................................................... 11  
Configuration requirements ........................................................................................................... 11  
Container repository (container image pipelines) ......................................................................... 12  
AWS identity and Access Management (IAM) .............................................................................. 12  
Access EC2 Image Builder ............................................................................................................. 13  
Create an image pipeline (AMI) ..................................................................................................... 13  
Step 1: Specify pipeline details ....................................................................................................... 13  
Step 2: Choose recipe ..................................................................................................................... 14  
Step 3: Define infrastructure configuration - optional ................................................................... 15  
Step 4: Define distribution settings - optional ............................................................................ 16  
Step 5: Review .............................................................................................................................. 16  
Step 6: Clean up ............................................................................................................................. 16  
Create an image pipeline (Docker) ................................................................................................ 18  
Step 1: Specify pipeline details ..................................................................................................... 18  
Step 2: Choose recipe ................................................................................................................... 18  
Step 3: Define infrastructure configuration - optional ................................................................ 20  
Step 4: Define distribution settings - optional ............................................................................ 20  
Step 5: Review .............................................................................................................................. 20  
Step 6: Clean up ............................................................................................................................. 21  
AWSTOE component manager ................................................................................................ ..... 23  
AWSTOE downloads ...................................................................................................................... 23  
Supported Regions ......................................................................................................................... 24  
Get started with AWSTOE ............................................................................................................. 25  
Verify signature ............................................................................................................................... 25  
Step 1: Install AWSTOE ................................................................................................................ 30  
Step 2: Set AWS credentials ......................................................................................................... 30  
Step 3: Develop component documents locally ............................................................................ 31  
Step 4: Validate AWSTOE components ....................................................................................... 32  
Step 5: Run AWSTOE components ............................................................................................... 32  
Use component documents .......................................................................................................... 33  
Component document workflow ................................................................................................... 34  
Component logging ......................................................................................................................... 34  
Input and output chaining ............................................................................................................... 35  
Document schema and definitions ................................................................................................. 36  

EC2 Image Builder User Guide
Create and update container image distribution
Manage security findings
Create a new version of a container recipe
SCAP compliance validator component

EC2 Image Builder User Guide

Document example schemas .......................................................... 39
Define variables ........................................................................ 41
Use looping constructs .............................................................. 46
Action modules ........................................................................... 53
General execution ........................................................................ 54
File download and upload ........................................................... 64
File system operation .................................................................. 72
Software installation actions ......................................................... 100
System actions ............................................................................ 114
Configure input ............................................................................ 118
Distributor package managed components for Windows ............... 121
Prerequisites .............................................................................. 121
Configure Systems Manager Distributor permissions ................. 121
Configure distributor-package-windows as a standalone component 123
Find Distributor packages ............................................................ 123
CIS hardening components .......................................................... 124
STIG hardening components ....................................................... 124
Windows STIG hardening components ....................................... 125
STIG version history log for Windows ......................................... 130
Linux STIG hardening components ............................................ 133
STIG version history log for Linux .............................................. 137
SCAP compliance validator component ....................................... 141
Command reference .................................................................... 143
run ............................................................................................. 144
validate ..................................................................................... 146
Manage resources ....................................................................... 148
Components ............................................................................... 148
Create a YAML component document ........................................ 150
Component parameters ............................................................. 151
List and view components .......................................................... 154
Create a component (console) ...................................................... 157
Create a component with the AWS CLI ..................................... 158
Import a component (AWS CLI) ................................................. 161
Clean up resources ..................................................................... 162
Recipes ...................................................................................... 162
List and view image recipes ....................................................... 163
List and view container recipes .................................................. 164
Create a new version of an image recipe ..................................... 165
Create a new version of a container recipe ................................. 172
Clean up resources ..................................................................... 177
Images ....................................................................................... 177
List images and build versions .................................................... 177
View image details ..................................................................... 182
Create images ............................................................................ 188
Import a VM image ..................................................................... 189
Manage security findings ............................................................. 192
Clean up resources ..................................................................... 195
Infrastructure configurations ....................................................... 195
List and view infrastructure configurations ............................... 196
Create an infrastructure configuration ....................................... 197
Update an infrastructure configuration ....................................... 198
VPC endpoints (AWS PrivateLink) .............................................. 200
Distribution settings ..................................................................... 203
List and view distribution settings .............................................. 204
Create and update AMI distribution .......................................... 205
Create and update container image distribution ......................... 213
Set up cross-account AMI distribution ....................................... 215
Monitor ................................................................................................................................. 264
  CloudTrail logs ................................................................................................................... 264
  Image Builder information in CloudTrail ........................................................................... 264
Security in EC2 Image Builder .............................................................................................. 266
  Data protection ..................................................................................................................... 266
  Encryption and Key Management ......................................................................................... 267
  Inter-network Traffic Privacy ............................................................................................... 267
Identity and Access Management ........................................................................................... 267
  Audience .............................................................................................................................. 268
  Authenticating with identities .............................................................................................. 268
  How EC2 Image Builder works with IAM ......................................................................... 268
  Identity-Based Policies ......................................................................................................... 275
  Resource-Based Policies ...................................................................................................... 277
  Managed policies ................................................................................................................ 278
  Service-linked roles ............................................................................................................. 294
Troubleshooting .................................................................................................................... 295
  Compliance validation ........................................................................................................ 297
Resilience ............................................................................................................................... 297
Infrastructure security ........................................................................................................ 297
Patch management ................................................................................................................ 298
Best practices ....................................................................................................................... 299
  Required post-build clean up .............................................................................................. 299
  Override the Linux clean up script .................................................................................... 303
Troubleshoot Image Builder ................................................................................................ 305
  Troubleshoot pipeline builds ............................................................................................. 305
  Troubleshooting scenarios .................................................................................................. 306
Document history .................................................................................................................. 310
What is EC2 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:

1. Choose a base image for your customizations.
2. Add to or remove software from your base image.
3. Customize settings and scripts with build components.
4. Run selected tests or create custom test components.
5. Distribute AMIs to AWS Regions and AWS accounts.
6. If your Image Builder pipeline creates a custom Amazon Machine Image (AMI) for distribution, you can authorize other AWS accounts, organizations, and OUs to launch it from your account. Your account is billed for charges that are associated with the AMI.

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.

Section Contents
- Features of EC2 Image Builder (p. 1)
- Supported operating systems (p. 2)
- Supported image formats (p. 2)
- Concepts (p. 3)
- Pricing (p. 5)
- Related AWS services (p. 5)

Features of EC2 Image Builder

EC2 Image Builder provides the following features:

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

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

Increase service uptime

Image Builder 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 EC2 Image Builder (p. 124).

Centralized enforcement and lineage tracking

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

Simplified sharing of resources across AWS accounts

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

- Components
- Images
- Image recipes
- Container recipes

For more information, see Share EC2 Image Builder resources (p. 223).

Supported operating systems

Image Builder supports the following operating system versions:

<table>
<thead>
<tr>
<th>Operating system/distribution</th>
<th>Supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Linux</td>
<td>2 and 2023</td>
</tr>
<tr>
<td>CentOS</td>
<td>7 and 8</td>
</tr>
<tr>
<td>CentOS Stream</td>
<td>8</td>
</tr>
<tr>
<td>Red Hat Enterprise Linux (RHEL)</td>
<td>7 and 8</td>
</tr>
<tr>
<td>SUSE Linux Enterprise Server (SUSE)</td>
<td>12 and 15</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>18.04 LTS, 20.04 LTS, and 22.04 LTS</td>
</tr>
<tr>
<td>Windows Server</td>
<td>2012 R2, 2016, 2019, and 2022</td>
</tr>
</tbody>
</table>

Supported image formats

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

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

**AMI**

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

**Image pipeline**

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

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

**Managed image**

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

**Image recipe**

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

**Container recipe**

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

**Base image**

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

**Components**

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

A component is created from a declarative, plain-text YAML or JSON document that describes the runtime configuration for building and validating, or testing an instance that is produced by your pipeline. Components run on the instance using a component management application. The component management application parses the documents and runs the desired steps.
After they are created, one or more components are grouped together using an image recipe or container recipe to define the plan for building and testing a virtual machine or container image. You can use public components that are owned and managed by AWS, or you can create your own. For more information about components, see AWS Task Orchestrator and Executor component manager (p. 23).

Component document

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

Runtime stages

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

Configuration phases

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

**Build stage:**

Build phase

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

Validate phase

After 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.
Pricing

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

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

Related AWS services

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

**AWS License Manager**

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

**AWS Organizations**

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

**Amazon Inspector**

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

**AWS Resource Access Manager**
AWS Resource Access Manager (AWS RAM) lets you share your resources with any AWS account or through AWS Organizations. If you have multiple AWS accounts, you can create resources centrally and use AWS RAM to share those resources with other accounts. EC2 Image Builder allows sharing for the following resources: components, images, and image recipes. For more information about AWS RAM, see the AWS Resource Access Manager User Guide. For information about sharing Image Builder resources, see Share EC2 Image Builder resources (p. 223).

Amazon CloudWatch Logs

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

Amazon Elastic Container Registry (Amazon ECR)

Amazon ECR is a managed AWS container image registry service that is secure, scalable, and reliable. Container images 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.
How EC2 Image Builder works

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

1. **Specify pipeline details** – Enter information about your pipeline, such as a name, description, tags, and a schedule to run automated builds. You can choose manual builds, if you prefer.

2. **Choose recipe** – Choose between building an AMI, or building a container image. For both types of output images, you enter a name and version for your recipe, select a base image, and choose components to add for building and testing. You can also choose automatic versioning, to ensure that you always use the latest available Operating System (OS) version for your base image. Container recipes additionally define Dockerfiles, and the target Amazon ECR repository for your output Docker container image.

   **Note**
   Components are the building blocks that are consumed by an image recipe or a container recipe. For example, packages for installation, security hardening steps, and tests. The selected base image and components make up an image recipe.

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

4. **Define distribution settings** – Choose the AWS Regions to distribute your image to after the build is complete and has passed all its tests. The pipeline automatically distributes your image to the Region where it runs the build, and you can add image distribution for other Regions.

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

Section contents
- AMI elements (p. 7)
- Default quotas (p. 8)
- AWS Regions and Endpoints (p. 8)
- Component management (p. 8)
- Semantic versioning (p. 9)
- Resources created (p. 9)
- Distribution (p. 10)
- Sharing Resources (p. 10)
- Compliance (p. 10)

AMI elements

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

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

## Default quotas

To view the default quotas for Image Builder, see [Image Builder Endpoints and Quotas](#).

## AWS Regions and Endpoints

To view the service endpoints for Image Builder, see [Image Builder Endpoints and Quotas](#).

## Component management

EC2 Image Builder uses a component management application AWS Task Orchestrator and Executor (AWSTOE) that helps you orchestrate complex workflows, modify system configurations, and test your systems with YAML-based script components. Because AWSTOE is a standalone application, it does not require any additional setup. It can run on any cloud infrastructure and on premises. To get started using AWSTOE as a standalone application, see [Get started with AWSTOE](#). For more information about how Image Builder uses AWSTOE to manage its components, see [Manage components with Image Builder](#). For more information about creating components with AWSTOE, see [AWS Task Orchestrator and Executor component manager](#).

## Image testing

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

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

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

\(<\text{major}>\cdot<\text{minor}>\cdot<\text{patch}>\cdot<\text{build}>\)

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: \(<\text{major}>\cdot<\text{minor}>\cdot<\text{patch}>\)
3. Build version ARNs have all four nodes, and point to a specific build for a specific version of an object.

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

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

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

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

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

Resources created

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

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

The following resources are created during the image build process:

**AMI image pipelines**
- EC2 instance (temporary)
Distribution

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

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

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

To update your distribution settings using the Image Builder console, follow the steps to Create a new image recipe version (console) (p. 165), or Create a new container recipe version with the console (p. 172).

Sharing Resources

To share components, recipes, or images with other accounts or within AWS Organizations, see Share EC2 Image Builder resources (p. 223).

Compliance

For CIS, EC2 Image Builder uses Amazon Inspector to perform assessments for exposure, vulnerabilities, and deviations from best practices and compliance standards. For example, 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 hardening, EC2 Image Builder validates with STIG. For a complete list of STIG components available through Image Builder, see Amazon managed STIG hardening components for EC2 Image Builder (p. 124). For more information, see Center for Internet Security (CIS) Benchmarks.
Get started with EC2 Image Builder

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

Contents

- Prerequisites (p. 11)
- Access EC2 Image Builder (p. 13)
- Create an image pipeline using the EC2 Image Builder console wizard (p. 13)
- Create a container image pipeline using the EC2 Image Builder console wizard (p. 18)

Prerequisites

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

EC2 Image Builder service-linked role

EC2 Image Builder uses a service-linked role to grant permissions to other AWS services on your behalf. You don't need to manually create a service-linked role. When you create your first Image Builder resource in the AWS Management Console, the AWS CLI, or the AWS API, Image Builder creates the service-linked role for you. For more information about the service-linked role that Image Builder creates in your account, see Using service-linked roles for EC2 Image Builder (p. 294).

Configuration requirements

- Image Builder supports AWS PrivateLink. For more information about configuring VPC endpoints for Image Builder, see EC2 Image Builder and interface VPC endpoints (AWS PrivateLink) (p. 200).
- Image Builder supports EC2-Classic.
- 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](https://aws.amazon.com/documentation/systemsmanager/) in the AWS Systems Manager User Guide.

## Container repository (container image pipelines)

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

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](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/creatingECRRepository.html) in the Amazon Elastic Container Registry User Guide.

## AWS Identity and Access Management (IAM)

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

- EC2InstanceProfileForImageBuilder
- EC2InstanceProfileForImageBuilderECRContainerBuilds
- AmazonSSMManagedInstanceCore

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

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

### Attach policy

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

1. Sign in to the AWS Management Console and open the IAM console at [https://console.aws.amazon.com/iam/](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.
Repeat steps 3-6 for the `EC2InstanceProfileForImageBuilderECRContainerBuilds` and `AmazonSSMManagedInstanceCore` policies.

**Note**
If you want to copy an image created with Image Builder to another account, you must create the `EC2ImageBuilderDistributionCrossAccountRole` role in all of the target accounts, and attach the `Ec2ImageBuilderCrossAccountDistributionAccess` policy (p. 289) managed policy to the role. For more information, see Share EC2 Image Builder resources (p. 223).

### Access EC2 Image Builder

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

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

### Create an image pipeline using the EC2 Image Builder console wizard

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

**Create image pipeline workflow**

- **Step 1: Specify pipeline details** (p. 13)
- **Step 2: Choose recipe** (p. 14)
- **Step 3: Define infrastructure configuration - optional** (p. 15)
- **Step 4: Define distribution settings - optional** (p. 16)
- **Step 5: Review** (p. 16)
- **Step 6: Clean up** (p. 16)

### Step 1: Specify pipeline details

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](https://aws.amazon.com/timezone/).
EC2 Image Builder User Guide
Step 2: Choose recipe

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

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

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

### Step 2: Choose recipe

1. Image Builder defaults to **Use existing recipe** in the **Recipe** section. For your first time through, choose the **Create new recipe** option.

2. In the **Image type** section, choose the **Amazon Machine Image (AMI)** option to create an image pipeline that will produce and distribute an AMI.

3. In the **General** section, enter the following required boxes:
   - **Name** – your recipe name
   - **Version** – your recipe version (use the format `<major>.<minor>.<patch>`, where major, minor, and patch are integer values). New recipes generally start with `1.0.0`.

4. In the **Source image** section, keep the default values for **Select image**, **Image Operating System (OS)**, and **Image origin**. This results in a list of Amazon Linux 2 AMIs, managed by Amazon, for you to choose from for your base image.
   a. From the **Image name** dropdown, choose an image.
   b. Keep the default for **Auto-versioning options (Use latest available OS version)**.
      
      **Note**
      This setting ensures that your pipeline uses semantic versioning for the base image, to detect dependency updates for automatically scheduled jobs. To learn more about semantic versioning for Image Builder resources, see [Semantic versioning](p. 9).

5. In the **Instance configuration** section, keep the default values for the **Systems Manager agent**. This results in Image Builder 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 – Amazon Linux** panel, you can browse through the components listed on the page. Use the pagination control in the upper right corner to navigate through additional components that are available for your base image OS. You can also search for specific components, or create your own build component using the Component manager.

   For this tutorial, choose a component that updates Linux with the latest security updates, as follows:
   a. Filter the results by entering the word **update** in the search bar that’s located at the top of the panel.
   b. Select the check box for the **update-linux** build component.
   c. Scroll down, and in the upper right corner of the **Selected components** list, choose **Expand all**.
d. Keep the default for **Versioning options (Use latest available component version)**.

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

If you selected a component that has input parameters, you would also see the parameters in this area. Parameters are not covered in this tutorial. For more information about using input parameters in your components, and setting them in your recipes, see [Manage AWSTOE component parameters with EC2 Image Builder](p. 151).

**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.

1. Scroll back up to the list of available components.
2. Select the check box for the `update-linux-kernel-mainline` build component (or any other component of your choice).
3. Scroll down to the **Selected components** list, to see that there are at least two results.
4. Newly added components might not have their versioning or input parameter settings expanded. To expand 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.
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.
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.
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 EC2 Image Builder distribution settings (p. 203).

For this tutorial, we are using the default settings.

Step 5: Review

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

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

Step 6: Clean up

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

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

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

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

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

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

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

Delete the image
Follow these steps to verify that you have deleted any image that was created from the tutorial pipeline. This tutorial is not likely to create an image unless enough time has elapsed since you created your pipeline that it runs, according to the build schedule.
1. To see a list of the images created under your account, choose Images from the navigation pane.
2. Choose the image Version for the image that you want to remove. This opens the Image build versions page.
3. Select the check box next to the Version for any image that you want to delete. You can select more than one image version at a time.
4. At the top of the Image build versions panel, choose Delete version.
5. To confirm the deletion, enter Delete in the box, and choose Delete.
Create a container image pipeline using the EC2 Image Builder console wizard

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

Create image pipeline workflow
- Step 1: Specify pipeline details (p. 18)
- Step 2: Choose recipe (p. 18)
- Step 3: Define infrastructure configuration - optional (p. 20)
- Step 4: Define distribution settings - optional (p. 20)
- Step 5: Review (p. 20)
- Step 6: Clean up (p. 21)

Step 1: Specify pipeline details

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 (p. 9).

5. Choose Next to proceed to the next step.

Step 2: Choose recipe

1. Image Builder defaults to Use existing recipe in the Recipe section. For your first time through, choose the Create new recipe option.
2. In the Image type section, choose the Docker image option to create a container pipeline that will produce a Docker image and distribute it to Amazon ECR repositories in target Regions.
3. In the General section, enter the following required boxes:
   - Name – your recipe name
   - Version – your recipe version (use the format <major>.<minor>.<patch>, where major, minor, and patch are integer values). New recipes generally start with 1.0.0.
4. In the Source image section, keep the default values for Select image, Image Operating System (OS), and Image origin. This results in a list of Amazon Linux 2 container images, managed by Amazon, for you to choose from for your base image.
a. From the **Image name** dropdown, choose an image.

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

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

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

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

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

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

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

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

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

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

If you had selected a component that has input parameters, you would also see the parameters in this area. Parameters are not covered in this tutorial. For more information about using input parameters in your components, and setting them in your recipes, see **Manage AWSTOE component parameters with EC2 Image Builder** *(p. 151)*.

**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.
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.
AWS Task Orchestrator and Executor component manager

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

Because it is a standalone application, it does not require additional server setup. It can run on any cloud infrastructure and on premises.

Contents
- AWSTOE downloads (p. 23)
- Supported Regions (p. 24)
- Get started with AWSTOE (p. 25)
- Use component documents in AWSTOE (p. 33)
- Action modules supported by AWSTOE component manager (p. 53)
- Configure input for the AWSTOE run command (p. 118)
- Distributor package managed components for Windows (p. 121)
- CIS hardening components (p. 124)
- Amazon managed STIG hardening components for EC2 Image Builder (p. 124)
- AWSTOE command reference (p. 143)

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 (p. 201).

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.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Platform</th>
<th>Download link</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RHEL 7 and 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ubuntu 16.04, 18.04, 20.04, and 22.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CentOS 7 and 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUSE 12 and 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Supported Regions**

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

<table>
<thead>
<tr>
<th>AWS Region name</th>
<th>AWS Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
</tr>
<tr>
<td>AWS GovCloud (US-East)</td>
<td>us-gov-east-1</td>
</tr>
<tr>
<td>AWS GovCloud (US-West)</td>
<td>us-gov-west-1</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
</tr>
<tr>
<td>Africa (Cape Town)</td>
<td>af-south-1</td>
</tr>
<tr>
<td>Asia Pacific (Hong Kong)</td>
<td>ap-east-1</td>
</tr>
<tr>
<td>Asia Pacific (Osaka)</td>
<td>ap-northeast-3</td>
</tr>
</tbody>
</table>
Get started with AWSTOE

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

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

Verify the signature of the AWSTOE installation download

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

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

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

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

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

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

Installing the GPG tools

If your operating system is Linux or Unix, the GPG tools are likely already installed. To test whether the tools are installed on your system, type gpg at a command prompt. If the GPG tools are installed, you see a GPG command prompt. If the GPG tools are not installed, you see an error message stating that the command cannot be found. You can install the GnuPG package from a repository.

To install GPG tools on Debian-based Linux

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

To install GPG tools on Red Hat–based Linux

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

Authenticating and importing the public key

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

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

   ```
   -----BEGIN PGP PUBLIC KEY BLOCK-----
   Version: GnuPG v2
   mQENBF8UqwsBCACdiRF2bkZYaFSDPF+CLfLWLFWtUClw8HztD8KIwTJ6LV3n3fHAU
   Ghk0ZH9mRmqRT2bq/xj6gsf9VqTj2AJQndG3dDjz75YCYfYM+oCZ+e5HSJaaW9i
   S5dykHj7Tx1z2He06G+5Wv7v5bP1zsPHsN7XWQ7+G2AMEPTz8pJxy//1DvMQns
   SlE19h7w6C1zl19LbBz1YHFSm5ucTThB8X5gM370CDMVf10Ctv8WfoLN
   6jxuA/sV7l1yKpm91Yp3+Gv6e7e878+Sn8/J00KE/U4sJv1ppbmuZDFh0zZ
   8e6W6IN9AFTlWzI2Z/L8SUuQs1575sZp1JAEBAAAg0kFXU1RPRSA6YxdedG9l
   QGGYXp6b15bJb0+1Q58BMBBCA6AjBGJfKksLAh95Bw83CAdAG6FQqGcQQoLBYYC
   AECqElCF46shGkQ3t3BVfVfGlw9FvmrB77+Qe/OvEz3JYoaf7If/FvDZ1
   6v3TC6p0V0e7uXlER6FTzjbh+7e5S5DX19HrPqzCnZfMqbpl4loeUuNd0f
   FcpuTCQfH+m+ElIPnoP1U0UjZuE10+++mxm0mB1/Kz+hly8h2L9n/vW5LB
   0Mbl1L9qmg6PbWiPcT8Krdz4USex9TXYYzjIwb0zw3uXetdaqY1M3KlKESiRsR
   YYDcpjcmwebhpur4xm91QoNoAHCdCtExJXJ/mkU3ewIR0cPyjA2ZeD2znk92LF
   79Dq8kr1arbCnvdDM5lBbbdhX36hezJE/b17VA0t1Y04MoEkn6OJg==
   =oyze
   -----END PGP PUBLIC KEY BLOCK-----
   ```
   
   - Make a note of the key value; you need it in the next step. In the preceding example, the key value is F5AEBC52.

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 = 6D6D E01C 869F D639 15E5 5742 DE0B C156 F5AE BC52
   uid  [ unknown] AWSTOE <awstoe@amazon.com>
   ```

   Additionally, the fingerprint string should be identical to F66D E01C 869F D639 15E5 5742 DE0B C156 F5AE BC52, as shown in the preceding example. Compare the key fingerprint that is returned to the one published on this page. They should match. If they don't match, do not install the AWSTOE installation script, and contact AWS Support.
Verify the signature of the package

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

To verify the installation script signature

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

   ```bash
   ```

   For example:

   ```bash
   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:

   ```bash
   ```

   For example:

   ```bash
   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.

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

   The output should look something like the following:

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

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

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

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

   - **WARNING:** This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner. Ideally, you would visit an AWS office and receive the key in person.
However, you would most likely download it from a website. In this case, the website is an AWS website.

- **gpg: no ultimately trusted keys found.** This means that the specific key is not "ultimately trusted" by you, or by other people that you trust.

For more information, see [http://www.gnupg.org](http://www.gnupg.org).

**Verify the signature of the AWSTOE installation download on Windows**

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

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

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

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

5B 77 F4 F0 C3 7A 8B 89 D9 A7 8F 54 B6 85 11 CE 9E A3 BF 17

**Note**

We are currently rolling out a new binary. If your signer certificate does not match the new thumbprint, verify that the thumbprint value is:

FC B6 AC 0D 24 E6 90 1D 35 9D 69 38 4C DF 31 04 A7 3E 91 7A

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:

  5B 77 F4 F0 C3 7A 8B 89 D9 A7 8F 54 B6 85 11 CE 9E A3 BF 17

  then your downloaded AWSTOE binary is authentic and can be safely installed.

  **Note**

  We are currently rolling out a new binary. If your signer certificate does not match the new thumbprint, verify that the thumbprint value is:

  FC B6 AC 0D 24 E6 90 1D 35 9D 69 38 4C DF 31 04 A7 3E 91 7A

  - If the thumbprint value in the lower details window is not identical to the previous value, do not run awstoe.exe.

**Get started steps**

- **Step 1: Install AWSTOE (p. 30)**
Step 1: Install AWSTOE

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

1. Download the AWSTOE application

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

   **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 (p. 26)]. To verify your download on a Windows server, see [Verify the signature on Windows (p. 29)].

   **Important**
   
   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.

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

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

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

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

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

### Step 3: Develop component documents locally

AWSTOE components are authored with plaintext YAML documents. For more information about document syntax, see [Use component documents in AWSTOE](p. 33).

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

**hello-world-windows.yml**

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

**hello-world-linux.yml**

```yaml
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:
            - Write-Host 'Hello World from the build phase.'
```

31
Step 4: Validate AWSTOE components

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

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

Windows

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

Linux

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

Step 5: Run AWSTOE components

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

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

To ensure that your logs are stored securely and retained for troubleshooting, we recommend configuring log storage in Amazon S3. In Image Builder, the Amazon S3 location for publishing logs is specified in the infrastructure configuration. For more information about infrastructure configuration, see Manage EC2 Image Builder infrastructure configuration (p. 195)

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
```
Use component documents

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

Topics
- Component document workflow (p. 34)
- Component logging (p. 34)
- Input and output chaining (p. 35)
- Document schema and definitions (p. 36)
- Document example schemas (p. 39)
- Define and reference variables in AWSTOE (p. 41)
- Use looping constructs in AWSTOE (p. 46)
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 (p. 53).

Component logging

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

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

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

Example: `/var/lib/amazon/toe/TOE_2021-07-01_12-34-56_UTC-0_a1bcd2e3-45f6-789a-bcde-0fa1b2c3def4`

AWSTOE stores the following core files in the log folder:

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

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

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

### Input and output chaining

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

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

or

```yaml
{{ 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 }}`.

```yaml
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 }}`.

```yaml
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.

```yaml
name: (optional)
description: (optional)
schemaVersion: "string"
phases:
  - name: "string"
    steps:
      - name: "string"
```
The schema definitions for a document are as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the document.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>description</td>
<td>Description of the document.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>schemaVersion</td>
<td>Schema version of the document, currently 1.0.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>phases</td>
<td>A list of phases with their steps.</td>
<td>List</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The schema definitions for a phase are as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the phase.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>steps</td>
<td>List of the steps in the phase.</td>
<td>List</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The schema definitions for a step are as follows.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>User-defined name for the step.</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>Keyword pertaining to the module that runs the step.</td>
<td>String</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timeoutSeconds</td>
<td>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.</td>
<td>Integer</td>
<td>No</td>
<td>7,200 sec (120 mins)</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Default value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>----------</td>
<td>----------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
| onFailure| Specifies what the step should do in case of failure. Valid values are as follows:  
  - **Abort** – Fails the step after the maximum number of attempts, and stops running. Sets status for phase and document to Failed.  
  - **Continue** – Fails the step after the maximum number of attempts, and continues to run remaining steps. Sets status for phase and document to Failed.  
  - **Ignore** – Sets the step to IgnoredFailure after the the maximum number of failed attempts, and continues to run remaining steps. Sets status for phase and document to SuccessWithIgnoredFailure. | String | No  | Abort |
| 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 example schemas

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

```json
name: RunConfig_UpdateWindows
description: 'This document will install all available Windows updates and run a configuration 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: 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}}'
```

39
The following is an example document schema to download and run a custom Linux binary file.

```json
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>mybucket</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 }}'
```

The following is an example document schema to install the AWS CLI on a Windows instance, using the setup file.

```json
name: InstallCLISetUp
```
Define and reference variables in AWSTOE

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

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

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

**Rules for parameter names**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Key name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Yes</td>
<td>The name of the parameter. Must be unique for the document (it must not be the same as any other parameter names or constants).</td>
</tr>
<tr>
<td>type</td>
<td>Yes</td>
<td>The data type of the parameter. Supported types include: string.</td>
</tr>
<tr>
<td>default</td>
<td>No</td>
<td>The default value for the parameter.</td>
</tr>
<tr>
<td>description</td>
<td>No</td>
<td>Describes the parameter.</td>
</tr>
</tbody>
</table>

**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:

```yaml
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:

```yaml
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
Define variables

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...
```

<table>
<thead>
<tr>
<th>CLI option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--parameters name=value,...</code></td>
<td>No</td>
<td>This option takes list of key-value pairs, with the parameter name as the key.</td>
</tr>
</tbody>
</table>

**Examples**

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

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

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

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

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

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

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

**Constants**

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

**Rules for constant names**

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

**Syntax**

```
constants:
  - <name>:
      type: <constant type>
      value: <constant value>
```
<table>
<thead>
<tr>
<th>Key name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Yes</td>
<td>Name of the constant. Must be unique for the document (it must not be the same as any other parameter names or constants).</td>
</tr>
<tr>
<td>value</td>
<td>Yes</td>
<td>Value of the constant.</td>
</tr>
<tr>
<td>type</td>
<td>Yes</td>
<td>Type of the constant. Supported type is string.</td>
</tr>
</tbody>
</table>

**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:
  
  ```yaml
  {{ 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**

```yaml
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**

```yaml
name: PingHosts
schemaVersion: 1.0
constants:
  - Hosts:
    type: 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 (p. 46)
- Types of looping constructs (p. 47)
- Step fields (p. 52)
- Step and iteration outputs (p. 53)

**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:
Use looping constructs

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

Resolve reference expressions

The AWSTOE resolves reference expressions as follows:

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

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

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

Types of looping constructs

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

**Looping construct types**

- **for loop** (p. 47)
- **forEach loop with input list** (p. 48)
- **forEach loop with delimited list** (p. 50)

**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 \([\text{start}, \text{end}]\) and includes boundary values.

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

**for loop schema**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Unique name of the loop. It must be unique compared to other loop names in the same phase.</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>start</td>
<td>Starting value of iteration. Does not accept chaining expressions.</td>
<td>Integer</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>end</td>
<td>Ending value of iteration. Does not accept chaining expressions.</td>
<td>Integer</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>updateBy</td>
<td>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.</td>
<td>Integer</td>
<td>Yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>

#### for loop input example

```powershell
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

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

forEach loop with input list input

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Unique name of the loop. It must be unique compared to other loop names in the same phase.</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>List of strings of forEach loop</td>
<td>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.</td>
<td>List of strings</td>
<td>Yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>

forEach loop with input list example 1

```yaml
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
Use looping constructs

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Unique name given to the loop. It should be unique when compared to other loop names in the same phase.</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>list</td>
<td>A string that is composed</td>
<td>String</td>
<td>Yes</td>
<td>n/a</td>
</tr>
</tbody>
</table>
### Field of constituent strings joined by a common delimiter character. Also accepts chained expressions. In case of chained expressions, ensure that those are enclosed by double quotes for correct interpretation by the YAML compiler.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
</table>
| delimiter | Character used to separate out strings within a block. Default is the comma character. Only one delimiter character is allowed from the given list:  
  - Dot: "."  
  - Comma: ","  
  - Semicolon: ";"  
  - Colon: ":"  
  - New line: "\n"  
  - Tab: "\t"  
  - Space: " "  
  - Hyphen: "-"  
  - Underscore: "_"  
  
  Chaining expressions cannot be used. | String | No | Comma: "," |

**Note**

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

```plaintext
forEach loop with delimited list example 1
```

```plaintext
// Uses chaining expression ({{ <phase_name>.<step_name>.inputs/outputs.<var_name> }})  
// to refer to another step's input/output variables for code re-use.

name: "RunMSIs"  
action: "ExecuteBinary"  
loop:
```
forEach loop with delimited list example 2

```yaml
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.
EC2 Image Builder User Guide
Action modules

• 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.
AWSTOE components are authored with plaintext YAML documents. For more information about
document syntax, see Use component documents in AWSTOE (p. 33).

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.
Action module types
• General execution modules (p. 54)
• File download and upload modules (p. 64)

53


General execution modules

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

**General execution action modules**
- ExecuteBash (p. 54)
- ExecuteBinary (p. 55)
- ExecuteDocument (p. 56)
- ExecutePowerShell (p. 62)

**ExecuteBash**

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

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

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

- The application hands the exit code to the caller if it is run by the Systems Manager Agent. The Systems Manager Agent handles the system reboot and runs the same step that initiated the restart, as described in Rebooting Managed Instance from Scripts.
- The application saves the current execution state, 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**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>commands</td>
<td>Contains a list of instructions or commands to run as per bash syntax. Multi-line YAML is allowed.</td>
<td>List</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: Before and after a reboot**

```yaml
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdout</td>
<td>Standard output of command execution.</td>
<td>string</td>
</tr>
</tbody>
</table>

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)**

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

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

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

**ExecuteBinary**

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

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

- The application hands the exit code to the caller if it is run by the Systems Manager Agent. The Systems Manager Agent handles restarting the system and runs the same step that initiated the restart, as described in [Rebooting Managed Instance from Scripts](#).
- The application saves the current execution state, 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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The path to the binary file for execution.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>arguments</td>
<td>Contains a list of command-line arguments to use when running the binary.</td>
<td>String List</td>
<td>No</td>
</tr>
</tbody>
</table>

Input example: install .NET

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

Output

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdout</td>
<td>Standard output of command execution.</td>
<td>string</td>
</tr>
</tbody>
</table>

Output example

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

ExecuteDocument

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.

**ExecuteDocument** action module
Component document nesting levels

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**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>Path of component document. Valid options include:</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Primitive

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Local file paths</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• S3 URIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• EC2 Image Builder component build version ARNs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>document-s3-bucket-owner</td>
<td>The account ID of the S3 bucket owner for the S3 bucket where component</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>documents are stored. (Recommended if you are using S3 URIs in your</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>component document.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phases</td>
<td>Phases to run in the component document, expressed as a comma-separated</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>list. If no phases are specified, then all phases run.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>Input parameters that are passed in to the component document at runtime</td>
<td>Parameter Map List</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>as key value pairs.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Parameter map input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the input parameter to pass to the component document that the</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>ExecuteDocument action module is running.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>The value of the input parameter.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Input examples

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

**Input example: Local document path**

```yaml
# 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:
Using a For loop to run documents

```yaml
# 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`.

```
"outputs": "[\"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`: ""
General execution

- "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 `main.yaml` component document successfully runs the `Sample-1.yaml` component document.

```json
{
  "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",
              "failureMessage": "",
              "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",
            }
          ]
        }
      ]
    }
  ]
}
```
ExecutePowerShell

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

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

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

- Hands the exit code to the caller if run by the Systems Manager Agent. The Systems Manager Agent handles the system reboot and runs the same step that initiated the restart, as described in Rebooting Managed Instance from Scripts.
- Saves the current execution state, 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

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

#### Input example: Before and after a reboot

```yaml
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdout</td>
<td>Standard output of command execution.</td>
<td>string</td>
</tr>
</tbody>
</table>

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

### Download and upload action modules

- **S3Download** (p. 64)
- **S3Upload** (p. 67)
- **WebDownload** (p. 69)

### S3Download

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

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

**Note**

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

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

### Supported use cases

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

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

### IAM requirements

The IAM role that you associate with your instance profile must have permissions to run the **S3Download** action module. The following IAM policies must be attached to the IAM role that is associated with the instance profile:
• **Single file**: s3:GetObject against the bucket/object (for example, arn:aws:s3:::BucketName/*).

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>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.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>destination</td>
<td>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, /myfolder/package.zip.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>expectedBucketOwner</td>
<td>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.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>overwrite</td>
<td>When set to true, if a file of the same name already exists in the destination folder for the specified local path, the</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
</tr>
</tbody>
</table>
File download and upload

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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 &quot;overwrite&quot; property for &quot;destination&quot; file is set to false. Cannot download.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

```
name: MyS3DownloadKeyprefix
action: S3Download
maxAttempts: 3
inputs:
```
S3Upload

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

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

Supported use cases

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

IAM requirements

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

Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The local path where source files/folders originate. The source supports an asterisk wildcard (*).</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>destination</td>
<td>The path for the destination Amazon S3 bucket where source files/folders are uploaded.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>recurse</td>
<td>When set to true, performs</td>
<td>String</td>
<td>No</td>
<td>false</td>
</tr>
</tbody>
</table>
### File download and upload

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3Upload</td>
<td>recursively.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expectedBucketOwner</td>
<td>expected owner account ID for the Amazon S3 bucket specified in the destination path. We recommend that you verify the ownership of the Amazon S3 bucket specified in the destination.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**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.

```yaml
name: MyS3UploadFile
action: S3Upload
onFailure: Abort
maxAttempts: 3
inputs:
  - source: C:\myfolder\package.zip
destination: s3://mybucket/path/to/package.zip
  expectedBucketOwner: 123456789022
```

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

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

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

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

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

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

None.

WebDownload

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

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

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

Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The valid HTTP/HTTPS URL <em>(HTTPS is recommended)</em>, which follows the RFC 3986 standard. Chaining expressions are permitted.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>destination</td>
<td>An absolute or relative file or folder path on the local system. Folder paths must end with <code>/</code>. If they do not end with <code>/</code>, they will be treated as file paths. The module creates any required file or folder for successful downloads. Chaining expressions are permitted.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>overwrite</td>
<td>When enabled, overwrites any existing files on the local system with the downloaded file or resource. When</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
</tr>
<tr>
<td>Primitive</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>checksum</td>
<td>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.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>algorithm</td>
<td>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.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>ignoreCertificateErrors</td>
<td>SSL certificate validation is ignored when enabled.</td>
<td>Boolean</td>
<td>No</td>
<td>false</td>
</tr>
</tbody>
</table>
### Output

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>destination</td>
<td>Newline character-delimited string that specifies the destination path where the downloaded files or resources are stored.</td>
<td>String</td>
</tr>
</tbody>
</table>

#### Input example: download remote file to local destination

```python
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

```python
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
/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

```python
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\n  checksum: ac68bbf921d953d1cfab916cb6120864
  algorithm: MD5
  overwrite: true

Output:
{}
AppendFile

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

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

The action module returns an error when the following occurs:

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

Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>content</td>
<td>The content to be appended to the file.</td>
<td>String</td>
<td>No</td>
<td>Empty string</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Input example: append file without encoding (Linux)

name: AppendingFileWithOutEncodingLinux
action: AppendFile
inputs:
  - path: ./Sample.txt
Input example: append file without encoding (Windows)

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

Input example: append file with encoding (Linux)

name: AppendingFileWithEncodingLinux
action: AppendFile
inputs:
  - path: /FolderName/SampleFile.txt
    content: "The string to be appended to the file"
    encoding: UTF-32

Input example: append file with encoding (Windows)

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

Input example: append file with empty string (Linux)

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

Input example: append file with empty string (Windows)

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

Output

None.

CopyFile

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

If a file with the specified name already exists in the specified folder, the action module, by default, overwrites the existing file. You can override this default behavior by setting the overwrite option to false. When the overwrite option is set to false, and there is already a file in the specified location with the specified name, the action module will return an error. This option works the same as the cp command in Linux, which overwrites by default.
The source file name can include a wildcard (*). Wildcard characters are accepted only after the last file path separator (/ or \). If wildcard characters are included in the source file name, all of the files that match the wildcard are copied to the destination folder. If you want to move more than one file by using a wildcard character, the input to the destination option must end with a file path separator (/ or \), which indicates that the destination input is a folder.

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The source file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>destination</td>
<td>The destination file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>overwrite</td>
<td>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.</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### Input example: copy a file (Linux)

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

#### Input example: copy a file (Windows)
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

76
CopyFolder

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

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

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

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The source folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>destination</td>
<td>The destination folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>overwrite</td>
<td>When set to false, the destination folders will not be replaced when there is already a folder in the specified location</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### 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\SourceFolder  
  destination: C:\MyFolder\

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

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

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

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

### Input example: copy a folder without overwriting (Linux)
name: CopyingFoldersWithoutOverwriteLinux
action: CopyFolder
inputs:
    - source: /Sample/MyFolder/SourceFolder
destination: /MyFolder/destinationFolder
overwrite: false

**Input example: copy a folder without overwriting (Windows)**

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

**Output**

None.

**CreateFile**

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

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

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

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

This action module can create a file 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**

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

**Input example: create a file without overwriting (Linux)**

```plaintext
ame: CreatingFileWithoutOverwriteLinux
action: CreateFile
inputs:
```
Input example: create a file without overwriting (Windows)

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

Input example: create a file with file properties

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

Input example: create a file without file properties

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

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 (p. 303)

Output
None.

CreateFolder

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

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

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

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>owner</td>
<td>The user name or ID.</td>
<td>String</td>
<td>No</td>
<td>The current user.</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>group</td>
<td>The group name or ID.</td>
<td>String</td>
<td>No</td>
<td>The group of the current user.</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>permissions</td>
<td>The folder permissions.</td>
<td>String</td>
<td>No</td>
<td>0777</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>overwrite</td>
<td>If the name of the specified file already exists, setting this value to false prevents the file from being truncated or overwritten by default.</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: create a folder (Linux)**

```plaintext
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/SampleFolder/
  permissions: 777

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

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

Output

None.

CreateSymlink

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

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

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

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

The action module returns an error when the following occurs:

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

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

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

```bash
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**

```bash
name: CreatingSymbolicLinkWithOutForce
action: CreateSymlink
inputs:
  - path: Symboliclink.txt
  target: /Folder/Sample.txt
```

### Output

None.

**DeleteFile**

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

The input of `path` should be a valid file path or a file path with a wildcard 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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: delete a single file (Linux)**

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

**Input example: delete a single file (Windows)**

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

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

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

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

name: DeletingFileEndingWithLogWindows  
action: DeleteFile  
inputs:  
  - path: C:\SampleFolder\MyFolder\*log

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

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

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

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

### Output

None.
DeleteFolder

The **DeleteFolder** action module deletes folders.

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>force</td>
<td>Removes the folder whether or not the folder is empty.</td>
<td>Boolean</td>
<td>No</td>
<td><code>false</code></td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**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\n    force: true
```

**Input example: delete a folder (Linux)**

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

**Input example: delete a folder (Windows)**

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

Output
None.

ListFiles

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

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

The action module returns an error when the following occurs:

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

Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>fileNamePattern</td>
<td>The pattern to match to list all files with names that match the pattern.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Lists files in the folder recursively.</td>
<td>Boolean</td>
<td>No</td>
<td>false</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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)

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

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

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

Input example: list files recursively

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

Output

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>files</td>
<td>The list of files.</td>
<td>String</td>
</tr>
</tbody>
</table>

Output example

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

MoveFile

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

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

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

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The source file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>destination</td>
<td>The destination file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>overwrite</td>
<td>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.</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: move a file (Linux)**

```plaintext
ame: MovingAFilenLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
destination: /MyFolder/destinationFile.txt
```

**Input example: move a file (Windows)**

```plaintext
name: MovingAFilenWindows
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)**

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

89
name: MovingFileWithSourceFileNameLinux
action: MoveFile
inputs:
  - source: /Sample/MyFolder/Sample.txt
destination: /MyFolder/

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

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

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

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

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

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

**Input example: move a file without overwriting (Linux)**

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

**Input example: move a file without overwriting (Windows)**

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

**Output**

None.

**MoveFolder**

The **MoveFolder** action module moves folders from the specified source to the specified destination. The input for the **source** option is the folder to move, and the input to the **destination** option is the folder where the contents of the source folders are moved.
If the destination parent folder or the input to the destination option does not exist at runtime, the default behavior of the module is to recursively create the folder at the specified destination.

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

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

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

The action module returns an error when the following occurs:

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>The source folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>destination</td>
<td>The destination folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>overwrite</td>
<td>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.</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example:** move a folder (Linux)
name: MovingAFolderLinux
action: MoveFolder
inputs:
  - source: /Sample/MyFolder/SourceFolder
destination: /MyFolder/destinationFolder

Input example: move a folder (Windows)

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

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

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

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

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

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

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

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

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

Input example: move a folder without overwriting (Linux)

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

Input example: move a folder without overwriting (Windows)
name: MovingFoldersWithoutOverwriteWindows
action: MoveFolder
inputs:
  - source: C:\Sample\MyFolder\SampleFolder
  destination: C:\MyFolder\destinationFolder
  overwrite: false

Output
None.

ReadFile

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

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

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

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

The action module returns an error when the following occurs:

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

Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Primitive**

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>printFileContent</td>
<td>Boolean</td>
<td>No</td>
<td>false</td>
<td>N/A</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: read a file (Linux)**

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

**Input example: read a file (Windows)**

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

**Input example: read a file and specify encoding standard**

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

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

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

**Output**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>The file content.</td>
<td>string</td>
</tr>
</tbody>
</table>

**Output example**

```json
{
}
```
**SetFileEncoding**

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

The action module returns an error when the following occurs:

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

**Input**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>encoding</td>
<td>The encoding standard.</td>
<td>String</td>
<td>No</td>
<td>utf8</td>
<td>utf8, utf-8, utf16, utf16-LE, utf-16-BE, utf-16-LE utf16-BE, utf-16-BE, utf32, utf32-LE, utf32-BE, utf32-BE, and utf32-BE. The value of the encoding option is case insensitive.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Input example: set file encoding property**

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

**Output**
SetFileOwner

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

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

The action module returns an error when the following occurs:

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

**Input**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>owner</td>
<td>The user name.</td>
<td>string</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the user group.</td>
<td>String</td>
<td>No</td>
<td>The name of the group that the user belongs to.</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
</tbody>
</table>

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

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

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

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

**Output**
None.

**SetFolderOwner**

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

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

The action module returns an error when the following occurs:

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

**Input**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The folder path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>owner</td>
<td>The user name.</td>
<td>string</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the user group.</td>
<td>String</td>
<td>No</td>
<td>The name of the group that the user belongs to.</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>recursive</td>
<td>Overrides the default behavior of modifying ownership for all of the contents of a folder when set to false.</td>
<td>Boolean</td>
<td>No</td>
<td>true</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>name: SettingFolderPropertyWithoutRecursively</th>
<th>action: SetFolderOwner</th>
<th>inputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- path: /SampleFolder/</td>
<td>owner: LinuxUser</td>
<td>recursive: false</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>name: SettingFolderPropertyWithGroup</th>
<th>action: SetFolderOwner</th>
<th>inputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- path: /SampleFolder/</td>
<td>owner: LinuxUser</td>
<td>group: LinuxUserGroup</td>
</tr>
</tbody>
</table>

Output

None.

SetFilePermissions

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

The input for permissions must be a string value.

This action module can create a file 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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
<th>Supported on all platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The file path.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
<tr>
<td>permissions</td>
<td>The file permissions.</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Not supported on Windows.</td>
</tr>
</tbody>
</table>

Input example: modify file permissions

| name: ModifyingFilePermissions |
SetFilePermissions

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

Output

None.

SetFolderPermissions

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

The input for permissions must be a string value.

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

The action module returns an error when the following occurs:

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

Input

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

Input example: set folder permissions

```plaintext
name: SettingFolderPermissions
action: SetFolderPermissions
inputs:
```
Software installation actions

This section describes action modules that perform software installation action commands and instructions.

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** *(p. 101)*
- **UninstallMSI** *(p. 107)*
InstallMSI

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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
</table>
| 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 RFC 3986 standard. Chaining expressions are allowed. | String | Yes | N/A | N/A |
<p>| reboot    | Configure the system reboot behavior that follows a successful run of the action module. | String | No | Allow | Allow, Force, Skip |</p>
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Settings:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Force – Initiates a system reboot after the msiexec command runs successfully.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Allow – Initiates a system reboot if the msiexec command returns an exit code that indicates a reboot is required.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• 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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Primitive | 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 product documentation. | String | No | *VX | $i,w,e,a,r,u,c,m,o,p,v,x,+,!,*$ |
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 |
## Software installation actions

### properties

- **MSI logging property key-value pairs**, for example:
  - `TARGETDIR: "C:\target\location"

**Note:**
Modification of the following properties is not allowed:

- `REBOOT="ReallySupress"
- `REINSTALLMODE="ecmus"
- `REINSTALL="ALL"

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>MSI logging property key-value pairs, for example: <code>TARGETDIR: &quot;C:\target\location&quot;</code></td>
<td>Map[String]String</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Primitive</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Default value</td>
<td>Acceptable values</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>------</td>
<td>----------</td>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ignoreAuthenticodeSignature</td>
<td>Flag to ignore authenticode signature validation errors for the installer specified in path. The Get-AuthenticodeSignature command is used to validate installers.</td>
<td>Boolean</td>
<td>No</td>
<td>false</td>
<td>true, false</td>
</tr>
</tbody>
</table>

**Settings:**

- **true** – Validation errors are ignored and the installer runs.
- **false** – Validation errors are not ignored. The installer runs only when validation is successful. This is the default behavior.
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 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: local document path installation**

```yaml
- 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

```yaml
- 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

```yaml
- 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**

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.

AWSCTOE 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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>path</strong></td>
<td>Specify the MSI file location using one of the following:</td>
<td>String</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>• The local file path. The path can be absolute or relative.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A valid S3 object URI.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A valid web HTTP/HTTPS URL (HTTPS is recommended) that follows the RFC 3986 standard.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chaining expressions are allowed.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>reboot</strong></td>
<td>Configures the system reboot behavior that follows a successful run of the action module.</td>
<td>String</td>
<td>No</td>
<td>Allow</td>
<td>Allow, Force, Skip</td>
</tr>
<tr>
<td>Settings:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• <strong>Force</strong> – Initiates a system reboot after the msiexec command runs successfully.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• <strong>Allow</strong> – Initiates a system reboot if the msiexec command</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitive</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Default value</td>
<td>Acceptable values</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>------</td>
<td>----------</td>
<td>---------------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>returns an exit code that indicates a reboot is required.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Skip</strong> – 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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Software installation actions

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
<tbody>
<tr>
<td>logOptions</td>
<td>Specify the options to use for MSI installation logging. Specified flags are passed to the MSI installer, along with the <code>/L</code> 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 <a href="#">Command Line Options</a> in the Microsoft Windows Installer product documentation.</td>
<td>String</td>
<td>No</td>
<td>*VX</td>
<td><em>VX, i</em>, w*, e*, a*, r*, u*, c*, m*, o*, p*, v*, x*, +, !, *</td>
</tr>
<tr>
<td>logFile</td>
<td>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.</td>
<td>String</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Primitive: properties

**Description**: MSI logging property key-value pairs, for example: TARGETDIR: "C:\target\location"

- **Type**: Map[String]String
- **Required**: No
- **Default value**: N/A
- **Acceptable values**: N/A

**Note**: Modification of the following properties is not allowed:
- REBOOT="ReallySuppress"
- REINSTALLMODE="ecmus"
- REINSTALL="ALL"
### Software installation actions

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Acceptable values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignoreAuthenticodeSignatureErrors</td>
<td>Flag to ignore authenticode signature validation errors for the installer specified in path. The Get-AuthenticodeSignature command is used to validate installers.</td>
<td>Boolean</td>
<td>No</td>
<td>false</td>
<td>true, false</td>
</tr>
</tbody>
</table>

**Settings:**

- **true** – Validation errors are ignored and the installer runs.
- **false** – Validation errors are not ignored. The installer runs only when validation is successful. This is the default behavior.
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 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**

```yaml
- 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 file system action commands and instructions.

**System action modules**

- [Reboot](#)
- [SetRegistry](#)
- [UpdateOS](#)

Reboot

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.

```bash
@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 SSM CommandRetryLimit. If the number of reboots exceeds the retry limit, the automation fails. You can change the limit by editing the SSM agent config file (Mds.CommandRetryLimit). See [Runtime Configuration](#) in the SSM 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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>delaySeconds</td>
<td>Delays a specific amount of time before initiating a reboot.</td>
<td>Integer</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>

**Input example: reboot step**

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

### Output

None.

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

**SetRegistry**

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

### Input

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path of registry key.</td>
<td>String</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
EC2 Image Builder User Guide
System actions

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of registry key.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>value</td>
<td>Value of registry key.</td>
<td>String/Number/Array</td>
<td>Yes</td>
</tr>
<tr>
<td>type</td>
<td>Value type of registry key.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**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**

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

**Output**

None.

**UpdateOS**

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

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

For Linux, you can specify one or more packages to be excluded from the list of updates for installation.

Input example: add support for installing Linux updates

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

Input example: add support for installing Windows updates

```json
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`).
- `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 Define and reference variables in AWSTOE (p. 41) page.

- `executionId` (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.

```json
{
  "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>"
}
```
Distributor package managed components for Windows

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 the build instances for your Windows images. For more information about how to use this component, see Configure distributor-package-windows as a standalone component (p. 123).
- The aws-vss-components-windows component uses AWS Systems Manager Distributor to install the AwsVssComponents package on your build instance.

For more information about how to use managed components in your Image Builder recipe, see Create a new version of an image recipe (p. 165) for image recipes or Create a new version of a container recipe (p. 172) for container recipes. For more information about the AwsVssComponents package, see Create a VSS application-consistent snapshot in the Amazon EC2 User Guide for Windows Instances.

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 doesn't currently support Systems Manager Distributor packages that reboot the instance. For example, the AWSNVMe, AWSPVD Presenters, and AwsEnaNetworkDriver Distributor packages reboot the instance, and so are not allowed.
- 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 (p. 121).

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:${AWS::Partition}:ssm:${AWS::Region}::document/AWS-ConfigureAWSPackage",
   "arn:${AWS::Partition}:ec2:${AWS::Region}:${AWS::AccountId}:instance/*"
   ],
   "Sid": "AllowGetCommandInvocation",
   "Effect": "Allow",
   "Action": [
   "ssm:GetCommandInvocation"
   ],
   "Resource": [
   "*"
   ]
   }
   }
   }
   ```

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 (p. 197).

Configure distributor-package-windows as a standalone component

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

- **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 (p. 123).

- **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.

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 EC2 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 Manage components with Image Builder (p. 148).
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 (p. 305).

**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 (p. 125)
- STIG version history log for Windows (p. 130)
- Linux STIG hardening components (p. 133)
- STIG version history log for Linux (p. 137)
- SCAP compliance validator component (p. 141)

**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 2022.4.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 1 Release 1**
  
  V-254335, V-254336, V-254337, V-254338, V-254351, V-254357, V-254363, and V-254481

- **Windows Server 2019 STIG Version 2 Release 5**
V-205691, V-205819, V-205858, V-205859, V-205860, V-205870, V-205871, and V-205923

• Windows Server 2016 STIG Version 2 Release 5

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


• Microsoft .NET Framework 4.0 STIG Version 2 Release 2

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

• Windows Firewall STIG Version 2 Release 1

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 3

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

• Microsoft Edge STIG Version 1 Release 6 (Windows Server 2022 only)

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

STIG-Build-Windows-Medium version 2022.4.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 1 Release 1

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

Windows STIG hardening components

- **Windows Server 2019 STIG Version 2 Release 5**

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


- **Windows Server 2016 STIG Version 2 Release 5**

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


- **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:
Windows STIG hardening components

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

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

- **Windows Firewall STIG Version 2 Release 1**

  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-242003

- **Internet Explorer 11 STIG Version 2 Release 3**

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


- **Microsoft Edge STIG Version 1 Release 6 (Windows Server 2022 only)**

  Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities, plus:
Defender STIG Version 2 Release 4 (Windows Server 2022 only)

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


STIG-Build-Windows-High version 2022.4.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 1 Release 1**
  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:


- **Windows Server 2019 STIG Version 2 Release 5**
  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:


- **Windows Server 2016 STIG Version 2 Release 5**
  Includes all supported STIG settings that the hardening component applies for Categories II and III (Medium and Low) vulnerabilities, plus:


- **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:
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.

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 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 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 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 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)

STIG-Build-Linux-Low version 2023.3.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 12
• RHEL 7/CentOS 7
  V-204452, V-204576, and V-204605
• AL2
  V-204452, V-204576, and V-204605

RHEL 8 STIG Version 1 Release 11
• RHEL 8/CentOS 8/AL 2023

Ubuntu 18.04 STIG Version 2 Release 11

Ubuntu 20.04 STIG Version 1 Release 9

STIG-Build-Linux-Medium version 2023.3.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](https://example.com). For information about how to view the complete list, see [STIG Viewing Tools](https://example.com).

**Note**
The STIG-Build-Linux-Medium hardening components include all listed STIG settings that AWSOTO 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 12**

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:**
  

**RHEL 8 STIG Version 1 Release 11**

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**
  
Ubuntu 18.04 STIG Version 2 Release 11
Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:


Ubuntu 20.04 STIG Version 1 Release 9
Includes all supported STIG settings that the hardening component applies for Category III (Low) vulnerabilities for this Linux distribution, plus:


STIG-Build-Linux-High version 2023.3.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.
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.

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
• 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
SCAP compliance validator component

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-checker-windows and scap-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-checker-windows version 2021.04.0**

The `scap-checker-windows` component runs on the Image Builder pipeline's build and test instances. 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*

The SCAP checker component for Windows includes the following benchmarks:

**SCC Version: 5.4.2**

2021 Q4 Benchmarks:

- `U_MS_DotNet_Framework_4-0_V2R1_STIG_SCAP_1-2_Benchmark`
- `U_MS_IE11_V2R1_STIG_SCAP_1-2_Benchmark`
- `U_MS_Windows_2012_and_2012_R2_MS_V3R2_STIG_SCAP_1-2_Benchmark`
- `U_MS_Windows_Defender_AV_V2R2_STIG_SCAP_1-2_Benchmark`
- `U_MS_Windows_Server_2016_V2R1_STIG_SCAP_1-2_Benchmark`
- `U_MS_Windows_Server_2019_V2R1_STIG_SCAP_1-2_Benchmark`
- `U_MS_Windows_Firewall_V2R1_STIG_SCAP_1-2_Benchmark`
- `U_CAN_Ubuntu_18-04_V2R4_STIG_SCAP_1-2_Benchmark`
- `U_RHEL_7_V3R5_STIG_SCAP_1-2_Benchmark`
- `U_RHEL_8_V1R3_STIG_SCAP_1-2_Benchmark`

**scap-checker-linux version 2021.04.0**

The `scap-checker-linux` component runs on the Image Builder pipeline's build and test instances. 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. The SCC application currently supports the x86 architecture for validation.

The SCAP checker component for Linux includes the following benchmarks:

**SCC Version: 5.4.2**

2021 Q4 Benchmarks:

- U_CAN_Ubuntu_18-04_V2R4_STIG_SCAP_1-2_Benchmark
- U_RHEL_7_V3R5_STIG_SCAP_1-2_Benchmark
- U_RHEL_8_V1R3_STIG_SCAP_1-2_Benchmark
- U_MS_DotNet_Framework_4-0_V2R1_STIG_SCAP_1-2_Benchmark
- U_MS_IE11_V2R1_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_2012_and_2012_R2_MS_V3R2_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Defender_AV_V2R2_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2016_V2R1_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Server_2019_V2R1_STIG_SCAP_1-2_Benchmark
- U_MS_Windows_Firewall_V2R1_STIG_SCAP_1-2_Benchmark

**SCAP version history**

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

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added SCAP components</td>
<td><strong>Introduced the following SCAP components:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Created scap-checker-windows version 2021.04.0 (SCC Version: 5.4.2)</td>
<td>December 20, 2021</td>
</tr>
<tr>
<td></td>
<td>• Created-scap-checker-linux version 2021.04.0 (SCC Version: 5.4.2)</td>
<td></td>
</tr>
</tbody>
</table>

**AWSTOE command reference**

AWSTOE is a component management application that runs in the AWS CLI.

**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 (p. 53)](p. 53).

**run** (p. 144)

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

**validate** (p. 146)

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 <?>]
    [--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 (p. 118).

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`).

**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)

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.
Manage EC2 Image Builder resources

Resources are the building blocks that make up image pipelines, as well as the images those pipelines produce. This chapter covers creating, maintaining, and sharing Image Builder resources, including components, recipes, and images, along with infrastructure configuration and distribution settings.

**Note**

To help you manage your Image Builder resources, you can assign your own metadata to each resource in the form of tags. You use tags to categorize your AWS resources in different ways; for example, by purpose, owner, or environment. This is useful when you have many resources of the same type. You can more readily identify a specific resource based on the tags you've assigned to it.

For more information about tagging your resources using Image Builder commands in the AWS CLI, see the [Tag resources](#) section of this guide.

**Contents**

- Manage components with Image Builder (p. 148)
- Manage recipes (p. 162)
- Manage EC2 Image Builder images (p. 177)
- Manage EC2 Image Builder infrastructure configuration (p. 195)
- Manage EC2 Image Builder distribution settings (p. 203)
- Import and export virtual machine (VM) images with EC2 Image Builder (p. 222)
- Share EC2 Image Builder resources (p. 223)
- Tag EC2 Image Builder resources (p. 229)
- Delete EC2 Image Builder resources (p. 230)

Manage components with Image Builder

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.
Components

1. **Build stage** (pre-snapshot) – During the build stage, you make changes to the Amazon EC2 build instance that’s running your base image, to create the baseline for your new image. For example, your image or container recipe can include components that install an application, or modify the operating system firewall settings.

   The following component phases run during the build stage:
   - build
   - validate

   After this stage completes successfully, Image Builder creates a snapshot or container image that it uses for the test stage and beyond.

2. **Test stage** (post-snapshot) – During the test stage, Image Builder launches an EC2 instance from the snapshot or container image that was created as the final step of the build stage. Tests run on the new instance to validate settings and ensure that the instance is functioning as expected.

   The following component phase runs for every component that is included in the recipe during the 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 AWSTOE allows you to define many phases in a component document, Image Builder has strict rules about what phases it runs, and during which stages it runs them. For a component to run during the build stage, the component document must define at least one of these phases: build or validate. For a component to run during the test stage, the component document must define the test phase, 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 [Manage AWSTOE component parameters with EC2 Image Builder](p. 151).

To assist with troubleshooting on your build or test instance AWSTOE creates a log folder that contains the input document and log files to track what's happening each time a component runs. If you configured an Amazon S3 bucket in your pipeline configuration, the logs are also written there. For more information about YAML documents and log output, see [Use component documents in AWSTOE](p. 33).

**Tip**
When you have many components to keep track of, tagging helps you to identify a specific component or version based on the tags you've assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the [Tag resources](p. 229) 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.

**Contents**
- Create a YAML component document (p. 150)
- Manage AWSTOE component parameters with EC2 Image Builder (p. 151)
- List and view component details (p. 154)
- Create a component using the Image Builder console (p. 157)
- Create a component with the AWS CLI (p. 158)
Create a YAML component document

To build a component, provide a YAML application component document. This represents the phases and steps that you need to create the component.

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 (p. 116). For more information about the phases, steps, and syntax for AWSTOE YAML application component documents, see Use documents in AWSTOE.

**Note**

Image Builder determines the component types in the pipeline workflow. This workflow corresponds to the **Build stage** and the **Test stage** in the build process. Image Builder determines the component type as follows:

- **Build** – This is the default component type. Anything that is not classified as a test component, is a build component. This type of component runs during the **Build stage**. 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 Manage components with Image Builder (p. 148).

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 a file named *update-linux-os.yaml*. Include the following content:

```yaml
name: update-linux-os
description: Updates Linux with the latest security updates.
```

```yaml
# 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.
EC2 Image Builder User Guide
Component parameters

schemaVersion: 1
phases:
  - name: build
    steps:
      - name: UpdateOS
        action: UpdateOS

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 a file named update-windows-os.yaml. Include 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

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.

Manage AWSTOE component parameters with EC2 Image Builder

You can manage AWSTOE components, including creating and setting component parameters, directly from the EC2 Image Builder console, or by using AWS CLI commands, or one of the Image Builder SDKs. In this section, we'll cover creating and using parameters in your component, and setting component parameters through the Image Builder console and AWS CLI commands.

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
Use parameters in your YAML component document

To build a component, provide a YAML application component document. This represents the phases and steps that you need to create the component. The recipe that references the component can set the parameters to customize the values at runtime, with default values that take effect if the parameter is not set to a specific value.

**Create a component document with input parameters**

This section shows you how to define and use input parameters in your YAML component document.

To create a YAML application component document that uses parameters and runs commands in your Image Builder build or test instances, follow the steps that match your image operating system:

**Linux**

**Create a YAML component document**

Use a file editing tool to create a file named `hello-world-test.yaml`. Include the following content:

```yaml
# 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 }}"
```

Tips

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 file named `hello-world-test.yaml`. Include the following content:

```yaml
# 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
```

Tips

Use a tool like this online YAML Validator, or a YAML lint extension in your code environment to verify that your YAML is well-formed.

For more information about the phases, steps, and syntax for AWSTOE YAML application component documents, see Use documents in AWSTOE. For more information about parameters and their requirements, see the Parameters (p. 42) section of the Define and reference variables in AWSTOE page.

Create a component from the YAML component document

153
Whatever method you use to create an AWSTOE component, the YAML application component document is always required as a baseline.

- To use the Image Builder console to create a component directly from your YAML document, see Create a component using the Image Builder console (p. 157).
- To use Image Builder commands in the AWS CLI to create your component, see Create AWSTOE components with Image Builder with the AWS CLI (p. 158). Replace the YAML document name in those examples with the name of your Hello World YAML document (hello-world-test.yaml).

Set component parameters in an Image Builder recipe (console)

Setting component parameters works the same for image recipes and container recipes. When you create a new recipe, or a new version of a recipe, you choose which components to include from the Build components and Test components lists. The component lists include components that are applicable for the base operating system you chose for your image.

After you select a component, it is displayed in the Selected components section, directly under the component lists. Configuration options are shown for each component that is selected. If your component has input parameters defined, they are displayed as an expandable section called Input parameters.

The following parameter settings are shown for each parameter that’s defined for your component:

- Parameter name (not editable) – The name of the parameter.
- Description (not editable) – The parameter description
- Type (not editable) – The data type for the parameter value.
- Value – The value for the parameter. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the Value box with greyed-out text. If no other value is entered, AWSTOE uses the default value.

List and view component details

This section describes how you can find information and view details for the AWS Task Orchestrator and Executor (AWSTOE) components that you use in your EC2 Image Builder recipes.

Component details
- List AWSTOE components (p. 154)
- List component build versions (AWS CLI) (p. 156)
- Get component details (AWS CLI) (p. 156)
- Get component policy details (AWS CLI) (p. 157)

List AWSTOE components

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

AWS Management Console

To display a list of components in the AWS Management Console, follow these steps:

2. Select Components from the navigation pane. By default, Image Builder shows a list of components that your account owns.
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:

- **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 AWS components that your account owns.

```bash
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**

- Self
- Amazon
- ThirdParty
- Shared

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

```bash
aws imagebuilder list-components --owner Self
```

```json
{
    "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"
        }
    ]
}
```
List and view components

aws imagebuilder list-components --owner Amazon
aws imagebuilder list-components --owner Shared
aws imagebuilder list-components --owner ThirdParty

List component build versions (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 (p. 9).

```bash
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/example-component/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 (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).

```bash
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/example-component/1.0.1/1",
    "name": "examplecomponent",
    "version": "1.0.1",
    "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" }
  }
}
```
Create a component (console)

```
"platform": "Linux",
"owner": "123456789012",
"data": "name: HelloWorldTestingDocument
description: This is hello world testing document... etc.\"\n",
"encrypted": true,
"dateCreated": "2020-09-24T16:58:24.444Z",
"tags": {} }
```

Get component policy details (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
```

Create a component using the Image Builder console

To create an AWSTOE application component from the Image Builder console, follow these steps:

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 (p. 150).
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.

To create a new component version, follow these steps:
1. Depending on where you start:
   - From the Components list page – Select the check box next to the component name, then select Create new version from the Actions menu.
   - From the component detail page – Choose the Create new version button in the upper right corner of the heading.

2. The component information is already populated with the current values when the Create Component page displays. Follow the create a component steps to update the component. This ensures that you enter a unique semantic version in the Component version. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

Create a component with the AWS CLI

This section describes how to use Image Builder commands to create AWS Task Orchestrator and Executor (AWSTOE) components from the AWS Command Line Interface. To build a component, provide a YAML application component document. This represents the phases and steps that you need to create the component. To create a new YAML component document, see Create a YAML component document (p. 150).

Create AWSTOE components with Image Builder with 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.

  ```bash
  aws s3 cp update-linux-os.yaml s3://my-s3-bucket/my-path/update-linux-os.yaml
  ```

**Create a component from the YAML document**

To streamline the 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 in the prior steps. 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 action 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:

   ```json
   {
     "name": "update-linux-os",
     "semanticVersion": "1.1.2",
     "description": "An example component that updates the Linux operating system",
     "changeDescription": "Initial version.",
     "platform": "Linux",
     "uri": "s3://my-s3-bucket/my-path/update-linux-os.yaml",
     "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/98765432-b123-456b-7f89-0123456f789c",
     "tags": {
       "MyTagKey-purpose": "security-updates"
     }
   }
   ```

   **Note**
   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. **Create the component**

   Use the following command to create the component, referencing the file name for the JSON file that you created in the prior step:

   ```bash
   aws imagebuilder create-component --cli-input-json file://create-update-linux-os-component.json
   ```

   **Note**
   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

**Windows**

**Store your application component document in Amazon S3**

You can use an S3 bucket as a repository for your AWSTOE application component source document. To store your component document, follow these steps:
EC2 Image Builder User Guide
Create a component with the AWS CLI

- **Upload the document to Amazon S3**

  *If your document is smaller than 64 KB, you can skip this step. Documents that are 64 KB or larger in size must be stored in Amazon S3.*

  ```
  aws s3 cp update-windows-os.yaml s3://my-s3-bucket/my-path/update-windows-os.yaml
  ```

**Create a component from the YAML document**

To streamline the `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 in the prior steps. 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 action 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://my-s3-bucket/my-path/update-windows-os.yaml",
   "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/98765432-b123-456b-7f89-0123456f789c",
   "tags": {
   "MyTagKey-purpose": "security-updates"
   }
   }
   ```

   **Note**

   - You must include the file:// notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. **Create the component**

   Use the following command to create the component, referencing the file name for the JSON file that you created in the prior step:

   ```
   aws imagebuilder create-component --cli-input-json file://create-update-windows-os-component.json
   ```
Note

- You must include the file:// notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

AWSTOE component versioning for updates (AWS CLI)

AWSTOE component names and versions are embedded in the component's Amazon Resource Name (ARN), after the component prefix. Each new version of a component has its own unique ARN. The steps to create a new version are exactly the same as the steps to create a new component, as long as the semantic version is unique for that component name. To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

To ensure that you assign the next logical version, first get a list of the existing versions for the component that you want to change. 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.

```bash
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 component (AWS CLI)

For some scenarios, it might be easier to start with a pre-existing script. For this scenario, you can use the following example.
This example assumes that you have a file called `import-component.json` (as shown). Note that the file directly references a PowerShell script called `AdminConfig.ps1` that is already uploaded to `my-s3-bucket`. Currently, SHELL is supported for the component format.

```json
{
    "name": "MyImportedComponent",
    "semanticVersion": "1.0.0",
    "description": "An example of how to import a component",
    "changeDescription": "First commit message.",
    "format": "SHELL",
    "platform": "Windows",
    "type": "BUILD",
    "uri": "s3://my-s3-bucket/AdminConfig.ps1",
    "kmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/60763706-b131-418b-8f85-3420912f020c"
}
```

To import the component, run the following command.

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

## Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see [Delete EC2 Image Builder resources](p. 230).

## Manage recipes

An EC2 Image Builder recipe defines the base image to use as your starting point to create a new image, along with the set of components that you add to customize your image and verify that everything works as expected. Image Builder provides automatic version choices for each component. The number of components that you can apply to a recipe is limited to 20 components overall. 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](p. 229) section of this guide.

**Tip**

You can use Amazon managed components in your recipes, or you can develop your own custom components with the AWS Task Orchestrator and Executor (AWSTOE) application. To get started, see [Get started with AWSTOE](p. 25).

This section covers how to list, view, and create recipes.

### Contents

- [List and view image recipe details](p. 163)
- [List and view container recipe details](p. 164)
- [Create a new version of an image recipe](p. 165)
- [Create a new version of a container recipe](p. 172)
- [Clean up resources](p. 177)
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 (console)** (p. 163)
- **List image recipes (AWS CLI)** (p. 163)
- **View image recipe details (console)** (p. 163)
- **Get image recipe details (AWS CLI)** (p. 164)
- **Get image recipe policy details (AWS CLI)** (p. 164)

**List image recipes (console)**

To see a list of the image recipes created under your account in the Image Builder console, follow these steps:

2. Choose **Image recipes** from the navigation pane. This shows a list of the image recipes that are created under your account.
3. To view details or create a new recipe version, choose the **Recipe name** link. This opens the detail view for the recipe.

   **Note**
   You can also select the check box next to the **Recipe name**, then choose **View details**.

**List image recipes (AWS CLI)**

The following example shows how to list all of your image recipes, using the AWS CLI.

```
aws imagebuilder list-image-recipes
```

**View image recipe details (console)**

To view details for a specific image recipe using the Image Builder console, select the image recipe to review, using the steps described in **List image recipes (console)** (p. 163).

On the recipe detail page, you can:

- Delete the recipe. For more information about deleting resources in Image Builder, see **Delete EC2 Image Builder resources** (p. 230).
- Create a new version.
- Create a pipeline from the recipe. After choosing **Create pipeline from this recipe**, you are taken to the pipeline wizard. For more information about creating an Image Builder pipeline using the pipeline wizard, see **Create an image pipeline using the EC2 Image Builder console wizard** (p. 13)

   **Note**
   When you create a pipeline from an existing recipe, the option to create a new recipe is not available.
Get image recipe details (AWS CLI)

The following example shows how to use an `imagebuilder` CLI command to get the details of an image recipe by specifying its Amazon Resource Name (ARN).

```
```

Get image recipe policy details (AWS CLI)

The following example shows how to use an `imagebuilder` CLI command to get the details of an image recipe policy by specifying its ARN.

```
```

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 (p. 164)
- List container recipes with the AWS CLI (p. 164)
- View container recipe details in the console (p. 164)
- Get container recipe details with the AWS CLI (p. 165)
- Get container recipe policy details with the AWS CLI (p. 165)

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:

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 (p. 164).
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 EC2 Image Builder resources (p. 230).
- Create a new version.
- Create a pipeline from the recipe. After you 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 Create an image pipeline using the EC2 Image Builder console wizard (p. 13).
  
  **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.

```bash
```

### 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.

```bash
```

### 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 (console) (p. 165)
- Create an image recipe with the AWS CLI (p. 167)
- Import a VM as your base image in the console (p. 170)

#### Create a new image recipe version (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 (p. 9).

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 ID** – Required. However, this setting is not pre-filled with your original entry. You must enter the AMI ID for your base image.
- **Instance configuration** – Settings are pre-selected, but you can edit them.
- **Systems Manager agent** – You can select or clear this check box to control installation of the Systems Manager agent on the new image. The check box is cleared by default to include the Systems Manager agent in your new image. To remove the Systems Manager agent from the final image, 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.

**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 EC2 Image Builder (p. 299).
- **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 or update components for your recipe, select the owner type for the component.
Create a new version of an image recipe

you're looking for. For example, if you want to add a component that's associated with a base image that you subscribed to in AWS Marketplace, select Third party managed from the owner type list, next to the search bar.

You can configure the following settings for your selected component:

- **Versioning options** – Pre-selected, but you can change them. We recommend that you choose the Use latest available component version option to ensure that your image builds always pick up the latest version of the component. If you need to use a specific component version in your recipe, you can choose Specify component version, and enter the version in the Component version box that appears.

- **Input parameters** – Displays input parameters that the component accepts. The Value is pre-filled with the value from the prior version of the recipe. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the Value box with greyed-out text. If no other value is entered, AWSTOE uses the default value.

  **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.

**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 image recipe.

For more information on how to create an image recipe when you create an image pipeline, see Step 2: Choose recipe (p. 14) 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 component with the AWS CLI (p. 158) 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 action request parameters. To review the API command 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. The value can be the base image ARN or an AMI ID.

Note
The Linux example uses an Image Builder AMI, and the Windows example uses an ARN.

- **semanticVersion** (string, required) – The semantic version of the image recipe, 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 (p. 9).

- **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.

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.
Create a new version of an image recipe

- **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==`:

  ```bash
  #!/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.

```json
{
  "name": "BB Ubuntu Image recipe",
  "description": "Hello World image recipe for Linux.",
  "parentImage": "ami-0a01b234c5de6fabc",
  "semanticVersion": "1.0.0",
  "components": [
    {
      "componentArn": "arn:aws:imagebuilder:us-west-2:123456789012:component/bb"
    }
  ],
  "additionalInstanceConfiguration": {
    "systemsManagerAgent": {
      "uninstallAfterBuild": true
    },
    "userDataOverride": "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 in the SKU based on the semantic
Create a new version of an image recipe


```
{
    "name": "MyBasicRecipe",
    "description": "This example image recipe creates a Windows 2016 image.",
    "semanticVersion": "1.0.0",
    "components": [
        {
            "componentArn": "arn:aws:imagebuilder:us-west-2:123456789012:component/my-example-component/2019.12.02/1"
        },
        {
            "componentArn": "arn:aws:imagebuilder:us-west-2:123456789012:component/my-imported-component/1.0.0/1"
        }
    ]
}
```

Note
To learn more about semantic versioning for Image Builder resources, see Semantic versioning (p. 9).

2. Create the recipe

Use the following command to create the recipe. 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, and Linux uses the forward slash (/).

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 Create an image pipeline (AMI) (p. 13). For additional steps to create a new image recipe or recipe version, see Create a new version of an image recipe (p. 165).

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 (p. 222).

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 (p. 172)
- Create a container recipe with the AWS CLI (p. 174)

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.

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 (p. 9)](#).
ECR image
- **Image Operating System (OS)** – Pre-selected, but editable.
- **OS version** – Pre-selected, but editable.
- **ECR image ID** – Pre-filled, but editable.

Docker Hub image
- **Image Operating System (OS)** – *Not editable.*
- **OS version** – Pre-selected, but editable.
- **Docker image ID** – Pre-filled, but editable.

Instance configuration
- **AMI ID** – Pre-filled, but editable.
- **Storage (volumes)**
  - **EBS volume 1 (AMI root)** – Pre-filled. You 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 or update components for your recipe, select the owner type for the component you're looking for. For example, if you want to add a component that's associated with a base image that you subscribed to in AWS Marketplace, select **Third party managed** from the owner type list, next to the search bar.

You can configure the following settings for your selected component:
- **Versioning options** – Pre-selected, but you can change them. We recommend that you choose the **Use latest available component version** option to ensure that your image builds always pick up the latest version of the component. If you need to use a specific component version in your recipe, you can choose **Specify component version,** and enter the version in the **Component version** box that appears.
- **Input parameters** – Displays input parameters that the component accepts. The **Value** is pre-filled with the value from the prior version of the recipe. If you are using this component for the first time in this recipe, and a default value was defined for the component, the default value appears in the **Value** box with greyed-out text. If no other value is entered, AWSTOE uses the default value.
**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?](https://docs.aws.amazon.com/secretsmanager/latest/userguide/what-is-secrets-manager.html) in the AWS Secrets Manager User Guide. For more information about AWS Systems Manager Parameter Store, see [AWS Systems Manager Parameter Store](https://docs.aws.amazon.com/systemsmanager/latest/userguide/parameter-store.html) 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.

**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](p. 18) 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 component with the AWS CLI](p. 158) 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 action request parameters. To review the API command request parameters, see the `CreateContainerRecipe` command in the [EC2 Image Builder API Reference](https://docs.aws.amazon.com/AmazonEC2/latest/APIReference/EC2-CreateContainerRecipe.html).
   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.

  **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) – Image that the container recipe uses as a base for your customized image. The value can be the base image ARN or an AMI ID.

- **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* (p. 9).

- **tags** (string map) – Tags that are attached to the container recipe.

- **instanceConfiguration** (object) – A group of options that can be used to configure an instance for building and testing container images.

  - **image** (string) – The AMI ID to use as the base image for a container build and test instance. If you don't specify this value, 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.
Create a new version of a container recipe

- **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-east-1:123456789012:component/helloworldal2/x.x.x"
    }
  ],
  "containerType": "DOCKER",
  "description": "My Linux Docker container image",
  "dockerfileTemplateData": "FROM
  "name": "amazonlinux-container-recipe",
  "parentImage": "amazonlinux:latest",
  "platformOverride": "Linux",
  "semanticVersion": "1.0.2",
  "tags": {
    "sometag": "Tag detail"
  },
  "instanceConfiguration": {
    "image": "ami-1234567890",
    "blockDeviceMappings": [
      {
        "deviceName": "/dev/xvda",
        "ebs": {
          "deleteOnTermination": true,
          "encrypted": false,
          "volumeSize": 8,
          "volumeType": "gp2"
        }
      }
    ],
    "targetRepository": {
      "repositoryName": "myrepo",
      "service": "ECR"
    },
    "workingDirectory": "/tmp"
  }
}
```

2. **Create the recipe**

Use the following command to create the recipe. 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
```
Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 230).

Manage EC2 Image Builder images

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 (p. 229) section of this guide.

This section covers how to list, view, and create images.

Contents

- List images and build versions (p. 177)
- View image details (p. 182)
- Create images (p. 188)
- Import a VM image (p. 189)
- Manage security findings for Image Builder images (p. 192)
- Clean up resources (p. 195)

List images and build versions

This section describes the different ways that you can list information about your EC2 Image Builder images.

Contents

- List images (p. 177)
- List image build versions (p. 180)

List images

On the Images list page in the Image Builder console, you can get a summary of the Image Builder image 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 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.

**Console**

**Image details**

Details on the image list page in the Image Builder console include the following:

- **Image name** – The name of the current version of the image resource.
- **Type** – The type of output that Image Builder distributed when it created this image resource (an AMI or a container image).
- **Version** – The current version of the image resource. In the Image Builder console, the version links to a list of build versions for the image.
- **Image source** – The origin of the base image that Image Builder used to build this image.
- **Platform** – The operating system platform of the image resource version, for example, "Windows" or "Linux".
- **Date created** – The date and time when Image Builder created the current version of the image resource.
- **Owner** – The owner of the current version of the image resource.
- **ARN** – The Amazon Resource Name (ARN) of the current version of the image resource.

**List images**

To list image resources in the Image Builder console, perform the following steps:

2. Choose **Images** from the navigation pane.

**Filter results**

On the Images list page, you can search for specific images, and filter these images with the available filters next to the search bar, as follows:

**Ownership**

Beside the search bar is a dropdown list that shows ownership. By default, Image Builder displays images that your account owns. You can change the selection to view other image resources that you have access to.

- **Owned by me** (default selection) – Image resources that your account owns.
- **Amazon-managed** – Image resources that Amazon owns and manages.
- **Shared with me** – Image resources that other accounts have shared with you.

**Operating system**

You can filter by the image operating system (OS), or view all types. The default is to view all operating systems.

- **Any** (default selection)
- **Linux**
List images and build versions

- **Windows**

*Image type*

You can filter by the type of image, or view all types. The default is to view all image types.

- **Any type** (default selection)
- **AMI**
- **Docker**

*Image source*

If you want to see only images that are imported from a virtual machine, you can choose **VMIE**. The default is to view all image types. If you see a dash in the **Image source** field, it means that the image was created before this detail was available.

**AWS CLI**

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 and 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"
      }
   ]
}
```

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*. 

### List images and build versions

**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 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](https://docs.aws.amazon.com/prophecypaperless-sdk/latest/APIReference/API_ListImageBuildVersions.html) 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:

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**

```bash
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.

```json
{
  "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 2023-03-10T01:04:40.609Z",
            "description": "First verison of image-recipe-name",
            "accountId": "123456789012"
          }
        ]
      }
    },
    {
      "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",
      "owner": "123456789012",
      "dateCreated": "2023-03-10T01:04:40.609Z",
      "outputResources": {
        "amis": [
          {
            "region": "us-west-2",
            "image": "ami-012b3456789012c3d",
            "name": "image-recipe-name 2023-03-10T01:04:40.609Z",
            "description": "First version of image-recipe-name",
            "accountId": "123456789012"
          }
        ]
      }
    }
  ],
  "tags": {}
}
```
"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 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
• View image details in the Image Builder console (p. 182)
• Get image policy details (AWS CLI) (p. 187)

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.

Console detail sections and tabs
• Summary section (p. 183)
• Output resources tab (p. 183)
• Infrastructure configuration tab (p. 184)
• Distribution settings tab (p. 184)
• Workflow tab (p. 185)
• Security findings tab (p. 187)
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 EC2 Image Builder distribution settings (p. 203).

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 EC2 Image Builder infrastructure configuration (p. 195).

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 –
- Associated license configurations – The License Manager license configuration ARNs to associate with the AMI in the specified Region.
- Launch template configuration –
• **Set launch template default version** –

**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 EC2 Image Builder distribution settings (p. 203).

**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 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, as follows:

**Image workflows (AMI output)**

• build-image
• test-image

**Container workflows (container output)**

• build-container
• test-container
• distribute-container

**Note**

If the workflow hasn't started yet, it doesn't appear in the list. For example, if your image build has just started, build-image is the only workflow type that appears in the list. When the next workflow begins, test-image in this case, 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 build 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 build instance. For more information, see Configure security scans for Image Builder images in the AWS Management Console (p. 193)

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 build 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 (p. 194).

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 (AWS CLI)

The following example shows how to get the details of an image policy with its Amazon Resource Name (ARN).

```bash
```
Create images

This section shows you how to create Image Builder images and cancel a build that's in progress.

Contents
  • Create an image (p. 188)
  • Cancel image creation (AWS CLI) (p. 189)

Create an image

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. 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 an image in the AWS Management Console, you can manually run an existing pipeline, as follows:

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 build 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.

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**

```
```

**Output:**

```
{
    "requestId": "1abcd234-e567-8fa9-0123-4567b890cd12",
    "imageVersionList": [
      {
        "arn": "arn:aws:imagebuilder:us-west-2:123456789012:image/recipe-name/1.0.0",
        "name": "recipe-name",
        ...
      }
    ]
}
```

**Cancel image creation (AWS CLI)**

To cancel an in-progress image build, use the cancel-image-creation command, as follows:

```
```

**Import a VM image**

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.

**Import a VM (console)**

To import a VM with the Image Builder console, follow these steps:

2. Choose **Images** from the navigation pane.
3. Choose **Import image**.
4. Provide details for the following sections on the Import image page. Then choose Import image when you're done.

**General**

1. Specify a unique Name for the base image.
2. Specify a Version for the base image. Use the following format: major.minor.patch.
3. You can also enter an optional Description for the base image.

**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**

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 (p. 222).

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.

**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](https://docs.aws.amazon.com/wizard/)

**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**.

**Import a VM (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](https://docs.aws.amazon.com/vmimport/)

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 action request parameters. To review the API command request parameters, see the ` importVmImage` command in the [EC2 Image Builder API Reference](https://docs.aws.amazon.com/)

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 (p. 9)](https://docs.aws.amazon.com/)
- **description** (string) – The description of the image recipe.
- **platform** (string, required) – The operating system platform for the imported VM.
Manage security findings

- **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.

- **clientToken** (string, required) – A unique, case-sensitive identifier you provide to ensure idempotency of the request. For more information, see [Ensuring idempotency](#) in the Amazon EC2 API Reference.

- **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.

```json
{
  "name": "example-request",
  "semanticVersion": "1.0.0",
  "description": "vm-import-test",
  "platform": "Linux",
  "vmImportTaskId": "import-ami-01ab234567890cd1e",
  "clientToken": "asz1231231234cs3z",
  "tags": {
    "Usage": "VMIE"
  }
}
```

3. **Import the image**

Run the `import-vm-image` command, with the file that you created as input:

```bash
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, and Linux uses the forward slash (`/`).

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 for your build instances 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
Manage security findings

- 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 for your image 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:

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:

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 details](#) (p. 182).
  - 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:
Clean up resources

To avoid unexpected charges, make sure to clean up resources and pipelines that you created from the examples in this guide. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 230).

Manage EC2 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.
- 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 Prerequisites (p. 11).
- The VPC, subnet, and security groups for your pipeline’s build and test instances.
List and view infrastructure configurations

- 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 (p. 258).

  **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

- List and view infrastructure configuration details (p. 196)
- Create an infrastructure configuration (p. 197)
- Update an infrastructure configuration (p. 198)
- EC2 Image Builder and interface VPC endpoints (AWS PrivateLink) (p. 200)

  **Tip**
  When you have multiple resources of the same type, tagging helps you to identify a specific resource based on the tags you’ve assigned to it. For more information about tagging your resources using Image Builder commands in the AWS CLI, see the Tag resources (p. 229) 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 (AWS CLI) (p. 196)
- Get infrastructure configuration details (AWS CLI) (p. 196)

List infrastructure configurations (AWS CLI)

The following example shows how to list all of your infrastructure configurations, using the `list-infrastructure-configurations` command in the AWS CLI.

```
aws imagebuilder list-infrastructure-configurations
```

Get infrastructure configuration details (AWS CLI)

The following example shows how to 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).

```
```
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:

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 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 all of the 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.
   - **VPC, subnet, and security groups** – Image Builder uses your default VPC and subnet. For more information about configuring VPC interface endpoints, see [EC2 Image Builder and interface VPC endpoints (AWS PrivateLink)](p. 200).
   - 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.

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 [EC2 Image Builder and interface VPC endpoints (AWS PrivateLink)](p. 200).

---

197
AWS CLI

The following example shows how to configure the infrastructure for your image with the Image Builder `create-infrastructure-configuration` command in the AWS CLI.

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

   This infrastructure configuration 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 Prerequisites (p. 11).

   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 `create-infrastructure-configuration.json`:

   ```json
   {
     "name": "MyExampleInfrastructure",
     "description": "An example that will retain instances of failed builds",
     "instanceTypes": [
       "m5.large", "m5.xlarge"
     ],
     "instanceProfileName": "myIAMInstanceProfileName",
     "securityGroupIds": [
       "sg-12345678"
     ],
     "subnetId": "sub-12345678",
     "logging": {
       "s3Logs": {
         "s3BucketName": "my-logging-bucket",
         "s3KeyPrefix": "my-path"
       }
     },
     "keyPair": "myKeyPairName",
     "terminateInstanceOnFailure": false,
   }
   ```

2. **Use the file you created as input when you run the following command.**

   ```sh
   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.

**Console**

You can edit the following infrastructure configuration details from the Image Builder console:
To update an infrastructure configuration resource from the Image Builder console, follow these steps:

**Choose an existing Image Builder infrastructure configuration**

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*:

   ```json
   {
     "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",
         ..."
2. Use the file you created as input when you run the following command.

```bash
aws imagebuilder update-infrastructure-configuration --cli-input-json file://update-infrastructure-configuration.json
```

### EC2 Image Builder and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and EC2 Image Builder by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Image Builder APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Image Builder APIs. Traffic between your VPC and Image Builder does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets. 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](p. 201).

For more information about VPC endpoints, see [Interface VPC endpoints (AWS PrivateLink)](ec2-image-builder-endpoints-and-quotas) in the [Amazon VPC User Guide](amazon-vpc-user-guide).

### Considerations for Image Builder VPC endpoints


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](https://docs.aws.amazon.com/vpc/latest/UserGuide/creating-an-endpoint.html) in the [Amazon VPC User Guide](https://docs.aws.amazon.com/vpc/latest/UserGuide/)

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](https://docs.aws.amazon.com/vpc/latest/UserGuide/creating-an-endpoint.html) in the [Amazon Web Services General Reference](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/creating-an-endpoint.html).
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:

```
arin: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](https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-endpoints.html) 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.
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 AWSTOE 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:region: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.

```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 EC2 Image Builder distribution settings

After you create distribution settings with Image Builder, you can manage them using the Image Builder console, the Image Builder API, or `imagebuilder` commands in the AWS CLI. 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`. See the examples for making an AMI public at EC2 `ModifyImageAttribute`.

- 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 Create distribution settings for output VM disks (AWS CLI) (p. 210).

**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 (p. 229) section of this guide.

This topic covers how to list, view, and create distribution settings.

### Contents

- List and view distribution settings detail (p. 204)
- Create and update AMI distribution configurations (p. 205)
- Create and update distribution settings for container images (p. 213)
- Set up cross-account AMI distribution with Image Builder (p. 215)
- Configure AMI distribution settings to use an Amazon EC2 launch template (p. 220)

### List and view distribution settings detail

This section describes the various ways that you can find information and view details for your EC2 Image Builder distribution settings.

**Distribution settings detail**

- List distribution configurations (console) (p. 204)
- View distribution configuration details (console) (p. 205)
- List distributions (AWS CLI) (p. 205)
- Get distribution configuration detail (AWS CLI) (p. 205)

### List distribution configurations (console)

To see a list of the distribution configurations created under your account in the Image Builder console, follow these steps:

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.
View distribution configuration details (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 (console) (p. 204).

On the distribution detail page, you can:

- **Delete** the distribution configuration. For more information about deleting resources in Image Builder, see Delete EC2 Image Builder resources (p. 230).
- **Edit** distribution details.

List distributions (AWS CLI)

The following example shows how to use the list-distribution-configurations command in the AWS CLI to list all of your distributions.

```bash
aws imagebuilder list-distribution-configurations
```

Get distribution configuration detail (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).

```bash
```

Create and update AMI distribution configurations

This section covers creating and updating distribution configurations for an Image Builder AMI.

Contents

- Create an AMI distribution configuration (console) (p. 205)
- Create distribution settings for output AMIs (AWS CLI) (p. 207)
- Update AMI distribution settings (console) (p. 209)
- Create distribution settings for a Windows AMI with EC2 Fast Launch enabled (AWS CLI) (p. 209)
- Create distribution settings for output VM disks (AWS CLI) (p. 210)
- Update AMI distribution settings (AWS CL) (p. 212)

Create an AMI distribution configuration (console)

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.
To create a new AMI distribution configuration:

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 target accounts in the target Regions, and attach the **Ec2ImageBuilderCrossAccountDistributionAccess policy** (p. 289) 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.

b. To add distribution settings for another Region, choose **Add Region**.

7. Choose **Create settings** when you are done.

## Create distribution settings for output AMIs (AWS CLI)

A distribution configuration allows you to specify the name and description of your output AMI, authorize other AWS accounts to launch the AMI, copy the AMI to other accounts, and replicate the AMI to other AWS Regions. It also allows you to export the AMI to Amazon Simple Storage Service (Amazon S3), 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.

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.

**Accounts**

This example distributes an AMI to two Regions, and specifies AWS accounts that have launch permissions in each Region.

```json
{
  "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",
```
Create and update AMI distribution

```
"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"
    ]
  }
}
}
```

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"
      ]
    }
    }
  ]
}
```

2. Run the following command, using the file you created as input.

```
aws imagebuilder create-distribution-configuration --cli-input-json file://create-ami-distribution-configuration.json
```

Note

- You must include the file:// notation at the beginning of the JSON file path.
The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see create-distribution-configuration in the AWS CLI Command Reference.

Update AMI distribution settings (console)

You can change your AMI distribution settings using the Image Builder console. The updated distribution settings are used for all automated and manual pipeline deployments going forward. 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.

Update AMI distribution configuration

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 (console) (p. 205).
5. Choose Save changes when you are done.

Create distribution settings for a Windows AMI with EC2 Fast Launch enabled (AWS CLI)

The following example shows how to use the create-distribution-configuration command to create distribution settings that have EC2 Fast Launch configured 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 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.

   ```json
   {
   "name": "WinFastLaunchDistribution",
   "description": "An example of Windows AMI EC2 Fast Launch settings in the distribution configuration.",
   "distributions": [
   {"region": "us-west-2",
   ```
Create and update AMI distribution

```
"amiDistributionConfiguration": {
    "name": "Name {{imagebuilder:buildDate}}",
    "description": "Includes Windows AMI EC2 Fast Launch settings with cross-account distribution.",
    "amiTags": {
        "KeyName": "Some Value"
    }
},
"fastLaunchConfigurations": [
    {
        "enabled": true,
        "snapshotConfiguration": {
            "targetResourceCount": 5
        },
        "maxParallelLaunches": 6,
        "launchTemplate": {
            "launchTemplateId": "lt-0ab1234c56d789012",
            "launchTemplateVersion": "1"
        },
        "accountId": "123456789012"
    }
],
"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, and Linux uses the forward slash (`/`).

For more detailed information, see [create-distribution-configuration](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/imagebuilder/create-distribution-configuration.html) in the AWS CLI Command Reference.

### Create distribution settings for output VM disks (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.
The naming convention for the data values in the JSON file follows the pattern that is specified for the Image Builder API action request parameters. To review the API command 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, and Linux uses the forward slash (/).
Update AMI distribution settings (AWS CLI)

The following example shows how to use the `update-distribution-configuration` command to update distribution settings for your AMI, using the AWS CLI.

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

   Use your favorite file-editing tool to create a JSON file with the keys shown in the following example, plus values that are valid for your environment. This example uses a file named `update-ami-distribution-configuration.json`.

   ```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.**

   ```bash
   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.
Create and update distribution settings for container images

This section covers creating and updating distribution settings for Image Builder container images.

Contents

• Create distribution settings for Image Builder container images (AWS CLI) (p. 213)
• Update distribution settings for your container image (AWS CLI) (p. 214)

Create distribution settings for Image Builder container images (AWS CLI)

A distribution configuration enables you to specify the name and description of your output container image and replicate the container image to other AWS Regions. You can also apply separate tags to the distribution configuration resource and to the container images within each Region.

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

   Use your favorite file-editing tool to create a JSON file with the keys shown in the following example, plus values that are valid for your environment. This example uses a file named `create-container-distribution-configuration.json`:

   ```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"]
     }
   ]
   }
   ```
Create and update container image distribution

2. Run the following command, using the file you created as input.

```
aws imagebuilder create-distribution-configuration --cli-input-json file://create-container-distribution-configuration.json
```

Note

- You must include the file:// notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see `create-distribution-configuration` in the AWS CLI Command Reference.

Update distribution settings for your container image (AWS CLI)

The following example shows how to use the `update-distribution-configuration` command to update distribution settings for your container image, using the AWS CLI. You can also update tags for the container images within each Region.

1. Create a CLI input JSON file

   Use your favorite file-editing tool to create a JSON file with keys shown in the following example, plus values that are valid for your environment. This example uses a file named `update-container-distribution-configuration.json`:

   ```
   {
   "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", "image2"]
   }
   ],
   "tags": {
   "DistributionConfigurationTestTagKey1": "DistributionConfigurationTestTagValue1",
   "DistributionConfigurationTestTagKey2": "DistributionConfigurationTestTagValue2"
   }
   }
   ```
Set up cross-account AMI distribution

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](p. 167). To create the JSON file for an infrastructure configuration, see [Create an infrastructure configuration](p. 197).

**Prerequisites**
To ensure that target accounts can successfully launch instances from your Image Builder image, you must configure the appropriate permissions for all destination accounts in all Regions.

If you encrypt your AMI using 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.

Run the following command, using the file you created as input:

```bash
aws imagebuilder update-distribution-configuration --cli-input-json file://update-container-distribution-configuration.json
```

**Note**
- You must include the `file://` notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

For more detailed information, see [update-distribution-configuration](in the [AWS CLI Command Reference](aws cli command reference)). To update tags for your distribution configuration resource, see the [Tag resources](p. 229) section.
Source account requirements

- Create a KMS key in your account in all Regions where you build and distribute your AMI. You can also use an existing key.
- Update the key policy for all of those keys to allow destination accounts to use your key.

Destination account requirements

- Add an inline policy to EC2ImageBuilderDistributionCrossAccountRole that allows the role to perform the required actions to distribute an encrypted AMI. For IAM configuration steps, see the IAM policies (p. 216) 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.

```json
{
  ...
  "blockDeviceMappings": [
  {
    "deviceName": "Example root volume",
    "ebs": {
      "deleteOnTermination": true,
      "encrypted": true,
      "iops": 100,
      "kmsKeyId": "image-owner-key-id",
      ...
    },
    ...
  }],
  ...
}
```

IAM policies

To configure cross-account distribution permissions in AWS Identity and Access Management (IAM), follow these steps:

1. To use Image Builder AMIs that are distributed across accounts, the destination account owner must create a new IAM role in their account called EC2ImageBuilderDistributionCrossAccountRole.
2. They must attach the Ec2ImageBuilderCrossAccountDistributionAccess policy (p. 289) to the role to enable cross-account distribution. For more information about managed policies, see Managed Policies and Inline Policies in the AWS Identity and Access Management User Guide.
3. Verify that the source account ID is added to the trust policy attached to the IAM role of the destination account. For more information about trust policies, see Resource-Based Policies in the AWS Identity and Access Management User Guide.
4. If the AMI you distribute is encrypted, the destination account owner must add the following inline policy to the EC2ImageBuilderDistributionCrossAccountRole in their account so that they can use your KMS keys. The Principal section contains their account number. This enables Image Builder to act on their behalf when it uses AWS KMS to encrypt and decrypt the AMI with the appropriate keys for each Region.

```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.

```json
{
    "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:ResourceTag/CreatedBy": "EC2 Image Builder"
                }
            }
        }
    ]
}
```
Limits for cross-account distribution

There are some limitations when distributing Image Builder images across accounts:

- The destination account is limited to 50 concurrent AMI copies for each destination Region.
- If you want to copy a paravirtual (PV) virtualization AMI to another Region, the destination Region must support PV virtualization AMIs. For more information, see Linux AMI virtualization types.
- You cannot create an unencrypted copy of an encrypted snapshot. If you 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 (console)

This section describes how to create and configure distribution settings for cross-account distribution of your Image Builder AMIs using the AWS Management Console. Configuring cross-account distribution requires specific IAM permissions. You must complete the Prerequisites (p. 215) for this section before you continue.

To create distribution settings in the Image Builder console, follow these steps:

2. Choose Distribution settings from the navigation pane. This shows a list of the distribution settings that are created under your account.
3. At the top of the Distribution settings page, choose Create distribution settings. This takes you to the Create distribution settings page.
4. In the Image type section, choose Amazon Machine Image (AMI) as the Output type. This is the default setting.
5. In the General section, enter the Name of the distribution settings resource that you want to create (required).
6. In the Region settings section, enter a 12-digit account ID that you want to distribute your AMI to in Target accounts for the selected Region, and press Enter. This checks for the correct formatting, and then displays the account ID that you entered below the box. Repeat the process to add more accounts.

   To remove an account that you entered, choose the X displayed to the right of the account ID.

   Enter the Output AMI name for each Region.

7. Continue specifying any additional settings that you require, and choose Create settings to create your new distribution settings resource.
Configure cross-account distribution for an Image Builder AMI (AWS CLI)

This section describes how to configure a distribution settings file and use the `create-image` command in the AWS CLI to build and distribute an Image Builder AMI across accounts.

Configuring cross-account distribution requires specific IAM permissions. You must complete the Prerequisites (p. 215) 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.

   ```json
   {
     "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.

   ```bash
   aws imagebuilder create-distribution-configuration --name my distribution name --cli-input-json file://create-distribution-configuration.json
   ```

   **Note**

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

   You can also provide JSON directly in the command, using the `--distributions` parameter.
Configure AMI distribution settings to use an Amazon EC2 launch template

To help ensure a consistent launch experience for your Image Builder AMI in target accounts and Regions, you can specify an Amazon EC2 launch template in your distribution settings, using launchTemplateConfigurations. When launchTemplateConfigurations are present during the 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

Add an Amazon EC2 launch template to your AMI distribution settings (console)

To provide a launch template with your output AMI, follow these steps in the console:

2. Choose Distribution settings from the navigation pane. This shows a list of the distribution settings that are created under your account.
3. At the top of the Distribution settings page, choose Create distribution settings. This opens the Create distribution settings page.
4. In the Image type section, choose the Amazon Machine Image (AMI) Output type. This is the default setting.
5. In the General section, enter the Name of the distribution settings resource that you want to create (required).
6. In the Region settings section, select the name of an EC2 launch template from the list. If there are no launch templates in your account, choose Create new launch template, which opens the Launch Templates in the EC2 Dashboard.

   Select the Set the default version check box to update the launch template default version to the new version that Image Builder creates with your output AMI.

   To add another launch template to the selected Region, choose Add launch template configuration.

   To remove a launch template, choose Remove.
7. Continue specifying any additional settings that you require, and choose Create settings to create your new distribution settings resource.

Add an Amazon EC2 launch template to your AMI distribution settings (AWS CLI)

This section describes how to configure a distribution settings file with a launch template, and use the create-image command in the AWS CLI to build and distribute an Image Builder AMI and a new version of the launch template that uses it.
Specifying an AMI launch template

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.

   ```json
   {
     "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.

   ```bash
   aws imagebuilder create-distribution-configuration --name my_distribution_name --cli-input-json file://create-distribution-config-launch-template.json
   
   Note
   
   - You must include the file:// notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

   You can also provide JSON directly in the command, using the `--distributions` parameter.
Import and export virtual machine (VM) images with EC2 Image Builder

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'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.

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 directly, see Import a VM (console) (p. 189). For more information about importing as part of your image recipe, see VM import configuration (p. 170).

Import a VM into Image Builder (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 action request parameters. To review the API command request parameters, see the ImportVmImage 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 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 (p. 9).
• 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.
• clientId (string, required) – A unique, case-sensitive identifier you provide to ensure idempotency of the request. For more information, see `Ensuring idempotency` in the `Amazon EC2 API Reference`.
• 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.

```json
{
    "name": "example-request",
    "semanticVersion": "1.0.0",
    "description": "vm-import-test",
    "platform": "Linux",
    "vmImportTaskId": "import-ami-01ab234567890cd1e",
    "clientId": "asz1231231234cs3z",
    "tags": {
        "Usage": "VMIE"
    }
}
```

3. Import the image

Run the `import-vm-image` command, with the file that you created as input:

```bash
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, and Linux uses the forward slash (`/`).

### Distribute VM disks from your image build (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 `Create distribution settings for output VM disks (AWS CLI)` (p. 210).

### Share EC2 Image Builder resources

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

• Components
• Images
Working with shared resources

Recipes

This section provides information to help you share these EC2 Image Builder resources.

Section contents

- Working with shared components, images, and recipes in EC2 Image Builder (p. 224)
- Prerequisites for sharing components, images, and recipes (p. 224)
- Related services (p. 225)
- Sharing across Regions (p. 225)
- Sharing a component, image, or recipe (p. 225)
- Unsharing a component, image, or recipe (p. 227)
- Identifying a shared component, image, or recipe (p. 228)
- Shared component, image, and recipe permissions (p. 228)
- Billing and metering (p. 228)
- Resource limits (p. 229)

Working with shared components, images, and recipes in EC2 Image Builder

Component, image, and recipe sharing enables resource owners to share software configurations with other AWS accounts or within an AWS organization. You can manage resource sharing centrally, and define a set of accounts with which the configuration can be shared.

In this model, the AWS account that owns the component, image, or recipe (owners) shares it with other AWS accounts (consumers). Consumers can associate a shared component with their image pipelines to automatically consume updates to the shared component, image, or recipe.

A component, image, or recipe owner can share these resources with:

- Specific AWS accounts inside or outside of its organization in AWS Organizations.
- An organizational unit (OU) inside of its organization in AWS Organizations.
- Its entire organization in AWS Organizations.
- AWS Organizations or OUs outside of its organization.

Prerequisites for sharing components, images, and recipes

To share an Image Builder component, image, or recipe:

- You must own the component, image, or recipe in your AWS account. You cannot share resources that have been shared with you.
- The AWS Key Management Service (AWS KMS) key associated with encrypted resources must be explicitly shared with the target accounts, 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 (p. 205). For more information about distribution settings for your container image in Amazon ECR, see Create and update distribution settings for container images (p. 213).

For more information about sharing your AMI with AWS Organizations and OUs, see Share an AMI with organizations or OUs.

Related services

AWS Resource Access Manager

Component, image, and recipe sharing integrate with AWS Resource Access Manager (AWS RAM). AWS RAM is a service that enables you to share your AWS resources with any AWS account or through AWS Organizations. With AWS RAM, you share resources that you own by creating a resource share. A resource share specifies the resources to share and the consumers with whom to share them. Consumers can be individual AWS accounts, organizational units, or an entire organization in AWS Organizations.

For more information about AWS RAM, see the AWS RAM User Guide.

Sharing across Regions

Shared components, images, and recipes can be shared only in a specified AWS Region. When you share these resources, they will not replicate across Regions.

Sharing a component, image, or recipe

To share an Image Builder component, image, or recipe, you must add it to a resource share. A resource share is an AWS RAM resource that lets you share your resources across AWS accounts. A resource share specifies the resources to share and the consumers with whom they are shared. To add the component, image, or recipe to a new resource share, you must first create the resource share using the AWS RAM console.

If you are part of an organization in AWS Organizations and sharing within your organization is enabled, consumers in your organization are automatically granted access to the shared component, image, or recipe. Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared resource after accepting the invitation.

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:
Sharing a component, image, or recipe

- **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 a RAM 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.

   ```
   ```

   **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.

   ```
   ```
Unsharing a shared component, image, or recipe

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.

```
```

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.

```
```

**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.

Unsharing a shared component, image, or recipe

To unshare a shared component, image, or recipe that you own, you must remove it from the resource share. You can do this using the AWS Resource Access Manager console or the AWS CLI.

**Note**
To unshare a component, image, or recipe, the consumer cannot have any dependencies on them. The consumer must remove any dependencies on the shared resources before the owner can unshare them.
To unshare a shared component, image, or recipe that you own using the AWS Resource Access Manager console

See Updating a Resource Share in the AWS 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.

Identifying a shared component, image, or recipe

Owners and consumers can identify shared components, images, and image recipes using Image Builder commands in the AWS CLI.

**Identify a shared component**

Run the list-components command to get a list of the components that you own and the components that are shared with you. The get-component command shows the AWS account ID of the component owner.

**Identify a shared image**

Run the list-images command to get a list of the images that you own and images that are shared with you. The get-image command shows the AWS account ID of the image owner.

**Identify a shared container image**

Run the list-images command to get a list of the images that you own and images that are shared with you. The get-image command shows the AWS account ID of the image owner.

**Identify a shared image recipe**

Run the list-image-recipes command to get a list of the image recipes that you own and image recipes that are shared with you. The get-image-recipe command shows the AWS account ID of the image recipe owner.

**Identify a shared container recipe**

Run the list-container-recipes command to get a list of the container recipes that you own and container recipes that are shared with you. The get-container-recipe command shows the AWS account ID of the container recipe owner.

**Shared component, image, and recipe permissions**

**Permissions for owners**

Owners cannot delete a shared component, image, or image recipe until it is no longer shared. An owner cannot unshare these resources until none of the consumers depend on them.

**Permissions for consumers**

Consumers can read a component, image, or image recipe, but cannot modify them in any way. They cannot view or modify these resources if they are owned by other consumers or the owner of the resource. Consumers can use shared components and images in image recipes to create custom images. Consumers can use shared image recipes to create their own custom images.

**Billing and metering**

There is no charge to use EC2 Image Builder.
Resource limits

Shared components, images, and image recipes count toward the corresponding resource limits of the owner only. The resource limits of the consumers are not affected by the resources that have been shared with them.

Tag EC2 Image Builder resources

Tagging your resources can be useful for filtering and tracking resource costs, or other categories. You can also control access based on tags. For more information about tag-based authorization, see Authorization based on Image Builder tags (p. 275)

Image Builder supports the following dynamic tags:

- {{imagebuilder:buildDate}}
  
  Resolves to the build date/time at build time.

- {{imagebuilder:buildVersion}}
  
  Resolves to a build version, which is a number that is located at the end of an Image Builder ARN. For example, "arn:aws:imagebuilder:us-west-2:123456789012:component/myexample-component/2019.12.02/1" shows the build version as 1.

Contents

- Tag a resource (AWS CLI) (p. 229)
- Untag a resource (AWS CLI) (p. 229)
- List all of the tags for a specific resource (AWS CLI) (p. 230)

Tag a resource (AWS CLI)

The following example shows how to use an imagebuilder CLI command 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:

```json
{
    "tags": {
        "KeyName": "KeyValue"
    }
}
```

Run the following command, which references the preceding tag-resource.json file.

```
aws imagebuilder tag-resource --cli-input-json file://tag-resource.json
```

Untag a resource (AWS CLI)

The following example shows how to use an imagebuilder CLI command to remove a tag from a resource. You must provide the resourceArn and the keys to remove the tag.
The example `untag-resource.json` contents are as follows:

```json
{
  "tagKeys": [
    "KeyName"
  ]
}
```

Run the following command, which references the preceding `untag-resource.json` file.

```bash
aws imagebuilder untag-resource --cli-input-json file://untag-resource.json
```

**List all of the tags for a specific resource (AWS CLI)**

The following example shows how to use an `imagebuilder` CLI command to list all the tags for a specific resource.

```bash
```

**Delete EC2 Image Builder resources**

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

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 using the AWS Management Console**

To delete an image pipeline and its resources, follow these steps:

**Delete the pipeline**

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

**Delete the recipe**

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

**Delete infrastructure configuration**

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

**Delete distribution settings**

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

**Delete an image**

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

**Delete an image pipeline using the AWS CLI**

The following examples show how to delete Image Builder resources using the AWS CLI. As mentioned previously, resources must be deleted in the following order to avoid dependency errors:

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

**Delete image pipeline (AWS CLI)**
The following example shows how to delete an image pipeline by specifying its ARN.

```
```

**Delete image recipe (AWS CLI)**

The following example shows how to delete an image recipe by specifying its ARN.

```
```

**Delete an infrastructure configuration**

The following example shows how to delete an infrastructure configuration resource by specifying its ARN.

```
```

**Delete distribution settings**

The following example shows how to delete a distribution settings resource by specifying its ARN.

```
```

**Delete an image**

The following example shows how to delete an image build version by specifying its ARN.

```
```

**Delete a component**

The following example shows how to use an `imagebuilder` CLI command to delete a component build version by specifying its ARN.

```
```

**Important**

Make sure there are no recipes that reference the component build version in any way before you delete it. Failing to do so could cause pipeline failures.
Manage EC2 Image Builder pipelines using the console

Image Builder image pipelines provide an automation framework for creating and maintaining custom AMIs and container images. Pipelines deliver the following functionality:

- Assemble the base image, components for building and testing, infrastructure configuration, and distribution settings.
- Facilitate scheduling for automated maintenance processes using the Schedule builder in the console wizard, or entering cron expressions for recurring updates to your images.
- Enable change detection for the base image and components, to automatically skip scheduled builds when there are no changes.
- Enable rule-based automation through Amazon EventBridge.

**Note**
For more information about using the EventBridge API to view or change rules, see the Amazon EventBridge API Reference. For more information about using EventBridge events commands in the AWS CLI to view or change rules, see events in the AWS CLI Command Reference.

Contents
- List and view pipeline details (p. 233)
- Create and update AMI image pipelines (p. 234)
- Create and update container image pipelines (p. 239)
- Use cron expressions in EC2 Image Builder (p. 244)
- Use EventBridge rules with Image Builder pipelines (p. 248)

List and view pipeline details

This section describes the various ways that you can find information and view details for your EC2 Image Builder image pipelines.

Pipeline details
- List image pipelines (AWS CLI) (p. 233)
- Get image pipeline details (AWS CLI) (p. 234)

List image pipelines (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.

```bash
aws imagebuilder list-image-pipelines
```
Get image pipeline details (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.

```
```

Create and update AMI image pipelines

You can set up, configure, and manage AMI image pipelines using the Image Builder console, through the Image Builder API, or with `imagebuilder` commands in the AWS CLI. The Create image pipeline console wizard provides starting artifacts, and guides you through steps to:

- Select a base image from quick-start managed images, or existing images in your account or that were shared with you from another account.
- Add and remove software
- Customize settings and scripts
- Run selected tests
- Distribute images to AWS Regions

For more information and a step-by-step tutorial about using the Create image pipeline console wizard, see [Create an image pipeline using the EC2 Image Builder console wizard](p. 13).

Contents
- Create an AMI image pipeline (AWS CLI) (p. 234)
- Update AMI image pipelines (console) (p. 235)
- Update AMI image pipelines (AWS CLI) (p. 237)
- Run your AMI image pipeline (p. 238)

Create an AMI image pipeline (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 EC2 Image Builder](p. 244).
- `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`:

   ```json
   {
     "name": "MyWindows2019Pipeline",
     "description": "Builds Windows 2019 Images",
     "enhancedImageMetadataEnabled": true,
     "imageTestsConfiguration": {
       "imageTestsEnabled": true,
       "timeoutMinutes": 60
     },
     "schedule": {
       "scheduleExpression": "cron(0 0 * * SUN *)",
       "pipelineExecutionStartCondition": "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
     },
     "status": "ENABLED"
   }
   ```

   **Note**

   - You must include the `file://` notation at the beginning of the JSON file path.
   - The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (`\`) to refer to the directory path, and Linux uses the forward slash (`/`).

2. **Run the following command, using the file you created as input.**

   ```bash
   aws imagebuilder create-image-pipeline --cli-input-json file://create-image-pipeline.json
   ```

**Update AMI image pipelines (console)**

After you have created an Image Builder image pipeline for your AMI image, you can make changes to the infrastructure configuration and distribution settings from the Image Builder console.

To update an image pipeline with a new image recipe, you must use the AWS CLI. For more information, see [Update AMI image pipelines (AWS CLI) (p. 237)] in this guide.

**Choose an existing Image Builder pipeline**

2. To see a list of the image pipelines created under your account, choose Image pipelines from the navigation pane.

  Note
  The list of image pipelines includes an indicator for the type of output image that is created by the pipeline – AMI or Docker.

3. To view details or edit a pipeline, choose the Pipeline name link. This opens the detail view for the pipeline.

  Note
  You can also select the check box next to the Pipeline name, then choose View detail.

Pipeline details

The pipeline details page includes the following sections:

Summary

The section at the top of the page summarizes key details for the pipeline that are visible with any of the detail tabs open. The details displayed in this section are editable only on their respective detail tabs.

Detail tabs

- **Output images** – Shows output images that the pipeline has produced.
- **Image recipe** – Shows recipe details. After you create a recipe, you cannot edit it. You must create a new version of the recipe from the Image recipes page in the Image Builder console, or by using Image Builder commands in the AWS CLI. For more information, see Manage recipes (p. 162).
- **Infrastructure configuration** – Shows editable information for configuring your build pipeline infrastructure.
- **Distribution settings** – Shows editable information for AMI distribution.
- **EventBridge rules** – For the selected Event Bus, shows EventBridge rules that target the current pipeline. Includes Create event bus and Create rule actions that link to the EventBridge console. For more information about this tab, see Use EventBridge rules (p. 248).

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 EC2 Image Builder](p. 244).

Update AMI image pipelines (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
Run pipeline

- 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`:

```json
{
   "imagePipelineArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
   "imageTestsConfiguration": {
      "imageTestsEnabled": true,
      "timeoutMinutes": 120
   },
   "schedule": {
      "scheduleExpression": "cron(0 0 * * MON *)",
      "pipelineExecutionStartCondition": "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
   },
   "status": "DISABLED"
}
```

**Note**
- You must include the file:// notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. **Run the following command, using the file you created as input.**

```bash
aws imagebuilder update-image-pipeline --cli-input-json file://update-image-pipeline.json
```

**Run your AMI image pipeline**

If you chose the manual schedule option for your pipeline, it will only run when you manually kick off the build. If you chose one of the automatic scheduling options, you can also run it manually, in between regularly scheduled runs. For example, if you have a pipeline that normally runs once a month, but you need to incorporate an update to one of your components two weeks after the prior run, you can choose to run your pipeline manually.
Create and update container image pipelines

You can set up, configure, and manage container image pipelines using the Image Builder console, through the Image Builder API, or with imagebuilder commands in the AWS CLI. The Create image pipeline console wizard provides starting artifacts, and guides you through steps to:

- Select a base image from quick-start managed images, Amazon ECR, or Docker Hub repositories
- Add and remove software
- Customize settings and scripts
- Run selected tests
- Create a Dockerfile using pre-configured build-time variables.
- Distribute images to AWS Regions

For more information and a step-by-step tutorial about using the Create image pipeline console wizard, see Create a container image pipeline using the EC2 Image Builder console wizard (p. 18).

Contents
- Create a container image pipeline (AWS CLI) (p. 239)
- Update a container image pipeline (console) (p. 241)
- Update container image pipelines (AWS CLI) (p. 243)
- Run your container image pipeline (p. 244)

Create a container image pipeline (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 EC2 Image Builder (p. 244).

• 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:

```json
{
   "name": "MyWindows2019Pipeline",
   "description": "Builds Windows 2019 Images",
   "enhancedImageMetadataEnabled": true,
   "imageTestsConfiguration": {
      "imageTestsEnabled": true,
      "timeoutMinutes": 60
   },
   "schedule": {
      "scheduleExpression": "cron(0 0 * * SUN *)",
      "pipelineExecutionStartCondition": "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
   },
   "status": "ENABLED"
}
```

**Note**

• You must include the file:// notation at the beginning of the JSON file path.
The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```bash
aws imagebuilder create-image-pipeline --cli-input-json file://create-image-pipeline.json
```

**Update a container image pipeline (console)**

After you have created an Image Builder container image pipeline for your Docker image, you can make changes to the infrastructure configuration and distribution settings from the Image Builder console.

To update a container image pipeline with a new container recipe, you must use the AWS CLI. For more information, see Update container image pipelines (AWS CLI) (p. 243) in this guide.

**Choose an existing Image Builder Docker image pipeline**

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 (p. 172).
- **Infrastructure configuration** – Shows editable information for configuring your build pipeline infrastructure.
- **Distribution settings** – Shows editable information for Docker image distribution.
- **EventBridge rules** – For the selected **Event Bus**, shows EventBridge rules that target the current pipeline. Includes **Create event bus** and **Create rule** actions that link to the EventBridge console. For more information about this tab, see Use EventBridge rules (p. 248).
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 settings.
- **Region settings** for the Regions where you distribute your image. Region 1 defaults to the Region where you created the pipeline. You can add Regions for distribution with the **Add Region** button, and you can remove all Regions except Region 1.

**Region settings** include:

- **Target Region**
- The **Service** defaults to "ECR", and is not editable.
- **Repository name** – the name of your target repository *(not including the Amazon ECR location)*. For example, the repository name with the location would look like the following pattern:

  `<account-id>.dkr.ecr.<region>.amazonaws.com/<repository-name>`

  **Note**
  
  If you change the **Repository name**, only the images created after the name change will be added under the new name. Any prior images that your pipeline created remain in their original repository.

To edit your distribution settings from the pipeline details page, follow these steps:

1. Choose the **Distribution settings** tab.
2. Choose **Edit** from the upper right corner of the **Distribution details** panel.
3. When you are ready to save updates you've made to your distribution settings, choose **Save changes**.

Edit the build schedule for your pipeline

The **Edit pipeline** page includes the following details that you can edit after creating the pipeline:

- The **Description** for your pipeline.
**Enhanced metadata collection.** This is turned on by default. To turn it off, clear the **Enable enhanced metadata collection** check box.

**The Build schedule** for your pipeline. You can change your **Schedule options** and all of the settings in this section.

To edit your pipeline from the pipeline details page, follow these steps:

1. In the upper right corner of the pipeline details page, choose **Actions**, and then **Edit pipeline**.
2. When you are ready to save your updates, choose **Save changes**.

**Note**
For more information about scheduling your build using cron expressions, see **Use cron expressions in EC2 Image Builder** (p. 244).

### Update container image pipelines (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**
UpdatImagePipeline 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`:

```json
{
    "imagePipelineArn": "arn:aws:imagebuilder:us-west-2:123456789012:image-pipeline/my-example-pipeline",
    "imageTestsConfiguration": {
        "imageTestsEnabled": true,
        "timeoutMinutes": 120
    }
}
```
"schedule": {
  "scheduleExpression": "cron(0 0 * * MON *)",
  "pipelineExecutionStartCondition": "EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE"
},
"status": "DISABLED"

Note

- You must include the file:// notation at the beginning of the JSON file path.
- The path for the JSON file should follow the appropriate convention for the base operating system where you are running the command. For example, Windows uses the backslash (\) to refer to the directory path, and Linux uses the forward slash (/).

2. Run the following command, using the file you created as input.

```
aws imagebuilder update-image-pipeline --cli-input-json file://update-image-pipeline.json
```

Run your container image pipeline

If you chose the manual schedule option for your pipeline, it will only run when you manually kick off the build. If you chose one of the automatic scheduling options, you can also run it manually, in between regularly scheduled runs. For example, if you have a pipeline that normally runs once a month, but you need to incorporate an update to one of your components two weeks after the prior run, you can choose to run your pipeline manually.

Console

To run your pipeline from the pipeline details page in the Image Builder console, choose Run pipeline from the Actions menu at the top of the page. A status message appears at the top of the page to notify you that your pipeline has started, or if there is an error.

1. In the upper left corner of the pipeline details page, choose Run pipeline, from the Actions menu.
2. You can see the current status of your pipeline on the Output images tab, in the Status column.

AWS CLI

The following example shows how to use the start-image-pipeline-execution command in the AWS CLI to manually start an image pipeline. Running this command results in the pipeline creating a new image on demand.

```
aws imagebuilder start-image-pipeline-execution --image-pipeline-arn
```

To see what resources are created when the build pipeline runs, see Resources created (p. 9).

Use cron expressions in EC2 Image Builder

Use cron expressions for EC2 Image Builder to set up a time window to refresh your image with updates that apply to your pipeline's base image and components. The time window for your pipeline refresh
starts with the time you set in the cron expression. You can set the time in your cron expression down to the minute. Your pipeline build can run on or after the start time.

It can sometimes take a few seconds, or up to a minute for your build to start running.

**Note**
Cron expressions use Universal Coordinated Time (UTC). For more information about UTC time, and to find the offset for your time zone, see [Time Zone Abbreviations – Worldwide List](#).

### Supported values for cron expressions in Image Builder

EC2 Image Builder uses a cron format that consists of six required fields. Each one is separated from the others by a space in between, with no leading or trailing spaces:

<Minute> <Hour> <Day> <Month> <Day of the week> <Year>

The following table shows supported values for required cron entries.

#### Supported values for cron expressions

<table>
<thead>
<tr>
<th>Field</th>
<th>Values</th>
<th>Wildcards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minute</td>
<td>0-59</td>
<td>, - */</td>
</tr>
<tr>
<td>Hour</td>
<td>0-23</td>
<td>, - */</td>
</tr>
<tr>
<td>Day</td>
<td>1-31</td>
<td>, - * ? / L W</td>
</tr>
<tr>
<td>Month</td>
<td>1-12 or jan-dec</td>
<td>, - * /</td>
</tr>
<tr>
<td>Day of the week</td>
<td>1-7 or sun-sat</td>
<td>, - * ? L #</td>
</tr>
<tr>
<td>Year</td>
<td>1970-2199</td>
<td>, - * /</td>
</tr>
</tbody>
</table>

#### 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

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>The , (comma) wildcard includes additional values. In the Month field, jan, feb, mar includes January, February, and March.</td>
</tr>
<tr>
<td>-</td>
<td>The - (dash) wildcard specifies ranges. In the day of the month field, 1-15 includes days 1 through 15 of the specified month.</td>
</tr>
<tr>
<td>*</td>
<td>The * (asterisk) wildcard includes all valid values for the field.</td>
</tr>
<tr>
<td>?</td>
<td>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</td>
</tr>
</tbody>
</table>
Examples of cron expressions in EC2 Image Builder

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>specified or includes all possible values (*), the other must be a ?. You cannot specify both. For example, if you enter a 7 in the Day field (run the build on the seventh day of the month), the Day-of-week position must contain a ?.</td>
</tr>
<tr>
<td>/</td>
<td>The / (forward slash) wildcard specifies increments. For example, if you want your build to run every other day, enter */2 in the day field.</td>
</tr>
<tr>
<td>L</td>
<td>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.</td>
</tr>
<tr>
<td>W</td>
<td>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.</td>
</tr>
<tr>
<td>#</td>
<td>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.</td>
</tr>
</tbody>
</table>

Restrictions

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or * in one of these fields, you must use a ? in the other.
- Cron expressions that lead to rates faster than one minute are not supported.

Examples of cron expressions in EC2 Image Builder

Cron expressions are entered differently for the Image Builder console, than they are for the API or CLI. To see examples, choose the tab that applies to you.

Image Builder console

The following examples show cron expressions that you can enter into the console for your build schedule. UTC time is specified using a 24-hour clock.

**Run daily at 10:00 AM (UTC)**

```
0 10 * * ? *
```

**Run daily at 12:15 PM (UTC)**

```
15 12 * * ? *
```
Rate expressions in EC2 Image Builder

A rate expression starts when you create the scheduled event rule, and then runs on its defined schedule.
Rate expressions have two required fields. Fields are separated by white space.

**Syntax**

\[ \text{rate}(\text{value \hspace{1mm} unit}) \]

- **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 \hspace{1mm} hours)` and `rate(5 \hspace{1mm} hours)` are not valid, but `rate(1 \hspace{1mm} hour)` and `rate(5 \hspace{1mm} hours)` are valid.

Use EventBridge rules with Image Builder pipelines

Events from a wide range of AWS and partner services are streamed to Amazon EventBridge event buses in near real-time. You can also generate custom events, and send events from your own applications to EventBridge. The event buses use rules to determine where to route event data.

Image Builder pipelines are available as EventBridge rule targets, which means that you can run an Image Builder pipeline based on rules that you create to respond to events on the bus, or on a schedule.

**Note**

Event buses are specific to a Region. The rule and the target must be in the same Region.

**Contents**

- EventBridge terms (p. 248)
- View EventBridge rules for your Image Builder pipeline (p. 249)
- Use EventBridge rules to schedule a pipeline build (p. 249)

**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.

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 EC2 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.

- **AWSTOE 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 CloudTrail** – Monitor Image Builder events that are sent to CloudTrail. For more information about CloudTrail, see [What Is AWS CloudTrail?](#) in the AWS CloudTrail User Guide.

- **Amazon CloudWatch Logs** – Monitor, store, and access your Image Builder log files. Optionally, you can save your logs to an S3 bucket. For more information about CloudWatch Logs, see [What is Amazon CloudWatch Logs?](#) in the Amazon CloudWatch Logs User Guide.

- **Amazon EventBridge** – Connect to a stream of real-time event data from Image Builder activities in your account. For more information about EventBridge, see [What Is Amazon EventBridge?](#) in the Amazon EventBridge User Guide.

- **Amazon Inspector** – Discover vulnerabilities in your software and network settings with automatic scans for the EC2 instances that Image Builder launches when you build and test a new image. Image Builder saves findings with the build version information for your output image resource so that you can investigate and remediate after the build completes. 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 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 the AWS Marketplace Buyer Guide.

• **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**
- AWS CloudTrail integration in Image Builder (p. 252)
- Amazon CloudWatch Logs integration in Image Builder (p. 252)
- Amazon EventBridge integration in Image Builder (p. 253)
- Amazon Inspector integration in Image Builder (p. 254)
- AWS Marketplace integration in Image Builder (p. 254)
- Amazon SNS integration in Image Builder (p. 258)
- Compliance products for your Image Builder images (p. 263)

### AWS CloudTrail integration in Image Builder

This service supports AWS CloudTrail. CloudTrail is a service that records AWS calls for your AWS account and delivers log files to an Amazon S3 bucket. By using information collected by CloudTrail, you can determine what requests were successfully made to AWS services, who made the request, when it was made, and so on. For more information about CloudTrail integration with Image Builder, see Logging EC2 Image Builder API calls using AWS CloudTrail (p. 264).

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 integration in Image Builder

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": [
  {
```

252
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 integrates with Image Builder image pipelines, see Use EventBridge rules with Image Builder pipelines (p. 248).
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 for your build instances 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 for your image 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 (p. 193).

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 (p. 194).

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?

**Note**

Data product providers must meet the AWS Data Exchange eligibility requirements. For more information, see Providing Data Products on AWS Data Exchange in the AWS Data Exchange User Guide.

**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.
- See a list of your current AWS Marketplace product subscriptions.
- Use an AWS Marketplace image product as the base image for an Image Builder recipe.

For products that include associated AWS Task Orchestrator and Executor (AWSTOE) components, you can filter on the product owner in the console and in the API, SDK, and CLI. For more information, see List AWSTOE components (p. 154).

**Find AWS Marketplace image products from the Image Builder console**

Image Builder integrates with AWS Marketplace to show your image product subscriptions directly from the AWS Marketplace section in the Image Builder console. You can also search for AWS Marketplace image products from the Image Products page without leaving the Image Builder console.

To find an AWS Marketplace image product from the Image Builder console, follow these steps:

2. From the navigation pane, choose Image products in the AWS Marketplace section.
3. The Image products page shows you a summary of the image products that you’ve subscribed to in the Subscriptions tab, or you can search for image products in the AWS Marketplace tab.

   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.

   **AWS Marketplace**

   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](https://docs.aws.amazon.com/marketplace/latest/userguide/buying-products.html) 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.

**Subscriptions**
This tab shows you a list of the image products that you've subscribed to in AWS Marketplace. 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 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 that you subscribed to.
- If there is an Associated component included with your subscribed product, Image Builder displays a link to the AWSTOE 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 product as the base image for 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.

### Use an AWS Marketplace image product in Image Builder recipes

In the Image Builder console, there are two ways that you can create a new image recipe based on one of your subscribed image products.

1. You can start from the **Image products** page as follows:
   2. From the navigation pane, choose **Image products** in the **AWS Marketplace** section.
   3. Open the **Subscriptions** tab.
   4. Select the subscribed image product to use as the base image in your recipe.
   5. Choose **Create new recipe**. This opens the Create recipe page with the **AWS Marketplace images** option and your subscribed image product pre-selected.
   6. Configure remaining settings for your recipe as you normally would. For more information about image recipes, see [Create a new version of an image recipe (p. 165)](#).

2. You can also open the **Create recipe** page and select an AWS Marketplace image product to use as your base image.

   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 [Find AWS Marketplace image products from the Image Builder console (p. 255)](#).

   6. Enter remaining details as usual, and choose **Create recipe**.

**Note**
If your image product subscription includes an AWSTOE build component, you can select it from the **Build components** list. Select **Third party managed** from the component owner type list to see it. If your product subscription includes an AWSTOE test component, follow the same procedure for the **Test components** list.
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 (p. 259). 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 Using service-linked roles for EC2 Image Builder (p. 294).

```json
{
   "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": 12379405928538274899124227,
  "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",
"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": {
  "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": "12345-example-linux-testbucket-uswest1"
    }
  },
  "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"
The following example shows the JSON payload for a typical message that Image Builder publishes for a pipeline build failure for a Linux 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"
  }
}
```
SNS message format

```
},
"accountId": "123456789012"
},
"infrastructureConfiguration": {
"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",
},
"terminateInstanceOnFailure": true,
"dateCreated": "Jul 5, 2022 7:31:53 PM",
"accountId": "123456789012"
},
"imageTestsConfigurationDocument": {
"imageTestsEnabled": true,
"timeoutMinutes": 720
},
"distributionConfiguration": {
"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": "1feecd23a-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,
```

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 AWSTOE 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 Guides in the CIS website support portal.

  **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 (p. 124).

- **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 EC2 Image Builder (p. 124).
Monitor events and logs in EC2 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

- Logging EC2 Image Builder API calls using AWS CloudTrail (p. 264)

Logging EC2 Image Builder API calls using AWS CloudTrail

EC2 Image Builder is integrated with AWS CloudTrail, a service that provides a record of actions all API calls for taken by a user, role, or an AWS service through the Image Builder API. CloudTrail captures Image Builder as events. The calls captured include calls from the Image Builder console and code calls to the Image Builder API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for Image Builder. If you don’t configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to Image Builder, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Image Builder information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Image Builder, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Image Builder, create a trail. A trail enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail.
- CloudTrail supported services and integrations.
- Configuring Amazon SNS notifications for CloudTrail.
- Receiving CloudTrail log files from multiple regions.
- Receiving CloudTrail log files from multiple accounts.

CloudTrail logs all Image Builder actions that are documented in the EC2 Image Builder API Reference. For example, calls to the CreateImagePipeline, UpdateInfrastructureConfiguration, and StartImagePipelineExecution actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information about determining who requested an event, see the CloudTrail userIdentity element.
Security in EC2 Image Builder

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to EC2 Image Builder, see AWS services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Image Builder. The following topics show you how to configure Image Builder to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Image Builder resources.

**Topics**
- Data protection in EC2 Image Builder (p. 266)
- Identity and Access Management for EC2 Image Builder (p. 267)
- Compliance validation for EC2 Image Builder (p. 297)
- Resilience in EC2 Image Builder (p. 297)
- Infrastructure security in Image Builder (p. 297)
- Patch Management in EC2 Image Builder (p. 298)
- Security best practices for EC2 Image Builder (p. 299)

Data protection in EC2 Image Builder

The AWS shared responsibility model applies to data protection in EC2 Image Builder. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual 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.
• 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 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers’ email addresses, into tags or free-form 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 EC2 Image Builder

Image Builder encrypts data in transit and at rest by default. Custom components defined in the service can be added to your image pipelines and shared with other customer accounts. You are not required to share your components to build images.

Custom components are encrypted with your default KMS key or a KMS key that’s owned by 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.

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.

Inter-network Traffic Privacy in EC2 Image Builder

Connections are secured between Image Builder and on-premises locations, between AZs within an AWS Region, and between AWS Regions through HTTPS. There are no direct connections between accounts.

Identity and Access Management for EC2 Image Builder

Topics
• Audience (p. 268)
• Authenticating with identities (p. 268)
• How EC2 Image Builder works with IAM (p. 268)
• EC2 Image Builder identity-based policies (p. 275)
• EC2 Image Builder resource-based policies (p. 277)
• Using managed policies for EC2 Image Builder (p. 278)
• Using service-linked roles for EC2 Image Builder (p. 294)
• Troubleshooting EC2 Image Builder identity and access (p. 295)
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 Troubleshooting EC2 Image Builder identity and access (p. 295).

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 EC2 Image Builder works with IAM (p. 268).

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 (p. 272).

Authenticating with identities

For detailed information about how to provide authentication for people and processes in your AWS account, see Identities in the IAM User Guide.

How EC2 Image Builder works with IAM

Before you use IAM to manage access to Image Builder, 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

<table>
<thead>
<tr>
<th>Supports identity-based policies</th>
<th>Yes</th>
</tr>
</thead>
</table>

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 Creating IAM 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 (p. 272).
Resource-based policies within Image Builder

| Supports resource-based policies | No |

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 How IAM roles differ from resource-based policies 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 (p. 272).
Policy resources for Image Builder

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports policy resources</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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.

Policy condition keys for Image Builder

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports service-specific policy condition keys</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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 (p. 272).
## 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 What is ABAC? 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 Switching to a 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 principal permissions | Yes |
When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for EC2 Image Builder in the Service Authorization Reference.

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 Creating 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 | No |

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 Using service-linked roles for EC2 Image Builder (p. 294).

Image Builder identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources, and also the conditions under which actions are allowed or denied. Image Builder supports specific actions, resources, and condition keys. For information about all of the elements that you use in a JSON policy, see Actions, Resources, and Condition Keys for Amazon EC2 Image Builder in the IAM User Guide.

Actions

Policy actions in Image Builder use the following prefix before the action: imagebuilder:. Policy statements must include either an Action or NotAction element. Image Builder defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
  "imagebuilder:action1",
  "imagebuilder:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word List, include the following action:

```
"Action": "imagebuilder:List*"
```
To see a list of Image Builder actions, see Actions, Resources, and Condition Keys for AWS services in the IAM User Guide.

Managing access using policies

For detailed information about how to manage access in AWS by creating policies and attaching them to IAM identities or AWS resources, see Policies and Permissions in the IAM User Guide.

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

- EC2InstanceProfileForImageBuilder
- EC2InstanceProfileForImageBuilderECRContainerBuilds
- AmazonSSMManagedInstanceCore

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as resource-level permissions.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*
```

The Image Builder instance resource has the following Amazon Resource Name (ARN).

```
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the i-1234567890abcdef0 instance in your statement, use the following ARN.

```
"Resource": "arn:aws:imagebuilder:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:imagebuilder:us-east-1:123456789012:instance/*"
```

Some Image Builder actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*
```

Many EC2 Image Builder API actions involve multiple resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
```
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 CreateInfrastructureConfiguration 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 CreateInfrastructureConfiguration and UpdateInfrastructureConfiguration 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 CreateInfrastructureConfiguration 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 CreateInfrastructureConfiguration and UpdateInfrastructureConfiguration APIs.

Examples

To view examples of Image Builder identity-based policies, see EC2 Image Builder identity-based policies (p. 275).

Image Builder resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on the Image Builder resource and under what conditions. Image Builder supports resource-
based permissions policies for components, images, and image recipes. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You can also use a resource-based policy to allow an AWS service to access your components, images, and image recipes.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, you must also grant the principal entity permission to access the resource. Grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.

For information about how to attach a resource-based policy to a component, image, or image recipe, see Share EC2 Image Builder resources (p. 223).

Note
When you update a resource policy using Image Builder, the update will appear in the RAM console.

Authorization based on Image Builder tags

You can attach tags to Image Builder resources or pass tags in a request to Image Builder. To control access based on tags, you provide tag information in the condition element of a policy using the imagebuilder:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. For more information about tagging Image Builder resources, see Tag a resource (AWS CLI) (p. 229).

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 Using service-linked roles for EC2 Image Builder (p. 294).

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.

EC2 Image Builder identity-based policies

Topics
- Identity-based policy best practices (p. 276)
- Using the Image Builder console (p. 276)
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 IAM Access Analyzer policy validation 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 Configuring MFA-protected API access 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** (p. 280)
- **AWSImageBuilderFullAccess policy** (p. 278)

For more information about Image Builder managed policies, see Using managed policies for EC2 Image Builder (p. 278).

**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": "sns topic arn"
        },
        {
            "Effect": "Allow",
            "Action": ["iam:GetInstanceProfile"],
            "Resource": "instance profile role arn"
        },
        {
            "Effect": "Allow",
            "Action": ["iam:PassRole"],
            "Resource": "instance profile role arn",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ec2.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": ["s3:ListBucket"],
            "Resource": "bucket arn"
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls to only the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

EC2 Image Builder resource-based policies

For information about how to create a component, see Manage components with Image Builder (p. 148).

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",
            "Principal": "*",
            "Action": "imagebuilder.GetComponent:*",
            "Resource": "arn:aws:imagebuilder:::examplecomponent/*",
            "Condition": {
                "IpAddress": {"aws:SourceIp": "54.240.143.0/24"},
                "NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}
            }
        }
    ]
}
```

Using managed policies for EC2 Image Builder

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".

### Policy example

The following is an example of the AWSImageBuilderFullAccess policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "imagebuilder:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["sns:ListTopics"],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["sns:Publish"],
      "Resource": "arn:aws:sns:*:*:imagebuilder"
    },
    {
      "Effect": "Allow",
      "Action": ["license-manager:ListLicenseConfigurations", "license-manager:ListLicenseSpecificationsForResource"],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetRole"],
      "Resource": "arn:aws:iam::*:role/aws-service-role/imagebuilder.amazonaws.com/AWSServiceRoleForImageBuilder"
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetInstanceProfile"],
      "Resource": "arn:aws:iam::*:instance-profile/*imagebuilder*"
    },
    {
      "Effect": "Allow",
      "Action": ["iam:ListInstanceProfiles", "iam:ListRoles"],
      "Resource": "*"
  ]
}
```
AWSImageBuilderReadOnlyAccess policy

The AWSImageBuilderReadOnlyAccess policy provides read-only access to all Image Builder resources. Permissions are granted to verify that the Image Builder service-linked role exists via the `iam:GetRole` API action.
Permissions details

This policy includes the following permissions:

- **Image Builder** – Access is granted for read-only access to Image Builder resources.
- **IAM** – Access is granted to verify the existence of the Image Builder service-linked role via the `iam:GetRole` API action.

Policy example

The following is an example of the AWSImageBuilderReadOnlyAccess policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:Get*",
                "imagebuilder:List*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/imagebuilder.amazonaws.com/AWSServiceRoleForImageBuilder"
        }
    ]
}
```

AWSServiceRoleForImageBuilder policy

The **AWSServiceRoleForImageBuilder** policy allows Image Builder to call AWS services on your behalf.

Permissions details

This policy is attached to the Image Builder service-linked role when the role is created through Systems Manager. To review specific permissions that are granted, see the policy example (p. 282) in this section. For more information about the Image Builder service-linked role, see Using service-linked roles for EC2 Image Builder (p. 294).

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 images and launch EC2 instances in your account, using related snapshots, volumes, network interfaces, security groups, 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, 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 the account.

- **Systems Manager** – Access is granted for Image Builder to list Systems Manager commands and their invocations, instance information, inventory entries and automation execution statuses. Image Builder can also send automation signals, and stop automation 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.

- **AWS STS** – Access is granted for Image Builder to assume roles named `EC2ImageBuilderDistributionCrossAccountRole` from your account to any account where the Trust policy on the role permits it. This is used for cross-account image distribution.

### Policy example

The following is an example of the AWSServiceRoleForImageBuilder policy.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["ec2:RunInstances"],
         "Resource": ["arn:aws:ec2::image/*",
                       "arn:aws:ec2::snapshot/*",
                       "arn:aws:ec2::subnet/*",
                       "arn:aws:ec2:*::image/*",
                       "arn:aws:ec2:*::snapshot/*",
                       "arn:aws:ec2:*::subnet/*"]
      }
   ]
}
```
Managed policies

```
{
  "Effect": "Allow",
  "Action": ["ec2:RunInstances"],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/CreatedBy": ["EC2 Image Builder", "EC2 Fast Launch"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": ["ec2.amazonaws.com", "ec2.amazonaws.com.cn", "vmie.amazonaws.com"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": ["ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances"],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
}
{
  "Effect": "Allow",
  ],
  "Effect": "Allow",
  "Action": ["ec2:RunInstances"],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/CreatedBy": ["EC2 Image Builder", "EC2 Fast Launch"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": ["ec2.amazonaws.com", "ec2.amazonaws.com.cn", "vmie.amazonaws.com"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": ["ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances"],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
}
{
  "Effect": "Allow",
  ],
  "Effect": "Allow",
  "Action": ["ec2:RunInstances"],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/CreatedBy": ["EC2 Image Builder", "EC2 Fast Launch"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": ["ec2.amazonaws.com", "ec2.amazonaws.com.cn", "vmie.amazonaws.com"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": ["ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances"],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
}
{
  "Effect": "Allow",
  ],
  "Effect": "Allow",
  "Action": ["ec2:RunInstances"],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/CreatedBy": ["EC2 Image Builder", "EC2 Fast Launch"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": ["ec2.amazonaws.com", "ec2.amazonaws.com.cn", "vmie.amazonaws.com"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": ["ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances"],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
}
{
  "Effect": "Allow",
  ],
  "Effect": "Allow",
  "Action": ["ec2:RunInstances"],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/CreatedBy": ["EC2 Image Builder", "EC2 Fast Launch"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": ["ec2.amazonaws.com", "ec2.amazonaws.com.cn", "vmie.amazonaws.com"]
    }
  }
}
{
  "Effect": "Allow",
  "Action": ["ec2:StopInstances", "ec2:StartInstances", "ec2:TerminateInstances"],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
}
```
"ec2:DescribeSubnets",
"ec2:DescribeTags",
"ec2:ModifyImageAttribute",
"ec2:DescribeImportImageTasks",
"ec2:DescribeExportImageTasks",
"ec2:DescribeSnapshots"
],
"Resource": "*
),
{
"Effect": "Allow",
"Action": [
"ec2:ModifySnapshotAttribute"
],
"Resource": "arn:aws:ec2:*::snapshot/**",
"Condition": {
"StringEquals": {
"ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
}
}
},
{
"Effect": "Allow",
"Action": [
"ec2:CreateTags"
],
"Resource": "*",
"Condition": {
"StringEquals": {
"ec2:CreateAction": [
"RunInstances",
"CreateImage"
],
"aws:RequestTag/CreatedBy": [
"EC2 Image Builder",
"EC2 Fast Launch"
]
}
}
},
{
"Effect": "Allow",
"Action": [
"ec2:CreateTags"
],
"Resource": [
"arn:aws:ec2:*::image/**",
"arn:aws:ec2:*:*:export-image-task/**"
]
},
{
"Effect": "Allow",
"Action": [
"ec2:CreateTags"
],
"Resource": [
"arn:aws:ec2:*::snapshot/**",
"arn:aws:ec2:*:*:launch-template/**"
],
"Condition": {
"StringEquals": {
"aws:RequestTag/CreatedBy": [
"EC2 Image Builder",
"EC2 Fast Launch"
]
}
}
Managed policies

```json
{
  "Effect": "Allow",
  "Action": [
    "license-manager:UpdateLicenseSpecificationsForResource"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": ["sns:Publish"],
  "Resource": "*"
},
{
  "Effect": "Allow",
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:SendCommand"]
},
{
  "Effect": "Allow",
  "Action": ["ssm:SendCommand"
  ],
  ],
  "Condition": {
    "StringEquals": {
      "ssm:resourceTag/CreatedBy": ["EC2 Image Builder"
      ]
    }
  }
},
{
  "Effect": "Allow",
  "Action": ["ssm:StartAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:RunCommand"
  ],
  "Condition": {
    "StringEquals": {
      "ssm:resourceTag/CreatedBy": ["EC2 Image Builder"
      ]
    }
  }
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuilder*"
},
{
  "Effect": "Allow",
  "Action": ["ssm:StopAutomationExecution"
  ],
  "Resource": "arn:aws:ssm:*:*:automation-definition/ImageBuild
Managed policies

"ssm:CreateAssociation",
"ssm:DeleteAssociation"
],
"Resource": [
"arn:aws:ssm:*:*:document/AWS-GatherSoftwareInventory",
"arn:aws:ssm:*:*:association/**",
"arn:aws:ec2:*:*:instance/**
]
},
{
"Effect": "Allow",
"Action": [
"kms:Encrypt",
"kms:Decrypt",
"kms:ReEncryptFrom",
"kms:ReEncryptTo",
"kms:GenerateDataKeyWithoutPlaintext"
],
"Resource": "*",
"Condition": {
"ForAllValues:StringEquals": {
"kms:EncryptionContextKeys": [
"aws:ebs:id"
]
},
"StringLike": {
"kms:ViaService": [
"ec2.*.amazonaws.com"
]
}
}
},
{
"Effect": "Allow",
"Action": [
"kms:DescribeKey"
],
"Resource": "*",
"Condition": {
"StringLike": {
"kms:ViaService": [
"ec2.*.amazonaws.com"
]
}
}
},
{
"Effect": "Allow",
"Action": "kms:CreateGrant",
"Resource": "*",
"Condition": {
"Bool": {
"kms:GrantIsForAWSResource": true
},
"StringLike": {
"kms:ViaService": [
"ec2.*.amazonaws.com"
]
}
}
}
},
{
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::*:role/EC2ImageBuilderDistributionCrossAccountRole"
Managed policies

```json
{
  "Effect": "Allow",
  "Action": [  
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
  ],
  "Resource": "arn:aws:logs:*::*:log-group:/aws/imagebuilder/*"
},
{
  "Effect": "Allow",
  "Action": [  
    "ec2:CreateLaunchTemplateVersion",
    "ec2:DescribeLaunchTemplates",
    "ec2:ModifyLaunchTemplate",
    "ec2:DescribeLaunchTemplateVersions"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [  
    "ec2:ExportImage"
  ],
  "Resource": "arn:aws:ec2:*::*:image/*",
  "Condition": {  
    "StringEquals": {  
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [  
    "ec2:ExportImage"
  ],
  "Resource": "arn:aws:ec2:*::*:export-image-task/*"
},
{
  "Effect": "Allow",
  "Action": [  
    "ec2:CancelExportTask"
  ],
  "Resource": "arn:aws:ec2:*::*:export-image-task/*",
  "Condition": {  
    "StringEquals": {  
      "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
    }
  }
},
{
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "*",
  "Condition": {  
    "StringEquals": {  
      "iam:AWSServiceName": [  
        "ssm.amazonaws.com",
        "ec2fastlaunch.amazonaws.com"
      ]
    }
  }
}
}
```
"ec2:EnableFastLaunch",
"Resource": [
  "arn:aws:ec2:*::image/**",
  "arn:aws:ec2:*::launch-template/**
],
"Condition": {
  "StringEquals": {
    "ec2:ResourceTag/CreatedBy": "EC2 Image Builder"
  }
},
"Effect": "Allow",
"Action": [
  "inspector2:ListCoverage",
  "inspector2:ListFindings"
],
"Resource": "*"
},
{
"Effect": "Allow",
"Action": [
  "ecr:CreateRepository"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "aws:RequestTag/CreatedBy": "EC2 Image Builder"
  }
}
},
{
"Effect": "Allow",
"Action": [
  "ecr:TagResource"
],
"Resource": "arn:aws:ecr:*:*:repository/image-builder-*",
"Condition": {
  "StringEquals": {
    "aws:RequestTag/CreatedBy": "EC2 Image Builder"
  }
}
},
{
"Effect": "Allow",
"Action": [
  "ecr:BatchDeleteImage"
],
"Resource": "arn:aws:ecr:*:*:repository/image-builder-*",
"Condition": {
  "StringEquals": {
    "ecr:ResourceTag/CreatedBy": "EC2 Image Builder"
  }
}
},
{
"Effect": "Allow",
"Action": [
  "events:DeleteRule",
  "events:DescribeRule",
  "events:PutRule",
  "events:PutTargets",
  "events:RemoveTargets"
],
"Resource": [
  "arn:aws:events:*::rule/image-builder-*",
  "arn:aws:events:*::ruleset/image-builder-*"
]
},
{
"Effect": "Allow",
"Action": [
  "sts:AssumeRole"
],
"Resource": "*"
}
Ec2ImageBuilderCrossAccountDistributionAccess policy

The Ec2ImageBuilderCrossAccountDistributionAccess policy grants permissions for Image Builder to distribute images across accounts in target Regions. Additionally, Image Builder can describe, copy, and apply tags to any Amazon EC2 image in the account. The policy also grants the ability to modify AMI permissions via the ec2:ModifyImageAttribute API action.

Permissions details

This policy includes the following permissions:

- **Amazon EC2** – Access is granted for Amazon EC2 to describe, copy, and modify attributes for an image, and to create tags for any Amazon EC2 images in the account.

Policy example

The following is an example of the Ec2ImageBuilderCrossAccountDistributionAccess policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:*::image/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeImages",
        "ec2:CopyImage",
        "ec2:ModifyImageAttribute"
      ],
      "Resource": "*"
    }
  ]
}
```

EC2InstanceProfileForImageBuilder policy

The EC2InstanceProfileForImageBuilder policy grants the minimum permissions required for an EC2 instance to work with Image Builder. This does not include permissions required to use the Systems Manager Agent.

Permissions details

This policy includes the following permissions:

- **CloudWatch Logs** – Access is granted to create and upload CloudWatch Logs to any log group whose name starts with /aws/imagebuilder/.
- **Image Builder** – Access is granted to get any Image Builder component.
- **AWS KMS** – Access is granted to decrypt an Image Builder component, if it was encrypted via AWS KMS.
Managed policies

• **Amazon S3** – Access is granted to get objects stored in an Amazon S3 bucket whose name starts with `ec2imagebuilder-`.

**Policy example**

The following is an example of the EC2InstanceProfileForImageBuilder policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:GetComponent"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "kms:EncryptionContextKeys": "aws:imagebuilder:arn",
                    "aws:CalledVia": [
                        "imagebuilder.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::ec2imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/imagebuilder/*"
        }
    ]
}
```

**EC2InstanceProfileForImageBuilderECRContainerBuilds policy**

The **EC2InstanceProfileForImageBuilderECRContainerBuilds** policy grants the minimum permissions required for an EC2 instance when working with Image Builder to build Docker images and then register and store the images in an Amazon ECR container repository. This does not include permissions required to use the Systems Manager Agent.

**Permissions details**

This policy includes the following permissions:
- **CloudWatch Logs** – Access is granted to create and upload CloudWatch Logs to any log group whose name starts with `/aws/imagebuilder/`.
- **Amazon ECR** – Access is granted for Amazon ECR to get, register, and store a container image, and to get an authorization token.
- **Image Builder** – Access is granted to get an Image Builder component or container recipe.
- **AWS KMS** – Access is granted to decrypt an Image Builder component or container recipe, if it was encrypted via AWS KMS.
- **Amazon S3** – Access is granted to get objects stored in an Amazon S3 bucket whose name starts with `ec2imagebuilder-`.

### Policy example

The following is an example of the `EC2InstanceProfileForImageBuilderECRContainerBuilds` policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "imagebuilder:GetComponent",
                "imagebuilder:GetContainerRecipe",
                "ecr:GetAuthorizationToken",
                "ecr:BatchGetImage",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:PutImage"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "kms:EncryptionContextKeys": "aws:imagebuilder:arn",
                    "aws:CalledVia": [
                        "imagebuilder.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::ec2imagebuilder*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```
Image Builder updates to AWS managed policies

This section provides information about updates to AWS managed policies for Image Builder since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Image Builder document history (p. 310) page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWSServiceRoleForImageBuilder</td>
<td>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.</td>
<td>March 30, 2023</td>
</tr>
<tr>
<td></td>
<td>• Added inspector2:ListCoverage and inspector2:ListFindings to allow Image Builder to determine when Amazon Inspector completes build instance scans, and to collect findings for images that are configured to allow it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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-*. The name constraint prevents the escalation of privileges and prevents changes to repositories that Image Builder didn't create.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added ecr:BatchDeleteImage for ECR repositories tagged with CreatedBy: EC2 Image Builder. This</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Permission requires the repository name to start with image-builder-<em>. • Added event permissions for Image Builder to create and manage Amazon EventBridge managed rules that include ImageBuilder-</em> in the name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>AWSServiceRoleForImageBuilder</strong> (p. 281) – 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.</td>
<td>March 22, 2022</td>
</tr>
<tr>
<td></td>
<td><strong>AWSServiceRoleForImageBuilder</strong> (p. 281) – 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.</td>
<td>February 21, 2022</td>
</tr>
<tr>
<td></td>
<td><strong>AWSServiceRoleForImageBuilder</strong> (p. 281) – 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.</td>
<td>November 20, 2021</td>
</tr>
<tr>
<td></td>
<td><strong>AWSServiceRoleForImageBuilder</strong> (p. 281) – 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.</td>
<td>August 11, 2021</td>
</tr>
</tbody>
</table>
### Using service-linked roles for EC2 Image Builder

EC2 Image Builder uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Image Builder. Service-linked roles are predefined by Image Builder and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Image Builder more efficient, because you don’t have to add the necessary permissions manually. Image Builder defines the permissions of its service-linked roles, and unless defined otherwise, only Image Builder can assume its roles. The defined permissions include the trust policy and the permissions policy. The permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS services That Work with IAM](https://aws.amazon.com/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 (p. 281)` 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.

   ```bash
   ```

2. Change all pipeline schedules to use a manual build process, or delete them if you won’t be using them again. For more information about deleting resources, see `Delete EC2 Image Builder resources (p. 230)`.

**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 EC2 Image Builder service-linked roles**

Image Builder supports using service-linked roles in all of the AWS Regions where the service is available. For the list of supported AWS Regions, see `AWS Regions and Endpoints (p. 8)`.

**Troubleshooting EC2 Image Builder identity and access**

**Topics**
- I am not authorized to perform an action in Image Builder (p. 296)
- I am not authorized to perform iam:PassRole (p. 296)
- I want to allow people outside of my AWS account to access my Image Builder resources (p. 296)
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 EC2 Image Builder works with IAM (p. 268).
- 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 How IAM roles differ from resource-based policies in the IAM User Guide.
Compliance validation for EC2 Image Builder

EC2 Image Builder is not in scope of any AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Image Builder is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **Evaluating Resources with Rules** in the AWS Config Developer Guide – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

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 (p. 263).

Resilience in EC2 Image Builder

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

The EC2 Image Builder service allows you to distribute images built in one Region with other Regions, giving them multi-Region resiliency for AMIs. There is no mechanism to “back up” image pipelines, recipes, or components. You can store the recipe and component documents outside of the Image Builder service, such as in an Amazon S3 bucket.

The EC2 Image Builder cannot be configured for High Availability (HA). You can distribute images to multiple Regions to make the images more highly available.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in 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](https://aws.amazon.com/security/tls/).

- 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)](https://aws.amazon.com/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 in EC2 Image Builder

EC2 Image Builder provides the latest Amazon Linux 2, Amazon Linux 2023, Red Hat Enterprise Linux (RHEL), CentOS, Ubuntu, SUSE Linux Enterprise Server, and Windows 2012 R2 and later AMIs as managed image sources. You maintain the Amazon EC2 system patching responsibility, per the [shared responsibility model](https://aws.amazon.com/sharedresponsibility/). If the EC2 instances in your application workload can be easily replaced, then it might be more efficient to update the base AMI and redeploy all compute nodes based on this image.

The following are two ways you can keep your Image Builder AMIs up to date.

- **AWS-provided patching components** – EC2 Image Builder provides two build components, `update-linux` and `update-windows`, which install all pending operating system updates. These components use the `UpdateOS` action module. For more information, see [UpdateOS](https://docs.aws.amazon.com/imagebuilder/latest/userguide/patching.html). 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](https://docs.aws.amazon.com/imagebuilder/latest/userguide/awstoe-component-manager.html).

### Component that uses the `UpdateOS` action module

```
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
```

298
Security best practices for EC2 Image Builder

EC2 Image Builder provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- Do not use overly-permissive security groups in Image Builder recipes.
- Do not share images with accounts that you do not trust.
- Do not make images public that have private or sensitive data.
- Apply all available Windows or Linux security patches during image builds.

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 EC2 Image Builder infrastructure configuration (p. 195). For more information about EC2 instance metadata options for Linux images, see Configure the instance metadata options in the Amazon EC2 User Guide for Linux Instances. For Windows images, see Configure the instance metadata options in the Amazon EC2 User Guide for Windows Instances.

**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.

```json
steps:
- name: InstallYumUpdates
  action: ExecuteBash
  inputs:
  commands:
  - sudo yum update -y
```
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.

```bash
#!/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"
```

300
Required post-build clean up

```
"/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 TOE 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

# 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
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 (p. 79) 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.

<table>
<thead>
<tr>
<th>Clean up section</th>
<th>Files removed</th>
<th>Skip section file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOUD_INIT_FILES</td>
<td>/etc/sudoers.d/90-cloud-init-users</td>
<td>skip_cleanup_cloudinit_files</td>
</tr>
<tr>
<td></td>
<td>/etc/locale.conf</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/log/cloud-init.log</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/log/cloud-init-output.log</td>
<td></td>
</tr>
</tbody>
</table>

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.
## Override the Linux cleanup script

### Clean up section

<table>
<thead>
<tr>
<th>Clean up section</th>
<th>Files removed</th>
<th>Skip section file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTANCE_FILES</td>
<td>/etc/.updated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/aliases.db</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/hostname</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/lib/misc/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>postfix.aliasesdb-stamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/lib/postfix/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>master.lock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/spool/postfix/pid/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>master.pid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/.updated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/cache/yum/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x86_64/2/.gpgkeyschecked.yum</td>
<td></td>
</tr>
<tr>
<td>SSH_FILES</td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_rsa_key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_rsa_key.pub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_ecdsa_key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_ecdsa_key.pub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_ed25519_key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/etc/ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssh_host_ed25519_key.pub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/root/.ssh/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>authorized_keys</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/home/&lt;all users&gt;/.ssh/authorized_keys;</td>
<td></td>
</tr>
<tr>
<td>INSTANCE_LOG_FILES</td>
<td>/var/log/audit/audit.log</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/log/boot.log</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/log/dmesg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/var/log/cron</td>
<td></td>
</tr>
<tr>
<td>TOE_FILES</td>
<td>{{workingDirectory}}/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOE_*</td>
<td></td>
</tr>
<tr>
<td>SSM_LOG_FILES</td>
<td>/var/log/amazon/ssm/*</td>
<td></td>
</tr>
</tbody>
</table>
Troubleshoot EC2 Image Builder

EC2 Image Builder integrates with AWS services for monitoring and troubleshooting to help you troubleshoot image build issues. Image Builder tracks and displays the progress for each step in the image building process. Additionally, Image Builder can export logs to an Amazon S3 location that you provide.

For advanced troubleshooting, you can run predefined commands and scripts using AWS Systems Manager Run Command.

Contents
- Troubleshoot pipeline builds (p. 305)
- Troubleshooting scenarios (p. 306)

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 (p. 198).

# Troubleshooting scenarios

This section lists the following detailed troubleshooting scenarios:

- **Access denied – status code 403** (p. 306)
- **Build times out while verifying the Systems Manager Agent availability on the build instance** (p. 307)
- **Windows secondary disk is offline at launch** (p. 308)
- **Build fails with CIS hardened base image** (p. 308)
- **AssertInventoryCollection fails (Systems Manager Automation)** (p. 309)

To see the details of a scenario, choose the scenario title to expand it. You can have multiple titles expanded at the same time.

**Access denied – status code 403**

**Description**

The pipeline build fails with "AccessDenied: Access Denied status code: 403".
Cause

Possible causes include:

- The instance profile does not have the required permissions (p. 12) to access APIs or component resources.
- The instance profile role is missing permissions that are required for logging to Amazon S3. Most commonly, this occurs when the instance profile role does not have **PutObject** permissions for your S3 buckets.

Solution

Depending on the cause, this issue can be resolved as follows:

- **Instance profile is missing managed policies** – Add the missing policies to your instance profile role. Then run the pipeline again.
- **Instance profile is missing write permissions for S3 bucket** – Add a policy to your instance profile role that grants **PutObject** permissions to write to your S3 bucket. Then run the pipeline again.

Build times out while verifying the Systems Manager Agent availability on the build instance

Description

The pipeline build fails with "status = 'TimedOut'" and "failure message = 'Step timed out while step is verifying the Systems Manager Agent availability on the target instance(s)'".

Cause

Possible causes include:

- The instance that was launched to perform the build operations and to run components was not able to access the Systems Manager endpoint.
- The instance profile does not have the required permissions (p. 12).

Solution

Depending on the possible cause, this issue can be resolved as follows:

- **Access issue, private subnet** – If you are building in a private subnet, make sure that you have set up PrivateLink endpoints for Systems Manager, Image Builder, and, if you want logging, Amazon S3/CloudWatch. For more information about setting up PrivateLink endpoints, see [VPC endpoints concepts (AWS PrivateLink)].
- **Missing permissions** – Add the following managed policies to your IAM service-linked role for Image Builder:
  - EC2InstanceProfileForImageBuilder
  - EC2InstanceProfileForImageBuilderECRContainerBuilds
  - AmazonSSMManagedInstanceCore

For more information about the Image Builder service-linked role, see [Using service-linked roles for EC2 Image Builder](p. 294).
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)

<table>
<thead>
<tr>
<th>PS C:\Users\Administrator&gt; get-disk</th>
<th>Number</th>
<th>Friendly Name</th>
<th>Serial Number</th>
<th>Health Status</th>
<th>Operational Status</th>
<th>Total Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>AWS PVDISK</td>
<td>vol0abc12d34e567f8a9</td>
<td>Healthy</td>
<td>Online</td>
<td>30 GB</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>AWS PVDISK</td>
<td>vol1bcd23e45f678a9b0</td>
<td>Healthy</td>
<td>Offline</td>
<td>8 GB</td>
</tr>
</tbody>
</table>

Cause

Because of Windows default settings, newly discovered disks are not automatically brought online and formatted. When the instance type is changed on EC2, Windows treats this as new disks being discovered. This is because of the underlying driver change.

Solution

We recommend that you use the same system of instance types when building your Windows AMI that you intend to launch from. Do not include instance types that are built on different systems in your infrastructure configuration. If any of the instance types you specify use the Nitro system, then they should all use the Nitro system.

For more information about instances that are built on the Nitro system, see Instances built on the Nitro System in the Amazon EC2 User Guide for Windows Instances.

Build fails with CIS hardened base image

Description

You are using a CIS hardened base image and the build fails.

Cause

When the /tmp directory is classified as noexec, it can cause Image Builder to fail.

Solution

Choose a different location for your working directory in the workingDirectory field of the image recipe. For more information, see the ImageRecipe data type description.
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 (console) (p. 235). For more information about updating your container image pipeline using the EC2 Image Builder console, see Update a container image pipeline (console) (p. 241).

- 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.

```json
{
  "name": "MyWindows2019Pipeline",
  "description": "Builds Windows 2019 Images",
  "enhancedImageMetadataEnabled": false,
  "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.
# Document history for EC2 Image Builder user guide

The following table describes important changes to the documentation by date. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version: 2023-03-30**

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>STIG Q3 updates (p. 124)</td>
<td>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.</td>
<td>October 5, 2023</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2023 second quarter release.</td>
<td>May 3, 2023</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2023 first quarter release. Added support for AL2023.</td>
<td>April 14, 2023</td>
</tr>
<tr>
<td>Update supported Regions for AWSTOE (p. 23)</td>
<td>Added AWSTOE support for the following AWS Regions: Asia Pacific (Hyderabad), Asia Pacific (Jakarta), Europe (Zurich), Europe (Spain), and Middle East (UAE).</td>
<td>April 13, 2023</td>
</tr>
<tr>
<td>AWSTOE application download updates (p. 23)</td>
<td>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.</td>
<td>March 31, 2023</td>
</tr>
<tr>
<td>Feature release: Enhanced build workflows (p. 310)</td>
<td>Added runtime details for image builds in the new workflow tab in the image build version details. Improved information for troubleshooting builds.</td>
<td>March 30, 2023</td>
</tr>
<tr>
<td>Feature release: CVE detection and reporting (p. 310)</td>
<td>For accounts that have activated Amazon Inspector scans, Image Builder can capture the common vulnerability and exposures (CVE) findings from</td>
<td>March 30, 2023</td>
</tr>
</tbody>
</table>
Amazon Inspector during 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.

<table>
<thead>
<tr>
<th>Added version history (p. 124)</th>
<th>Added version history to the Windows and Linux sections.</th>
</tr>
</thead>
<tbody>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2022 fourth quarter release.</td>
</tr>
<tr>
<td>Feature release: AWS Marketplace integration and CIS hardening (p. 310)</td>
<td>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.</td>
</tr>
<tr>
<td>CIS hardening components (p. 23)</td>
<td>Added CIS hardening components that are owned and maintained by CIS.</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Introduced Ubuntu support, updated STIG versions, and applied STIGS for 2022 second quarter release.</td>
</tr>
<tr>
<td>Document update: Navigation for Create YAML component document page (p. 310)</td>
<td>Moved the Create YAML component document content to its own page, and updated other pages to reference it.</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2022 first quarter release.</td>
</tr>
<tr>
<td>Added ExecuteDocument action module</td>
<td>Added documentation for the ExecuteDocument action module under General execution.</td>
</tr>
<tr>
<td>Feature release: Support for faster launching Windows AMI (p. 310)</td>
<td>Added distribution configuration settings to support faster launching for Windows AMIs.</td>
</tr>
<tr>
<td>Maintenance release: Update AWSTOE binary thumbprint (p. 310)</td>
<td>Updated binary thumbprint for AWSTOE signer certificate.</td>
</tr>
<tr>
<td>Feature release: Configure input for AWSTOE (p. 310)</td>
<td>Added support for using a JSON configuration file as input for the AWSTOE run command.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2021 fourth quarter release. Also added a section for new SCAP Compliance Checker (SCC) components.</td>
</tr>
<tr>
<td>Feature release: VM Import/Export (VMIE) integration (p. 310)</td>
<td>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.</td>
</tr>
<tr>
<td>Feature release: AMI sharing for AWS Organizations and OUs (p. 310)</td>
<td>Updated distribution configuration to add support for sharing output AMIs with AWS Organizations and OUs.</td>
</tr>
<tr>
<td>Document update: Update component stages and phases (p. 310)</td>
<td>Expanded content for component stages in Image Builder, and how those interact with AWSTOE component phases.</td>
</tr>
<tr>
<td>Document update: Add CloudTrail integration content (p. 310)</td>
<td>Added monitoring summary and CloudTrail integration content.</td>
</tr>
<tr>
<td>New STIG Versions (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2021 third quarter release.</td>
</tr>
<tr>
<td>Feature release: Amazon EventBridge integration (p. 310)</td>
<td>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.</td>
</tr>
<tr>
<td>Feature release: Parameterized components and additional instance configuration (p. 310)</td>
<td>Added support for specifying parameters to customize components for recipes. Expanded configuration of the EC2 instances that are used for building and testing images, including the ability to specify commands to run on launch, and more control over installation and removal of the Systems Manager agent.</td>
</tr>
<tr>
<td>Update Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>New STIG versions</strong> (p. 124)</td>
<td>Updated STIG versions and applied STIGS for 2021 second quarter release.</td>
</tr>
<tr>
<td><strong>Enhancement: Tagging enhancements</strong> (p. 310)</td>
<td>Improved messaging around resource tagging.</td>
</tr>
<tr>
<td><strong>Feature release: Launch template integration</strong> (p. 310)</td>
<td>Added support for using Amazon EC2 launch templates for AMI distribution in the Distribution settings.</td>
</tr>
<tr>
<td><strong>Feature release: Container build enhancements</strong> (p. 310)</td>
<td>Added support for configuring block device mappings and specifying AMIs to use as the base image for container builds.</td>
</tr>
<tr>
<td><strong>New STIG versions</strong> (p. 124)</td>
<td>Updated STIG versions and applied STIGS.</td>
</tr>
<tr>
<td><strong>Update cron expressions</strong> (p. 310)</td>
<td>Image Builder cron processing is updated to increase cron expression granularity to the minute, and use a standard cron scheduling engine. Examples are updated with the new format.</td>
</tr>
<tr>
<td><strong>Feature release: Container support</strong> (p. 310)</td>
<td>Added support for creating Docker container images using Image Builder, with registration and storage of the resulting images on Amazon Elastic Container Registry (Amazon ECR). Content has been rearranged to reflect new functionality and accommodate future growth.</td>
</tr>
<tr>
<td><strong>Restructured cron documentation</strong> (p. 310)</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Console version 2.0: updated pipeline editing</strong> (p. 310)</td>
<td>Content changes in getting started and create pipeline tutorials, plus the manage image pipelines page, to incorporate new console features and flow.</td>
</tr>
<tr>
<td><strong>New STIG versions</strong> (p. 124)</td>
<td>Updated STIG versions and applied STIGS. Note - list format changed to show STIGs that are applied by default.</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Support for looping constructs in AWSTOE</strong></td>
<td>Create looping constructs to define a repeated sequence of instructions in the AWSTOE application.</td>
</tr>
<tr>
<td><strong>Support for local development of AWSTOE components</strong></td>
<td>Develop and test image components locally with the AWSTOE application.</td>
</tr>
<tr>
<td><strong>Encrypted AMIs</strong></td>
<td>EC2 Image Builder adds support for encrypted AMI distribution.</td>
</tr>
<tr>
<td><strong>AutoScaling deprecation</strong></td>
<td>Deprecation of the use of AutoScaling to launch instances.</td>
</tr>
<tr>
<td><strong>Support for connectivity through AWS PrivateLink</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>New STIG versions</strong></td>
<td>Updated STIG versions and applied STIGS.</td>
</tr>
<tr>
<td><strong>Troubleshooting</strong></td>
<td>Added general troubleshooting scenarios.</td>
</tr>
<tr>
<td><strong>STIG Components</strong></td>
<td>You can create STIG-compliant images with AWSTOE STIG components.</td>
</tr>
</tbody>
</table>