



Manufacturing devices at scale with AWS IoT Greengrass golden images

AWS Prescriptive Guidance



AWS Prescriptive Guidance: Manufacturing devices at scale with AWS IoT Greengrass golden images

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Introduction	1
Intended audience	1
Assumed knowledge	1
What is a golden image?	2
Extracting a golden image from a golden device	2
Unique configuration	3
Why use golden images?	4
Using the Greengrass CLI instead of a golden image	5
Golden image anatomy	6
Greengrass directory tree	6
Contents of the packages directory	7
artifacts	8
artifacts-unarchived	9
recipes	11
System service	12
Docker images	12
Secrets	12
When secrets are a hard requirement	13
Golden image methods	15
Full Greengrass snapshot	15
Partial Greengrass snapshot	16
Golden image procedure	17
Create your golden device	17
Read the golden image from your golden device	17
Write the golden image to your production devices	18
Configure your production devices	19
Full Greengrass snapshot method	15
Partial Greengrass snapshot method	16
Conclusion	25
Resources	26
Document history	27
Glossary	28
#	28
A	29

B	32
C	34
D	37
E	41
F	43
G	45
H	46
I	47
L	50
M	51
O	55
P	58
Q	60
R	61
S	64
T	68
U	69
V	70
W	70
Z	71

Manufacturing devices at scale with AWS IoT Greengrass golden images

Greg Breen, Amazon Web Services (AWS)

October 2024 ([document history](#))

[AWS IoT Greengrass](#) is an edge runtime and cloud service that helps you compose, deploy, and manage Internet of Things (IoT) device software at scale. By convention, an AWS IoT Greengrass [deployment](#) distributes your application's software [components](#) from the cloud to your devices. If you're a device maker who manufactures at scale, it's unlikely that you'd want to perform an AWS IoT Greengrass deployment from the cloud to each device that comes off the manufacturing line in your factory. Instead, you would likely want to bundle your entire software stack into a single image, and flash each device by wire. This guide outlines approaches you can take to bundling the AWS IoT Greengrass edge runtime, and your application components and configuration, into a *golden image*. This helps to facilitate efficient and scalable factory-line programming of your devices. These approaches help increase the productivity of your device manufacturing operations and reduce your per-unit manufacturing costs.

Intended audience

This guide is intended for architects, technical leads, and engineers who are responsible for designing and developing the manufacturing stations on the production line of an IoT device or product that uses AWS IoT Greengrass.

Assumed knowledge

This guide assumes that you're familiar with:

- The AWS IoT Greengrass service and features such as components, deployments, recipes, and artifacts. For information, see the [AWS IoT Greengrass documentation](#).
- The Linux operating system.
- Device manufacturing lines and processes.

What is a golden image?

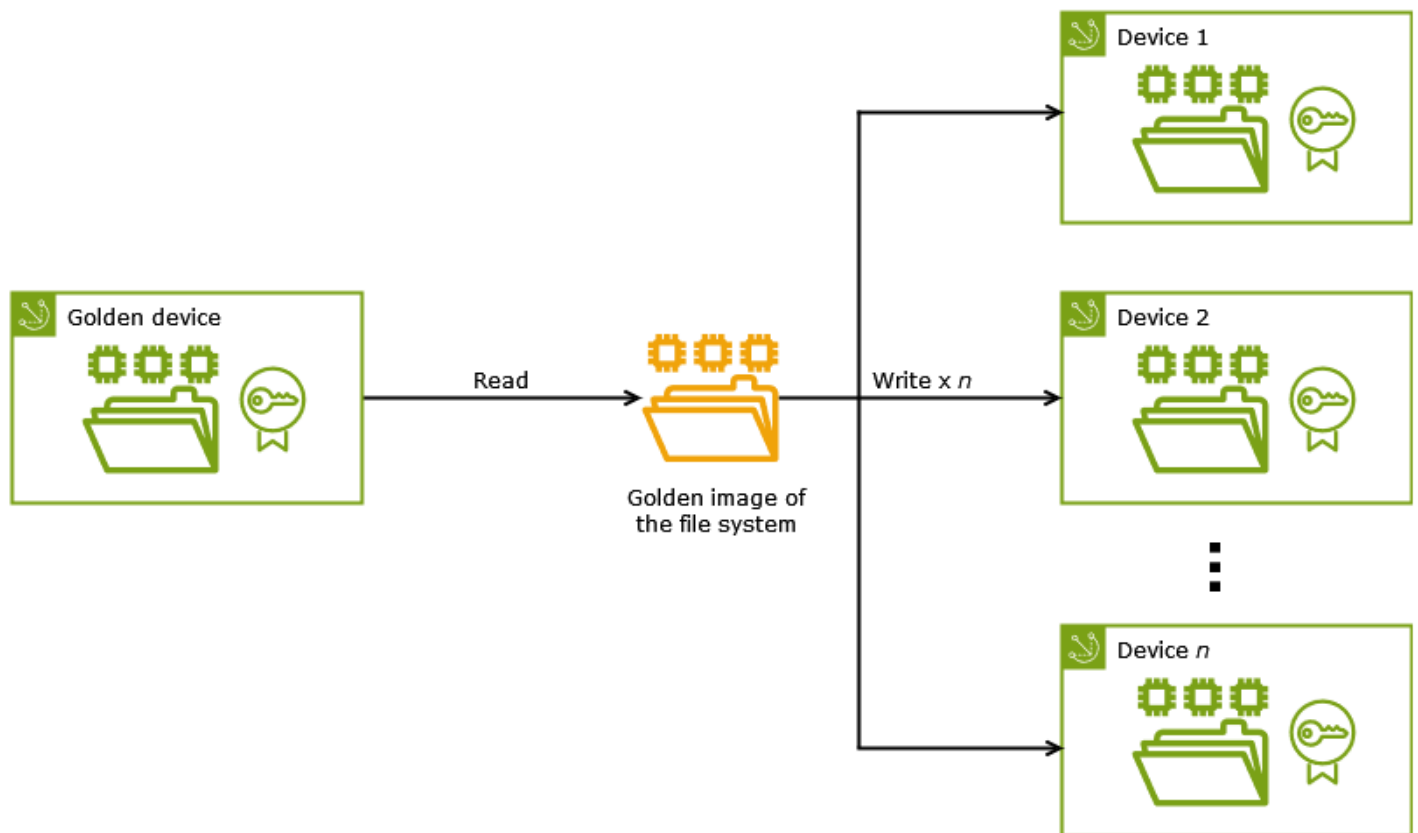
A golden image is a snapshot of software used to flash or provision many devices. Here are some examples of golden images in other domains that you might already be familiar with:

- **Raspberry Pi:** The [Raspberry Pi OS ISO files](#) that you can download and use to flash the Raspberry Pi SD card.
- **Amazon Elastic Compute Cloud (Amazon EC2):** The [Amazon Machine Images \(AMIs\)](#) you use to launch an [Amazon EC2](#) instance.
- **Docker:** The Docker images you download from registries such as [Docker Hub](#) and use to launch Docker containers.
- **Micro-controllers:** For highly-constrained micro-controllers, it's common to combine the bootloader, the application, and data sections into a single Motorola S-record file, Intel HEX file, or binary file for flashing by wire in the factory.

Extracting a golden image from a golden device

A golden image can be composed, or it can be created by taking a snapshot of a golden device whose image represents the desired state. In the case of AWS IoT Greengrass, using a snapshot of a golden device is the recommended approach.

As shown in the following illustration, a golden device is created, its file system is read to create the golden image, and this image is then written to many devices, at scale.



Unique configuration

Although the same golden image is written to every device, a small amount of unique configuration or personalization (for example, unique serial numbers, unique device names, and unique credentials) is typically also needed for each device. In the Raspberry Pi example, the `raspi-config` utility is used to create the unique configuration after flashing. In the case of AWS IoT Greengrass, a core device requires at least a unique thing name, a unique device certificate, and a unique private key.

Why use golden images?

There are numerous reasons for using a golden image to program devices in the factory.

If you're planning to manufacture AWS IoT Greengrass devices at scale, you might consider it problematic to perform an AWS IoT Greengrass cloud deployment for each device on the manufacturing line. Your reasons might include:

- **It's too slow:** The time required to perform a deployment for every device can affect your manufacturing line productivity.
- **There's no internet connectivity:** Contract manufacturers sometimes don't have internet connectivity on the manufacturing line, so a cloud deployment might not even be possible.
- **There's intermittent internet connectivity:** Even if internet connectivity is nominally available, your company might require that your manufacturing line not be stalled by a temporary loss of connectivity, so a cloud deployment might not be desirable.

If you choose not to perform an AWS IoT Greengrass deployment on the manufacturing line, you could ship your device with no application components installed. However, your devices might still require application components to be installed during manufacture for the following reasons:

- **Factory acceptance testing (FAT):** You might need to perform FAT on every device, or a sampling of devices, as part of your quality assurance (QA) processes. Your device will likely have its own domain-specific I/O and interfaces that can be tested only if your device is in a fully working state and runs your own application software.
- **Field internet connectivity isn't reliable:** When your device is installed or unboxed in the field, you can't be sure that there will be internet connectivity in its environment. Therefore, you can't rely on a cloud deployment to put the device into a fully working state.
- **Device needs to work immediately:** Your device might have to be in a fully working state when it's first unboxed in the field and not able to wait for the first cloud deployment to complete, even if internet connectivity is reliable.
- **Cellular network data charges:** Your device might have cellular network connectivity, and downloading all your application component artifacts during an initial cloud deployment in the field might incur significant cost.

Consequently, an AWS IoT Greengrass device typically needs to be manufactured with application components partially or fully installed and configured, but might not be able to use an AWS IoT Greengrass cloud deployment. This guide discusses how you can achieve that objective.

Using the Greengrass CLI instead of a golden image

An alternative to using golden images is to locally compose your software image on each device by automating the installation of the Greengrass runtime, and then automating the installation of components by using the [Greengrass command line interface \(CLI\)](#). However:

- You would need to install the Greengrass CLI first, in an offline environment. Although this type of installation is possible, there is no documented procedure for it.
- You would need to collect and organize all component artifacts and recipes, and have them available on the manufacturing line.
- You would need to use the [Greengrass CLI local deployment command](#) to deploy each component.
- You would need to be careful when you use the `groupId` and component versions for the local deployments to make sure that subsequent cloud deployments won't be in conflict.
- If the Greengrass CLI is not required on your device in the field, you would typically remove it as a final step.

Overall, the Greengrass CLI approach is slower, and more prone to error, than a golden image.

Golden image anatomy

AWS IoT Greengrass core devices that are manufactured at scale are generally embedded Linux devices that have a Linux distribution that's constructed by using tools such as [Yocto](#). Typically, the Greengrass edge runtime is baked into the distribution, as demonstrated by the [Meta AWS](#) project.

Such devices often have their file system organized into multiple partitions. This guide uses *golden image* as a catch-all term. Your device might have several golden images, to flash the various partitions.

Your golden image might encompass the entirety of the device's file system or just part of it. This guide focuses on the parts of the file system you need to consider for AWS IoT Greengrass, without prescribing how to assemble your images more broadly.

Greengrass directory tree

To understand the golden image methods discussed in this guide, review the structure of the Greengrass directory tree shown in the following table.

Directory	Description
alts	Launch parameters and symbolic links to the Greengrass nucleus version that is currently active.
bin	Binaries, if any are installed (for example, the Greengrass CLI binary if that component is installed). This directory is often empty.
cli_ipc_info	Scratchpad for Greengrass CLI interprocess communication (IPC). This directory is empty if you haven't installed the Greengrass CLI.
config	All Greengrass configuration, including component configuration.
deployments	Data for managing the state of deployments and rollbacks.

logs	Log files for the nucleus and other components.
packages	Artifacts and recipes for all components.
plugins	Storage for components of type <code>aws.greengrass.plugin</code> that you manually installed. Otherwise, this directory holds no data.
telemetry	Scratchpad used by Greengrass to aggregate telemetry data ready for publishing.
work	Scratchpad for components.

The logs, telemetry, and work directories contain only ephemeral data. They don't need to be included in a golden image, so omit them if you want to minimize the size of the image.

The Greengrass CLI isn't usually installed on production devices, so the `bin` and `cli_ipc_info` directories are often empty and don't typically need to be included in a golden image.

The `plugins` directory includes data only if you manually installed a plugin (such as the [fleet provisioning plugin](#) or a custom provisioning plugin) when you installed Greengrass.

The data in the `deployments` directory is used only when a deployment is in progress and therefore isn't needed in a golden image.

Consequently the `alts`, `config` and `packages` directories are of the greatest interest. Sometimes these are the only Greengrass directories you need to include in a golden image, if you want to minimize the image size.

Contents of the packages directory

The `packages` directory has three subdirectories, as shown in the following table.

Subdirectory	Description
artifacts	The zipped component artifacts that Greengrass downloads during deployments.

artifacts-unarchived

For artifacts that are .zip archives, this directory contains the same artifacts, but they are unzipped so that components can use the artifact contents.

recipes

The component recipe files.

artifacts

The following example tree listing of packages/artifacts shows how artifacts are stored.

```
user@machine:~$ sudo tree /greengrass/v2/packages/artifacts
/greengrass/v2/packages/artifacts
### aws.greengrass.DockerApplicationManager
### aws.greengrass.LogManager
#   ### 2.3.7
#       ### aws.greengrass.LogManager.jar
### aws.greengrass.Nucleus
#   ### 2.12.6
#       ### aws.greengrass.nucleus.zip
### aws.greengrass.SecretManager
#   ### 2.1.8
#       ### aws.greengrass.SecretManager.jar
### aws.greengrass.SecureTunneling
#   ### 1.0.19
#       ### GreengrassV2SecureTunnelingComponent-1.0-all.jar
### aws.greengrass.labs.CertificateRotator
#   ### 1.1.0
#       ### certificate-rotator.zip
### aws.greengrass.labs.HomeAssistant
#   ### 1.0.0
#       ### home-assistant.zip
### aws.greengrass.telemetry.NucleusEmitter
#   ### 1.0.8
#       ### aws.greengrass.telemetry.NucleusEmitter.jar

15 directories, 7 files
```

artifacts-unarchived

The following example tree listing of packages/artifacts-unarchived shows artifacts that are extracted from .zip files.

```
user@machine:~$ sudo tree /greengrass/v2/packages/artifacts-unarchived
/greengrass/v2/packages/artifacts-unarchived
### aws.greengrass.Nucleus
#   ### 2.12.6
#       ### aws.greengrass.nucleus
#           ### LICENSE
#           ### NOTICE
#           ### README.md
#           ### THIRD-PARTY-LICENSES
#           ### bin
#               #   ### greengrass.exe
#               #   ### greengrass.service
#               #   ### greengrass.service.procd.template
#               #   ### greengrass.service.template
#               #   ### greengrass.xml.template
#               #   ### loader
#               #   ### loader.cmd
#           ### conf
#               #   ### recipe.yaml
#           ### lib
#               ### Greengrass.jar
### aws.greengrass.SecureTunneling
#   ### 1.0.19
### aws.greengrass.labs.CertificateRotator
#   ### 1.1.0
#       ### certificate-rotator
#           ### __pycache__
#               #   ### config.cpython-310.pyc
#               #   ### config.cpython-311.pyc
#               #   ### effective_config.cpython-310.pyc
#               #   ### effective_config.cpython-311.pyc
#               #   ### main.cpython-311.pyc
#               #   ### pki.cpython-310.pyc
#               #   ### pki.cpython-311.pyc
#               #   ### pki_file.cpython-310.pyc
#               #   ### pki_file.cpython-311.pyc
#               #   ### pki_hsm.cpython-310.pyc
#               #   ### pki_hsm.cpython-311.pyc
```

```
#      #      ### pubsub.cpython-310.pyc
#      #      ### pubsub.cpython-311.pyc
#      #      ### state.cpython-310.pyc
#      #      ### state.cpython-311.pyc
#      #      ### state_committing_certificate.cpython-310.pyc
#      #      ### state_committing_certificate.cpython-311.pyc
#      #      ### state_creating_certificate.cpython-310.pyc
#      #      ### state_creating_certificate.cpython-311.pyc
#      #      ### state_getting_job.cpython-310.pyc
#      #      ### state_getting_job.cpython-311.pyc
#      #      ### state_idle.cpython-310.pyc
#      #      ### state_idle.cpython-311.pyc
#      #      ### state_machine.cpython-310.pyc
#      #      ### state_machine.cpython-311.pyc
#      #      ### state_updating_job.cpython-310.pyc
#      #      ### state_updating_job.cpython-311.pyc
#      #      ### topic_base.cpython-310.pyc
#      #      ### topic_base.cpython-311.pyc
#      ### config.py
#      ### effective_config.py
#      ### main.py
#      ### pki.py
#      ### pki_file.py
#      ### pki_hsm.py
#      ### pubsub.py
#      ### requirements.txt
#      ### scripts
#      #      ### run.cmd
#      ### state.py
#      ### state_committing_certificate.py
#      ### state_creating_certificate.py
#      ### state_getting_job.py
#      ### state_idle.py
#      ### state_machine.py
#      ### state_updating_job.py
#      ### topic_base.py
### aws.greengrass.labs.HomeAssistant
### 1.0.0
### home-assistant
### config
#      ### automations.yaml
#      ### configuration.yaml
#      ### groups.yaml
```

```
#   ### scenes.yaml
#   ### scripts.yaml
### docker-compose.yml
### install.py
### secret.py
```

17 directories, 67 files

Note that the `alts` directory links to the Nucleus `.jar` file in `packages/artifacts-unarchived`. For example:

```
user@machine:~$ sudo ls -l /greengrass/v2/alts/init
total 8
lrwxrwxrwx 1 root root 97 Jun 27 08:12 distro -> /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.6/aws.greengrass.nucleus
-rw-r--r-- 1 root root 16 Jun 27 07:07 launch.params
```

Therefore, `packages/artifacts-unarchived` must be included in your golden image.

recipes

The following example tree listing of `packages/recipes` shows how recipes are stored. As the listing indicates, recipes are stored with digests to help Greengrass determine whether it already has the correct files when it receives a deployment. This highly specific format makes it difficult to compose a golden image. Consequently, taking a snapshot of a golden device is the recommended method for creating a golden image.

```
user@machine:~$ sudo tree /greengrass/v2/packages/recipes
/greengrass/v2/packages/recipes
### 0ya1warrMfzlq5PUTv0gfH0ununru_xCLUFACECM_R0@2.3.7.metadata.json
### 0ya1warrMfzlq5PUTv0gfH0ununru_xCLUFACECM_R0@2.3.7.recipe.yaml
### 89r1-ak7xPauDt407EG03sSXVU08ysdHTk-YdF0NAAc@2.12.6.metadata.json
### 89r1-ak7xPauDt407EG03sSXVU08ysdHTk-YdF0NAAc@2.12.6.recipe.yaml
### VAZ-Grqe5g43y07UtasQ0R5jcQGILgPeRZQhVikLd9o@1.0.0.metadata.json
### VAZ-Grqe5g43y07UtasQ0R5jcQGILgPeRZQhVikLd9o@1.0.0.recipe.yaml
### ViMYPYs99-AzSt1gL2L2YD5P7sIN-yEhy23wWJK_JN8@1.0.8.metadata.json
### ViMYPYs99-AzSt1gL2L2YD5P7sIN-yEhy23wWJK_JN8@1.0.8.recipe.yaml
### _1hT2A6X0ZYtB_CfI_ZU0EMDV96DfQVksmZh2bbGYXg@1.0.19.metadata.json
### _1hT2A6X0ZYtB_CfI_ZU0EMDV96DfQVksmZh2bbGYXg@1.0.19.recipe.yaml
### gQWwM7MSL2k0sBADU9b0QJ1Qq08ZI3hqpBKT5Bv4Ijk@1.1.0.metadata.json
### gQWwM7MSL2k0sBADU9b0QJ1Qq08ZI3hqpBKT5Bv4Ijk@1.1.0.recipe.yaml
```

```
### j_j5Seyy01F0cIh95nBFy4HYf8P1kT-jW_nmV18ldbK@2.1.8.metadata.json
### j_j5Seyy01F0cIh95nBFy4HYf8P1kT-jW_nmV18ldbK@2.1.8.recipe.yaml
```

```
0 directories, 14 files
```

System service

If Greengrass is installed as a system service, as it commonly is for embedded Linux devices, the golden image must also include the directories that contain the systemd startup scripts.

Docker images

If your device uses AWS IoT Greengrass components that feature Docker images as artifacts, these artifacts sit outside the Greengrass directory tree. Therefore, you need to include the golden device's Docker image registry in your golden image. This registry is typically stored in `/var/lib/docker`.

Alternatively, you can use Docker commands to make a copy of the Docker images that are stored on your golden device, and then load those Docker images onto each device on your manufacturing line. In general, this method is slower and becomes less scalable as the number of Docker images increases.

Secrets

If your devices use the [secret manager component](#) to synchronize secrets from [AWS Secrets Manager](#), these secrets are stored in the `config/config.tlog` file in the Greengrass directory tree of your golden device. For example:

```
{
  "TS": 1718878001465,
  "TP": ["services", "aws.greengrass.SecretManager", "runtime", "secretResponse"],
  "W": "changed",
  "V": {
    "\secrets\": [
      {
        "\arn\": "\arn:aws:secretsmanager:us-east-1:111122223333:secret:greengrass-home-assistant-KIzJfZ",
        "\name\": "\greengrass-home-assistant",
        "\versionId\": "\8e481177-9250-4458-9f1f-3690d28e4ae9",
        "\encryptedSecretString\": "\AgV4j+We ... A7QjdE1w=="
```

```

        \ "versionStages\":[\ "AWSCURRENT\"],
        \ "createdDate\":"1660648425915
      }
    ]
  }
}

```

These secrets are also stored in the corresponding `config/effectiveConfig.yaml` file:

```

aws.greengrass.SecretManager:
  componentType: "PLUGIN"
  configuration:
    cloudSecrets:
      - arn: "arn:aws:secretsmanager:us-east-1:111122223333:secret:greengrass-home-
assistant-KIzJfZ"
    dependencies:
      - "aws.greengrass.Nucleus:SOFT"
    lifecycle: {}
    runtime:
      secretResponse: "{\ "secrets\":[{\ "arn\":"arn:aws:secretsmanager:us-
east-1:111122223333:secret:greengrass-home-assistant-KIzJfZ"\
        ,\ "name\":"greengrass-home-assistant",\ "versionId\":"
        \ "8e481177-9250-4458-9f1f-3690d28e4ae9"\
        ,\ "encryptedSecretString\":"AgV4Rpc9 ... MYeVALYQ=="\
        ,\ "versionStages\":[\ "AWSCURRENT"],\ "createdDate\":"1660648425915}}}"
    version: "2.1.8"

```

Even if you include the `config` directory in your golden image, it's important to remember that the Greengrass [secret manager component encrypts the secret with the golden device's private key](#). Because each device has a unique private key, keys that were encrypted by your golden device can't be decrypted by your production devices.

For this reason, we recommend that you remove encrypted secrets from the golden image to prevent your production devices from incorrectly decrypting golden device secrets. Your application components should function adequately, or at least fail gracefully, when secrets aren't present on disk, before a device has its first communication with the cloud.

When secrets are a hard requirement

If your organization requires your production devices to be populated with secrets during manufacture, your production line needs a script or program that replicates the behavior of the

secret manager component, to populate secrets on each production device. We don't recommend this approach because of its complexity and the possibility that your secrets might briefly be held in cleartext on your production programming station.

Golden image methods

At a high level, you can choose between two methods for creating a golden image that includes the Greengrass directory tree:

- Take a snapshot of the entire Greengrass directory tree on the golden device.
- Take only a partial snapshot of the Greengrass directory tree on the golden device.

The following table summarizes the key differences between these methods.

	Full Greengrass snapshot	Partial Greengrass snapshot
Greengrass directories included	All Greengrass directories (optionally excluding empty directories or those with only ephemeral data, as explained in the previous section).	alts, packages, and plugins directories only.
Advantage	Production devices are in a fully working state immediately.	Supports simple and robust configuration of each production device.
Disadvantage	Requires more complicated configuration of each production device.	Production devices must receive a cloud deployment to enter a fully working state.

Full Greengrass snapshot

By including a full snapshot of the Greengrass directory tree in your golden image, you retain the history and context of the deployments and component configuration that were applied to your golden device. This means that the devices produced from this image also have this history and context. Consequently, your production devices can boot up and run all application components, and each device can be in a fully working state immediately.

The complication with this method is that the history and context can make it more difficult to personalize each production device with unique configuration. These configuration steps are detailed in the [next section](#).

Limitations of the fully working state:

Although your application components will run, they need to gracefully deal with the following limitations:

- Your device will not synchronize any secrets to disk until it contacts the cloud.
- Your device will not have valid AWS Identity and Access Management (IAM) credentials through [token exchange](#) until it contacts the cloud.
- If your device uses [fleet provisioning](#) or [just-in-time provisioning](#), it will not connect to AWS IoT Core until provisioning is complete.

Partial Greengrass snapshot

When you use the partial Greengrass snapshot method, the golden image includes only the `alts`, `packages`, and `plugins` directories. The `config` directory is excluded, so the history and context of deployments and configuration are omitted.

The chief advantage of this method is that the unique configuration steps required for each production device are simple and robust. These configuration steps are detailed in the [next section](#).

The chief disadvantage is that your application components will not be in a running state until your production devices receive a cloud deployment. This might not meet all your requirements. However, because the component artifacts are present on disk, they will not be re-downloaded when your devices receive the cloud deployment, if the cloud deployment still matches the artifacts on disk. Consequently, the cloud deployment is relatively quick and will not incur large data charges on a cellular network.

Golden image procedure

The following sections describe the procedure for creating a golden image and writing that image to each device on the production line.

Create your golden device

Step 1: Install and provision Greengrass.

[Install the Greengrass edge runtime](#) on your golden device as you typically would, and provision it by using your preferred [provisioning method](#).

Step 2: Deploy your components.

From the cloud, deploy your application components and configuration to your golden device, as you typically would.

Read the golden image from your golden device

You can now snapshot your golden device to create a golden image.

Step 1: Shut down Greengrass.

Shut down the Greengrass service on your golden device. For example:

```
sudo systemctl stop greengrass.service
```

Step 2: Snapshot the Greengrass file tree.

Depending on your preferred golden method, take a full or partial snapshot of the Greengrass file tree, and include it in your golden image.

Step 3: Copy your Docker images.

If your components use Docker images as artifacts, take a snapshot of the Docker image registry and include it in your golden image.

Alternatively, you can use `docker save` to create an archive of each Docker image, and copy these from the golden device. For example:

```
user@machine:~/environment $ docker images
REPOSITORY              TAG          IMAGE ID          CREATED          SIZE
homeassistant/home-assistant  2024.6      2b4c9de733f1     9 days ago     1.6GB
user@machine:~/environment $ docker save homeassistant/home-assistant | gzip > home-
assistant.tar.gz
```

In general, this approach is slower than adding the Docker image registry to your golden image.

Write the golden image to your production devices

You now need to write the golden image to each of your production devices that come through your manufacturing line. The following steps would typically be automated by using scripting.

Step 1: Write the image.

Write your golden image to the production device. Make sure that it includes the full or partial snapshot of the Greengrass directory tree and the Docker image registry, as applicable.

Step 2: Load Docker images.

If you elected to use `docker save` to copy Docker image artifacts from your golden device, use `docker load` to load these images onto your production device. For example:

```
user@machine:~/environment $ docker load < home-assistant.tar.gz
user@machine:~/environment $ docker images
REPOSITORY              TAG          IMAGE ID          CREATED          SIZE
homeassistant/home-assistant  2024.6      2b4c9de733f1     9 days ago     1.6GB
```

In general, this approach is slower than adding the Docker image registry to your golden image.

Step 3: Add credentials.

If your device doesn't use [fleet provisioning](#), install the unique X.509 certificate and private key for the production device, replacing the golden device credentials. The exact method of achieving this depends on whether you use a [hardware security module \(HSM\)](#) and your particular public key infrastructure (PKI).

Configure your production devices

After you write the golden image, you must configure each production device. This configuration depends on whether you use the full or partial Greengrass snapshot method. The steps should be automated.

Full Greengrass snapshot method

Important

Follow these steps only if you're using the **full** Greengrass snapshot method.

Step 1: Edit the configuration.

Edit the `config/effectiveConfig.yaml` file. For example:

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

Step 2: Set the thing name.

The actions required to set the thing name of the new production device depend on whether you're using fleet provisioning. Make the correct modifications for your use case.

If you aren't using fleet provisioning, you can set the thing name of your production device by editing the `thingName` property to replace the golden device thing name with the production device thing name. For example:

```
system:
  certificateFilePath: "/greengrass/v2/thingCert.crt"
  privateKeyPath: "/greengrass/v2/privKey.key"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "ProductionDeviceThingName1"
```

If you are using fleet provisioning, you must first erase part of the golden device configuration to ensure that the [fleet provisioning plugin](#) will run on the production device. You can do this by replacing the golden device thing name in the `thingName` property with an empty string. For example:

```
system:
  certificateFilePath: "/greengrass/v2/thingCert.crt"
  privateKeyPath: "/greengrass/v2/privKey.key"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: ""
```

Additionally, you must add the fleet provisioning plugin configuration to the services property, and pass the production device thing name as a property within the templateParameters property. The following example assumes that the [fleet provisioning template](#) has a parameter named ThingName, and therefore there's a ThingName property within the templateParameters of the plugin configuration:

```
system:
  certificateFilePath: "/greengrass/v2/thingCert.crt"
  privateKeyPath: "/greengrass/v2/privKey.key"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: ""
services:
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-east-1"
      iotDataEndpoint: "<prefix>-ats.iot.us-east-1.amazonaws.com"
      iotCredentialEndpoint: "<prefix>.credentials.iot.us-east-1.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/rootCA.pem"
      templateParameters:
        ThingName: "ProductionDeviceThingName1"
```

Your use case might require some different fleet provisioning plugin configuration parameters than what's shown here. For more information, see [Configure the AWS IoT fleet provisioning plugin](#) in the AWS IoT Greengrass documentation.

Step 3: Update interpolated thing names.

If you have any components that use the `iot:thingName` [recipe variable](#) to [interpolate](#) the thing name, you will likely have access control statements in which the golden device thing name has been interpolated. For example, in the case of the [Certificate Rotator community component](#):

```
services:
  aws.greengrass.labs.CertificateRotator:
    componentType: "GENERIC"
    configuration:
      accessControl:
        aws.greengrass.ipc.mqttproxy:
          aws.greengrass.labs.CertificateRotator:mqttproxy:1:
            operations:
              - "aws.greengrass#PublishToIoTCore"
            policyDescription: "Allows access to publish to relevant topics"
            resources:
              - "$aws/things/GoldenDeviceThingName/jobs/+/get"
              - "$aws/things/GoldenDeviceThingName/jobs/+/update"
              - "awslabs/things/GoldenDeviceThingName/certificate/create"
              - "awslabs/things/GoldenDeviceThingName/certificate/commit"
          aws.greengrass.labs.CertificateRotator:mqttproxy:2:
            operations:
              - "aws.greengrass#SubscribeToIoTCore"
            policyDescription: "Allows access to subscribe to relevant topics"
            resources:
              - "$aws/things/GoldenDeviceThingName/jobs/notify-next"
              - "$aws/things/GoldenDeviceThingName/jobs/+/get/accepted"
              - "$aws/things/GoldenDeviceThingName/jobs/+/get/rejected"
              - "$aws/things/GoldenDeviceThingName/jobs/+/update/accepted"
              - "$aws/things/GoldenDeviceThingName/jobs/+/update/rejected"
              - "awslabs/things/GoldenDeviceThingName/certificate/create/accepted"
              - "awslabs/things/GoldenDeviceThingName/certificate/create/rejected"
              - "awslabs/things/GoldenDeviceThingName/certificate/commit/accepted"
              - "awslabs/things/GoldenDeviceThingName/certificate/commit/rejected"
            keyAlgorithm: "RSA-2048"
            signingAlgorithm: "SHA256WITHRSA"
```

In all such cases, you must replace all instances of the golden device thing name with your production device thing name.

Step 4: Remove golden device secrets.

Examine `effectiveConfig.yaml` for any encrypted secrets that were stored by your golden device. These will be stored within the secret manager component configuration. For example:

```
aws.greengrass.SecretManager:
  componentType: "PLUGIN"
  configuration:
    cloudSecrets:
      - arn: "arn:aws:secretsmanager:us-east-1:111122223333:secret:greengrass-home-
assistant-KIzJfZ"
    dependencies:
      - "aws.greengrass.Nucleus:SOFT"
    lifecycle: {}
    runtime:
      secretResponse: "{\"secrets\": [{\"arn\": \"arn:aws:secretsmanager:us-
east-1:111122223333:secret:greengrass-home-assistant-KIzJfZ\",
      \"name\": \"greengrass-home-assistant\", \"versionId\":
      \"8e481177-9250-4458-9f1f-3690d28e4ae9\",
      \"encryptedSecretString\": \"AgV4Rpc9 ... MYeVALYQ==\",
      \"versionStages\": [\"AWSCURRENT\"], \"createdDate\": 1660648425915}]}"
    version: "2.1.8"
```

You should delete this secret to ensure that your production device will not attempt to decrypt a secret that was encrypted by the golden device. Delete the `secretResponse` property. For example:

```
aws.greengrass.SecretManager:
  componentType: "PLUGIN"
  configuration:
    cloudSecrets:
      - arn: "arn:aws:secretsmanager:us-east-1:111122223333:secret:greengrass-home-
assistant-KIzJfZ"
    dependencies:
      - "aws.greengrass.Nucleus:SOFT"
    lifecycle: {}
    runtime:
      version: "2.1.8"
```

Step 5: Apply the altered configuration.

Apply the altered configuration, to update `config/config.tlog`. For example:

```
sudo java -Droot="/greengrass/v2" \
```

```
-jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \  
--start false \  
--init-config /greengrass/v2/config/effectiveConfig.yaml
```

Partial Greengrass snapshot method

Configuration of your production devices is trivial if you use the partial Greengrass snapshot method. The procedure is identical to the normal bootstrap partial configuration procedure when you [install Greengrass](#), except that the initial configuration is passed to the Greengrass runtime that's already installed.

Important

Follow these steps only if you're using the **partial** Greengrass snapshot method.

Step 1: Create the configuration.

The required partial configuration depends on whether you're using fleet provisioning. Be sure to make the correct modifications for your use case.

If you aren't using fleet provisioning, you can create a partial configuration file that doesn't include any configuration for the fleet provisioning plugin. Set the `thingName` property to your production device thing name. For example:

```
system:  
  certificateFilePath: "/greengrass/v2/device.pem.crt"  
  privateKeyPath: "/greengrass/v2/private.pem.key"  
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"  
  rootpath: "/greengrass/v2"  
  thingName: "ProductionDeviceThingName1"  
services:  
  aws.greengrass.Nucleus:  
    componentType: "NUCLEUS"  
    version: "2.12.6"  
    configuration:  
      awsRegion: "us-east-1"  
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"  
      iotDataEndpoint: "<prefix>-ats.iot.us-east-1.amazonaws.com"  
      iotCredEndpoint: "<prefix>.credentials.iot.us-east-1.amazonaws.com"
```

If you are using fleet provisioning, the partial configuration should include configuration for the [fleet provisioning plugin](#). This configuration should pass the production device thing name as a property within the `templateParameters` property. The following example assumes that the [fleet provisioning template](#) has a parameter named `ThingName`, and therefore there's a `ThingName` property within the `templateParameters` of the plugin configuration:

```
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-east-1"
      iotDataEndpoint: "<prefix>-ats.iot.us-east-1.amazonaws.com"
      iotCredentialEndpoint: "<prefix>.credentials.iot.us-east-1.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "ProductionDeviceThingName1"
```

Your use case might require different fleet provisioning plugin configuration parameters from what's shown here. For more information, see [Configure the AWS IoT fleet provisioning plugin](#) in the AWS IoT Greengrass documentation.

Step 2: Apply the configuration.

Apply the configuration to create `config/config.tlog`. For example:

```
sudo java -Droot="/greengrass/v2" \
-jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
--start false \
--init-config config.yaml
```

Conclusion

This guide provided you with the knowledge to harness golden images to scale your manufacturing processes. We recommend that you select your preferred [provisioning method](#) for your devices carefully, and consider whether your hardware design will use a [hardware security module \(HSM\)](#) for high security posture. These choices affect how you implement your production line and subsequent field deployments.

To send us feedback about this guide, use the [Provide feedback link](#) on this page. If you have any questions, please ask them at [repost.aws](#).

Resources

AWS IoT Greengrass Developer Guide

- [Install the AWS IoT Greengrass Core software](#)
- [Install AWS IoT Greengrass Core software with AWS IoT fleet provisioning](#)
- [Hardware security integration](#)
- [Recipe variables](#)
- [Interpolate component configuration](#)
- [Secret manager component](#)
- [Deploy AWS IoT Greengrass components to devices](#)

AWS IoT Core Developer Guide

- [Device provisioning](#)

Whitepapers

- [Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core](#)

Support

- [repost.aws](#)

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an [RSS feed](#).

Change	Description	Date
Initial publication	—	October 23, 2024

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- **Refactor/re-architect** – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- **Replatform (lift and reshape)** – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- **Repurchase (drop and shop)** – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- **Rehost (lift and shift)** – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- **Relocate (hypervisor-level lift and shift)** – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- **Retain (revisit)** – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- **Retire** – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See [attribute-based access control](#).

abstracted services

See [managed services](#).

ACID

See [atomicity, consistency, isolation, durability](#).

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than [active-passive migration](#).

active-passive migration

A database migration method in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include SUM and MAX.

AI

See [artificial intelligence](#).

AI Ops

See [artificial intelligence operations](#).

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to [the portfolio discovery and analysis process](#) and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see [What is Artificial Intelligence?](#)

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the [operations integration guide](#).

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see [ABAC for AWS](#) in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the [AWS CAF website](#) and the [AWS CAF whitepaper](#).

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A [bot](#) that is intended to disrupt or cause harm to individuals or organizations.

BCP

See [business continuity planning](#).

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see [Data in a behavior graph](#) in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also [endianness](#).

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as "Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of [bots](#) that are infected by [malware](#) and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see [About branches](#) (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the [Implement break-glass procedures](#) indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and [greenfield](#) strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the [Organized around business capabilities](#) section of the [Running containerized microservices on AWS](#) whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See [AWS Cloud Adoption Framework](#).

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See [Cloud Center of Excellence](#).

CDC

See [change data capture](#).

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use [AWS Fault Injection Service \(AWS FIS\)](#) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See [continuous integration and continuous delivery](#).

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the [CCoE posts](#) on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to [edge computing](#) technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see [Building your Cloud Operating Model](#).

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post [The Journey Toward Cloud-First & the Stages of Adoption](#) on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the [migration readiness guide](#).

CMDB

See [configuration management database](#).

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of [AI](#) that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see [Conformance packs](#) in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see [Benefits of continuous delivery](#). CD can also stand for *continuous deployment*. For more information, see [Continuous Delivery vs. Continuous Deployment](#).

CV

See [computer vision](#).

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see [Data classification](#).

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see [Building a data perimeter on AWS](#).

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See [database definition language](#).

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see [Services that work with AWS Organizations](#) in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See [environment](#).

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see [Detective controls](#) in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a [star schema](#), a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a [disaster](#). For more information, see [Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#) in the AWS Well-Architected Framework.

DML

See [database manipulation language](#).

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

DR

See [disaster recovery](#).

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to [detect drift in system resources](#), or you can use AWS Control Tower to [detect changes in your landing zone](#) that might affect compliance with governance requirements.

DVSM

See [development value stream mapping](#).

E

EDA

See [exploratory data analysis](#).

EDI

See [electronic data interchange](#).

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with [cloud computing](#), edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see [What is Electronic Data Interchange](#).

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See [service endpoint](#).

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see [Create an endpoint service](#) in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, [MES](#), and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see [Envelope encryption](#) in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the [program implementation guide](#).

ERP

See [enterprise resource planning](#).

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a [star schema](#). It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see [AWS Fault Isolation Boundaries](#).

feature branch

See [branch](#).

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see [Machine learning model interpretability with AWS](#).

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the "2021-05-27 00:15:37" date into "2021", "May", "Thu", and "15", you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an [LLM](#) with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also [zero-shot prompting](#).

FGAC

See [fine-grained access control](#).

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through [change data capture](#) to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See [foundation model](#).

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see [What are Foundation Models](#).

G

generative AI

A subset of [AI](#) models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see [What is Generative AI](#).

geo blocking

See [geographic restrictions](#).

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see [Restricting the geographic distribution of your content](#) in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the [trunk-based workflow](#) is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction

of compatibility with existing infrastructure, also known as [brownfield](#). If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See [high availability](#).

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. [AWS provides AWS SCT](#) that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a [machine learning](#) model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See [infrastructure as code](#).

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See [Industrial Internet of Things](#).

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than [mutable infrastructure](#). For more information, see the [Deploy using immutable infrastructure](#) best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by [Klaus Schwab](#) in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see [Building an industrial Internet of Things \(IIoT\) digital transformation strategy](#).

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see [What is IoT?](#)

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see [Machine learning model interpretability with AWS](#).

IoT

See [Internet of Things](#).

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the [operations integration guide](#).

ITIL

See [IT information library](#).

ITSM

See [IT service management](#).

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see [Setting up a secure and scalable multi-account AWS environment](#).

large language model (LLM)

A deep learning [AI](#) model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see [What are LLMs](#).

large migration

A migration of 300 or more servers.

LBAC

See [label-based access control](#).

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see [Apply least-privilege permissions](#) in the IAM documentation.

lift and shift

See [7 Rs](#).

little-endian system

A system that stores the least significant byte first. See also [endianness](#).

LLM

See [large language model](#).

lower environments

See [environment](#).

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see [Machine Learning](#).

main branch

See [branch](#).

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See [Migration Acceleration Program](#).

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see [Building mechanisms](#) in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See [manufacturing execution system](#).

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the [publish/subscribe](#) pattern, for resource-constrained [IoT](#) devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see [Integrating microservices by using AWS serverless services](#).

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed,

and scaled to meet demand for specific functions of an application. For more information, see [Implementing microservices on AWS](#).

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the [AWS migration strategy](#).

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the [discussion of migration factories](#) and the [Cloud Migration Factory guide](#) in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO

comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The [MPA tool](#) (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the [migration readiness guide](#). MRA is the first phase of the [AWS migration strategy](#).

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the [7 Rs](#) entry in this glossary and see [Mobilize your organization to accelerate large-scale migrations](#).

ML

See [machine learning](#).

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see [Strategy for modernizing applications in the AWS Cloud](#).

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see [Evaluating modernization readiness for applications in the AWS Cloud](#).

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can

use a microservices architecture. For more information, see [Decomposing monoliths into microservices](#).

MPA

See [Migration Portfolio Assessment](#).

MQTT

See [Message Queuing Telemetry Transport](#).

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of [immutable infrastructure](#) as a best practice.

O

OAC

See [origin access control](#).

OAI

See [origin access identity](#).

OCM

See [organizational change management](#).

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See [operations integration](#).

OLA

See [operational-level agreement](#).

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See [Open Process Communications - Unified Architecture](#).

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see [Operational Readiness Reviews \(ORR\)](#) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for [Industry 4.0](#) transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the [operations integration guide](#).

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the

organization and tracks the activity in each account. For more information, see [Creating a trail for an organization](#) in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the [OCM guide](#).

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic PUT and DELETE requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also [OAC](#), which provides more granular and enhanced access control.

ORR

See [operational readiness review](#).

OT

See [operational technology](#).

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The [AWS Security Reference Architecture](#) recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see [Permissions boundaries](#) in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See [personally identifiable information](#).

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See [programmable logic controller](#).

PLM

See [product lifecycle management](#).

policy

An object that can define permissions (see [identity-based policy](#)), specify access conditions (see [resource-based policy](#)), or define the maximum permissions for all accounts in an organization in AWS Organizations (see [service control policy](#)).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store

best adapted to their requirements. For more information, see [Enabling data persistence in microservices](#).

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see [Evaluating migration readiness](#).

predicate

A query condition that returns true or false, commonly located in a WHERE clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see [Preventative controls](#) in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in [Roles terms and concepts](#) in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see [Working with private hosted zones](#) in the Route 53 documentation.

proactive control

A [security control](#) designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the [Controls reference guide](#) in the

AWS Control Tower documentation and see [Proactive controls](#) in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See [environment](#).

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one [LLM](#) prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based [MES](#), a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

RAG

See [Retrieval Augmented Generation](#).

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See [responsible, accountable, consulted, informed \(RACI\)](#).

RCAC

See [row and column access control](#).

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See [7 Rs](#).

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See [7 Rs](#).

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see [Specify which AWS Regions your account can use](#).

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See [7 Rs](#).

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See [7 Rs](#).

replatform

See [7 Rs](#).

repurchase

See [7 Rs](#).

resiliency

An application's ability to resist or recover from disruptions. [High availability](#) and [disaster recovery](#) are common considerations when planning for resiliency in the AWS Cloud. For more information, see [AWS Cloud Resilience](#).

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see [Responsive controls](#) in *Implementing security controls on AWS*.

retain

See [7 Rs](#).

retire

See [7 Rs](#).

Retrieval Augmented Generation (RAG)

A [generative AI](#) technology in which an [LLM](#) references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see [What is RAG](#).

rotation

The process of periodically updating a [secret](#) to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See [recovery point objective](#).

RTO

See [recovery time objective](#).

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see [About SAML 2.0-based federation](#) in the IAM documentation.

SCADA

See [supervisory control and data acquisition](#).

SCP

See [service control policy](#).

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see [What's in a Secrets Manager secret?](#) in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: [preventative](#), [detective](#), [responsive](#), and [proactive](#).

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as [detective](#) or [responsive](#) security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see [Service control policies](#) in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see [AWS service endpoints](#) in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a [service-level indicator](#).

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see [Shared responsibility model](#).

SIEM

See [security information and event management system](#).

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See [service-level agreement](#).

SLI

See [service-level indicator](#).

SLO

See [service-level objective](#).

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid

innovation. For more information, see [Phased approach to modernizing applications in the AWS Cloud](#).

SPOF

See [single point of failure](#).

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a [data warehouse](#) or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was [introduced by Martin Fowler](#) as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see [Modernizing legacy Microsoft ASP.NET \(ASMX\) web services incrementally by using containers and Amazon API Gateway](#).

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use [Amazon CloudWatch Synthetics](#) to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an [LLM](#) to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see [Tagging your AWS resources](#).

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See [environment](#).

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see [What is a transit gateway](#) in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see [Using AWS Organizations with other AWS services](#) in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the [Quantifying uncertainty in deep learning systems](#) guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See [environment](#).

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see [What is VPC peering](#) in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See [write once, read many](#).

WQF

See [AWS Workload Qualification Framework](#).

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered [immutable](#).

Z

zero-day exploit

An attack, typically malware, that takes advantage of a [zero-day vulnerability](#).

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an [LLM](#) with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also [few-shot prompting](#).

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.