Cloud Deployment Guide

/ ForgeRock Identity Platform 6.5

Latest update: 6.5.2

David Goldsmith
Shankar Raman

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Provides guidance for deploying the ForgeRock Identity Platform in the cloud.

Table of Contents

Preface

The *Cloud Deployment Guide* provides instructions to help you deploy ForgeRock Identity Platform™ in the cloud, and then keep it up and running smoothly. The guide contains information about creating base Docker images, monitoring, securing, backing up, and restoring the CDM.

Before You Begin

Before deploying the ForgeRock Identity Platform on Kubernetes, read the important information in Start Here.

About ForgeRock Identity Platform Software

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

The platform includes the following components:

- ForgeRock® Access Management (AM)
- ForgeRock® Identity Management (IDM)
- ForgeRock® Directory Services (DS)
- ForgeRock® Identity Gateway (IG)

About This Guide

This *ForgeRock Cloud Deployment Guide* supplements the *ForgeRock Cloud Deployment Model (CDM) Cookbooks*. The cookbooks provide steps for a simplified and repeatable deployment process:

1. To get started, see the cookbook for your cloud platform:
    - Cloud Deployment Model Cookbook for GKE
    - Cloud Deployment Model Cookbook for Amazon EKS

- Cloud Deployment Model Cookbook for AKS

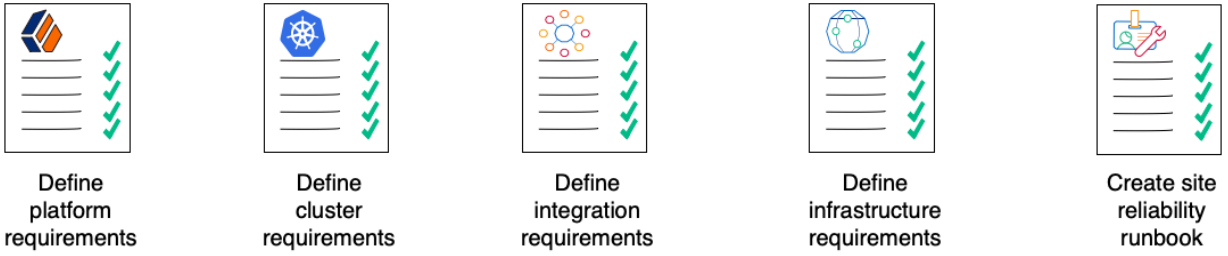
2. After you've deployed the CDM, validate the base deployment against these criteria:
   - The Kubernetes cluster and pods are up and running.
   - DS, AM, IDM, and IG are installed and running. You can access each ForgeRock component.
   - DS is provisioned with user entries. Replication and failover work as expected.
   - Monitoring tools are installed and running. You can access a monitoring console for DS, AM, IDM, and IG.
   - The benchmarking tests in the cookbook run without errors.
   - Your benchmarking test results are similar to the benchmarks reported in this cookbook.

After you've validated your CDM deployment and done some exploration, you're ready to work with a project team to plan your production deployment. You'll need a team with expertise in the ForgeRock Identity Platform, in your cloud provider, and in Kubernetes on your cloud provider. We strongly recommend that you engage a ForgeRock technical consultant or partner to assist you with deploying the platform in production.

You'll perform these major activities:

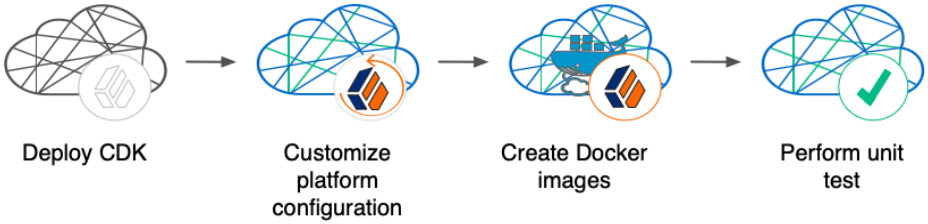
In the *project planning* activity, ForgeRock Identity Platform experts and cloud technology experts define the requirements for your production deployment. Requirements might include integrating systems, such as identity databases and applications, with the platform, and specifying deployment infrastructure requirements, such as backup, system monitoring, Git repository management, CI/CD, quality assurance, security, and load testing.

Project Planning Overview



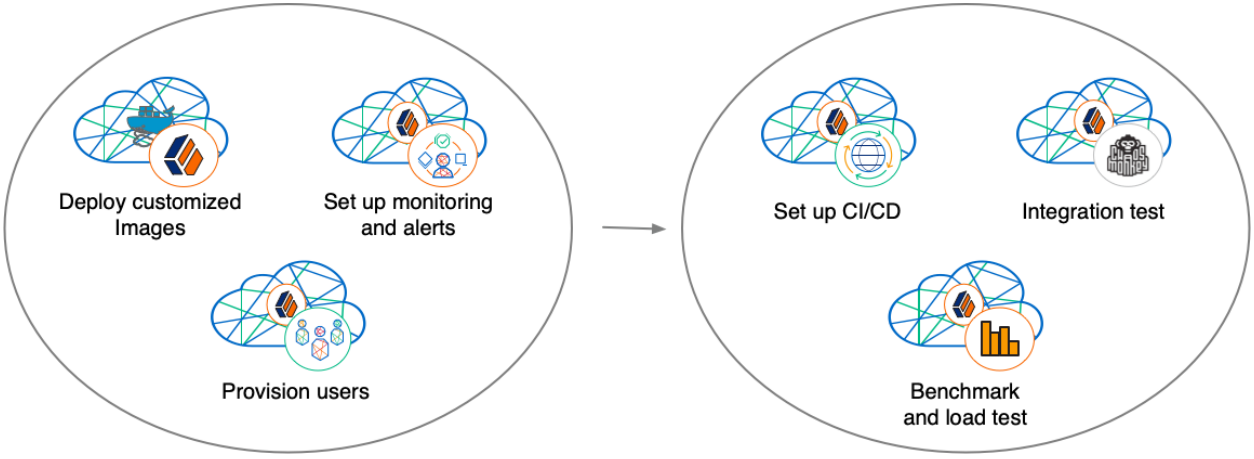
In the *platform configuration* activity, ForgeRock Identity Platform experts configure AM and IDM using the CDK, and build custom Docker images for the ForgeRock Identity Platform. The *DevOps Guides* (For Minikube | For Shared Clusters) provide information about platform configuration tasks.

Platform Configuration Overview



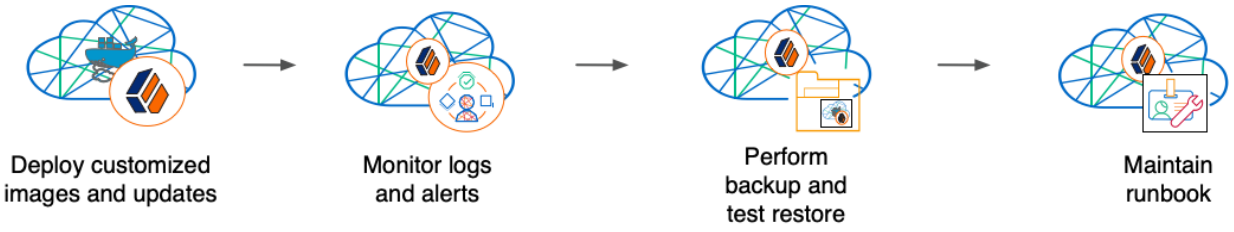
In the *cluster configuration* activity, cloud technology experts configure the Kubernetes cluster that will host the ForgeRock Identity Platform for optimal performance and reliability. Tasks include: modifying the cluster configuration to suit your business needs; setting up monitoring and alerts to track site health and performance; backing up configuration and user data for disaster preparedness; and securing your deployment. The Cloud Deployment Guide, and READMEs in the `forgeops` repository, provide information about cluster configuration.

Cluster Configuration Overview



In the *site reliability engineering* activity, site reliability engineers monitor the ForgeRock Identity Platform deployment, and keep the deployment up and running based on your business requirements. These might include use cases, service-level agreements, thresholds, and load test profiles. The Cloud Deployment Guide, and READMEs in the `forgeops` repository, provide information about site reliability.

Site Reliability Engineering Overview



ForgeRock Identity Platform deployment work is never really finished. It's likely you'll want to customize the configurations for your cluster to meet your service level requirements. You might also want to alter the ForgeRock Identity Platform configuration to provide your users with additional capabilities.

Even after you've modified the configurations, you'll need to continually monitor the system for changes in real-world performance and availability. You should also periodically review changes to your IAM needs. Understanding that your requirements can change rapidly, you'll always be prepared to make incremental changes to the system.

Use this guide to customize your ForgeRock Identity Platform site, to maintain the site, and to make changes to the system as needed.

Chapter 1

About the forgeops Repository

Use ForgeRock's `forgeops` repository to customize and deploy the ForgeRock Identity Platform on a Kubernetes cluster.

The repository contains files needed for customizing and deploying the ForgeRock Identity Platform on a Kubernetes cluster:

- Files used to build Docker images for the ForgeRock Identity Platform:
  - Dockerfiles
  - Scripts and configuration files incorporated into ForgeRock's Docker images
  - Canonical configuration profiles for the platform
- Kustomize bases and overlays
- Skaffold configuration files

In addition, the repository contains numerous utility scripts and sample files. The scripts and samples are useful for:

- Deploying ForgeRock's CDM quickly and easily.
- Exploring monitoring, alerts, and security customization.
- Modeling a CI/CD solution for cloud deployment.

See "Repository Reference" for information about the files in the repository, recommendations about how to work with them, and the support status for the files.

Repository Updates

Every several weeks, ForgeRock tags the `forgeops` repository. Each tag marks a new release of the repository, with enhancements and bug fixes.

The current release tag is `6.5-2020.06.24`.

When you start working with the `forgeops` repository:

1. Check out the current release tag and create a branch based on the tag. For example:

```
$ git checkout tags/6.5-2020.06.24 -b my-working-branch
```

2. Start watching the repository on GitHub, so that you'll receive notifications about new releases.

ForgeRock recommends that you regularly incorporate changes from newer release tags into your working branch. When GitHub notifies you that a new release is available:

1. Read the Release Notes, which contain information about the new release tag.
2. Use the **git fetch --tags** command to fetch new `forgeops` repository tags into your clone.
3. Create a new branch based on the new release tag:

   ```
   $ git checkout tags/new-release-tag -b releases/new-release-tag
   ```

4. Rebase the commits from the new branch into your working branch in your `forgeops` repository clone.

   If you like, you can remove the branch based on the new release tag when you've finished rebasing.

It's important to understand the impact of rebasing changes from a new release tag from ForgeRock into your branches. "Repository Reference" provides advice about which files in the `forgeops` repository to change, which files not to change, and what to look out for when you rebase. Follow the advice in "Repository Reference" to reduce merge conflicts, and to better understand how to resolve them when you rebase changes from a new release tag.

Caution

If your organization chooses to work with a `forgeops` repository fork as the upstream repository, the steps for initially obtaining and updating your repository clone will differ from the steps provided in the documentation. For more information, see "Working With a Fork", and ask your administrator to see if your organization is using a common upstream fork.

Repository Reference

Directories

`bin`

Example scripts you can use or model for a variety of deployment tasks.

Recommendation: Don't modify the files in this directory. If you want to add your own scripts to the `forgeops` repository, create a subdirectory under `bin`, and store your scripts there.

Support Status: Sample files. Not supported by ForgeRock.

`cicd`

Example files for working with several CI/CD systems, including Tekton and Google Cloud Build.

Recommendation: Don't modify the files in this directory. If you want to add your own CI/CD support files to the `forgeops` repository, create a subdirectory under `cicd`, and store your files there.

Support Status: Sample files. Not supported by ForgeRock.

`cluster`

Example scripts and artifacts that automate cluster creation.

Recommendation: When you deploy the CDM, you run **pulumi config set** commands to configure your cluster. The configuration is stored in YAML files in the `cluster/pulumi` subdirectory. Your modifications will cause merge conflicts when you rebase changes from a new release tag into your branches. When you resolve merge conflicts after a rebase, be sure to keep the modifications to the YAML files if you need them.

Don't make any additional modifications to the files in this directory.

When you're ready to automate cluster creation for your own deployment, create a subdirectory under `cluster`, and store your scripts and other files there.

Support Status: Sample files. Not supported by ForgeRock.

`config`

Configuration profiles, including the canonical `cdk` profile from ForgeRock and user-customized profiles.

Recommendation: Add your own profiles to this directory using the **config.sh** command. Do not modify the canonical `cdk` profile.

Support Status: Configuration profiles. Support is available from ForgeRock for the canonical `cdk` configuration profile, but not for customized configuration profiles you've added to the `config` directory.

`docker`

Dockerfiles and other support files needed to build Docker images for version 6.5 of the ForgeRock Identity Platform.

Recommendation: When customizing ForgeRock's default deployments, you'll need to add files under the `docker/6.5` directory. For example, to customize the AM WAR file, you might need to add plugin JAR files, user interface customization files, or image files.

If you only add new files under the `docker/6.5` directory, you should not encounter merge conflicts when you rebase changes from a new release tag into your branches. However, if you need to

modify any files from ForgeRock, you might encounter merge conflicts. Be sure to track changes you've made to any files in the `docker` directory, so that you're prepared to resolve merge conflicts after a rebase.

Support Status: Dockerfiles and other files needed to build Docker images for the ForgeRock Identity Platform. Support is available from ForgeRock.

`etc`

Files used to support several examples, including the CDM.

Recommendation: Don't modify the files in this directory (or its subdirectories). If you want to use CDM automated cluster creation as a model or starting point for your own automated cluster creation, then create your own subdirectories under `etc`, and copy the files you want to model into the subdirectories.

Support Status: Sample files. Not supported by ForgeRock.

`jenkins-scripts`

For ForgeRock internal use only. Do not modify or use.

`kustomize`

Artifacts for orchestrating the ForgeRock Identity Platform using Kustomize.

Recommendation: Common deployment customizations, such as changing the deployment namespace and providing a customized FQDN, require modifications to files in the `kustomize/6.5` directory. You'll probably change, at minimum, the `kustomize/6.5/all/kustomization.yaml` file.

Expect to encounter merge conflicts when you rebase changes from a new release tag into your branches. Be sure to track changes you've made to files in the `kustomize` directory, so that you're prepared to resolve merge conflicts after a rebase.

Support Status: Kustomize bases and overlays. Support is available from ForgeRock.

`legacy-docs`

Deprecated. Will be removed from the repository.

`samples`

Deprecated. Will be removed from the repository.

Files in the Top-Level Directory

`.gcloudignore`, **`.gitchangelog.rc`**, **`.gitignore`**

For ForgeRock internal use only. Do not modify.

`cloudbuild.yaml`

Example files for working with Google Cloud Build.

Recommendation: Don't modify this file. If you want to add your own Cloud Build configuration to the `forgeops` repository, use a different file name.

Support Status: Sample file. Not supported by ForgeRock.

`LICENSE`

Software license for artifacts in the `forgeops` repository. Do not modify.

`Makefile`

For ForgeRock internal use only. Do not modify.

`notifications.json`

For ForgeRock internal use only. Do not modify.

`README.md`

The top-level `forgeops` repository README file. Do not modify.

`skaffold.yaml`

Early availability version of the declarative configuration for running Skaffold to deploy the next version of the ForgeRock Identity Platform.

For early availability use only.

Support Status: Early availability file. Not supported by ForgeRock.

`skaffold-6.5.yaml`

The declarative configuration for running Skaffold to deploy version 6.5 of the ForgeRock Identity Platform.

Recommendation: If you need to customize the `skaffold-6.5.yaml` file, you might encounter merge conflicts when you rebase changes from a new release tag into your branches. Be sure to track changes you've made to this file, so that you're prepared to resolve merge conflicts after a rebase.

Support Status: Skaffold configuration file. Support is available from ForgeRock.

Working With a Fork

For the simplest use cases—a single user in an organization installing the CDK or CDM for a proof of concept, or for exploring of the platform—cloning ForgeRock's public `forgeops` repository from GitHub provides a quick and adequate way to access the repository.

If, however, your use case is more complex, you might want to fork the `forgeops` repository, and use the fork as your common upstream repository. For example:

- Multiple users in your organization need to access a common version of the repository and share changes made by other users.
- Your organization plans to incorporate `forgeops` repository changes from ForgeRock.
- Your organization wants to use pull requests when making repository updates.

If you've forked the `forgeops` repository:

- You'll need to synchronize your fork with ForgeRock's public repository on GitHub when ForgeRock releases a new release tag.
- Your users will need to clone your fork before they start working, instead of cloning the public `forgeops` repository on GitHub. Because procedures in the *DevOps Guides* and the *CDM Cookbooks* tell users to clone the public repository, you'll need to make sure your users follow different procedures to clone the forks instead.

Caution

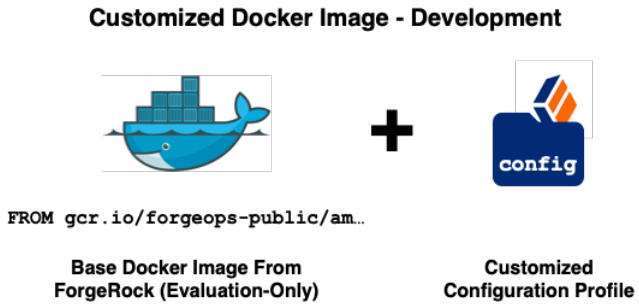
If your organization chooses to work with a `forgeops` repository fork as the upstream repository, the steps for initially obtaining and updating your repository clone will differ from the steps provided in the documentation. You'll need to let users know how to work with the fork as the upstream instead of following the steps in the documentation.

Chapter 2

Building Base Docker Images

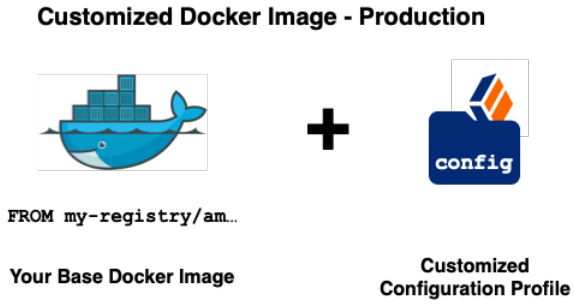
ForgeRock provides a set of unsupported, evaluation-only base images for the ForgeRock Identity Platform. These images are available in ForgeRock's public Docker registry.

Developers working with the CDK use the base images from ForgeRock to build customized Docker images for a fully-configured ForgeRock Identity Platform deployment:



Users working with the CDM also use the base images from ForgeRock to perform proof-of-concept deployments, and to benchmark the ForgeRock Identity Platform.

The base images from ForgeRock are evaluation-only. *They are unsupported for production use.* Because of this, you must build your own base images before you deploy in production:



This chapter tells you how to build your own base images, which you can deploy in production.

Obtaining ForgeRock Software Binary Files

The ForgeRock Dockerfiles expect ForgeRock software binary files to have specific names and locations within your `forgeops` repository clone.

The following procedure provides steps to obtain binary files with the ForgeRock software, rename the binary files, and copy the files to locations required by the Dockerfiles. Note that if ForgeRock releases updates to version 6.5, you must perform this procedure again to ensure that your Docker images incorporate the latest fixes and changes:

To Use ForgeRock Binary Files in Docker Images

1. Download the latest versions of the AM `.war` file, the IDM `.zip` file, the DS `.zip`, and the Amster `.zip` file. Optionally, you can also download the latest version of the IG `.war` file. Obtain these binary files from the ForgeRock BackStage download site.

2. Rename the binary files as follows:

Original File Name	New File Name
`AM-a.b.c.d.war`	`openam.war`
`Amster-a.b.c.d.zip`	`amster.zip`
`DJ-a.b.c.d.zip`	`opendj.zip`
`IDM-a.b.c.d.zip`	`openidm.zip`
(Optional) `IG-a.b.c.war`	`openig.war`

3. Copy the renamed binary files to the following locations in your clone of the `forgeops` repository:

Binary File	Location
`openam.war`	`/path/to/forgeops/docker/6.5/am-base/openam.war`
`amster.zip`	`/path/to/forgeops/docker/6.5/amster-base/amster.zip`
`opendj.zip`	`/path/to/forgeops/docker/6.5/ds-base/opendj.zip`
`openidm.zip`	`/path/to/forgeops/docker/6.5/idm-base/openidm.zip`
(Optional) `openig.war`	`/path/to/forgeops/docker/6.5/ig-base/openig.war`

Deploying Your Own Base Docker Images

Perform the following procedure to build base Docker images that you can use in production deployments of the ForgeRock Identity Platform. After you've built the base images, push them to your Docker registry:

To Deploy Your Own Base Docker Images

1. Change to the directory that contains content required to build Docker images:

```
$ cd /path/to/forgeops/docker/6.5
```

2. Build base Docker images for the ForgeRock Identity Platform:

   a. Build the AM base image:

```
$ docker build am-base --tag my-registry/am-base:6.5
Sending build context to Docker daemon  179.9MB
Step 1/12 : FROM tomcat:8-jdk8-adoptopenjdk-hotspot
 ---> 6175a3cb5c1f
Step 2/12 : RUN rm -fr "$CATALINA_HOME"/webapps/*
 ---> Using cache
 ---> 79e8d91e0774
Step 3/12 : COPY openam.war "$CATALINA_HOME"/webapps/am.war
 ---> Using cache
 ---> 1fdadf9cd6b8
. . .
```

   b. Build the Amster base image:

```
$ docker build amster-base --tag my-registry/amster-base:6.5
Sending build context to Docker daemon  34.44MB
Step 1/10 : FROM adoptopenjdk:8-jre-hotspot
 ---> 1ff0cbaf52a8
Step 2/10 : ADD amster.zip /var/tmp/
 ---> 32c21d07403b
Step 3/10 : RUN apt-get update && apt-get install -y unzip curl &&     useradd -ms /bin/bash
 forgerock --gid 0 --uid 11111 &&     unzip /var/tmp/amster.zip -d /opt/amster
 ---> Running in af78c2f77a3d
. . .
```

   c. Build the DS base image:

```
$ docker build ds-base --tag my-registry/ds-base:6.5
Step 1/30 : FROM krallin/ubuntu-tini:bionic as download_tini
bionic: Pulling from krallin/ubuntu-tini
6abc03819f3e: Pull complete
05731e63f211: Pull complete
0bd67c50d6be: Pull complete
d04b80bdaf6e: Pull complete
002bb203453b: Pull complete
Digest: sha256:fe681211bc1e1c55caa631a379189d728b0239a2bb580ed1c59f513dd2c28f05
Status: Downloaded newer image for krallin/ubuntu-tini:bionic
 ---> a45da59ab6f9
Step 2/30 : FROM adoptopenjdk/openjdk8:debian-slim as unpack
debian-slim: Pulling from adoptopenjdk/openjdk8
16ea0e8c8879: Pull complete
4a2acb47610b: Pull complete
2cf5a11ce7fe: Pull complete
1656c8f6586d: Pull complete
Digest: sha256:ec082dd0fb075ef2e582214017138faece53593fa492ac98345d09f7ac50bc74
Status: Downloaded newer image for adoptopenjdk/openjdk8:debian-slim
 ---> 7745ab206e33
Step 3/30 : RUN apt-get update && apt-get install -y unzip && apt-get clean all
 ---> Running in f8a9cef9e35a
 . . .
 Step 4/30 : COPY opendj.zip /var/tmp/opendj.zip
 ---> b52409818ffe
Step 5/30 : WORKDIR /var/tmp/bootstrap
 ---> Running in 399c6bd5fae0
 . . .
```

d. Build the IDM base image:

```
$ docker build idm-base --tag my-registry/idm-base:6.5
Sending build context to Docker daemon  167.1MB
Step 1/22 : FROM krallin/ubuntu-tini:bionic as download_tini
 ---> a45da59ab6f9
Step 2/22 : FROM adoptopenjdk/openjdk8:debian-slim as unpack
 ---> 7745ab206e33
Step 3/22 : RUN apt-get update && apt-get install -y unzip && apt-get clean all
 ---> Running in 124ede95c4e6
. . .
Step 4/22 : COPY openidm.zip /openidm.zip
 ---> 8d3a3b14b8c1
 . . .
```

e. (Optional) Build the IG base image:

```
$ docker build ig-base --tag my-registry/ig-base:6.5
Sending build context to Docker daemon  44.52MB
Step 1/10 : FROM tomcat:8.5-alpine
 ---> 8b8b1eb786b5
Step 2/10 : ENV OPENIG_BASE /var/openig
 ---> Running in 7a673f4ab0f4
Removing intermediate container 7a673f4ab0f4
 ---> 3f81dd933f82
Step 3/10 : ENV FORGEROCK_HOME /opt/forgerock
 ---> Running in 09592642a854
Removing intermediate container 09592642a854
 ---> 65689588ffd0
Step 4/10 : RUN rm -fr ${CATALINA_HOME}/webapps/*  && mkdir -p ${OPENIG_BASE}
 ---> Running in 2a176b050cc4
Removing intermediate container 2a176b050cc4
 ---> 29037956857a
Step 5/10 : ADD /openig.war "$CATALINA_HOME"/webapps/ROOT.war
 ---> 4cb9c62ab8cd
 . . .
```

3. Run the **docker images** to verify that you built the base images correctly:

```
$ docker images
REPOSITORY                      TAG      IMAGE ID        CREATED        SIZE
my-registry/am-base             6.5      d115125b1c3f    1 hour ago     739MB
my-registry/amster-base         6.5      d9e1c735f415    1 hour ago     462MB
my-registry/ds-base             6.5      ac8e8ab0fda6    1 hour ago     519MB
my-registry/idm-base            6.5      0cc1b7f70ce6    1 hour ago     943MB
my-registry/ig-base             6.5      9728c30c1829    1 hour ago     239MB
. . .
```

4. Push the new base Docker images to your Docker registry.

   See your registry provider documentation for detailed instructions. For most Docker registries, you run the **docker login** to log in to the registry. Then, you run the **docker push** command to push a Docker image to the registry.

   However, some Docker registries have different requirements. For example, to push Docker images to Google Container Registry, you use Google Cloud SDK commands instead of using the **docker push** command.

   Push the following images:

   - `my-registry/am-base:6.5`
   - `my-registry/amster-base:6.5`
   - `my-registry/ds-base:6.5`
   - `my-registry/idm-base:6.5`

   If you're deploying your own IG base image, also push the `my-registry/am-base:6.5` image.

Updating Developer Dockerfiles

After you've pushed your own base images to your Docker registry, update the Dockerfiles that your developers use when creating customized Docker images for the ForgeRock Identity Platform. The Dockerfiles can now reference your own base images instead of the evaluation-only images from ForgeRock.

To Update Developer Dockerfiles to Use Your Own Base Images

1. Update the AM Dockerfile:
   a. Change to the `/path/to/forgeops/docker/6.5/am` directory.
   b. Open the file, `Dockerfile`, in that directory.
   c. Change the line:
      ```
      FROM gcr.io/forgeops-public/am-base:6.5
      ```
      to:
      ```
      FROM my-registry/am-base:6.5
      ```
2. Make a similar change to the file, `/path/to/forgeops/docker/6.5/amster/Dockerfile`.
3. Make a similar change to the file, `/path/to/forgeops/docker/6.5/ds/Dockerfile`.
4. Make a similar change to the file, `/path/to/forgeops/docker/6.5/idm/Dockerfile`.
5. (Optional) Make a similar change to the file, `/path/to/forgeops/docker/6.5/ig/Dockerfile`.

You can now build customized Docker images for the ForgeRock Identity Platform based on your own Docker images and use them in production deployments.

Chapter 3
Monitoring Your Deployment

This chapter describes the CDM's monitoring architecture. It also covers common customizations you might perform to change the way monitoring, reporting, and sending alerts works in your environment.

About CDM Monitoring

The CDM uses Prometheus to monitor ForgeRock Identity Platform components and Kubernetes objects, Prometheus Alertmanager to send alert notifications, and Grafana to analyze metrics using dashboards.

Prometheus and Grafana are deployed when you run the `prometheus-deploy.sh` script. This script installs Helm charts from the prometheus-operator project into the `monitoring` namespace of a CDM cluster. These Helm charts deploy Kubernetes pods that run the Prometheus and Grafana services.

The following Prometheus and Grafana pods from the `prometheus-operator` project run in the `monitoring` namespace:

Pod	Description
`alertmanager-prometheus-operator-alertmanager-0`	Handles Prometheus alerts by grouping them together, filtering them, and then routing them to a receiver, such as a Slack channel.
`prometheus-operator-kube-state-metrics-...`	Generates Prometheus metrics for cluster node resources, such as CPU, memory, and disk usage. One pod is deployed for each CDM node.
`prometheus-operator-prometheus-node-exporter-...`	Generates Prometheus metrics for Kubernetes API objects, such as deployments and nodes.
`prometheus-operator-grafana-...`	Provides the Grafana service.
`prometheus-prometheus-operator-prometheus-0`	Provides the Prometheus service.

See the `prometheus-operator` Helm chart README file for more information about the pods in the preceding table.

In addition to the pods from the prometheus-operator project, the `import-dashboards-...` pod from the forgeops project runs after Grafana starts up. This pod imports Grafana dashboards from the ForgeRock Identity Platform and terminates after importing has completed.

To access CDM monitoring dashboards, see:

- "Monitoring the CDM" in the *Cloud Deployment Model Cookbook for GKE*.
- "Monitoring the CDM" in the *Cloud Deployment Model Cookbook for Amazon EKS*
- "Monitoring the CDM" in the *Cloud Deployment Model Cookbook for AKS*

Note

The CDM uses Prometheus and Grafana for monitoring, reporting, and sending alerts. If you prefer to use different tools, deploy infrastructure in Kubernetes to support those tools.

Prometheus and Grafana are evolving technologies. Descriptions of these technologies were accurate at the time of this writing, but might differ when you deploy them.

Importing Custom Grafana Dashboards

The CDM includes a set of Grafana dashboards. You can customize, export and import Grafana dashboards using the Grafana UI or HTTP API.

For information about importing custom Grafana dashboards, see the *Import Custom Grafana Dashboards* section in the Prometheus and Grafana Deployment README file in the `forgeops` repository.

Modifying the Prometheus Operator Configuration

The CDM's monitoring framework is based on the Prometheus Operator for Kubernetes project. The Prometheus Operator project provides monitoring definitions for Kubernetes services and deployment, and management of Prometheus instances.

When deployed, the Prometheus Operator watches for ServiceMonitor CRDs—Kubernetes Custom Resource Definitions. CRDs are Kubernetes class types that you can manage with the **kubectl** command. The ServiceMonitor CRDs define targets to be scraped.

In the CDM, the Prometheus Operator configuration is defined in the `prometheus-operator.yaml` file in the `forgeops` repository. To customize the CDM monitoring framework, change values in these files, following the examples documented in README files in the Prometheus Operator project on GitHub.

Configuring Additional Alerts

CDM alerts are defined in the `fr-alerts.yaml` file in the `forgeops` repository.

To configure additional alerts, see the *Configure Alerting Rules* section in the Prometheus and Grafana Deployment README file in the `forgeops` repository.

Chapter 4

Backing Up and Restoring Directory Data

This chapter describes the backup and restore functionality in CDM. It explains how to customize backup and restore operations in your deployment.

This chapter includes the following sections:

About Backup and Restore

The following diagram shows the backup topology of DS used in CDM.

Backup and Restore Overview

Following are some important backup and restore considerations, as shown in the diagram above.

- Three DS instances are deployed using Kubernetes stateful sets. Each stateful set is associated with a dedicated, persistent disk volume with a persistent volume claim (PVC). Each DS instance is a combined directory and replication server.
- A backup volume (shown as `bak` in the diagram) is mounted using a PVC on each DS pod, for storing backups of directory data. The mount point for the backup volume is the`/opt/opendj/bak` directory in each DS pod.
- The scripts necessary for performing backup and restore operations are available in the `scripts` folder of each directory server pod.

- Backups are scheduled in the DS pods.
- When enabled, the default backup schedule is:
  - An incremental backup is performed every hour.
  - A full backup is performed once a day.
- The backup schedule can be customized based on your volume of activity and recovery objectives.

Enabling CDM-Scheduled Backups

After you deploy the CDM, backups will *not* be made until you schedule them.

To schedule backups, run the **schedule-backup.sh** script in each DS pod to be backed up. The default backup schedule creates a full backup every day at 12:00 midnight, and an incremental backup every hour.

To start making backups using the default schedule, perform these steps:

- For the `ds-idrepo-0` pod:

```
$ kubectl exec ds-idrepo-0 -it scripts/schedule-backup.sh
```

- For the `ds-cts-0` instance:

```
$ kubectl exec ds-cts-0 -it scripts/schedule-backup.sh
```

Customizing the Backup Schedule

You can alter the backup schedule using the following procedure:

To Customize the Backup Schedule for a DS Instance

1. Log in to the DS instance using **kubectl exec** command. For example:

```
$ kubectl exec ds-idrepo-0 -it /bin/bash
```

2. Edit the `schedule-backup.sh` file in the `scripts` directory to match your required schedule, and change the following two lines in the `schedule-backup.sh` file to suit your schedule:
   - `FULL_CRON="0 0 * * *"`
   - `INCREMENTAL_CRON="0 * * * *"`

   For example the following two lines schedule a full backup every day at 3:00 AM, and an incremental backup every 30 minutes:

- `FULL_CRON="0 3 * * *"`
- `INCREMENTAL_CRON="*/30 * * * *"`

3. Log out of the DS instance and run the **schedule-backup.sh** script in the DS pod. For example, to schedule backup in *ds-idrepo-0* pod:

```
$ kubectl exec ds-idrepo-0 -it scripts/schedule-backup.sh
```

The **schedule-backup.sh** script stops any previously scheduled backup jobs before initiating the new schedule.

Performing User-Initiated Backups

In addition to scheduled backups, you might want to make occasional backups—for example, before applying security patches. Backups made outside of the scope of scheduled backups are called *user-initiated backups*.

The scripts necessary to perform user-initiated backups are available in the `scripts` folder of the Directory Services pods. For more information on backing up DS data, see the Backing Up Directory Data section of the *ForgeRock Directory Services Maintenance Guide*.

To Initiate a Full Backup

1. To make a full backup of the identity store, use the following **kubectl exec** command:

```
$ kubectl exec ds-idrepo-0 -it -- scripts/backup.sh --start 0
```

2. Verify that the backup you made is available on your local backup volume by using the **list-backup.sh** script:

```
$ kubectl exec ds-idrepo-0 -it -- scripts/list-backup.sh
```

Exporting User Data

Exporting directory data to LDIF is very useful to:

- Back up directory data
- Copy or move directory data from one directory to another
- Initialize a new directory server

Use the following steps to export user data with the **export-ldif.sh** command. For more information on exporting directory data, see Importing and Exporting Data in the *ForgeRock Directory Services Maintenance Guide*.

To Export User Data

1. Run the **export-ldif** command in the Directory Services pod. For example:

```
$ kubectl exec ds-idrepo-0 -it -- scripts/export-ldif.sh
```

2. Run the **ls** command in the DS pod and verify that the LDIF files have been created on the `backup` storage volume:

```
$ kubectl exec ds-idrepo-0 -it -- ls -l ./bak/default/ds-idrepo-0/
```

Initializing Directory Data From Binary Backup

When you deploy the platform, new directory instances are created with no data. If you want to restore a new platform environment after a disaster occurs, or when directory services are lost, or when you want to port a test environment data to production, you need to have directory instances created with the data.

To create PVCs and initialize directory instances with data from binary backup, use the **backup-loader.sh** script after configuring the cluster, but before deploying the platform:

To Create DS PVCs From a Previous Binary Backup

1. If the binary backup files are not stored locally, perform one of these tasks:
   - Copy the binary backup data to a predetermined directory on your local computer.
   - Mount the remote volume containing binary backup data on your local computer.
2. Change to the `/path/to/forgeops/bin` directory.
3. Run the **backup-loader.sh** script as follows[1]:

```
$ ./backup-loader.sh -n prod -r 3 \
 -s CTS-Disk-Size-in-GB \
 -z IDRepo-Size-in-GB \
 -b /path/to/CTS/backup \
 -u /path/to/IDRepo/backup
```

After the **backup-loader.sh** script has created the PVCs, and iniialized the DS, deploy the platform.

When you deploy DS, the entire DS environment is restored. To restore specific portions of the directory data without redeploying the DS environment, see "User-Initiated Restore".

[1] The sample command is for restoring both CTS and ID Repo directories. If you want to to restore only CTS instances, then you should not specify `-z` and `-u` options. Conversely, if you want to to restore only ID Repo instances, then you should not specify `-s` and `-b` options.

Note

You should note that **backup-loader.sh** is a sample script to show the task of deploying DS in the Kubernetes cluster using previously backed up data.

In a production environment involving voluminous data (a few terabytes in size), copy data directly from a cloud storage provider rather than porting data through your local computer.

User-Initiated Restore

To manually restore DS data, use a backup from the backup volume. First, run the **list-backup.sh** script to list the backups available on your local backup volume. Then, run the **restore** command in the `bin` directory to restore directory data. For more information about the **restore** command, see the ForgeRock Directory Services Maintenance Guide.

Consider following these best practices:

- Use a backup that is newer than the last replication purge.
- When you restore a DS replica using backups that are older than the purge delay, that replica will no longer be able to participate in replication. Reinitialize the replica to restore the replication topology.
- If the available backups are older than the purge delay, then initialize the DS replica from an up-to-date master instance. For more information on how to initialize a replica from a master instance, see Initializing Replicas section of the *ForgeRock Directory Services Server Configuration Guide*.

Chapter 5

Securing Your Deployment

This chapter describes options for securing your ForgeRock Identity Platform deployment.

Changing Default Secrets

The ForgeRock secrets generator randomly generates all secrets for AM, IDM, and DS services running in the CDK and the CDM. The secrets generator runs as a Kubernetes job before AM, IDM, and DS are deployed.

See the `forgeops-secrets` README file for more information about the secrets generator, including a list of which secrets it generates, and how to override default secrets.

Granting Access to Multiple Users (EKS Only)

It's common for team members to share the use of a cluster. To share a cluster with your team members, as the cluster owner, you must grant access to each user.

To Grant Users Access to an EKS Cluster

1. Get the ARNs and names of users who need access to your cluster.
2. Set the Kubernetes context to your Amazon EKS cluster.
3. Edit the authorization configuration map for the cluster using **kubectl edit** command:

   ```
   $ kubectl edit -n kube-system configmap/aws-auth
   ```

4. Under the `mapRoles` section, insert the `mapUser` section. An example is shown here with the following parameters:

   - The user ARN is `arn:aws:iam::012345678901:user/new.user`.
   - The user name registered in AWS is *new.user*.

```
...
 mapUsers: |
    - userarn: arn:aws:iam::012345678901:user/new.user
      username: new.user
      groups:
        - system:masters
...
```

5. For each additional user, insert the `- userarn:` entry in the `mapUsers:` section:

```
...
 mapUsers: |
    - userarn: arn:aws:iam::012345678901:user/new.user
      username: new.user
      groups:
        - system:masters
    - userarn: arn:aws:iam::901234567890:user/second.user
      username: second.user
      groups:
        - system:masters
...
```

6. Save the configuration map.

Controlling Access by Configuring a CIDR Block

When installing the ingress controller in production environments, you should consider configuring a CIDR block in the Helm chart for the ingress controller so that you restrict access to worker nodes from a specific IP address or a range of IP addresses.

To specify a range of IP addresses allowed to access resources controlled by the ingress controller, specify the `--set controller.service.loadBalancerSourceRanges="your IP range"` option when you install your ingress controller.

For example:

```
$ helm install --namespace nginx --name nginx \
 --set rbac.create=true \
 --set controller.publishService.enabled=true \
 --set controller.stats.enabled=true \
 --set controller.service.externalTrafficPolicy=Local \
 --set controller.service.type=LoadBalancer \
 --set controller.image.tag="0.21.0" \
 --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="nlb" \
 --set controller.service.loadBalancerSourceRanges="{81.0.0.0/8,3.56.113.4/32}" \
 stable/nginx-ingress
```

Securing Communication With ForgeRock Identity Platform Servers

The CDK and CDM enable secure communication with ForgeRock Identity Platform services using an SSL-enabled ingress controller. Incoming requests and outgoing responses are encrypted. SSL is terminated at the ingress controller.

You can configure communication with ForgeRock Identity Platform services using one of the following options:

- **Over HTTPS using a self-signed certificate**. Communication is encrypted, but users will receive warnings about insecure communication from some browsers.
- **Over HTTPS using a certificate with a trust chain that starts at a trusted root certificate**. Communication is encrypted, and users will not receive warnings from their browsers.
- **Over HTTPS using a dynamically obtained certificate from Let's Encrypt**. Communication is encrypted and users will not receive warnings from their browsers. A `cert-manager` pod installed in your Kubernetes cluster calls Let's Encrypt to obtain a certificate, and then automatically installs a Kubernetes secret.

You install a Helm chart from the `cert-manager` project to provision certificates. By default, the pod issues a self-signed certificate. You can also configure the pod to issue a certificate with a trust chain that begins at a trusted root certificate, or to dynamically obtain a certificate from Let's Encrypt.

Automating Certificate Management

In the CDM, certificate management is provided by the `cert-manager` add-on. The certificate manager deployed in CDM generates a self-signed certificate to secure CDM communication.

In your own deployment, you can specify a different certificate issuer or DNS challenge provider by changing values in the `ingress.yaml` file.

For more information about configuring certificate management, see the `cert-manager` documentation.

Appendix A. Getting Support

This appendix contains information about support options for the ForgeRock Cloud Developer's Kit, the ForgeRock Cloud Deployment Model, and the ForgeRock Identity Platform.

ForgeRock DevOps Support

ForgeRock has developed artifacts in the `forgeops` Git repository for the purpose of deploying the ForgeRock Identity Platform in the cloud. The companion ForgeRock DevOps documentation provides examples, including the ForgeRock Cloud Developer's Kit (CDK) and the ForgeRock Cloud Deployment Model (CDM), to help you get started.

These artifacts and documentation are provided on an "as is" basis. ForgeRock does not guarantee the individual success developers may have in implementing the code on their development platforms or in production configurations.

Commercial Support

ForgeRock provides commercial support for the following DevOps resources:

- Artifacts in the `forgeops` Git repository:
  - Files used to build Docker images for the ForgeRock Identity Platform:
    - Dockerfiles
    - Scripts and configuration files incorporated into ForgeRock's Docker images
    - Canonical configuration profiles for the platform

  - Kustomize bases and overlays
  - Skaffold configuration files
- ForgeRock DevOps guides.

ForgeRock provides commercial support for the ForgeRock Identity Platform. For supported components, containers, and Java versions, see the following:

- *ForgeRock Access Management Release Notes*
- *ForgeRock Identity Management Release Notes*
- *ForgeRock Directory Services Release Notes*
- *ForgeRock Identity Gateway Release Notes*

Support Limitations

ForgeRock provides no commercial support for the following:

- Artifacts other than Dockerfiles, Kustomize bases, Kustomize overlays, and Skaffold YAML configuration files in the `forgeops` Git repository. Examples include scripts, example configurations, and so forth.
- Non-ForgeRock infrastructure. Examples include Docker, Kubernetes, Google Cloud Platform, Amazon Web Services, and so forth.
- Non-ForgeRock software. Examples include Java, Apache Tomcat, NGINX, Apache HTTP Server, Certificate Manager, Prometheus, and so forth.
- Production deployments that use ForgeRock's evaluation-only Docker images. When deploying the ForgeRock Identity Platform using Docker images, you must build and use your own images for production deployments. For information about how to build your own Docker images for the ForgeRock Identity Platform, see "*Building Base Docker Images*".

Third-Party Kubernetes Services

ForgeRock supports deployments on Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (Amazon EKS), Microsoft Azure Kubernetes Service (AKS), and Red Hat OpenShift.

Red Hat OpenShift is a tested and supported platform using Kubernetes for deployment. ForgeRock uses OpenShift tools such as the OpenShift installer, as well as other representative environments such as Amazon AWS for the testing. We do not test using bare metal due to the many customer permutations of deployment and configuration that may exist, and therefore cannot guarantee that we have tested in the same way a customer chooses to deploy. We will make commercially reasonable efforts to provide first-line support for any reported issue. In the case we are unable to reproduce a

reported issue internally, we will request the customer engage OpenShift support to collaborate on problem identification and remediation. Customers deploying on OpenShift are expected to have a support contract in place with IBM/Red Hat that ensures support resources can be engaged if this situation may occur.

Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock developer documentation, such as this document, aims to be technically accurate with respect to the sample that is documented. It is visible to everyone.

How to Report Problems or Provide Feedback

If you are a named customer Support Contact, contact ForgeRock using the Customer Support Portal to request information or report a problem with Dockerfiles, Kustomize bases, Kustomize overlays, or Skaffold YAML configuration files in the CDK or the CDM.

If you have questions regarding the CDK or the CDM that are not answered in the documentation, file an issue at https://github.com/ForgeRock/forgeops/issues.

When requesting help with a problem, include the following information:

- Description of the problem, including when the problem occurs and its impact on your operation.
- Steps to reproduce the problem.

  If the problem occurs on a Kubernetes system other than Minikube, GKE, EKS, OpenShift, or AKS, we might ask you to reproduce the problem on one of those.

- HTML output from the **debug-logs.sh** script. For more information, see "Reviewing Pod Descriptions and Container Logs" in the *DevOps Developer's Guide: Using Minikube*.
- Description of the environment, including the following information:
  - Environment type: Minikube, GKE, EKS, AKS, or OpenShift.
  - Software versions of supporting components:
    - Oracle VirtualBox (Minikube environments only).

  - Docker client (all environments).
  - Minikube (all environments).
  - **kubectl** command (all environments).
  - Kustomize (all environments).
  - Skaffold (all environments).
  - Google Cloud SDK (GKE environments only).
  - Amazon AWS Command Line Interface (EKS environments only).
  - Azure Command Line Interface (AKS environments only).
- `forgeops` repository branch.
- Any patches or other software that might be affecting the problem.

Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service-level agreements (SLAs), visit https://www.forgerock.com/support.

Glossary

affinity (AM)	AM affinity based load balancing ensures that the CTS token creation load is spread over multiple server instances (the token origin servers). Once a CTS token is created and assigned to a session, all subsequent token operations are sent to the same token origin server from any AM node. This ensures that the load of CTS token management is spread across directory servers. Source: *Best practices for using Core Token Service (CTS) Affinity based load balancing in AM*
Amazon EKS	Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on Amazon Web Services without needing to set up or maintain your own Kubernetes control plane. Source: *What is Amazon EKS* in the Amazon EKS documentation.
ARN (AWS)	An Amazon Resource Name (ARN) uniquely identifies an Amazon Web Service (AWS) resource. AWS requires an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies and API calls. Source: *Amazon Resource Names (ARNs) and AWS Service Namespaces* in the AWS documentation.
AWS IAM Authenticator for Kubernetes	The AWS IAM Authenticator for Kubernetes is an authentication tool that enables you to use *Amazon Web Services (AWS)* credentials for authenticating to a Kubernetes cluster. Source: *AWS IAM Authenticator for Kubernetes* `README` file on `GitHub`.

Azure Kubernetes Service (AKS)	AKS is a managed container orchestration service based on Kubernetes. AKS is available on the Microsoft Azure public cloud. AKS manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications. Source: *Microsoft Azure AKS* documentation.
cloud-controller-manager	The `cloud-controller-manager` daemon runs controllers that interact with the underlying cloud providers. `cloud-controller-manager` is an alpha feature introduced in Kubernetes release 1.6. The `cloud-controller-manager` daemon runs cloud-provider-specific controller loops only. Source: *cloud-controller-manager* section in the Kubernetes Concepts documentation.
Cloud Developer's Kit (CDK)	The developer artifacts in the `forgeops` Git repository, together with the ForgeRock Identity Platform documentation form the Cloud Developer's Kit (CDK). Use the CDK to stand up the platform in your developer environment.
Cloud Deployment Model (CDM)	The Cloud Deployment Model (CDM) is a common use ForgeRock Identity Platform architecture, designed to be easy to deploy and easy to replicate. The ForgeRock Cloud Deployment Team has developed Kustomize bases and overlays, Skaffold configuration files, Docker images, and other artifacts expressly to build the CDM.
CloudFormation (AWS)	CloudFormation is a service that helps you model and set up your Amazon Web Services (AWS) resources. You create a template that describes all the AWS resources that you want. AWS CloudFormation takes care of provisioning and configuring those resources for you. Source: *What is AWS CloudFormation?* in the AWS documentation.
CloudFormation template (AWS)	An AWS CloudFormation template describes the resources that you want to provision in your AWS stack. AWS CloudFormation templates are text files formatted in JSON or YAML. Source: *Working with AWS CloudFormation Templates* in the AWS documentation.
cluster	A container cluster is the foundation of Kubernetes Engine. A cluster consists of at least one cluster master and multiple worker machines called nodes. The Kubernetes objects that represent your containerized applications all run on top of a cluster. Source: *Container Cluster Architecture* in the Kubernetes Concepts documentation.

cluster master	A cluster master schedules, runs, scales and upgrades the workloads on all nodes of the cluster. The cluster master also manages network and storage resources for workloads. Source: *Container Cluster Architecture* in the Kubernetes Concepts docuementation.
ConfigMap	A configuration map, called `ConfigMap` in Kubernetes manifests, binds the configuration files, command-line arguments, environment variables, port numbers, and other configuration artifacts to the assigned containers and system components at runtime. The configuration maps are useful for storing and sharing non-sensitive, unencrypted configuration information. Source: *ConfigMap* in the Kubernetes Cocenpts documentation.
container	A container is an allocation of resources such as CPU, network I/O, bandwidth, block I/O, and memory that can be “contained” together and made available to specific processes without interference from the rest of the system. Source *Container Cluster Architecture* in the Google Cloud Platform documentation
DaemonSet	A set of daemons, called `DaemonSet` in Kubernetes manifests, manages a group of replicated pods. Usually, the daemon set follows an one-pod-per-node model. As you add nodes to a node pool, the daemon set automatically distributes the pod workload to the new nodes as needed. Source *DaemonSet* in the Google Cloud Platform documentation.
Deployment	A Kubernetes deployment represents a set of multiple, identical pods. A Kubernetes deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. Source: *Deployment* in the Google Cloud Platform documentation.
deployment controller	A deployment controller provides declarative updates for pods and replica sets. You describe a desired state in a deployment object, and the deployment controller changes the actual state to the desired state at a controlled rate. You can define deployments to create new replica sets, or to remove existing deployments and adopt all their resources with new deployments. Source: *Deployments* in the Google Cloud Platform documentation.
Docker Cloud	Docker Cloud provides a hosted registry service with build and testing facilities for Dockerized application images; tools to help you set up

	and manage host infrastructure; and application lifecycle features to automate deploying (and redeploying) services created from images. Source: About Docker Cloud in the Docker Cloud documentation.
Docker container	A Docker container is a runtime instance of a Docker image. A Docker container is isolated from other containers and its host machine. You can control how isolated your container's network, storage, or other underlying subsystems are from other containers or from the host machine. Source: Containers section in the Docker architecture documentation.
Docker daemon	The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A Docker daemon can also communicate with other Docker daemons to manage Docker services. Source: *Docker daemon* section in the Docker Overview documentation.
Docker Engine	The Docker Engine is a client-server application with these components: • A server, which is a type of long-running program called a daemon process (the `dockerd` command) • A REST API, which specifies interfaces that programs can use to talk to the daemon and tell it what to do • A command-line interface (CLI) client (the `docker` command) Source: Docker Engine section in the Docker Overview documentation.
Dockerfile	A Dockerfile is a text file that contains the instructions for building a Docker image. Docker uses the Dockerfile to automate the process of building a Docker image. Source: *Dockerfile* section in the Docker Overview documentation.
Docker Hub	Docker Hub provides a place for you and your team to build and ship Docker images. You can create public repositories that can be accessed by any other Docker Hub user, or you can create private repositories you can control access to. An image is an application you would like to run. A container is a running instance of an image.

	Source: *Overview of Docker Hub* section in the Docker Overview documentation.
Docker image	A Docker image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. A Docker image includes the application code, a runtime engine, libraries, environment variables, and configuration files that are required to run the application. An image is an application you would like to run. A container is a running instance of an image. Source: *Docker objects* section in the Docker Overview documentation. Hello Whales: Images vs. Containers in Dockers.
Docker namespace	Docker namespaces provide a layer of isolation. When you run a container, Docker creates a set of namespaces for that container. Each aspect of a container runs in a separate namespace and its access is limited to that namespace. The `PID` namespace is the mechanism for remapping process IDs inside the container. Other namespaces such as net, mnt, ipc, and uts provide the isolated environments we know as containers. The user namespace is the mechanism for remapping user IDs inside a container. Source: *Namespaces* section in the Docker Overview documentation.
Docker registry	A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can also run your own private registry. Source: *Docker registries* section in the Docker Overview documentation.
Docker repository	A Docker repository is a public, certified repository from vendors and contributors to Docker. It contains Docker images that you can use as the foundation to build your applications and services. Source: *Repositories on Docker Hub* section in the Docker Overview documentation.
Docker service	In a distributed application, different pieces of the application are called “services.” Docker services are really just “containers in production.” A Docker service runs only one image, but it codifies the way that image runs including which ports to use, the number replicas

	the container should run, and so on. By default, the services are load-balanced across all worker nodes. Source: *About services* in the Docker Get Started documentation.
dynamic volume provisioning	The process of creating storage volumes on demand is called dynamic volume provisioning. Dynamic volume provisioning allows storage volumes to be created on-demand. It automatically provisions storage when it is requested by users. Source: *Dynamic Volume Provisioning* in the Kubernetes Concepts documentation.
egress	An egress controls access to destinations outside the network from within a Kubernetes network. For an external destination to be accessed from a Kubernetes environment, the destination should be listed as an allowed destination in the whitelist configuration. Source: *Network Policies* in the Kubernetes Concepts documentation.
firewall rule	A firewall rule lets you allow or deny traffic to and from your virtual machine instances based on a configuration you specify. Each Kubernetes network has a set of firewall rules controlling access to and from instances in its subnets. Each firewall rule is defined to apply to either incoming glossary-ingress(ingress) or outgoing (egress) traffic, not both. Source: *Firewall Rules Overview* in the Google Cloud Platform documentation.
garbage collection	Garbage collection is the process of deleting unused objects. Kubelets perform garbage collection for containers every minute and garbage collection for images every five minutes. You can adjust the high and low threshold flags and garbage collection policy to tune image garbage collection. Source: *Garbage Collection* in the Kubernetes Concepts documentation.
Google Kubernetes Engine (GKE)	The Google Kubernetes Engine (GKE) is an environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machine instances grouped together to form a container cluster. Source: *Kubernetes Engine Overview* in the Google Cloud Platform documentation.
ingress	An ingress is a collection of rules that allow inbound connections to reach the cluster services.

	Source: *Ingress* in the Kubernetes Concepts documentation.
instance group	An instance group is a collection of instances of virtual machines. The instance groups enable you to easily monitor and control the group of virtual machines together. Source: *Instance Groups* in the Google Cloud Platform documentation.
instance template	An instance template is a global API resource that you can use to create VM instances and managed instance groups. Instance templates define the machine type, image, zone, labels, and other instance properties. They are very helpful in replicating the environments. Source: *Instance Templates* in the Google Cloud Platform documentation.
kubectl	The **kubectl** command-line tool supports several different ways to create and manage Kubernetes objects. Source: *Kubernetes Object Management* in the Kubernetes Concepts documentation.
kube-controller-manager	The Kubernetes controller manager is a process that embeds core controllers that are shipped with Kubernetes. Logically each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. Source: *kube-controller-manager* in the Kubernetes Reference documentation.
kubelet	A kubelet is an agent that runs on each node in the cluster. It ensures that containers are running in a pod. Source: *kubelets* in the Kubernetes Concepts documentation.
kube-scheduler	The `kube-scheduler` component is on the master node and watches for newly created pods that do not have a node assigned to them, and selects a node for them to run on. Source: *Kubernetes components* in the Kubernetes Concepts documentation.
Kubernetes	Kubernetes is an open source platform designed to automate deploying, scaling, and operating application containers. Source: *Kubernetes Concepts*

Kubernetes DNS	A Kubernetes DNS pod is a pod used by the kubelets and the individual containers to resolve DNS names in the cluster. Source: *DNS for services and pods* in the Kubernetes Concepts documentation.
Kubernetes namespace	A Kubernetes namespace is a virtual cluster that provides a way to divide cluster resources between multiple users. Kubernetes starts with three initial namespaces: • `default`: The default namespace for user created objects which don't have a namespace • `kube-system`: The namespace for objects created by the Kubernetes system • `kube-public`: The automatically created namespace that is readable by all users Kubernetes supports multiple virtual clusters backed by the same physical cluster. Source: *Namespaces* in the Kubernetes Concepts documentation.
Let's Encrypt	Let's Encrypt is a free, automated, and open certificate authority. Source: Let's Encrypt web site.
Microsoft Azure	Microsoft Azure is the Microsoft cloud platform, including infrastructure as a service (IaaS) and platform as a service (PaaS) offerings. Source: *Cloud computing terms* in the Microsoft Azure documentation.
network policy	A Kubernetes network policy specifies how groups of pods are allowed to communicate with each other and with other network endpoints. Source: *Network policies* in the Kubernetes Concepts documentation.
node (Kubernetes)	A Kubernetes node is a virtual or physical machine in the cluster. Each node is managed by the master components and includes the services needed to run the pods. Source: *Nodes* in the Kubernetes Concepts documentation.
node controller (Kubernetes)	A Kubernetes node controller is a Kubernetes master component that manages various aspects of the nodes such as: lifecycle operations on the nodes, operational status of the nodes, and maintaining an internal list of nodes.

	Source: *Node Controller* in the Kubernetes Concepts documentation.
persistent volume	A persistent volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins that have a lifecycle independent of any individual pod that uses the PV. Source: *Persistent Volumes* in the Kubernetes Concepts documentation.
persistent volume claim	A persistent volume claim (PVC) is a request for storage by a user. A PVC specifies size, and access modes such as: • Mounted once for read and write access • Mounted many times for read-only access Source: *Persistent Volumes* in the Kubernetes Concepts documentation.
pod anti-affinity (Kubernetes)	Kubernetes pod anti-affinity allows you to constrain which nodes can run your pod, based on labels on the **pods** that are already running on the node rather than based on labels on nodes. Pod anti-affinity enables you to control the spread of workload across nodes and also isolate failures to nodes. Source: *Inter-pod affinity and anti-affinity*
pod (Kubernetes)	A Kubernetes pod is the smallest, most basic deployable object in Kubernetes. A pod represents a single instance of a running process in a cluster. Containers within a pod share an IP address and port space. Source: *Understanding Pods* in the Kubernetes Concepts documentation.
region (Azure)	An Azure region, also known as a location, is an area within a geography, containing one or more data centers. Source: *region* in the Microsoft Azure glossary.
replication controller (Kubernetes)	A replication controller ensures that a specified number of Kubernetes pod replicas are running at any one time. The `replication controller` ensures that a pod or a homogeneous set of pods is always up and available. Source: *ReplicationController* in the Kubernetes Concepts documentation.

resource group (Azure)	A resource group is a container that holds related resources for an application. The resource group can include all of the resources for an application, or only those resources that are logically grouped together. Source: *resource group* in the Microsoft Azure glossary.
secret (Kubernetes)	A Kubernetes secret is a secure object that stores sensitive data, such as passwords, OAuth 2.0 tokens, and SSH keys in your clusters. Source *Secrets* in the Kubernetes Concepts documentation.
security group (AWS)	A security group acts as a virtual firewall that controls the traffic for one or more compute instances. Source: *Amazon EC2 Security Groups* in the AWS documentation.
service (Kubernetes)	A Kubernetes service is an abstraction which defines a logical set of pods and a policy by which to access them. This is sometimes called a microservice. Source: *Services* in the Kubernetes Concepts documentation.
service principal (Azure)	An Azure service principal is an identity created for use with applications, hosted services, and automated tools to access Azure resources. Service principals enable applications to access resources with the restrictions imposed by the assigned roles instead of accessing resources as a fully privileged user. Source: *Create an Azure service principal with Azure PowerShell* in the Microsoft Azure PowerShell documentation.
shard	Sharding is a way of partitioning directory data so that the load can be shared by multiple directory servers. Each data partition, also known as a *shard*, exposes the same set of naming contexts, but only a subset of the data. For example, a distribution might have two shards. The first shard contains all users whose name begins with A-M, and the second contains all users whose name begins with N-Z. Both have the same naming context. Source: *Class Partition* in the *OpenDJ Javadoc*.
stack (AWS)	A stack is a collection of AWS resources that you can manage as a single unit. You can create, update, or delete a collection of resources by using stacks. All the resources in a stack are defined by the template. Source: *Working with Stacks* in the AWS documentation.

stack set (AWS)	A stack set is a container for stacks. You can provision stacks across AWS accounts and regions by using a single AWS template. All the resources included in each stack of a stack set are defined by the same template. Source: *StackSets Concepts* in the AWS documentation.
subscription (Azure)	An Azure subscription is used for pricing, billing and payments for Azure cloud services. Organizations can have multiple Azure subscriptions, and subscriptions can span multiple regions. Source: *subscription* in the Microsoft Azure glossary.
volume (Kubernetes)	A Kubernetes volume is a storage volume that has the same lifetime as the pod that encloses it. Consequently, a volume outlives any containers that run within the pod, and data is preserved across container restarts. When a pod ceases to exist, the Kubernetes volume also ceases to exist. Source: *Volumes* in the Kubernetes Concepts documentation.
VPC (AWS)	A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. Source: *What Is Amazon VPC?* in the AWS documentation.
worker node (AWS)	An Amazon Elastic Container Service for Kubernetes (Amazon EKS) worker node is a standard compute instance provisioned in Amazon EKS. Source: *Worker Nodes* in the AWS documentation.
workload (Kubernetes)	A Kubernetes workload is the collection of applications and batch jobs packaged into a container. Before you deploy a workload on a cluster, you must first package the workload into a container. Source: *Understanding Pods* in the Kubernetes Concepts documentation.