DevOps Developer's Guide: Using Minikube

/ 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

Guide to ForgeRock Identity Platform™ deployment on Kubernetes.

Table of Contents

Preface

DevOps Developer's Guide: Using Minikube explains basic concepts and strategies for developing custom Docker images for ForgeRock software on a single-user Minikube Kubernetes cluster.

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)

Chapter 1

Introducing the Cloud Developer's Kit

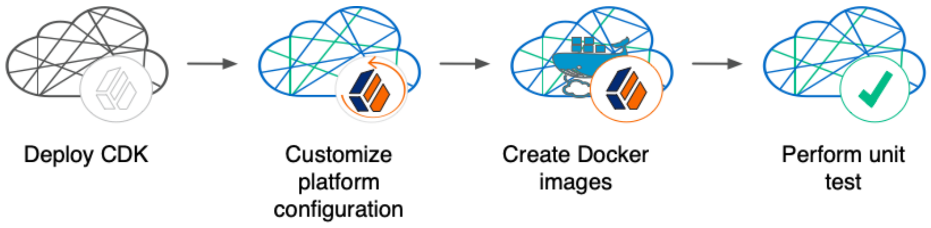
This chapter introduces you to ForgeRock's Cloud Developer's Kit (CDK). Sections in this chapter provide the following conceptual information:

- A CDK overview
- A description of the CDK deployment on Minikube
- Information about data used by the CDK

About the Cloud Developer's Kit

The CDK is a minimal sample deployment for development purposes. It includes fully integrated AM, IDM, and DS installations, and randomly generated secrets. Developers deploy the CDK, and then access AM's and IDM's GUI consoles and REST APIs to configure the platform and build customized Docker images for the platform.

This guide describes how to use the CDK to stand up the platform in your developer environment, then create and test customized Docker images containing your custom AM and IDM configurations:



Customizing the platform using the CDK is one of the major activities required before deploying the platform in production. To better understand how this activity fits in to the overall deployment process, see "Configure the Platform" in the *Start Here* guide.

Containerization

The CDK uses Docker for containerization. The CDK leverages the following Docker capabilities:

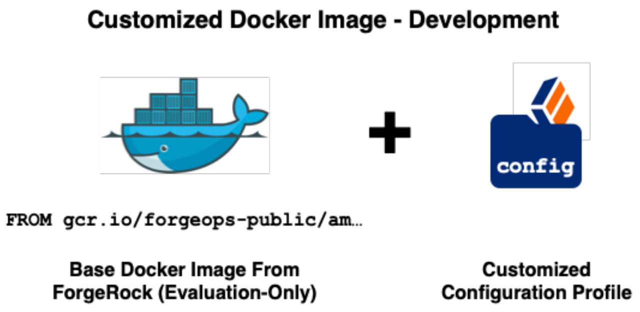
- **File-Based Representation of Containers**. Docker *images* contain a file system and run-time configuration information. Docker *containers* are running instances of Docker images.

- **Modularization**. Docker images are based on other Docker images. For example, an AM image is based on a Tomcat image that is itself based on an OpenJDK JRE image. In this example, the AM container has AM software, Tomcat software, and the OpenJDK JRE.

- **Collaboration**. Public and private Docker registries let users collaborate by providing cloud-based access to Docker images. Continuing with the example, the public Docker registry at https://hub.docker.com/ has Docker images for Tomcat and the OpenJDK JRE that any user can download. You build Docker images for the ForgeRock Identity Platform based on the Tomcat and OpenJDK JRE images in the public Docker registry. You can then push the Docker images to a private Docker registry that other users in your organization can access.

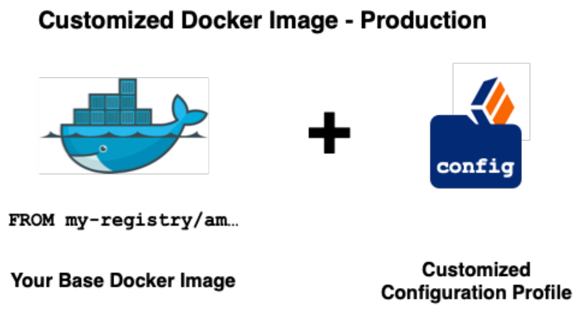
ForgeRock provides a set of unsupported, evaluation-only base images for the ForgeRock Identity Platform. These images are available in ForgeRock's public Docker registry.

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


Customized Docker Image - Development
FROM gcr.io/forgeops-public/am...
config
Base Docker Image From ForgeRock (Evaluation-Only)
Customized Configuration Profile

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:


Customized Docker Image - Production
FROM my-registry/am...
config
Your Base Docker Image
Customized Configuration Profile

For information about how to build base images for deploying the ForgeRock Identity Platform in production, see "*Building Base Docker Images*" in the *Cloud Deployment Guide*.

Orchestration

The CDK uses Kubernetes for container orchestration. The CDK has been tested on the following Kubernetes implementations:

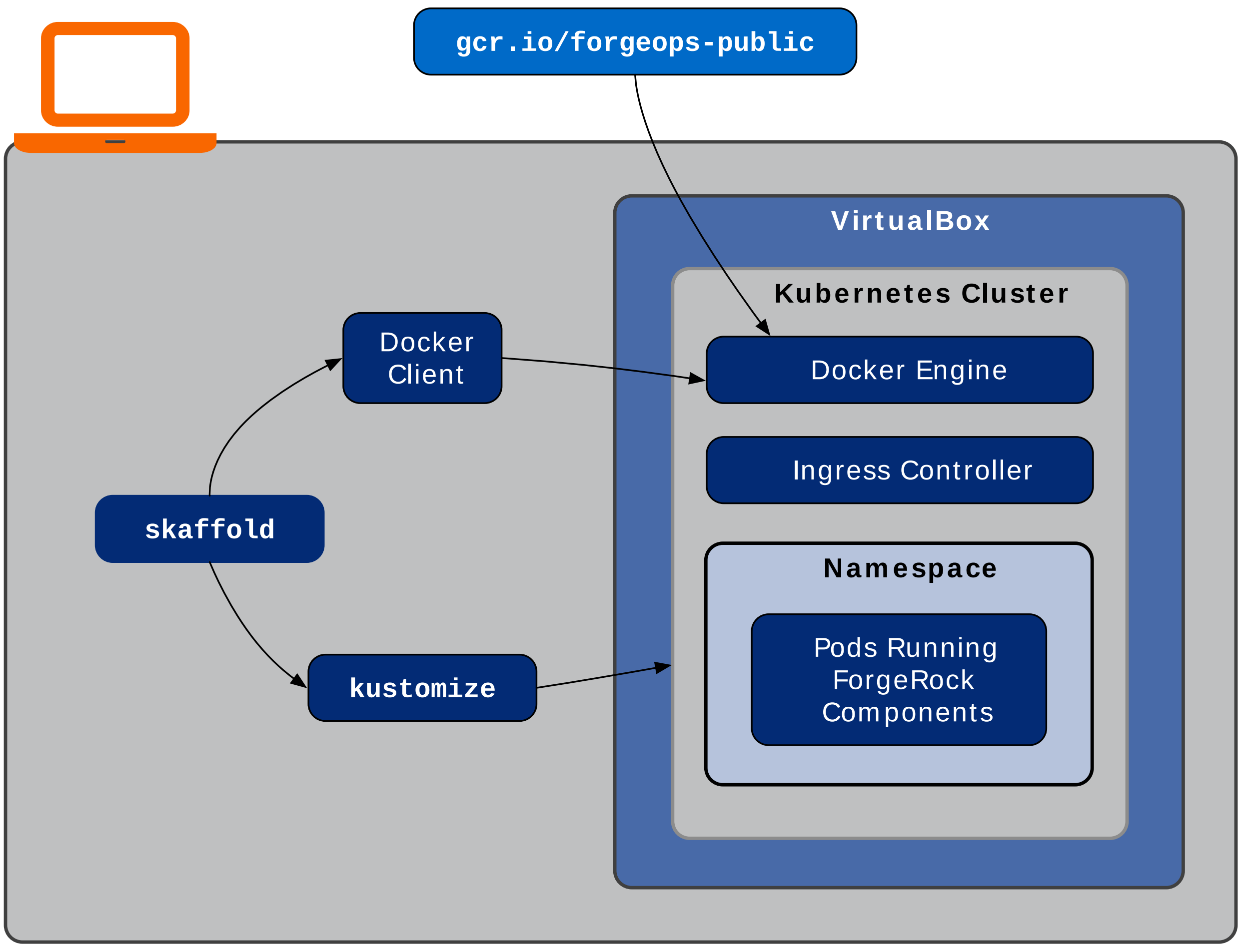
- Single-node deployments suitable for proofs of concept and development:
  - Minikube
  - Minishift
- Cloud-based Kubernetes orchestration frameworks. These are suitable for both development and production deployment of the platform:
  - Google Kubernetes Engine (GKE)
  - Amazon Elastic Kubernetes Service (Amazon EKS)
  - Azure Kubernetes Service (AKS)
  - Red Hat OpenShift

About the Minikube Environment

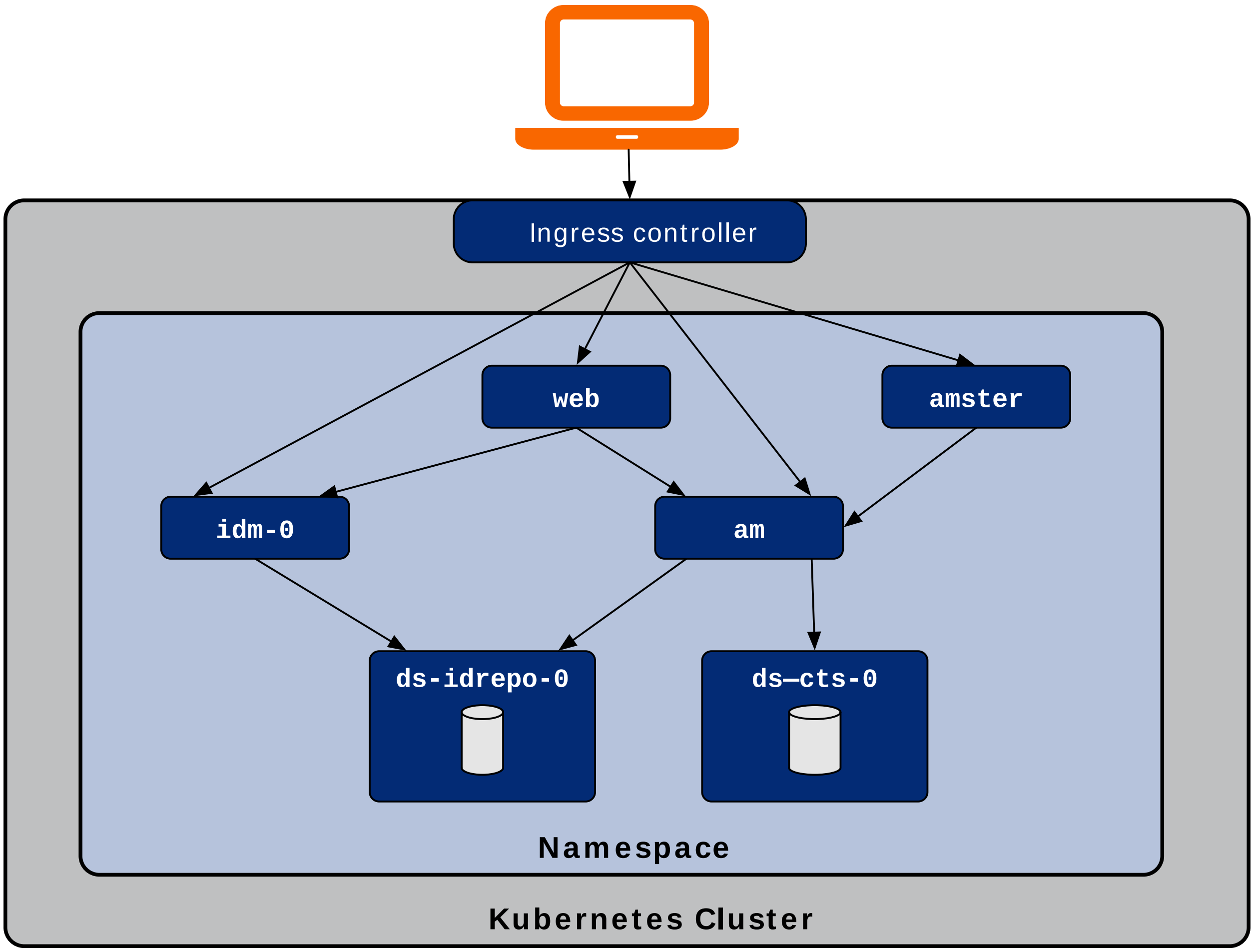
The CDK uses Skaffold to trigger Docker image builds and Kubernetes orchestration. Here's what Skaffold does:

1. Calls the Docker client on the local computer to build and tag Docker images for the ForgeRock Identity Platform. The images are based on Docker images in ForgeRock's registry, `gcr.io/forgeops-public`.
2. Pushes the Docker images to the Docker engine that's part of the Minikube VM.
3. Calls Kustomize to orchestrate the ForgeRock Identity Platform in your namespace. Kustomize uses the Docker images that Skaffold pushed to your Docker registry.

The following diagram illustrates how the CDK uses Skaffold to build and orchestrate Docker images on Minikube:



After deploying the ForgeRock Identity Platform, you'll see the following pods running in your namespace:



`am`

The `am` pod runs AM.

Note that the `amster` pod runs a script to provide AM's initial configuration.

`amster`

The `amster` pod is available to run Amster jobs.

When the `amster` pod starts, it runs the `/path/to/forgeops/docker/amster-install.sh` script. This script checks whether AM has successfully started in the `am` pod. If AM has not started, the script waits until it has. Once AM has started, the script configures AM, using the configuration files in the `/opt/amster/config` directory[1].

After the `amster-install.sh` script has finished configuring AM, the `amster` pod's initial job is complete. The pod remains available to run Amster jobs as needed.

`ds-cts-0`

The `ds-cts-0` pod runs the directory service used by the AM Core Token Service.

`ds-idrepo-0`

The `ds-idrepo-0` pod runs the following directory services:

- AM configuration store
- Identity repository shared by AM and IDM
- IDM repository

`idm-0`

The `idm-0` pod runs IDM.

When IDM starts, it obtains its configuration from the `/opt/openidm/conf` directory[2].

In containerized deployments, IDM must retrieve its configuration from the file system and not from the IDM repository. The default values for the `openidm.fileinstall.enabled` and `openidm.config.repo.enabled` properties in the CDK's `system.properties` file ensure that IDM retrieves its configuration from the file system. Do not override the default values for these properties.

`web`

The `web` pod runs a web application that lets you access the AM console and the IDM Admin UI.

About Data Used by the Platform

The ForgeRock Identity Platform uses two types of data: configuration data and run-time data.

Configuration Data

Configuration data consists of properties and settings used by the ForgeRock Identity Platform. You update configuration data during the development phase of ForgeRock Identity Platform implementation. You should not change configuration data during the testing and production phases.

[1] When you build the `amster` Docker image, the AM configuration files are copied from the `/path/to/forgeops/docker/amster/config` directory to the `/opt/amster/config` directory.
[2] When you build the `idm` Docker image, the IDM configuration files are copied from the `/path/to/forgeops/docker/idm/conf` directory to the `/opt/openidm/conf` directory.

You change configuration data iteratively in a development environment. After changing configuration data, you rebuild Docker images and restart ForgeRock Identity Platform services when you're ready to test sets of changes. If you make incorrect changes to configuration data, the platform might become inoperable. After testing modifications to configuration data, you promote your changes to test and production environments.

Examples of configuration data include AM realms, AM authentication trees, IDM social identity provider definitions, and IDM data mapping models for reconciliation.

Configuration Profiles

A ForgeRock Identity Platform *configuration profile* is a named set of configuration data that describes the operational characteristics of a running ForgeRock deployment.

Configuration profiles reside in two locations in the `forgeops` repository:

- **The master directory**. Holds a canonical configuration profile for the CDK and user-customized configuration profiles. User-customized configuration profiles in this directory are considered to be the *source of truth* for ForgeRock Identity Platform deployments.

  The master directory for configuration profiles is located at the path `/path/to/forgeops/config/6.5`. You use Git to manage the configuration profiles in this directory.

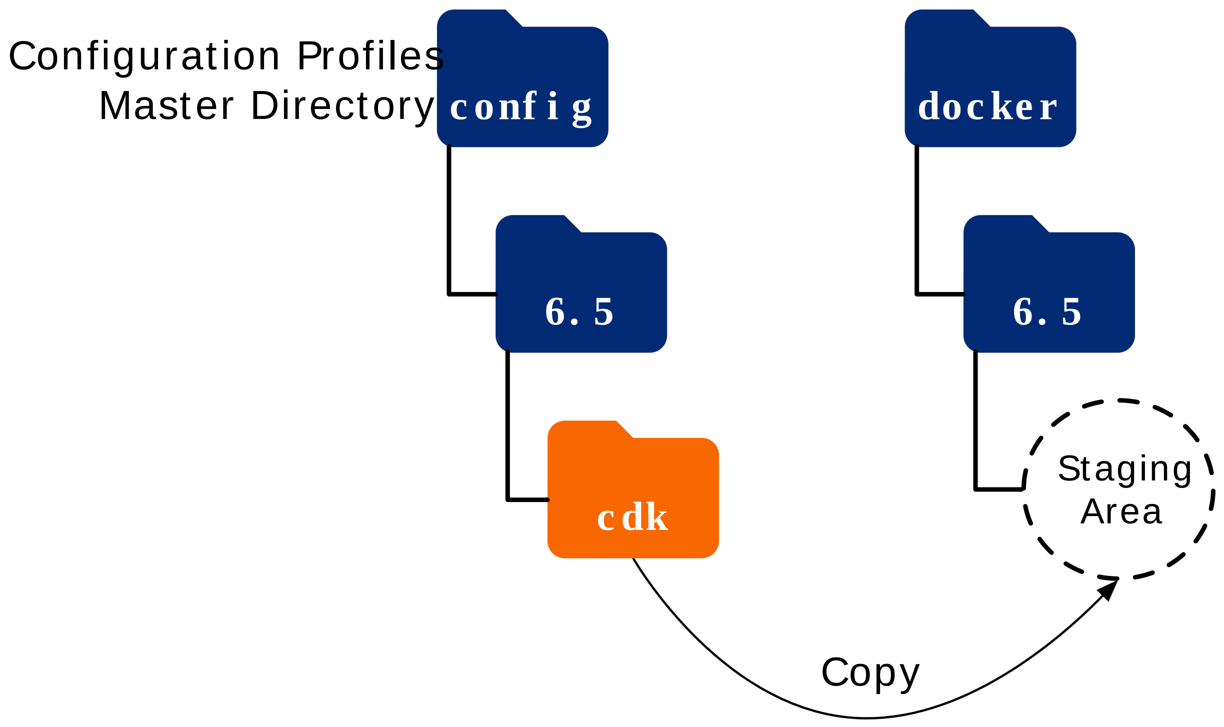
- **The staging area**. Holds a single configuration profile. You copy a profile from the master directory to the staging area before building a customized Docker image for the ForgeRock Identity Platform.

  The staging area is located in subdirectories of the path, `/path/to/forgeops/docker/6.5`. Configuration profiles copied to the staging area are transient and are not managed with Git.

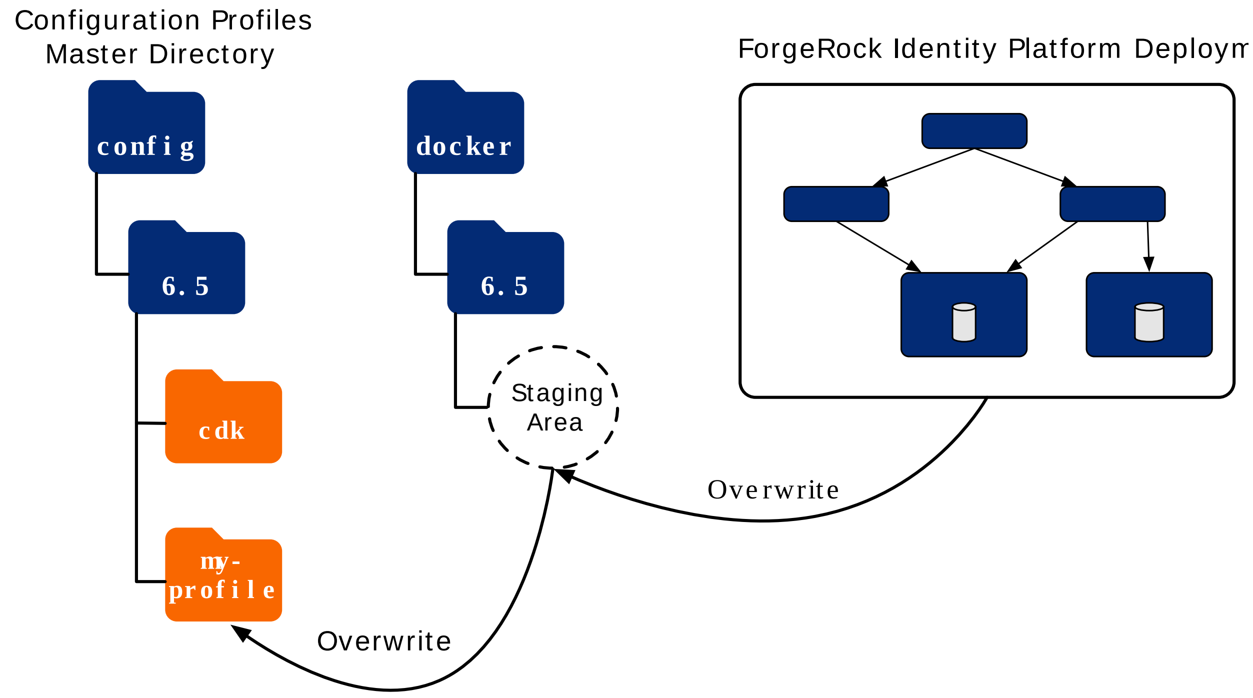
The `config.sh` script lets you copy configuration profiles between the master directory and the staging area. It also lets you copy profiles from Kubernetes pods running ForgeRock Identity Platform components to the staging area.

You run this script before you build a customized Docker image for the platform. The script lets you specify which configuration profile to copy to the staging area. Skaffold uses this profile when it builds a Docker image.

For example, when you start developing customized images for the platform, you run the **config.sh init** command to initialize the staging area with the canonical CDK profile:



You run the **config.sh sync** command to synchronize configuration changes you've made in a running deployment back to the staging area, and then to the master directory:


Configuration Profiles Master Directory
config
6.5
cdk
my-profile
docker
6.5
Staging Area
ForgeRock Identity Platform Deploym
Overwrite
Overwrite

For more information about the **config.sh** script, see Managing Configurations in the `forgeops` repository's top-level README file.

Run-Time Data

Run-time data consists of identities, policies, applications, and data objects used by the ForgeRock Identity Platform. You might extract sample run-time data while developing configuration data. Run-time data is volatile throughout ForgeRock Identity Platform implementation. Expect it to change even when the ForgeRock Identity Platform is in production.

You usually use sample data for run-time data. Run-time data that's changed during development is not typically promoted to test and production environments. There's no need to modify Docker images or restart ForgeRock Identity Platform services when run-time data is modified.

Examples of run-time data include AM and IDM identities, AM policies, AM OAuth 2.0 client definitions, and IDM relationships.

In the ForgeRock Identity Platform, run-time data is stored in databases and is not file-based. For more information about how run-time data is stored in AM and IDM, see:

- *Preparing External Stores* in the *AM Installation Guide*.

- *Managing the Repository* in the *IDM Integrator's Guide*.

Chapter 2
Setting up Your Development Environment

This chapter describes how to set up your local computer for developing custom Docker images for the ForgeRock Identity Platform on a Minikube cluster.

+ *Windows users*

ForgeRock supports deploying the CDK and CDM using macOS and Linux. If you have a Windows computer, you'll need to create a Linux VM. We tested using the following configurations:

- Hypervisor: Hyper-V, VMWare Player, or VMWare Workstation
- Guest OS: Ubuntu 19.10 with 12 GB memory and 60 GB disk space
- Nested virtualization enabled in the Linux VM.

Perform all the procedures in this guide within the Linux VM. In this guide, the local computer refers to the Linux VM for Windows users.

Complete all of the tasks in the following sections to set up your local computer:

1. "Obtaining the forgeops Repository"
2. "Installing Third-Party Software"
3. "Creating a Minikube Virtual Machine"
4. "Creating a Namespace"
5. "Setting up Hostname Resolution"
6. "Setting up Your Local Computer to Use Minikube's Docker Engine"

When you've completed the setup tasks, you'll have an environment like the one shown in this diagram.

Obtaining the forgeops Repository

Before you can deploy the CDK or the CDM, you must first get the `forgeops` repository[1]:

To Obtain the forgeops Repository

1. Clone the `forgeops` repository:

   ```
   $ git clone https://github.com/ForgeRock/forgeops.git
   ```

   The `forgeops` repository is a public Git repository. You do not need credentials to clone it.

2. Check out the `6.5-2020.06.24` release tag, creating a branch named *`my-branch`*:

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

Installing Third-Party Software

After you've obtained the `forgeops` repository, you'll need to install a set of third-party software on your local computer.

ForgeRock recommends that you install third-party software using Homebrew on macOS and Linux. For a list of the Homebrew packages to install, see "*Homebrew Package Names*".

The versions listed in the following table have been validated for building custom Docker images for the ForgeRock Identity Platform. Earlier and later versions will *probably* work. If you want to try using versions that are not in the tables, it is your responsibility to validate them.

Install all of the following third-party software:

Software	Version	URL for More Information
Docker Desktop[a]	2.3.0.3	https://www.docker.com/products/docker-desktop
Kubernetes client (**kubectl**)	1.18.4	https://kubernetes.io/docs/tasks/kubectl/install
Skaffold	1.11.0	https://skaffold.dev
Kustomize	3.6.1	https://kustomize.io
Kubernetes context switcher (**kubectx**)	0.9.0	https://github.com/ahmetb/kubectx
Kubernetes log display utility (**stern**)	1.11.0	https://github.com/wercker/stern

[1] For the short term, follow the steps in the procedure to clone the `forgeops` repository and check out the `6.5-2020.06.24` tag.

For the long term, you'll need to implement a strategy for managing updates, especially if a team of people in your organization works with the repository. For example, you might want to adopt a workflow that uses a fork as your organization's common upstream repository. For more information, see "*About the forgeops Repository*" in the *Cloud Deployment Guide*.

Software	Version	URL for More Information
VirtualBox	6.1.10	https://www.virtualbox.org/wiki/downloads
Minikube	1.11.0	https://kubernetes.io/docs/tasks/tools/install-minikube/

[a] Docker Desktop is available for macOS only. On Linux computers, install Docker CE instead. For more information, see the Docker documentation.

Creating a Minikube Virtual Machine

Now that you've installed third-party software on your local computer, you're ready to create a Minikube VM. When you create a Minikube VM, a Kubernetes cluster is created in the VM.

The following configuration has been validated for building custom Docker images for the ForgeRock Identity Platform:

- Kubernetes version: 1.17.4
- Memory: 8 GB or more
- Disk space: 40 GB or more

Perform the following procedure to set up Minikube:

To Set up Minikube

1. Use the **minikube start** command to create a Minikube VM. In this example, the Minikube VM is created with a Kubernetes cluster suitable for building custom Docker images for the ForgeRock Identity Platform:

```
$ minikube start --memory=12288 --cpus=3 --disk-size=40g \
 --vm-driver=virtualbox --bootstrapper kubeadm --kubernetes-version=1.17.4
##  minikube v1.8.2 on Darwin 10.14.6
#  Using the virtualbox driver based on user configuration
##  Downloading VM boot image ...
    > minikube-v1.8.0.iso.sha256: 65 B / 65 B [--------------] 100.00% ? p/s 0s
    > minikube-v1.8.0.iso: 173.56 MiB / 173.56 MiB [-] 100.00% 4.61 MiB p/s 38s
##  Creating virtualbox VM (CPUs=2, Memory=8192MB, Disk=40000MB) ...
##  Preparing Kubernetes v1.17.4 on Docker 19.03.6 ...
    > kubelet.sha1: 41 B / 41 B [---------------------------] 100.00% ? p/s 0s
    > kubectl.sha1: 41 B / 41 B [---------------------------] 100.00% ? p/s 0s
    > kubeadm.sha1: 41 B / 41 B [---------------------------] 100.00% ? p/s 0s
    > kubeadm: 38.32 MiB / 38.32 MiB [---------------] 100.00% 1.29 MiB p/s 30s
    > kubectl: 40.99 MiB / 40.99 MiB [---------------] 100.00% 1.31 MiB p/s 31s
    > kubelet: 114.08 MiB / 114.08 MiB [-------------] 100.00% 2.00 MiB p/s 57s
##  Launching Kubernetes ...
##  Enabling addons: default-storageclass, storage-provisioner
#  Waiting for cluster to come online ...
##  Done! kubectl is now configured to use "minikube"
```

2. Run the following command to enable the ingress controller plugin built into Minikube:

```
$ minikube addons enable ingress
##  The 'ingress' addon is enabled
```

3. Before attempting to work with the ForgeRock Identity Platform on Minikube, you *must* implement the workaround for Minikube issue 1568. The workaround lets pods deployed on Minikube reach themselves on the network.

   Run the following command to work around the issue:

   ```
   $ minikube ssh sudo ip link set docker0 promisc on
   ```

   Note that you must run this command *every time you restart the Minikube VM*.

Creating a Namespace

After you've created the Minikube VM and Kubernetes cluster, create a namespace in your new cluster.

ForgeRock recommends that you deploy the ForgeRock Identity Platform in a namespace other than the default namespace. Deploying to a non-default namespace lets you separate workloads in a cluster. Separating a workload into a namespace lets you delete the workload easily; just delete the namespace.

Perform the following procedure to create a namespace:

To Create a Namespace

1. Create a namespace in your Kubernetes cluster:

   ```
   $ kubectl create namespace my-namespace
   namespace/my-namespace created
   ```

2. Make the new namespace your current namespace:

   ```
   $ kubens my-namespace
   Context "my-context" modified.
   Active namespace is "my-namespace".
   ```

Setting up Hostname Resolution

After you've created a namespace, set up hostname resolution for the ForgeRock Identity Platform servers you'll deploy in your namespace.

1. Run the **minikube ip** command to get the Minikube ingress controller's IP address.

2. Add an entry similar to the following to the `/etc/hosts` file:

```
minikube-ip-address my-namespace.iam.example.com
```

Setting up Your Local Computer to Use Minikube's Docker Engine

Now you've prepared your cluster by creating a namespace and setting up hostname resolution. Your last step before you can deploy the ForgeRock Identity Platform is to set up your local computer to execute **docker** commands on Minikube's Docker engine.

ForgeRock recommends using the built-in Docker engine when developing custom Docker images using Minikube. When you use Minikube's engine, you don't have to build Docker images on a local engine and then push the images to a local or cloud-based Docker registry. Instead, you build images using the same Docker engine that Minikube uses. This streamlines development.

Set up your local computer to use Minikube's Docker engine as follows:

To Set Up Your Local Computer to Use Minikube's Docker Engine

1. Run the **docker-env** command in your shell:

   ```
   $ eval $(minikube docker-env)
   ```

2. Stop Skaffold from pushing Docker images to a remote Docker registry [2]:

   ```
   $ skaffold config set --kube-context minikube local-cluster true
   ```

For more information about using Minikube's built-in Docker engine, see Use local images by re-using the Docker daemon in the Minikube documentation.

[2] If your cluster's context is not `minikube`, replace `minikube` with the actual context name in the **skaffold config set** command.

Chapter 3

Deploying the Platform

After you've set up your development environment, your next step is to deploy the platform.

Perform the following procedure to deploy the ForgeRock Identity Platform in your namespace:

To Deploy the ForgeRock Identity Platform

1. Change the deployment namespace for the `all` environment from the `default` namespace to your namespace:

   a. Change to the directory containing the `all` environment:

   ```
   $ cd /path/to/forgeops/kustomize/overlay/6.5/all
   ```

   b. Open the `kustomization.yaml` file.

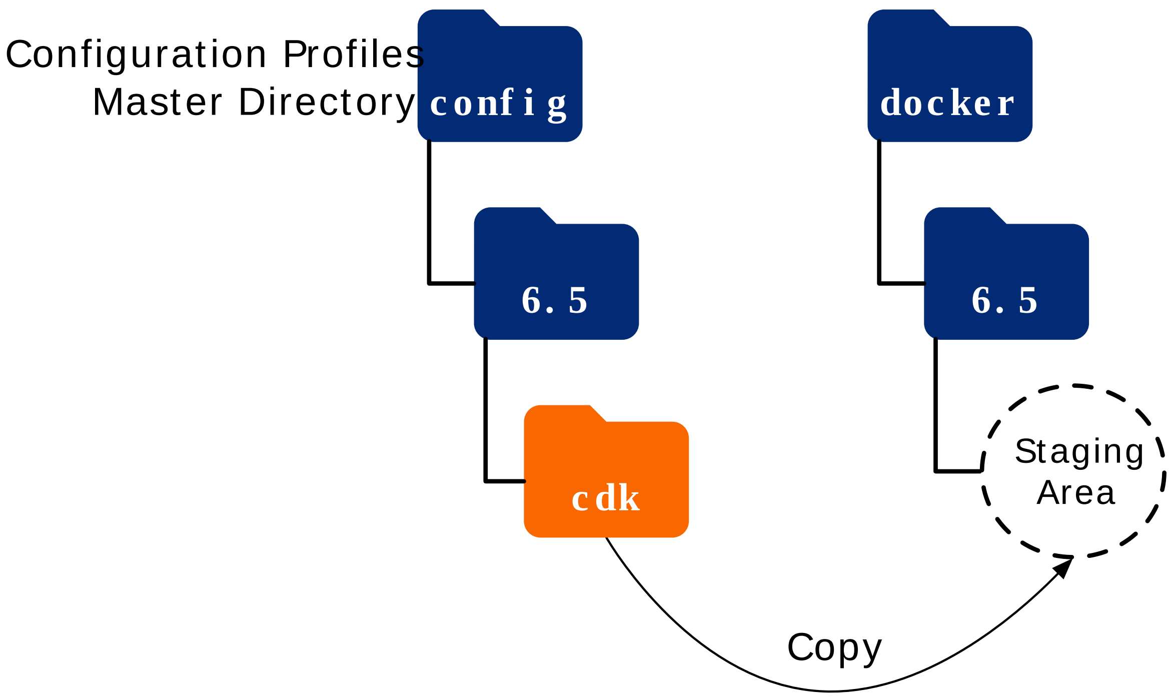
   c. Change the text, `namespace: default`, to `namespace: my-namespace`

   d. Save the updated `kustomization.yaml` file.

2. Initialize the staging area for configuration profiles with the canonical CDK configuration profile for the ForgeRock Identity Platform:

   ```
   $ cd /path/to/forgeops/bin
   $ ./config.sh init --profile cdk --version 6.5
   ```

   The **config.sh init** command copies the canonical CDK configuration profile from the master directory for configuration profiles to the staging area:



For more information about the management of ForgeRock Identity Platform configuration profiles in the `forgeops` repository, see "Configuration Profiles" in the *DevOps Developer's Guide: Using a Shared Cluster*.

3. Run Skaffold to build Docker images and deploy the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops
$ skaffold dev -f skaffold-6.5.yaml
```

4. In a separate terminal tab or window, run the **kubectl get pods** command to monitor status of the deployment. Wait until all the pods are ready.

   Your namespace should have the pods shown in this diagram.

   You're now ready to access tools that will help you customize ForgeRock Identity Platform Docker images. Proceed to "*Using the Platform*" for more information about using ForgeRock's administration consoles and REST APIs from your development environment.

Chapter 4
Using the Platform

Now that you've deployed the ForgeRock Identity Platform, you'll need to know how to access its administration tools. You'll use these tools to build customized Docker images for the platform.

This chapter shows you how to access the ForgeRock Identity Platform's administrative consoles and REST APIs.

You access AM and IDM services through the Kubernetes ingress controller. Access components using their normal interfaces:

- For AM, the console and REST APIs.
- For IDM, the Admin UI and REST APIs.

You can't access DS through the ingress controller, but you can use Kubernetes methods to access the DS pods.

For more information about how AM and IDM are configured in the CDK, see Configuration in the `forgeops` repository's top-level README file.

Accessing AM Services

Access the AM console and REST APIs as follows:

- "To Access the AM Console"
- "To Access the AM REST APIs"

To Access the AM Console

1. Open a new window or tab in a web browser.
2. Obtain the `amadmin` user's password:

   ```
   $ cd /path/to/forgeops/bin
   $ ./print-secrets.sh amadmin
   ```

3. Navigate to the AM deployment URL, https://*my-namespace*.iam.example.com/am.

   The Kubernetes ingress controller handles the request, routing it to a running AM instance.

AM prompts you to log in.

4. Log in as the `amadmin` user.

   The AM console appears in the browser.

To Access the AM REST APIs

1. Start a terminal window session.

2. Run a **curl** command to verify that you can access the REST APIs through the ingress controller. For example:

```
$ curl \
 --insecure \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: 179rd8en9rffa82rcf1qap1z0gv1hcej" \
 --header "Accept-API-Version: resource=2.0" \
 --data "{}" \
 'https://my-namespace.iam.example.com/am/json/realms/root/authenticate'
{
    "tokenId":"AQIC5wM2...",
    "successUrl":"/am/console",
    "realm":"/"
}
```

Accessing IDM Services

Access the IDM Admin UI and REST APIs as follows:

- "To Access the IDM Admin UI Console"
- "To Access the IDM REST APIs"

To Access the IDM Admin UI Console

1. Open a new window or tab in a web browser.

2. Obtain the `openidm-admin` user's password:

```
$ cd /path/to/forgeops/bin
$ ./print-secrets.sh idmadmin
```

3. Navigate to the IDM Admin UI deployment URL, https://*my-namespace*.iam.example.com/admin.

   The Kubernetes ingress controller handles the request, routing it to a running IDM instance.

   IDM prompts you to log in.

4. Log in as the `openidm-admin` user.

   The IDM Admin UI appears in the browser.

To Access the IDM REST APIs

1. Start a terminal window session.
2. Run a **curl** command to verify that you can access the REST APIs through the ingress controller. For example:

```
 $ curl \
 --request GET \
 --insecure \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: 2732jd6bxpw1108ccdjsq4zkeoep0zsb" \
 --data "{}" \
 https://my-namespace.iam.example.com/openidm/info/ping
{
    "_id":" ",
    "_rev":"",
    " shortDesc":"OpenIDM ready",
    "state":"ACTIVE_READY"
}
```

Accessing DS

The DS pods in the CDK are not exposed outside of the cluster. If you need to access one of the DS pods, use a standard Kubernetes method:

- Execute shell commands in DS pods using the **kubectl exec** command.
- Forward a DS pod's LDAP port (1389) to your local computer. Then you can run LDAP CLI commands like **ldapsearch**. You can also use an LDAP editor such as Apache Directory Studio to access the directory.

For all CDK directory pods, the directory superuser DN is `cn=Directory Manager`. Obtain this user's password by running the `print-secrets.sh dsadmin` command.

Chapter 5

Developing Custom Docker Images for the Platform

After following the instructions in the preceding chapters, you have deployed the ForgeRock Identity Platform and learned how to access its administration GUIs and REST APIs. Now you're ready to configure the platform to meet your needs. As you configure the platform, you can decide at any point to build new custom Docker images that will incorporate the configuration changes you've made.

This chapter contains information about building Docker images for the ForgeRock Identity Platform:

- "Custom Docker Image Development Overview"
- "Developing a Customized Amster Docker Image"
- "Developing a Customized IDM Docker Image"

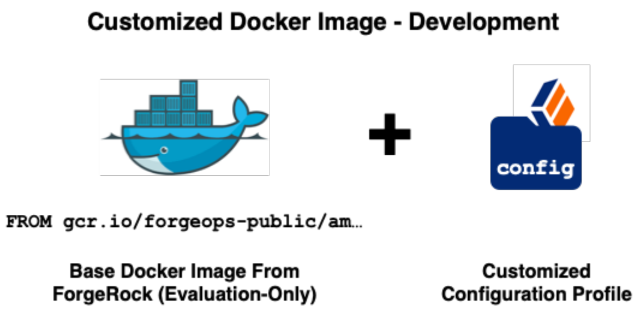
Custom Docker Image Development Overview

After you've deployed the platform and verified you can access its GUIs and REST APIs, you're ready to develop customized Docker images. During development, you iterate on the following process:

- Access AM and IDM running in the CDK, and customize them using their GUIs and REST APIs.
- Export your customizations from the CDK to a Git repository on your local computer.
- Rebuild the Docker images for the platform with your new customizations.
- Redeploy the platform on the CDK.

Before you build customized Docker images for the platform, be sure you're familiar with the types of data used by the platform. This conceptual information helps you understand which type of data is included in custom Docker images.

To develop customized Docker images, start with base images and a canonical configuration profile from ForgeRock. Then, build up a configuration profile, customizing the platform to meet your needs. The configuration profile is integrated into the customized Docker image:


Customized Docker Image - Development
FROM gcr.io/forgeops-public/am...
config
Base Docker Image From ForgeRock (Evaluation-Only)
Customized Configuration Profile

Before you deploy the platform in production, you must change from using ForgeRock's evaluation-only base images to using base images you build yourself. Building your own base images is covered in "*Building Base Docker Images*" in the *Cloud Deployment Guide*.

Developing a Customized Amster Docker Image

With AM up and running, you can iteratively:

- Customize AM's configuration using the console and the REST APIs.
- Capture your configuration changes by synchronizing them from the AM service running on Kubernetes back to the staging area and the master directory for configuration profiles in your `forgeops` repository clone.
- Rebuild the `amster` Docker image.
- Restart the platform.
- Test the deployment based on the updated Docker image.

Perform the following procedure iteratively to develop a customized `amster` Docker image:

To Develop a Customized Amster Docker Image

1. Perform version control activities on your `forgeops` repository clone:

   a. Run the **git status** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

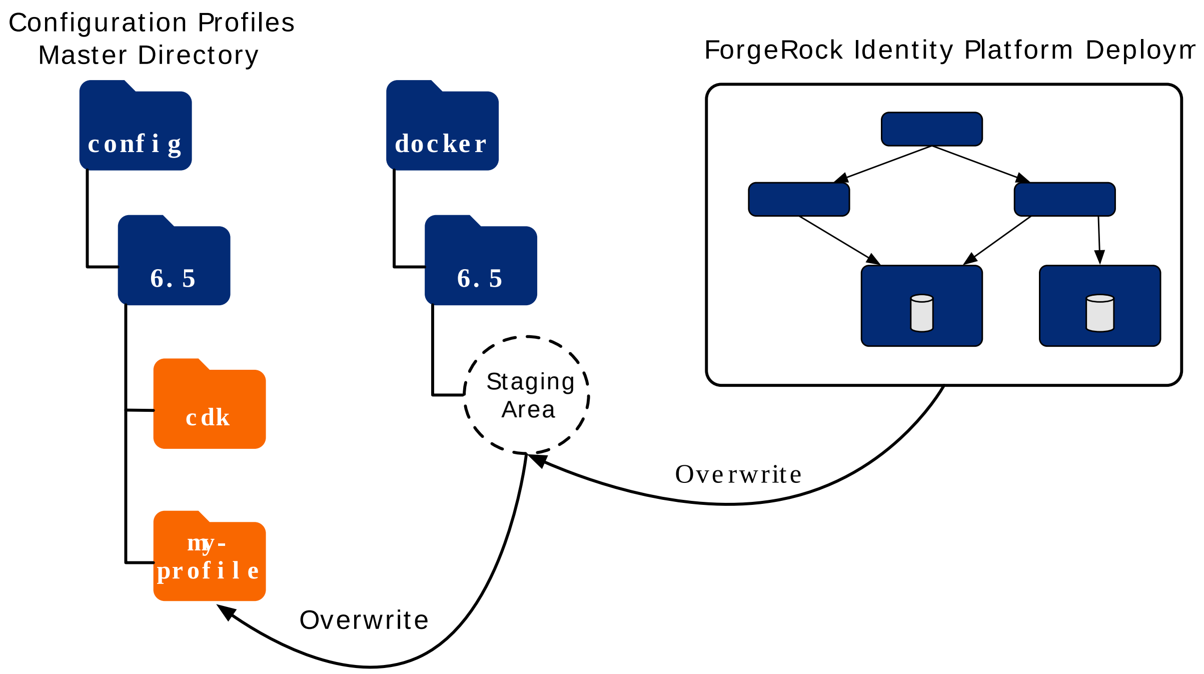
2. Modify the AM configuration using the AM console or the REST APIs.

   For information about how to access the AM console or REST APIs, see "Accessing AM Services".

3. Synchronize the changes you made to the AM configuration to your `forgeops` repository clone:

```
$ cd /path/to/forgeops/bin
$ ./config.sh sync --profile my-profile --component amster --version 6.5
Finding the amster pod
Executing amster export from amster-c684d69f9-q9p5r
Amster OpenAM Shell (7.0.0-SNAPSHOT build @build.number@, JVM: 1.8.0_212)
Type ':help' or ':h' for help.
-------------------------------------------------------------------------
am> :load /tmp/do_export.amster
Export completed successfully
tar: removing leading '/' from member names
```

   The **config.sh sync** command exports the modified AM configuration profile from the running ForgeRock Identity Platform to the staging area. Then, it saves the configuration profile as *my-profile* in the master directory for configuration profiles:



   For more information about the management of ForgeRock Identity Platform configuration profiles in the `forgeops` repository, see "Configuration Profiles".

4. Perform version control activities on your `forgeops` repository clone:

   a. Run the **git status** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

5. Shut down your ForgeRock Identity Platform deployment and delete PVCs used by the deployment from your namespace. See "*Shutting Down Your Deployment*" for details.

6. Redeploy the ForgeRock Identity Platform:

```
$ cd /path/to/forgeops
$ skaffold dev -f skaffold-6.5.yaml
```

7. To validate that AM has the expected configuration, start the console and verify that your configuration changes are present.

Developing a Customized IDM Docker Image

With IDM up and running, you can iteratively:

- Customize IDM's configuration using the Admin UI and the REST APIs.
- Capture your configuration changes by synchronizing them from the IDM service running on Kubernetes back to the staging area and the master directory for configuration profiles in your `forgeops` repository clone.

  Skaffold detects the changes and rebuilds the `idm` Docker image. Then, it restarts IDM.
- Test the deployment based on the updated Docker image.

Perform the following procedure iteratively to develop a customized `idm` Docker image:

To Develop a Customized IDM Docker Image

1. Perform version control activities on your `forgeops` repository clone:

   a. Run the **git status** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

2. Modify the IDM configuration using the IDM Admin UI or the REST APIs.

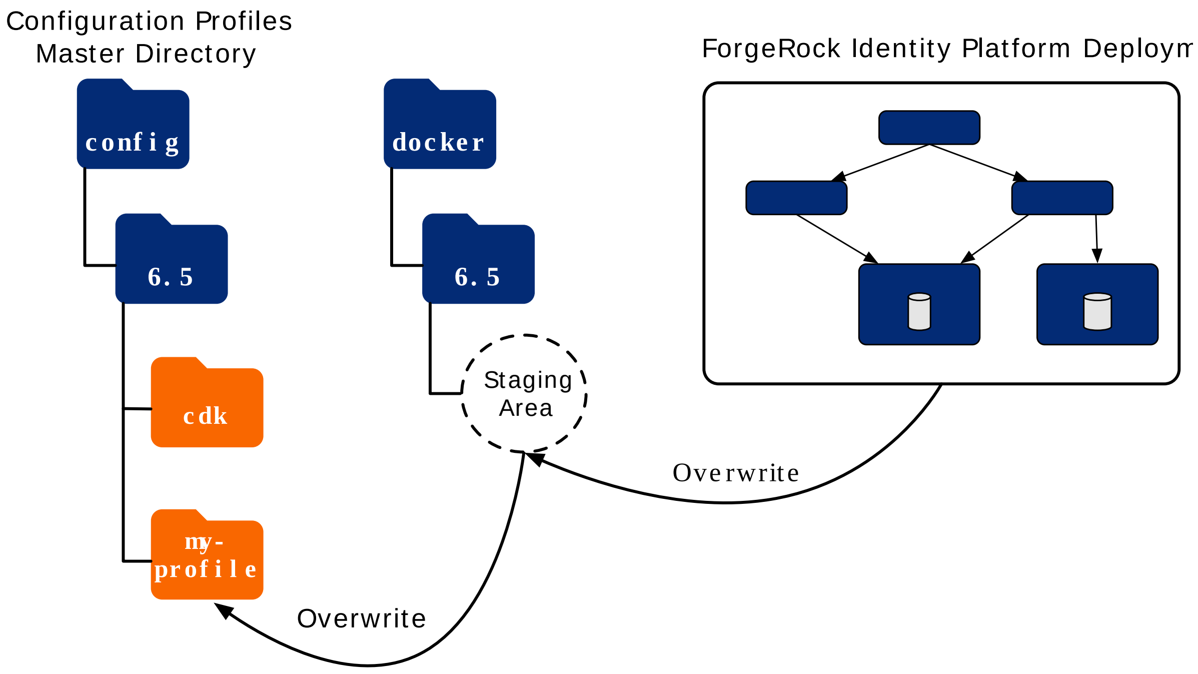
   See "Using Property Value Substitution for Passwords" for important information about configuring passwords in containerized IDM deployments.

   For information about how to access the IDM Admin UI or REST APIs, see "Accessing IDM Services".

3. Synchronize the changes you made to the IDM configuration to your `forgeops` repository clone:

```
$ cd /path/to/forgeops/bin
$ ./config.sh sync --profile my-profile --component idm --version 6.5
tar: Removing leading `/' from member names
```

The **config.sh sync** command exports the modified IDM configuration from the running ForgeRock Identity Platform to the staging area. Then, it saves the configuration profile as *my-profile* in the master directory for configuration profiles:



Skaffold automatically builds a new Docker image for IDM when you export the configuration profile from the running deployment to the staging area. After building the Docker image, it redeploys IDM. For the short period that it takes Skaffold to redeploy IDM, you won't be able to access the IDM Admin UI.

For more information about the management of ForgeRock Identity Platform configurations in the `forgeops` repository, see "Configuration Profiles".

4. Perform version control activities on your `forgeops` repository clone:

   a. Run the **git status** command.

   b. Review the state of the working directory and staging area.

   c. (Optional) Run the **git commit** command to commit changes to files that have been modified.

5. To validate that IDM has the expected configuration, start the Admin UI and verify that your configuration changes are present.

Using Property Value Substitution for Passwords

ForgeRock recommends using property value substitution for all passwords in the IDM configuration when deploying IDM in a container.

To use property value substitution for passwords:

- Specify passwords using configuration expressions in the IDM Admin UI, or when using the REST API.
- Specify passwords' run-time values in the `/path/to/forgeops/docker/6.5/idm/resolver/boot.properties` file.

The following example illustrates how the default IDM configuration in the CDK uses property value substitution for the shared DS repository's password:

- In the IDM configuration, the `repo.ds.json` file specifies properties for the shared DS repository. One of the properties is the bind password for the repository. The value of the password is set to a configuration expression—`&{openidm.repo.password}`.
- The `/path/to/forgeops/docker/6.5/idm/resolver/boot.properties` file sets the run-time value of the `openidm.repo.password` property to `password`.

Property value substitution for passwords eases promotion of the IDM configuration from a development environment to a test or production environment. When passwords are specified without property value substitution, IDM encrypts them before storing them in its configuration.

The encrypted passwords present a problem when you're ready to promote your configuration; the same encryption keys in your development environment must be present in the new environment so that the passwords can be decrypted. Resolving the passwords' values at run-time with property value substitution solves the problem by eliminating the requirement for the same keys to be available in multiple environments. Good security practice requires different encryption keys for different environments.

Caution

Specifying `boot.properties` passwords as described in the preceding section is *extremely insecure*. The passwords appear in cleartext in the `/path/to/forgeops/docker/6.5/idm/resolver/boot.properties` file.

It is expected that a future build of IDM 6.5 will be able to resolve configuration expressions for passwords from password management systems; for example, HashiCorp Vault and Google Cloud Key Management System (KMS). For more information, see IDM issue #13262.

Chapter 6

Shutting Down Your Deployment

When you're done working with your ForgeRock Identity Platform deployment, shut it down and remove it from your namespace as follows:

To Shut Down and Remove a ForgeRock Identity Platform Deployment

1. Navigate to the terminal window where you started Skaffold.
2. Shut down your deployment and remove it from your namespace:
   a. Determine whether the Skaffold process is still running in the foreground.
   b. If the Skaffold process is still running in the foreground, press **CTRL+c** in the terminal window running Skaffold.
   c. If the Skaffold process is no longer running in the foreground, run the **skaffold delete** command.
3. Delete DS persistent volume claims (PVCs) from your namespace:

```
$ kubectl delete pvc --all
persistentvolumeclaim "data-ds-cts-0" deleted
persistentvolumeclaim "data-ds-idrepo-0" deleted
```

Chapter 7
Troubleshooting Your Deployment

Kubernetes deployments are multi-layered and often complex.

Errors and misconfigurations can crop up in a variety of places. Performing a logical, systematic search for the source of a problem can be daunting.

This chapter provides troubleshooting techniques you can use when attempting to resolve an issue:

1. "Verifying Versions of Third-Party Software"
2. "Verifying the Minikube VM's Configuration"
3. "Checking for Sufficient Disk Space"
4. "Enabling kubectl bash Tab Completion"
5. "Generating Kubernetes YAML Files from Kustomize"
6. "Debugging Skaffold Issues"
7. "Reviewing Pod Descriptions and Container Logs"
8. "Accessing Files in Kubernetes Containers"

Verifying Versions of Third-Party Software

ForgeRock recommends installing tested versions of third-party software in environments where you'll run the CDK. See "*Setting up Your Development Environment*" for tested versions of third-party software.

If you used Homebrew to install third-party software, you can use the following commands to obtain software versions:

- Homebrew: **brew list --versions**
- Homebrew casks: **brew cask list --versions**

Verifying the Minikube VM's Configuration

The **minikube start** command example in "Creating a Minikube Virtual Machine" specifies the virtual hardware requirements for a Minikube VM.

Run the **VBoxManage showvminfo "minikube"** command to verify that your Minikube VM meets the stated memory requirement (`Memory Size` in the output), and to gather other information that might be of interest when troubleshooting issues running the CDK in a Minikube environment.

Checking for Sufficient Disk Space

When the Minikube VM runs low on disk space, it acts unpredictably. Unexpected application errors can appear.

Verify that adequate disk space is available by logging in to the Minikube VM and running a command to display free disk space:

```
$ minikube ssh
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           3.9G  383M  3.6G  10% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
tmpfs           3.9G   64K  3.9G   1% /tmp
/dev/sda1        25G  7.7G   16G  33% /mnt/sda1
/Users          465G  219G  247G  48% /Users
$ exit
logout
```

In the preceding example, 16 GB of disk space is available on the Minikube VM.

Enabling kubectl bash Tab Completion

The bash shell contains a feature that lets you use the Tab key to complete file names.

A bash shell extension that provides similar Tab key completion for the **kubectl** command is available. While not a troubleshooting tool, this extension can make troubleshooting easier, because it lets you enter **kubectl** commands more easily.

For more information about the **kubectl** bash Tab completion extension, see *Enabling shell autocompletion* in the Kubernetes documentation.

Note that to install the bash Tab completion extension, you must be running version 4 or later of the bash shell. To determine your bash shell version, run the **bash --version** command.

Generating Kubernetes YAML Files from Kustomize

If you've modified any of the Kustomize bases and overlays that come with the CDK, you might want to see how your changes affect CDK deployment. Use the **kustomize build** command to see how Kustomize expands your bases and overlays into YAML files.

For example:

```
$ cd /path/to/forgeops/kustomize/overlay/6.5
$ kustomize build all
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app: forgeops-secrets
  name: forgeops-secrets-serviceaccount
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    app: forgeops-secrets
  name: forgeops-secrets-role
  namespace: default
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  - configmaps
  verbs:
  - get
  - list
. . .
```

Debugging Skaffold Issues

Skaffold provides different levels of debug logging information. When you encounter issues deploying the platform with Skaffold, you can set the logging verbosity to display more messages. The additional messages might help you identify problems.

For example:

```
$ cd /path/to/forgeops
$ skaffold dev -v debug -f skaffold-6.5.yaml
INFO[0000] starting gRPC server on port 50051
INFO[0000] starting gRPC HTTP server on port 50052
INFO[0000] Skaffold &{Version:v0.38.0 ConfigVersion:skaffold/v1beta14 GitVersion:
 GitCommit:1012d7339d0055ab93d7f88e95b7a89292ce77f6 GitTreeState:clean BuildDate:2019-09-13T02:16:09Z
 GoVersion:go1.13 Compiler:gc Platform:darwin/amd64}
DEBU[0000] config version (skaffold/v1beta12) out of date: upgrading to latest (skaffold/v1beta14)
DEBU[0000] found config for context "minikube"
DEBU[0000] Defaulting build type to local build
DEBU[0000] validating yamltags of struct SkaffoldConfig
DEBU[0000] validating yamltags of struct Metadata
. . .
```

Reviewing Pod Descriptions and Container Logs

Look at pod descriptions and container log files for irregularities that indicate problems.

Pod descriptions contain information about active Kubernetes pods, including their configuration, status, containers (including containers that have finished running), volume mounts, and pod-related events.

Container logs contain startup and run-time messages that might indicate problem areas. Each Kubernetes container has its own log that contains output written to `stdout` by the application running in the container. `am` container logs are especially important for troubleshooting AM issues in Kubernetes deployments: AM writes its debug logs to `stdout`. Therefore, the `am` container logs include all the AM debug logs.

Here's an example of how you can use pod descriptions and container logs to troubleshoot. Events in the pod description indicate that Kubernetes was unsuccessful in pulling a Docker image required to run a container. You can review your Docker registry's configuration to determine whether a misconfiguration caused the problem.

The **debug-logs.sh** script generates the following HTML-formatted output, which you can view in a browser:

- Descriptions of all the Kubernetes pods running the ForgeRock Identity Platform in your namespace
- Logs for all of the containers running in these pods

Perform the following procedure to run the **debug-logs.sh** script and then view the output in a browser:

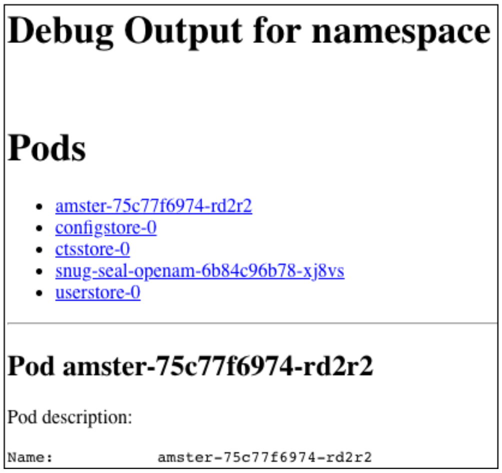
To Run the debug-logs.sh Script

1. Make sure that your namespace is the active namespace in your Kubernetes context.
2. Make sure you've checked out the master branch of the `forgeops` repository.
3. Change to the `/path/to/forgeops/bin` directory in your `forgeops` repository clone.
4. Run the **debug-logs.sh** script:

```
$ ./debug-logs.sh
Generating debug log for namespace my-namespace
rm: /tmp/forgeops/*: No such file or directory
Generating amster-75c77f6974-rd2r2 logs
Generating configstore-0 logs
Generating ctsstore-0 logs
Generating snug-seal-openam-6b84c96b78-xj8vs logs
Generating userstore-0 logs
open file:///tmp/forgeops/log.html in your browser
```

5. In a browser, navigate to the URL shown in the **debug-logs.sh** output. For example, `file:///tmp/forgeops/log.html`. The browser displays a screen with a link for each ForgeRock Identity Platform pod in your namespace:

debug-logs.sh Output



Debug Output for namespace

Pods

- amster-75c77f6974-rd2r2
- configstore-0
- ctsstore-0
- snug-seal-openam-6b84c96b78-xj8vs
- userstore-0

Pod amster-75c77f6974-rd2r2

Pod description:

```
Name:           amster-75c77f6974-rd2r2
```

6. (Optional) To navigate to the information for a pod, select its link from the start of the **debug-logs.sh** output.

   Selecting the link takes you to the pod's description. Logs for each of the pod's containers follow the pod's description.

7. (Optional) To modify the output to contain the latest updates to the pod descriptions and container logs, run the **debug-logs.sh** script again, and then refresh your browser.

Accessing Files in Kubernetes Containers

You can log in to the bash shell of any container in the CDK with the **kubectl exec** command. From the shell, you can access ForgeRock-specific files, such as audit, debug, and application logs, and other files that might help you troubleshoot problems.

For example, access the AM authentication audit log as follows:

```
$ kubectl exec openam-960906639-wrjd8 -c openam -it /bin/bash
bash-4.3$ pwd
/usr/local/tomcat
bash-4.3$ cd
bash-4.3$ pwd
/home/forgerock
bash-4.3$ cd openam/openam/log
bash-4.3$ ls
access.audit.json  activity.audit.json authentication.audit.json config.audit.json
bash-4.3$ cat authentication.audit.json
{"realm":"/","transactionId":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-86","component":"Authentication","eventName":"AM
LOGIN-MODULE-COMPLETED","result":"SUCCESSFUL","entries":[{"moduleId":"Amster","info":
{"authIndex":"service","authControlFlag":"REQUIRED","moduleClass":"Amster","ipAddress":"172.17.0.3","authLevel":"0"
["amadmin"],"timestamp":"2017-09-29T18:14:46.200Z","trackingIds":
["29aac0af-4b62-48cd-976c-3bb5abbed8c8-79"],"_id":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-88"}
{"realm":"/","transactionId":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-86","userId":"id=amadmin,ou=user,dc=openam,dc=fo
LOGIN-COMPLETED","result":"SUCCESSFUL","entries":[{"moduleId":"Amster","info":
{"authIndex":"service","ipAddress":"172.17.0.3","authLevel":"0"}}],"timestamp":"2017-09-29T18:14:46.454Z","tracking
["29aac0af-4b62-48cd-976c-3bb5abbed8c8-79"],"_id":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-95"}
bash-4.3$ exit
```

You can also copy files from a Kubernetes pod to your local system using the **kubectl cp** command. For more information, see the **kubectl** command reference.

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*" in the *Cloud Deployment Guide*.

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".

- 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.

Appendix B. Homebrew Package Names

The following table lists the Homebrew package names for third-party software used in Minikube development environments:

Software	Homebrew (macOS)	Homebrew (Linux)
Docker Desktop[a]	`docker` (cask)	Not available[b]
`kubectl`	`kubernetes-cli`	`kubernetes-cli`
Skaffold	`skaffold`	`skaffold`
Kustomize	`kustomize`	`kustomize`
Kubernetes context switcher (`kubectx`)	`kubectx`	`kubectx`
Kubernetes log display utility (`stern`)	`stern`	`stern`
VirtualBox	`virtualbox` (cask)	Not available[b]
Minikube	`minikube`	`minikube`

[a] Docker Desktop is available for macOS. On Linux computers, install Docker CE instead. For more information, see the Docker documentation.

[b] The Linux version of Homebrew does not support installing software it maintains as casks. Because of this, if you're setting up an environment on Linux, you won't be able to use Homebrew for this package. Instead, refer to the package's documentation for installation instructions.

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.