# Cloud Deployment Model Cookbook for AKS (Evaluation Edition)

/ ForgeRock Identity Platform 6.5

Latest update: 6.5.2

Gina Cariaga
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

Step-by-step instructions for getting the CDM up and running on Azure Kubernetes Service (AKS).

# Table of Contents

Preface ..... iv
1. About the ForgeRock Cloud Deployment Model ..... 1
    1.1. How the CDM Relates to Your Deployment ..... 1
    1.2. About This Evaluation Edition ..... 2
    1.3. CDM Overview ..... 2
    1.4. Third-Party Software Deployed With the CDM ..... 4
    1.5. Best Practices for Implementing the CDM ..... 4
2. Setting Up the Deployment Environment ..... 6
    2.1. Installing Required Third-Party Software ..... 6
    2.2. Cloning the forgeops Repository ..... 6
    2.3. Setting Up an Azure Subscription for the CDM ..... 7
    2.4. Creating and Setting up a Kubernetes Cluster ..... 8
3. Deploying the CDM ..... 15
4. Using the CDM ..... 25
    4.1. Accessing ForgeRock Identity Platform Services ..... 25
    4.2. Monitoring the CDM ..... 28
A. Getting Support ..... 31
    A.1. ForgeRock DevOps Support ..... 31
    A.2. Accessing Documentation Online ..... 32
    A.3. How to Report Problems or Provide Feedback ..... 33
    A.4. Getting Support and Contacting ForgeRock ..... 34
Glossary ..... 35

# Preface

The ForgeRock Cloud Deployment Model (CDM) demonstrates a common use ForgeRock Identity Platform™ architecture installed in a DevOps environment. This guide describes the CDM and its default behaviors, and provides steps for replicating the model on AKS.

## Before You Begin

Before deploying the ForgeRock Identity Platform in a DevOps environment, 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

# About the ForgeRock Cloud Deployment Model

The `forgeops` repository on GitHub contains artifacts you can use to deploy ForgeRock Identity Platform in a cloud environment. The ForgeRock Cloud Deployment Team has developed Helm charts, Docker images, and other artifacts expressly to build the Cloud Deployment Model (CDM). The CDM is a common use ForgeRock Identity Platform architecture, designed to be easy to deploy and easy to replicate.

This chapter explains how the CDM relates to your deployment, and provides an overview of CDM components and architecture.

## 1.1. How the CDM Relates to Your Deployment

Using the CDM artifacts and *CDM Cookbook* instructions, you can quickly get the ForgeRock Identity Platform running in a Kubernetes cloud environment. The CDM is not a deployment template. But you can use the CDM to evaluate deploying and running the ForgeRock Identity Platform in the Azure cloud. Then you can customize your evaluation deployment to simulate your specific business environment. From there, you can develop your own custom template, and write scripts to automate re-deploying your evaluation deployment.

**Standardizes the process.** The ForgeRock Cloud Deployment Team develops CDM artifacts based on interactions with ForgeRock customers. The Team is made up of technical consultants and cloud software developers who encounter common deployment issues at customer sites. Our mission is to standardize a process for deploying ForgeRock Identity Platform natively in the cloud. We began by standardizing on Kubernetes for our cloud platform.

**Simplifies deployment.** We then developed artifacts such as Helm charts and configuration files to simplify the deployment process. We deployed a small Kubernetes cluster in the Azure cloud. We conduct continuous integration and continuous deployment as we add new capabilities and fix problems in the cluster. We maintain, troubleshoot, and tune the system for optimized performance. Most importantly, we are documenting and testing the deployment process.

**Eliminates guesswork.** If you use our CDM artifacts and follow the *CDM Cookbook* instructions without deviation, you can have ForgeRock Identity Platform running with minimum product viability within a relatively short time. The CDM takes the guesswork out of setting up your cloud environment. It bypasses the deploy-test-integrate-test-repeat cycle many customers struggle through when spinning up the ForgeRock Identity Platform in the cloud for the first time.

# 1.2. About This Evaluation Edition

ForgeRock has published multiple *CDM Cookbooks*, each providing instructions for deploying ForgeRock Identity Platform on a specific cloud service. Unlike the other CDM Cookbooks, this evaluation edition for AKS is limited in depth and scope. Although the ForgeRock Identity Platform you deploy in the Azure cloud is fully operational, the deployment will not meet the ForgeRock standard for production quality until these features can be included in the model:

- Multiple-zone high availability
- DS replication
- Backup and recovery

The Cloud Deployment Team has deferred integrating these capabilities into the ForgeRock CDM because AKS does not support multiple availability zones for high-availability clusters at this time.

When high availability for AKS clusters becomes a viable option, we will update the CDM as well as this guide.

# 1.3. CDM Overview

Once you deploy the CDM, the ForgeRock Identity Platform is fully operational within a Kubernetes cluster. Values in the CDM Helm charts override some of the values in the `forgeops` Helm charts. When you deploy CDM, `forgeops` artifacts are overlaid with JVM tuning, memory and CPU limits, and other CDM configurations. Here are some of the benefits of deploying the CDM in your pre-production environment:

**Immutable Kubernetes cluster**

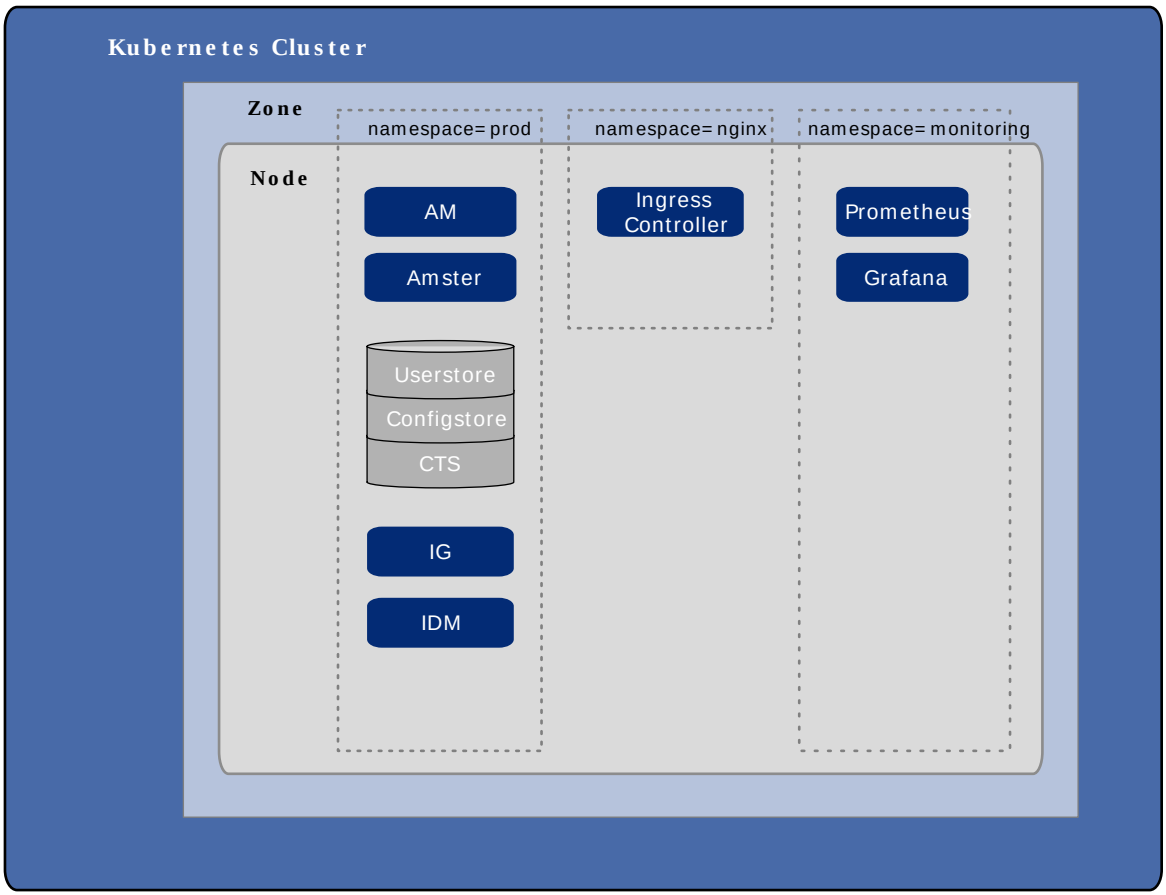
ForgeRock Identity Platform is deployed in a small Kubernetes cluster capable of handling up to 1,000,000 test users. When you want to add new features or update existing configurations, you redeploy the whole cluster. This immutability makes continuous integration and continuous deployment possible.

**Ready-to-use ForgeRock Identity Platform components**

- A DS instance is deployed with separate data stores for users, configuration, and session tokens.
- The `amster` pod facilitates AM deployment by:
  - Obtaining the AM post-installation configuration.
  - Installing the AM server based on configuration from Helm charts in the `forgeops` repository.
  - Importing the post-installation configuration into the AM configuration store.
- An AM instance is deployed and configured to access the DS user, token, configuration stores.

- An IDM instance is deployed and configured to work with the DS user store.
- An IG instance is deployed to extend access to web applications, APIs, devices and things.

*CDM Kubernetes Architecture*




**Secured communication**

The ingress controller is SSL-enabled. SSL is terminated at the ingress controller. Incoming requests and outgoing responses are encrypted.

**StatefulSets**

The CDM uses Kubernetes stateful sets to manage the DS pods. Stateful sets protect against data loss if Kubernetes client containers fail. The AM configuration, policies, and application data reside in the configuration store.

**Authentication and authorization**

ForgeRock Access Management authentication and OAuth 2 authorization are fully enabled.

# 1.4. Third-Party Software Deployed With the CDM

Before you can deploy the CDM, you must install a set of third-party software on your local computer. "Installing Required Third-Party Software" in the *DevOps Release Notes* lists the software you need.

When you deploy the CDM, you deploy not only the ForgeRock Identity Platform in your cluster, but also the following third-party software:

| Software | Purpose | More Information |
|---|---|---|
| Helm (tiller) | Helm chart deployment | https://helm.sh |
| Prometheus | Monitoring and Alerting Toolkit | https://prometheus.io/ |
| Grafana | Monitoring Metrics Visualization | https://grafana.com/ |
| Gatling | Load Testing for Benchmarking | https://gatling.io/ |
| Certificate Manager | Certificate Management | https://docs.cert-manager.io |

# 1.5. Best Practices for Implementing the CDM

As you work through the *CDM Cookbook* instructions, you can leverage some of the best practices the Cloud Deployment Team has developed over time.

## 1.5.1. Begin at the Beginning

Using the *CDM Cookbook* simplifies deploying ForgeRock Identity Platform in the Azure cloud. But if you deviate from the documented steps sequence or customize any of the `forgeops` artifacts, your results will be unpredictable.

## 1.5.2. Provide End-to-End Project Management

Engage at least one deployment professional to provide oversite and guidance during the entire deployment process. We found that at many customer sites, one person architects a deployment plan, then hands off implementation specifications to other team members. Your chances of using the CDM to your best advantage are much higher if everyone on the team understands the context and goals of the CDM.

## 1.5.3. Engage a ForgeRock Cloud Expert

If you intend to implement the CDM as the starting point for a production-quality deployment, then you are probably already working with a ForgeRock technical consultant or partner. If you're not working with a qualified ForgeRock cloud professional, contact your ForgeRock salesperson for more information.

Chapter 2

# Setting Up the Deployment Environment

This chapter describes how to set up your local computer and create a Kubernetes cluster in AKS environment before you deploy the CDM.

The chapter covers the following topics:


- "Installing Required Third-Party Software"
- "Cloning the forgeops Repository"
- "Setting Up an Azure Subscription for the CDM"
- "Creating and Setting up a Kubernetes Cluster"


## 2.1. Installing Required Third-Party Software

Before installing the CDM, you must obtain non-ForgeRock software and install it on your local computer.

The CDM has been validated with specific third-party software versions. Review "Installing Required Third-Party Software" in the *DevOps Release Notes* to determine which software you need. Then install the software on your local computer.

The CDM *might* also work with older or newer versions of the third-party software.

## 2.2. Cloning the forgeops Repository

Before you can deploy the CDM environment, you must clone the `forgeops` repository. The `samples` directory in the `forgeops` repository contains the CDM configuration.

### *To Obtain the forgeops Repository*

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

1. Clone the `forgeops` repository:

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

2. Check out the `release/6.5.2` branch:

```
$ cd forgeops
$ git checkout release/6.5.2
```

# 2.3. Setting Up an Azure Subscription for the CDM

The CDM is deployed and runs in an Azure subscription.

This section outlines the steps that the Cloud Deployment Team performed to configure an Azure subscription for deploying the CDM.

To replicate the Azure subscription configuration, follow this procedure:

### *To Configure an Azure Subscription for the CDM*

1. Assign the following roles to users who will be deploying CDM:
   - Azure Kubernetes Service Cluster Admin Role
   - Azure Kubernetes Service Cluster User Role
   - Contributor
   - User Access Administrator
2. Log in to Azure services using an user account with the roles you have assigned in the previous step.
3. Set up the Azure subscription ID you will use to deploy the CDM:

   ```
   $ az account set --subscription your-subscription-id
   ```

   The resources configured for use with the CDM on AKS are associated with the same Azure subscription.
4. Choose the Azure region in which you will deploy the CDM.

   The Cloud Deployment Team deployed the CDM in the `eastus` region.

   To use any other region, note the following:

   - The region must support AKS.
   - Objects required for your AKS cluster must reside in the same region.
   - You must change the `--location` argument when you run the **az** command in the procedures in this chapter.
5. Create a resource group with the name `ip-resource-group`.

6. Create a static public IP address in Azure with the name `cdm-ip` in the `ip-resource-group` resource group.

   This static public IP address is used later to configure the ingress controller in the CDM.

# 2.4. Creating and Setting up a Kubernetes Cluster

This section describes how to create and set up a Kubernetes cluster that can run the CDM, and covers the following topics:

- "Creating a Cluster"
- "Assigning the Network Contributor Role to the Service Principal"
- "Creating Namespaces"
- "Creating Storage Classes"
- "Configuring RBAC for the Cluster"
- "Deploying an Ingress Controller"
- "Deploying the Certificate Manager"
- "Deploying Monitoring Infrastructure"

## 2.4.1. Creating a Cluster

This section outlines how the Cloud Deployment Team created our cluster. After creating the cluster, the `Network Contributor` role is assigned to the service principal that is used to deploy the CDM.

### *To Create a Cluster for the CDM*

1. Run the **az account show** command and note the ID value, which is your subscription ID.
2. Create a resource group that will contain the resources you need for your cluster. Specify any string as the name of the resource group, and specify the subscription ID in which the resource group should be created.

   ```
   $ az group create \
    --name my-fr-resource-group \
    --location eastus
   ```

3. Obtain the latest patch version of Kubernetes supported in your AKS region:

   ```
   $ az aks get-versions --location eastus
   ```

   Kubernetes patch versions are expressed as *x.y.z*, where *x* is the major version, *y* is the minor version, and *z* is the patch version.

4. Identify values that you will need when you run the command to create a Kubernetes cluster in the next step:
   - **Resource Group**. The name of the resource group you created in the previous step.
   - **Name**. Any string that you want to use as the name of your Kubernetes cluster.
   - **Kubernetes Version**. The latest patch version of Kubernetes supported in your region. Use the patch version you obtained in Step 3 of this procedure.
   - **Tag**. Specify any value to identify who created the cluster.
5. Using the **az aks create** command, create a cluster for your deployment:

```
$ az aks create \
 --resource-group my-fr-resource-group \
 --name my-fr-cluster \
 --admin-username forgerock \
 --location eastus \
 --kubernetes-version "kubernetes-version" \
 --node-vm-size "Standard_D4s_v3" \
 --node-osdisk-size 50 \
 --node-count 1 \
 --tag "createdby=my-user-name"  \
 --enable-addons monitoring \
 --generate-ssh-keys
```

6. To ensure that you are working on the correct Kubernetes cluster, set the Kubernetes context to the cluster you created:

```
$ az aks get-credentials \
 --resource-group my-fr-resource-group \
 --name my-fr-cluster \
 --subscription your subscription ID \
 --overwrite-existing
```

You can write your own script to automate cluster creation. For an example script, see `aks-create-cluster.sh`, which is called by the example environment setup script, `aks-up.sh`.

## 2.4.2. Assigning the Network Contributor Role to the Service Principal

During cluster creation, AKS automatically creates a service principal in the resource group in which you created the cluster. After you create the cluster, you must assign the `Network Contributor` role of the IP resource group to this service principal.

### *To Assign the Network Contributor Role to the Service Principal*

1. Obtain the `clientId` value for the service principal:

```
$ az aks list --resource-group my-fr-resource-group | grep -i clientId
```

2. Obtain the Azure subscription ID:

```
$ az account show
```

The `ID` value is your subscription ID.

3. Assign the `Network Contributor` role to the service principal:

```
$ az role assignment create \
 --assignee your-clientID \
 --role "Network Contributor" \
 --scope /subscriptions/your-subscriptionID/resourceGroups/ip-resource-group
```

You can write your own script to automate assigning the `Network Contributor` role. For an example script, see `aks-assign-role-to-sp.sh`, which is called by the example environment setup script, `aks-up.sh`.

## 2.4.3. Creating Namespaces

The CDM uses the `monitoring` and `prod` namespaces. The `monitoring` namespace contains monitoring and notification tools and related resources. The `prod` namespace contains all the other resources of the CDM.

For more information on Kubernetes namespaces, see Namespaces in the Kubernetes Concepts documentation.

To create the namespaces required in CDM, perform the following procedure:

### *To Create Namespaces for CDM*

1. Create the `monitoring` namespace:

```
$ kubectl create namespace monitoring
```

2. Create the `prod` namespace:

```
$ kubectl create namespace prod
```

3. Set context to the `prod` namespace:

```
$ kubens prod
```

You can write your own script to automate namespace creation. A section of the `aks-up.sh` script contains example commands.

## 2.4.4. Creating Storage Classes

Kubernetes storage classes named `fast` and `standard` define the characteristics of the storage volumes used in an environment. In CDM, the `userstore` pod and `ctsstore` pod use the `fast` storage class, and `configstore` pod uses the `standard` storage class.

For more information on Kubernetes storage classes, see Storage Classes in the Kubernetes Concepts documentation.

To replicate creating storage classes used by CDM, perform the following procedure:

*To Create Storage Classes*

- To create two storage classes named `standard` and `fast`, run the following **kubectl** command:

```
$ kubectl create -f - <<EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: managed
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Premium_LRS
  kind: Managed
EOF
```

You can write your own script to automate storage class creation. For an example script, see `aks-create-sc.sh`, which is called by the example environment setup script, `aks-up.sh`.

## 2.4.5. Configuring RBAC for the Cluster

Role-based access control (RBAC) in Kubernetes enables you to control how users access the API resources running on your cluster.

For information about RBAC in Kubernetes, see Using RBAC Authorization in the Kubernetes documentation.

To replicate the RBAC implemented in CDM, perform the following procedure:

*To Configure RBAC*

1. Create the `tiller` service account:

```
$ kubectl --namespace kube-system create serviceaccount tiller
```

2. Bind the `tiller` service account to the `cluster-admin` role:

```
$ kubectl create clusterrolebinding tiller --clusterrole cluster-admin \
 --serviceaccount=kube-system:tiller
```

3. Initialize the `tiller` service account:

```
$ helm init --wait --service-account tiller
```

You can write your own script to automate RBAC configuration. For an example script, see `helm-rbac-init.sh`, which is called by the example environment setup script, `aks-up.sh`.

## 2.4.6. Deploying an Ingress Controller

An ingress controller must be running in the Kubernetes cluster before the ingress rules are applied and cluster resources can be accessed.

For information about the ingress controller used in CDM, see NGINX Ingress Controller.

While the ingress controller can be configured with or without a static IP, for performance and ease of administration, the CDM uses a reserved static IP address for its ingress controller. The static IP address resides in the same region as the CDM deployment.

To replicate CDM ingress controller deployment in your Kubernetes cluster, follow this procedure:

### *To Deploy the Ingress Controller*

1. Get the static public IP address you created in `ip-resource-group`:

```
$ az network public-ip show \
 --resource-group ip-resource-group \
 --name cdm-ip --query ipAddress --output tsv
```

2. Run the following **helm install** command to deploy the ingress controller:

```
$ 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.service.annotations."service\.beta\.kubernetes\.io/azure-load-balancer-resource-
group"=ip-resource-group \
 --set controller.image.tag="0.21.0" \
 --set controller.service.loadBalancerIP=IP value from the previous command \
 stable/nginx-ingress
```

You can write your own script to automate ingress controller deployment. For an example script, see `aks-create-ingress-cntlr.sh`, which is called by the example environment setup script, `aks-up.sh`.

## 2.4.7. Deploying the Certificate Manager

In the CDM, certificate management is provided by the `cert-manager` add-on. In the AKS environment, CDM is configured to use the self-signed certificate issued by `cert-manager` to secure communication.

*To Deploy the Certificate Manager*

1. Deploy the Kubernetes `cert-manager` add-on:

```
$ helm install stable/cert-manager --namespace kube-system --version v0.5.0
NAME:   brawny-marmot
LAST DEPLOYED: Tue Jul 30 13:46:55 2019
NAMESPACE: kube-system
STATUS: DEPLOYED
. . .
```

2. Run the **kubectl get pods -n kube-system** command to determine when the certificate manager is available.

   Examine the **kubectl get pods** output and look for a pod whose name contains the string `cert-manager`. If this pod has `Running` status, the certificate manager is available. Note that you might have to run the **kubectl get pods** command several times before the `cert-manager` pod attains `Running` status.

## 2.4.8. Deploying Monitoring Infrastructure

The CDM uses Prometheus and Grafana for CDM monitoring and reporting.

To replicate CDM monitoring in your cluster, perform the following procedure:

*To Deploy CDM Monitoring Tools*

1. Refresh your local Helm repository cache:

```
$ helm repo update
```

2. Change to the directory that contains configuration values for installing Prometheus:

```
$ cd /path/to/forgeops/etc/prometheus-values
```

3. Install the `prometheus-operator` chart:

```
$ helm install stable/prometheus-operator \
 --name monitoring-prometheus-operator --values prometheus-operator.yaml \
 --set=rbac.install=true --namespace=monitoring
NAME:   monitoring-prometheus-operator
LAST DEPLOYED: Fri Aug  2 10:07:41 2019
NAMESPACE: monitoring
STATUS: DEPLOYED
. . .
```

4. Change to the directory that contains the `forgerock-metrics` Helm chart:

```
$ cd /path/to/forgeops/helm
```

5. Install the `forgerock-metrics` chart:

```
$ helm install forgerock-metrics \
 --name=monitoring-forgeops-metrics \
 --set=rbac.install=true --namespace=monitoring
NAME:   monitoring-forgerock-metrics
LAST DEPLOYED: Fri Aug  2 10:08:25 2019
NAMESPACE: monitoring
STATUS: DEPLOYED
. . .
```

After the CDM is completely deployed, you can access the monitoring dashboards. For instructions for accessing CDM monitoring dashboards, see "Monitoring the CDM".

You can write your own script to automate monitoring infrastructure deployment. For an example script, see `deploy-prometheus.sh`, which is called by the example environment setup script, `aks-up.sh`.

Chapter 3
# Deploying the CDM

Now that you've set up your deployment environment following the instructions in the previous chapter, you're ready to deploy the CDM. This chapter shows you how to deploy the CDM in your Kubernetes cluster using artifacts from the `forgeops` repository.

Perform the following procedures:

1. "To Customize the Deployment YAML Files"
2. "To Set Your Kubernetes Context"
3. "To Deploy Supporting Components"
4. "To Deploy DS"
5. "To Deploy AM"
6. "To Deploy IDM"
7. "To Deploy IG"

You can write your own script to automate completing all these procedures. For an example script, see `deploy.sh`.

After successfully deploying the CDM, you can access AM, IDM, and DS using graphical and command-line interfaces. For more information, see "*Using the CDM*".

When you're finished working with the CDM, you can remove it from your cluster. See "Deleting ForgeRock Identity Platform From Your Namespace" in the *DevOps Developer's Guide* for details.

### *To Customize the Deployment YAML Files*

Make the following changes to the Helm charts in your `forgeops` repository clone:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. In your clone of the `forgeops` repository, change to the `/path/to/forgeops/samples/config/dev/mini` directory.
3. Make the following changes to the `common.yaml` file:
   - Change the value of the `domain` key to your own domain, making sure you put a period before the domain name.

- Change the value of the `fqdn` key to `login.prod.domain`, where *`domain`* is your own domain.

For example:

```
domain: .mydomain.com
fqdn: login.prod.mydomain.com
. . .
```

### *To Set Your Kubernetes Context*

Ensure that your Kubernetes context is set to the Kubernetes cluster and namespace in which you want to deploy the CDM:

1. Run the **kubectx** command to determine the Kubernetes cluster in your current context:

   ```
   $ kubectx
   ```

2. Examine the **kubectx** command output. Highlighted text shows which Kubernetes cluster is set in your current context.

   If the Kubernetes cluster in your current context does not match the cluster you want to deploy to, set the cluster in your current context to the correct cluster. See "Setting up a Kubernetes Context" in the *DevOps Developer's Guide*.

3. Run the **kubens** command to determine the Kubernetes namespace in your current context. For example:

   ```
   $ kubens
   default
   kube-public
   kube-system
   monitoring
   nginx
   prod
   ```

4. Examine the **kubens** command output. Highlighted output text shows which Kubernetes namespace is set in your current context.

   If the Kubernetes namespace in your current context is not the `prod` namespace, run the following command to set the namespace in your current context:

   ```
   $ kubens prod
   ```

### *To Deploy Supporting Components*

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. In your clone of the `forgeops` repository, change to the `/path/to/forgeops/samples/config/dev/mini` directory.
3. Install the `frconfig` chart, which creates Kubernetes objects used by other ForgeRock pods:

```
$ helm install --name prod-frconfig --namespace prod \
 --values common.yaml --values frconfig.yaml /path/to/forgeops/helm/frconfig
NAME:   prod-frconfig
LAST DEPLOYED: Wed Jul 31 16:00:31 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME      AGE
frconfig  0s

==> v1/ConfigMap
frconfig  0s
```

### *To Deploy DS*

Install the `ds` chart to create a single DS server for the configuration, user, and CTS stores:

1. Install the AM configuration store:

```
$ helm install --name prod-configstore --namespace prod \
 --values common.yaml --values configstore.yaml /path/to/forgeops/helm/ds
NAME:   prod-configstore
LAST DEPLOYED: Wed Jul 31 16:13:57 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME         AGE
configstore  0s

==> v1/Service
configstore  0s

==> v1beta1/StatefulSet
configstore  0s

==> v1/Pod(related)

NAME           READY  STATUS   RESTARTS  AGE
configstore-0  0/1    Pending  0         0s

==> v1/Secret

NAME         AGE
configstore  0s
```

2. Review the logs from the `configstore-0` pod to verify that the DS pod started up successfully.

   For example, the following command verifies that the `configstore-0` pod started successfully:

```
$ kubectl logs configstore-0 | grep successfully
[31/Jul/2019:22:50:50 +0000] category=CORE severity=NOTICE msgID=135 msg=The Directory Server has
 started successfully
[31/Jul/2019:22:50:50 +0000] category=CORE severity=NOTICE msgID=139 msg=The Directory Server has
 sent an alert notification generated by class org.opends.server.core.DirectoryServer (alert type org
.opends.server.DirectoryServerStarted, alert ID org.opends.messages.core-135): The Directory Server
 has started successfully
```

This step is complete when you have verified that the pod started successfully.

### *To Deploy AM*

1. Install the `openam` chart:

```
$ helm install --name prod-openam --namespace prod \
 --values common.yaml /path/to/forgeops/helm/openam
NAME:   prod-openam
LAST DEPLOYED: Thu Aug  1 12:26:58 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME          AGE
am-configmap  1s
boot-json     1s

==> v1/Service
openam  1s

==> v1beta1/Deployment
prod-openam-openam  1s

==> v1beta1/Ingress
openam  1s

==> v1/Pod(related)

NAME                                READY  STATUS    RESTARTS  AGE
prod-openam-openam-6f846bbf69-gt82c  0/1    Init:0/3  0         1s

==> v1/Secret

NAME            AGE
openam-secrets  1s
```

2. Install the `amster` chart:

```
$ helm install --name prod-amster --namespace prod \
 --values common.yaml --values amster.yaml /path/to/forgeops/helm/amster
NAME:   prod-amster
LAST DEPLOYED: Thu Aug  1 12:28:00 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME            AGE
amster-secrets  0s

==> v1/ConfigMap
amster-config       0s
amster-prod-amster  0s

==> v1beta1/Deployment
amster  0s

==> v1/Pod(related)

NAME                      READY  STATUS    RESTARTS  AGE
amster-7b8b6d9667-hxq57  0/2    Init:0/1  0         0s
```

3. Check the status of the pods in the deployment until all pods are ready:

   a. Run the **kubectl get pods** command:

   ```
   $ kubectl get pods
   NAME                                    READY     STATUS    RESTARTS   AGE
   amster-7b8b6d9667-hxq57                 2/2       Running   0          1m
   configstore-0                           1/1       Running   0          10m
   prod-openam-openam-6f846bbf69-gt82c     1/1       Running   0          2m
   ```

   b. Review the output. Deployment is complete when:

   - The `READY` column indicates all containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
   - All entries in the `STATUS` column indicate `Running`.

   c. If necessary, continue to query your deployment's status until all the pods are ready.

4. Review the Amster pod's log to determine whether the deployment completed successfully.

   Use the **kubectl logs amster-xxxxxxxxxx-yyyyy -c amster -f** command to stream the Amster pod's log to standard output.

   The deployment clones the Git repository containing the initial AM configuration. Before the AM and DS servers become available, the following output appears:

```
. . .
+ ./amster-install.sh
Waiting for AM server at http://openam:80/openam/config/options.htm
Got Response code 000
response code 000. Will continue to wait
Got Response code 000
response code 000. Will continue to wait
Got Response code 000
. . .
```

When Amster starts to configure AM, the following output appears:

```
. . .
Got Response code 200
AM web app is up and ready to be configured
About to begin configuration
Executing Amster to configure AM
Executing Amster script /opt/amster/scripts/00_install.amster
Aug  1, 2019 4:51:29 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /opt/amster/scripts/00_install.amster
08/01/2019 04:51:32:699 PM GMT: Checking license acceptance...
08/01/2019 04:51:32:700 PM GMT: License terms accepted.
08/01/2019 04:51:32:706 PM GMT: Checking configuration directory /home/forgerock/openam.
08/01/2019 04:51:32:707 PM GMT: ...Success.
08/01/2019 04:51:32:712 PM GMT: Tag swapping schema files.
08/01/2019 04:51:32:747 PM GMT: ...Success.
08/01/2019 04:51:32:747 PM GMT: Loading Schema odsee_config_schema.ldif
08/01/2019 04:51:32:825 PM GMT: ...Success.
08/01/2019 04:51:32:825 PM GMT: Loading Schema odsee_config_index.ldif
08/01/2019 04:51:32:855 PM GMT: ...Success.
08/01/2019 04:51:32:855 PM GMT: Loading Schema cts-container.ldif
08/01/2019 04:51:32:945 PM GMT: ...Success
.
. . .
```

The following output indicates that deployment is complete:

```
. . .
08/01/2019 04:51:53:215 PM GMT: Setting up monitoring authentication file.
Configuration complete!
Executing Amster script /opt/amster/scripts/01_import.amster
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /opt/amster/scripts/01_import.amster
Importing directory /git/config/default/am/empty-import
Import completed successfully
Configuration script finished
+ pause
+ echo Args are 0
+ echo Container will now pause. You can use kubectl exec to run export.sh
+ true
+ sleep 1000000
Args are 0
Container will now pause. You can use kubectl exec to run export.sh
```

5. Delete the AM pod to force an AM server restart.

   After the restart, AM uses the appropriate CTS configuration.

   a. Run the **kubectl get pods** command to get the AM pod's name.

      The name of the AM pod starts with the string `prod-openam`.

   b. Delete the pod:

      ```
      $ kubectl delete pod openam-pod
      ```

   c. Wait several seconds and then run the **kubectl get pods** command again. Observe that a new AM pod has started up.

   When you installed the `openam` chart in Step 1, the AM server started up before the configuration was available. The `amster` chart then installed AM configuration, including the CTS configuration. But the CTS configuration is not hot-swappable; AM cannot change the CTS configuration without a restart.

### *To Deploy IDM*

1. Install the `postgres-openidm` chart:

```
$ helm install --name prod-postgres-openidm --namespace prod \
 --values common.yaml /path/to/forgeops/helm/postgres-openidm
NAME:   prod-postgres-openidm
LAST DEPLOYED: Thu Aug  1 12:41:53 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/PersistentVolumeClaim
NAME              AGE
```

```
postgres-openidm  1s

==> v1/Service
postgresql  1s

==> v1beta1/Deployment
postgres-openidm  1s

==> v1/Pod(related)

NAME                               READY  STATUS   RESTARTS  AGE
postgres-openidm-6b6fb898cb-mthrj  0/1    Pending  0         1s

==> v1/Secret

NAME              AGE
postgres-openidm  1s

==> v1/ConfigMap
openidm-sql  1s


NOTES:
PostgreSQL can be accessed via port 5432 on the following DNS name from within your cluster:
prod-postgres-openidm-postgres-openidm.prod.svc.cluster.local

To get your user password run:

    PGPASSWORD=$(printf $(printf '\%o' `kubectl get secret --namespace prod prod-postgres-openidm-
postgres-openidm -o jsonpath="{.data.postgres-password[*]}"`);echo)

To connect to your database run the following command (using the env variable from above):

   kubectl run prod-postgres-openidm-postgres-openidm-client --rm --tty -i --image postgres \
   --env "PGPASSWORD=$PGPASSWORD" \
   --command -- psql -U openidm \
   -h prod-postgres-openidm-postgres-openidm postgres
```

2. Install the `openidm` chart:

```
$ helm install --name prod-openidm --namespace prod \
 --values common.yaml /path/to/forgeops/helm/openidm
NAME:   prod-openidm
LAST DEPLOYED: Thu Sep 27 12:46:24 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                 AGE
openidm-secrets-env  2s
openidm-secrets      2s

==> v1/ConfigMap
prod-openidm-openidm    1s
idm-boot-properties     1s
idm-logging-properties  1s

==> v1/Service
```

```
openidm  1s

==> v1beta1/StatefulSet
prod-openidm-openidm  1s

==> v1beta1/Ingress
openidm  1s

==> v1/Pod(related)

NAME                                READY  STATUS    RESTARTS  AGE
prod-openidm-openidm-0              0/2    Init:0/1  0         1s


NOTES:
OpenIDM should be available soon at the ingress address of http://openidm.prod.example.com

It can take a few minutes for the ingress to become ready.
```

3. Check the status of the pods in the deployment until all pods are ready:

   a. Run the **kubectl get pods** command:

   ```
   $ kubectl get pods
   NAME                                    READY     STATUS    RESTARTS   AGE
   amster-7b8b6d9667-hxq57                 2/2       Running   0          31m
   configstore-0                           1/1       Running   0          40m
   postgres-openidm-6b6fb898cb-mthrj       1/1       Running   0          17m
   prod-openam-openam-6f846bbf69-867rc     1/1       Running   0          24m
   prod-openidm-openidm-696685d8cb-pt5x4   2/2       Running   0          12m
   ```

   b. Review the output. Deployment is complete when:

   - The `READY` column indicates all containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
   - All entries in the `STATUS` column indicate `Running`.

   c. If necessary, continue to query your deployment's status until all the pods are ready.

*To Deploy IG*

1. Install the `openig` chart:

```
$ helm install --name prod-openig --namespace prod \
 --values common.yaml /path/to/forgeops/helm/openig
NAME:   prod-openig
LAST DEPLOYED: Thu Sep 27 12:46:24 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
openig  1s

==> v1beta1/Deployment
prod-openig-openig  1s

==> v1beta1/Ingress
openig  1s

==> v1/Pod(related)
NAME                                READY  STATUS             RESTARTS  AGE
prod-openig-openig-9b4dfb574-jdp4n  0/1    ContainerCreating  0         0s


NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace prod -l "app=prod-openig-openig" -o jsonpath="{
.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:8080


If you have an ingress controller, you can also access IG at
http://openig.prod.example.com
```

2. Check the status of the pods in the deployment until all pods are ready:

   a. Run the **kubectl get pods** command:

   ```
   $ kubectl get pods
   NAME                                  READY     STATUS    RESTARTS   AGE
   amster-7b8b6d9667-hxq57               2/2       Running   0          31m
   configstore-0                         1/1       Running   0          40m
   postgres-openidm-6b6fb898cb-mthrj     1/1       Running   0          17m
   prod-openam-openam-6f846bbf69-867rc   1/1       Running   0          24m
   prod-openidm-openidm-696685d8cb-pt5x4 2/2       Running   0          12m
   prod-openig-openig-9b4dfb574-jdp4n    1/1       Running   0          1m
   ```

   b. Review the output. Deployment is complete when:

      - The `READY` column indicates all containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
      - All entries in the `STATUS` column indicate `Running`.

   c. If necessary, continue to query your deployment's status until all the pods are ready.

Chapter 4

# Using the CDM

This chapter shows you how to access and monitor the ForgeRock Identity Platform components that make up the CDM.

## 4.1. Accessing ForgeRock Identity Platform Services

This section shows you how to access ForgeRock Identity Platform components within the CDM.

AM, IDM, and IG are configured for access through the CDM cluster's Kubernetes ingress controller. You can access these components using their normal interfaces:

- For AM, the console and REST APIs.
- For IDM, the Admin UI and REST APIs.
- For IG, HTTP URLs. Note that because the CDM deploys IG in production mode, IG Studio is not available.

DS cannot be accessed through the ingress controller, but you can use Kubernetes methods to access the DS pods.

For more information about how AM, IDM, and IG have been configured in the CDM, see the 6.5 CDM README file in the `forgeops-init` repository.

### 4.1.1. Before You Begin

The URLs you use to access ForgeRock Identity Platform services must be resolvable from your local computer.

If DNS does not resolve the following hostnames, add entries for them to your `/etc/hosts` file:

- `login.prod.`*`my-domain`*
- `openidm.prod.`*`my-domain`*
- `openig.prod.`*`my-domain`*

### 4.1.2. 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. By default, the `amster` Helm chart dynamically generates a random password for the `amadmin` user. It stores the password in the `amster-config` configmap.

   To obtain the password, look for the `adminPwd` value in the `amster-config` configmap:

   ```
   $ kubectl get configmaps amster-config -o yaml | grep adminPwd
    --adminPwd "74xW6IShJF" \
   ```

2. (Optional) Delete the `amster-config` configmap so that the `amadmin` password is not publicly available.

   ```
   $ kubectl delete configmaps amster-config
   ```

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

4. Navigate to the AM deployment URL, https://login.prod.*my-domain*/XUI/?service=adminconsoleservice.

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

   AM prompts you to log in.

5. Log in to AM as the `amadmin` user.

### *To Access the AM REST APIs*

1. Start a terminal window session.

2. If you have not already done so, obtain the password for the `amadmin` user. To obtain the password, perform Step 1 in "To Access the AM Console".

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

```
$ curl \
 --request POST \
 --insecure \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: amadmin password" \
 --header "Accept-API-Version: resource=2.0" \
 --data "{}" \
 'https://login.prod.my-domain/json/realms/root/authenticate?
service=ldapService&authIndexType=service&authIndexValue=ldapService'
{
    "tokenId":"AQIC5wM2...",
    "successUrl":"/console",
    "realm":"/"
}
```

### 4.1.3. Accessing IDM Services

In the CDM, IDM is deployed with a read-only configuration. You can access the IDM Admin UI and REST APIs, but you cannot use them to modify the IDM configuration.

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. Navigate to the IDM Admin UI URL, https://openidm.prod.*my-domain*/admin.

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

   IDM prompts you to log in.
3. Log in to the IDM Admin UI as the `openidm-admin` user with password `openidm-admin`.

#### *To Access the IDM REST APIs*

1. Start a terminal window sesion.
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: openidm-admin" \
 --data "{}" \
 https://openidm.prod.my-domain/openidm/info/ping
{
    "_id":" ",
    "_rev":"",
    " shortDesc":"OpenIDM ready",
    "state":"ACTIVE_READY"
}
```

### 4.1.4. Accessing IG Services

In the CDM, IG is deployed in production mode, and therefore, IG Studio is disabled. You can access IG using a browser to verify that it is operational:

#### *To Verify IG Is Operational*

1. Open a new window or tab in a web browser.
2. Navigate to https://openig.prod.*my-domain*.

   The Kubernetes ingress controller handles the request, routing it to IG.

   You should see a message similar to the following:

   ```
   hello and Welcome to OpenIG. Your path is /. OpenIG is using the default handler for this route.
   ```

### 4.1.5. Accessing DS

The DS pods in the CDM 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, for example **ldapsearch**. You can also use an LDAP editor such as Apache Directory Studio to access the directory.

For all directory pods in the CDM, the directory manager entry is `cn=Directory Manager` and the password is `password`.

## 4.2. Monitoring the CDM

This section contains procedures for accessing Grafana dashboards and the Prometheus web UI:

- "To Access Grafana Dashboards"
- "To Access the Prometheus Web UI"

### *To Access Grafana Dashboards*

1. Forward port 3000 on your local computer to port 3000 on the Grafana web server:

```
$ kubectl \
 port-forward \
 $(kubectl get pods --selector=app=grafana \
 --output=jsonpath="{.items..metadata.name}" --namespace=monitoring) \
 3000 --namespace=monitoring
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

2. In a web browser, navigate to http://localhost:3000 to start the Grafana user interface.

   For information about the Grafana UI, see the Grafana documentation.

3. When you're done using the Grafana UI, enter Cntl+c in the terminal window to stop port forwarding.

### *To Access the Prometheus Web UI*

1. Forward port 9090 on your local computer to port 9090 on the Prometheus web server:

```
$ kubectl \
 port-forward \
 $(kubectl get pods --selector=app=prometheus \
 --output=jsonpath="{.items..metadata.name}" --namespace=monitoring) \
 9090 --namespace=monitoring
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

2. In a web browser, navigate to http://localhost:9090.

   The Prometheus web UI appears in the browser.

   To use the Prometheus web UI, see the Prometheus documentation.

3. When you're done using the Prometheus web UI, enter Cntl+c in the terminal window to stop port forwarding.

To enable monitoring in the CDM, see:

- "Deploying Monitoring Infrastructure" in the *Cloud Deployment Model Cookbook for GKE*
- "Deploying Monitoring Infrastructure" in the *Cloud Deployment Model Cookbook for Amazon EKS*

For a description of the CDM monitoring architecture and information about how to customize CDM monitoring, see:

- "*Monitoring Your Deployment*" in the *Site Reliability Guide for GKE*
- "*Monitoring Your Deployment*" in the *Site Reliability Guide for Amazon EKS*

# Appendix A. Getting Support

This appendix contains information about support options for the ForgeRock DevOps Examples and the ForgeRock Identity Platform.

## A.1. ForgeRock DevOps Support

ForgeRock has developed artifacts in the `forgeops` and `forgeops-init` Git repositories for the purpose of deploying the ForgeRock Identity Platform in the cloud. The companion ForgeRock DevOps documentation provides examples, including 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.

### A.1.1. Commercial Support

ForgeRock provides commercial support for the following DevOps resources:

- Dockerfiles and Helm charts in the `forgeops` Git repository
- 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 Message Broker Release Notes*
- *ForgeRock Identity Gateway Release Notes*

### A.1.2. Support Limitations

ForgeRock provides no commercial support for the following:

- Artifacts other than Dockerfiles or Helm charts in the `forgeops` and `forgeops-init` repositories. 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, and so forth.
- Production deployments that use the DevOps 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 Docker images for the ForgeRock Identity Platform, see "*Building and Pushing Docker Images*" in the *DevOps Developer's Guide*.

### A.1.3. 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 Minishift, 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.

## A.2. 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 product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

## A.3. 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 or Helm charts in the DevOps Examples or the CDM.

If you have questions regarding the DevOps Examples 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 "Running the debug-logs.sh Script" in the *DevOps Developer's Guide*.
- 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).
    - Kubernetes Helm (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.

## A.4. 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.<br><br>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.<br><br>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.<br><br>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.<br><br>Source: *AWS IAM Authenticator for Kubernetes* `README` file on `GitHub`. |

| | |
|---|---|
| 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.<br><br>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 Helm charts, 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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>Source: *Docker daemon* section in the Docker Overview documentation. |
| Docker Engine | The Docker Engine is a client-server application with these components:<br><br>• A server, which is a type of long-running program called a daemon process (the `dockerd` command)<br><br>• A REST API, which specifies interfaces that programs can use to talk to the daemon and tell it what to do<br><br>• A command-line interface (CLI) client (the `docker` command)<br><br>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.<br><br>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.<br><br>An image is an application you would like to run. A container is a running instance of an image.<br><br>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.<br><br>An image is an application you would like to run. A container is a running instance of an image.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>Source: *Kubernetes components* in the Kubernetes Concepts documentation. |
| Kubernetes | Kubernetes is an open source platform designed to automate deploying, scaling, and operating application containers.<br><br>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.<br><br>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:<br>• `default`: The default namespace for user created objects which don't have a namespace<br>• `kube-system`: The namespace for objects created by the Kubernetes system<br>• `kube-public`: The automatically created namespace that is readable by all users<br>Kubernetes supports multiple virtual clusters backed by the same physical cluster.<br>Source: *Namespaces* in the Kubernetes Concepts documentation. |
| Let's Encrypt | Let's Encrypt is a free, automated, and open certificate authority.<br>Source: Let's Encrypt web site. |
| network policy | A Kubernetes network policy specifies how groups of pods are allowed to communicate with each other and with other network endpoints.<br>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.<br>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.<br>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.<br>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<br>• Mounted many times for read-only access<br>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.<br>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.<br>Source: *Understanding Pods* in the Kubernetes Concepts documentation. |
| replication controller | 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.<br>Source: *ReplicationController* in the Kubernetes Concepts documentation. |
| 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.<br>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.<br>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.<br>Source: *Services* in the Kubernetes Concepts 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.<br><br>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.<br><br>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.<br><br>Source: *StackSets Concepts* in the AWS documentation. |
| 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.<br><br>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.<br><br>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.<br><br>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.<br><br>Source: *Understanding Pods* in the Kubernetes Concepts documentation. |