Cloud Deployment Model Cookbook for AKS

/ 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

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

Table of Contents

Preface

The ForgeRock Cloud Deployment Model (CDM) demonstrates a common use ForgeRock Identity Platform™ architecture installed on Kubernetes. 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 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

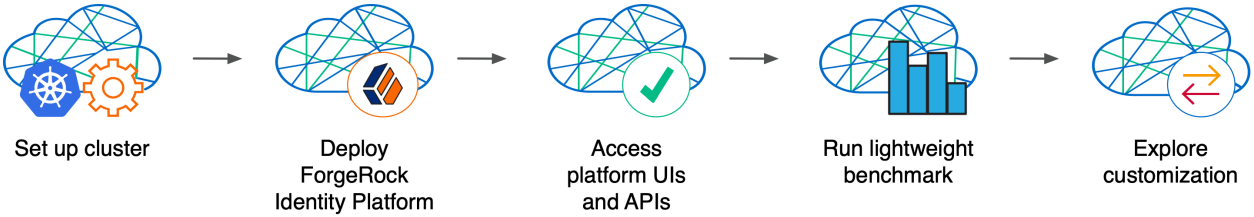
About the ForgeRock Cloud Deployment Model

The ForgeRock Cloud Deployment Team has developed Docker images, Kustomize bases and overlays, Skaffold workflows, Pulumi scripts, and other artifacts expressly to build the Cloud Deployment Model (CDM). The `forgeops` repository on GitHub contains the CDM artifacts you can use to deploy the ForgeRock Identity Platform in a cloud environment.

The CDM is a reference implementation for ForgeRock cloud deployments. You can get a sample ForgeRock Identity Platform deployment up and running in the cloud quickly using the CDM. After deploying the CDM, you can use it to explore how you might configure your Kubernetes cluster before you deploy the platform in production.

The CDM is a robust sample deployment for demonstration and exploration purposes only. *It is not a production deployment*.

This guide describes how to use the CDM to stand up a Kubernetes cluster in the cloud that runs the ForgeRock Identity Platform, and then access the platform's GUIs and REST APIs and run lightweight benchmarks. When you're done, you can use the CDM to explore deployment customizations:



Standing up a Kubernetes cluster and deploying the platform using the CDM is an activity you might want to perform as a learning and exploration exercise before you put together a project plan for deploying the platform in production. To better understand how this activity fits in to the overall deployment process, see "Deploy the CDM" in the *Start Here* guide.

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

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. You deploy the CDM to begin to familiarize yourself with some of the steps you'll need to perform when deploying the platform in the cloud for production use. These steps include creating a cluster suitable for deploying the ForgeRock Identity Platform, installing the platform, accessing its UIs and APIs, and running simple benchmarks.

Standardizes the process. The ForgeRock Cloud Deployment Team's mission is to standardize a process for deploying ForgeRock Identity Platform natively in the cloud. The Team is made up of technical consultants and cloud software developers. We've had numerous interactions with ForgeRock customers, and discussed common deployment issues. Based on our interactions, we standardized on Kubernetes as the cloud platform, and we developed the CDM artifacts to make deployment of the platform easier in the cloud.

Simplifies baseline deployment. We then developed artifacts—Dockerfiles, Kustomize bases and overlays, Skaffold workflows, and Pulumi scripts—to simplify the deployment process. We deployed a production-quality Kubernetes cluster, and kept it up and running 24x7. We conduct continuous integration and continuous deployment as we add new capabilities and fix problems in the system. We maintain, troubleshoot, and tune the system for optimized performance. Most importantly, we documented the process, and captured benchmark results—a process with results you can replicate.

Eliminates guesswork. If you use our CDM artifacts and follow the *CDM Cookbook* instructions without deviation, you can attain results similar to the benchmark results reported in this document. The CDM takes the guesswork out of setting up a 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.

Prepares you to deploy in production. After you've deployed the CDM, you'll be ready to start working with experts on deploying in production. We strongly recommend that you engage a ForgeRock technical consultant or partner to assist you to deploy the platform in production.

CDM Overview

Once you deploy the CDM, the ForgeRock Identity Platform is fully operational within a Kubernetes cluster. `forgeops` artifacts provide well-tuned JVM settings, memory, CPU limits, and other CDM configurations. Here are some of the characteristics of the CDM:

Multi-zone Kubernetes cluster

ForgeRock Identity Platform is deployed in a Kubernetes cluster. For high availability, CDM clusters are distributed across three zones.

For better node sizing, pods in CDM clusters are organized in two node pools.

Go here for a diagram that shows the organization of pods in zones and node pools in a CDM cluster.

Third-party deployment and monitoring tools

- Pulumi for cluster creation.
- NGINX Ingress Controller for Kubernetes ingress support.
- Prometheus for monitoring and notifications.
- Prometheus Alertmanager for setting and managing alerts.
- Grafana for metrics visualization.
- Gatling for benchmark load testing.
- Certificate Manager for obtaining and installing security certificates.
- Helm for deploying Helm charts for the NGINX Ingress Controller, Prometheus, and Grafana.

Ready-to-use ForgeRock Identity Platform components

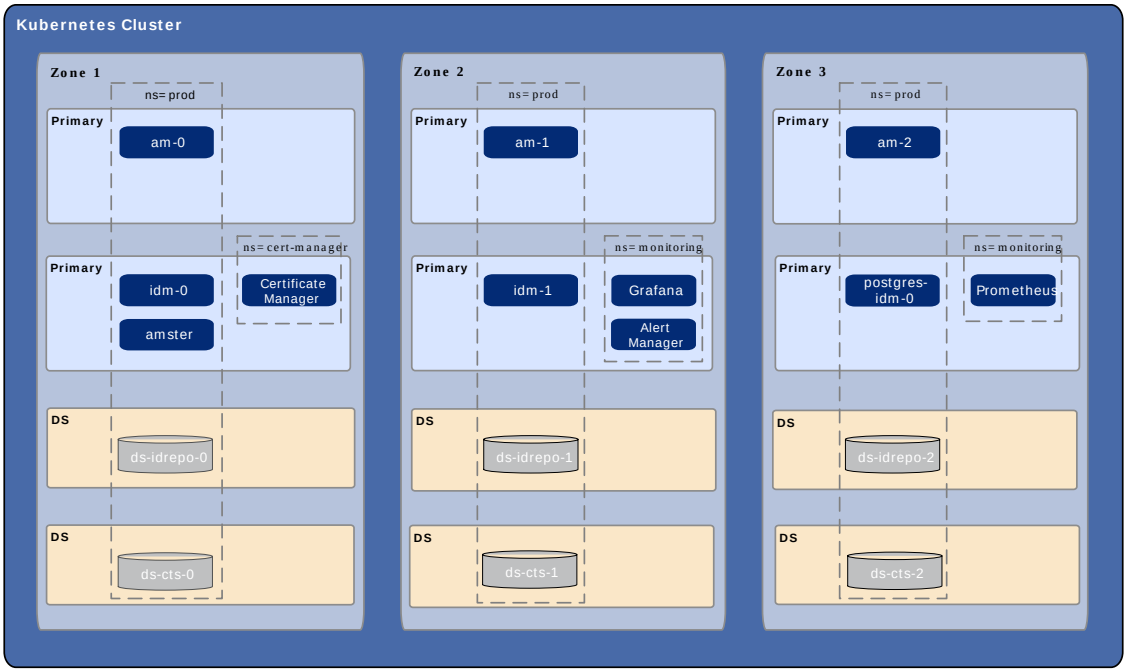
- Multiple DS instances are deployed for higher availbility. Separate instances are deployed for Core Token Service (CTS) tokens and identities. The instances for identities also contain:
  - AM configuration data, policies, and application data.
  - IDM run-time data.
- The `amster` pod facilitates AM deployment by setting up the AM server with configuration data in the Amster Docker image.
- Multiple AM instances are deployed for higher availability. The AM instances are configured to access the DS data stores.
- Multiple IDM instances are deployed for higher availability. The IDM instances are configured to access the DS data stores.

Highly available, distributed deployment

Deployment across the three zones ensures that the ingress controller and all ForgeRock Identity Platform components are highly available.

Distribution across the two node pools—primary and DS—groups like pods together, enabling appropriate node sizing.

The following diagram shows how pods are organized in node pools and zones on CDM clusters:



Load balancing

The NGINX Ingress Controller provides load balancing services for CDM deployments. Ingress controller pods run in the `nginx` namespace. Implementation varies by cloud provider.

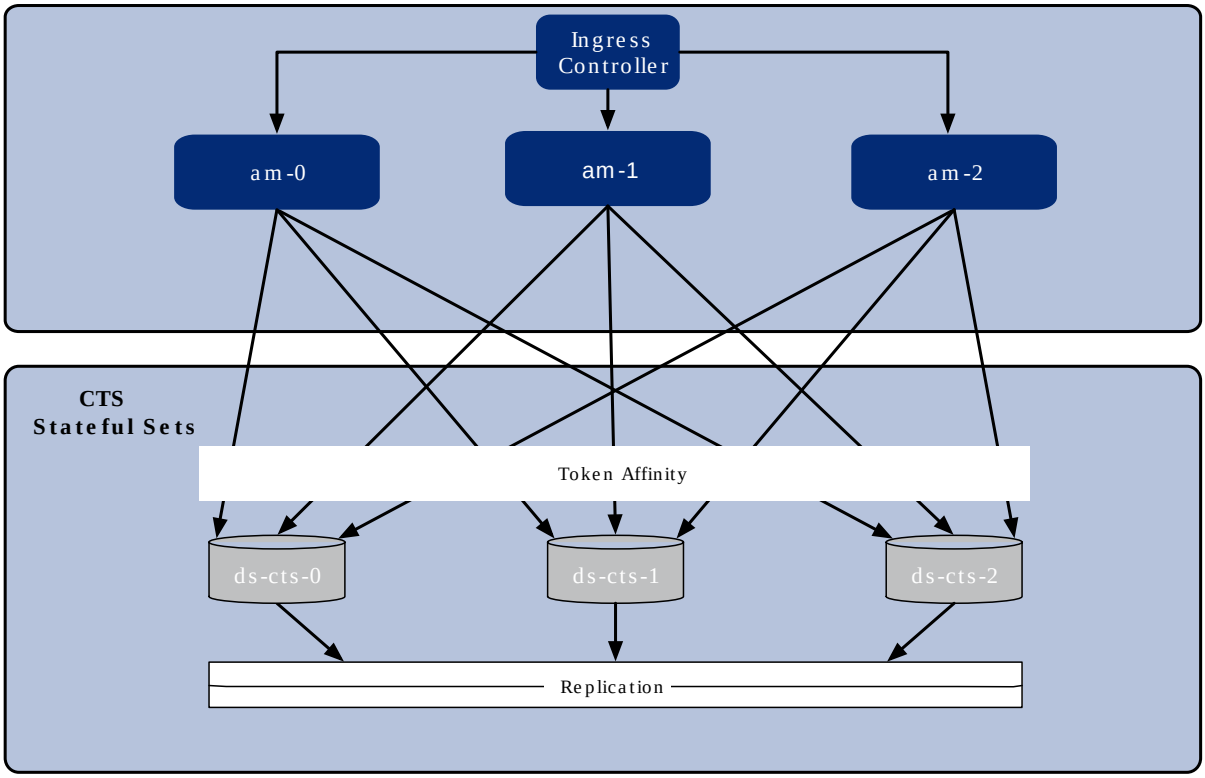
Secured communication

The ingress controller is SSL-enabled. SSL is terminated at the ingress controller. Incoming requests and outgoing responses are encrypted. For more information, see "Securing Communication With ForgeRock Identity Platform Servers" in the *Cloud Deployment Guide*.

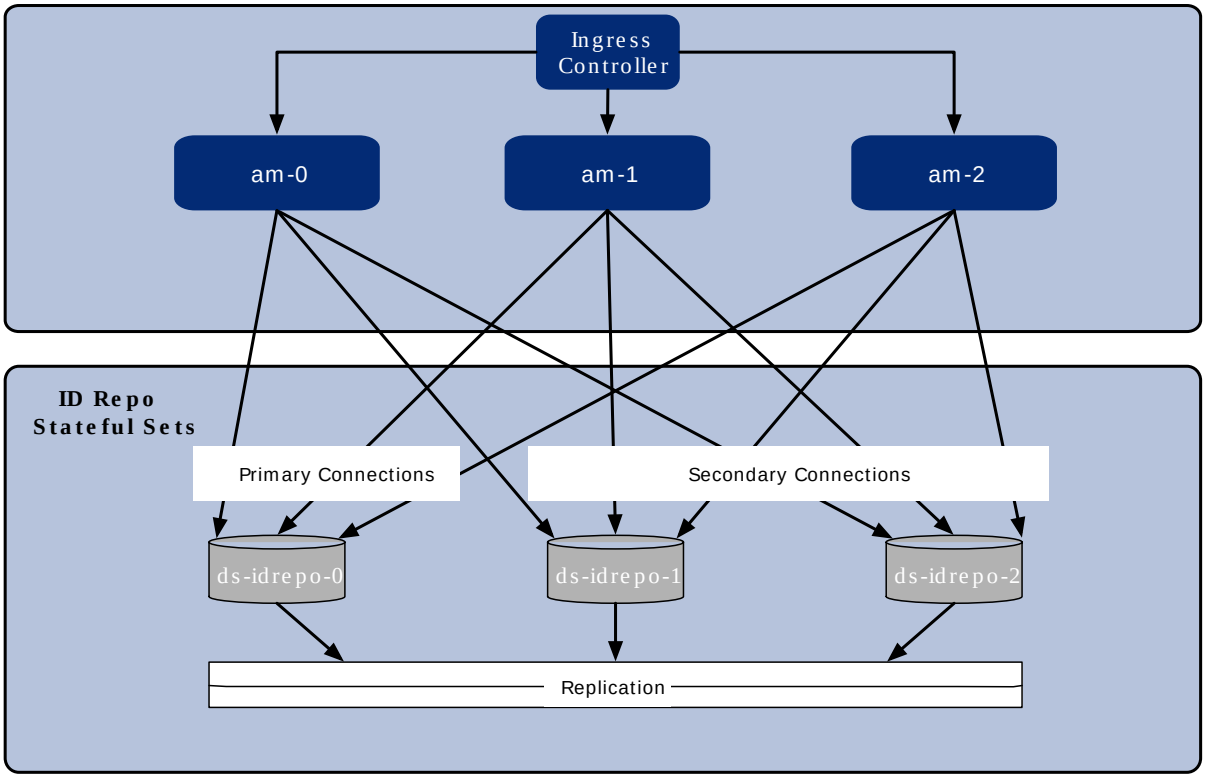
Stateful Sets

The CDM uses Kubernetes stateful sets to manage the DS pods. Stateful sets protect against data loss if Kubernetes client containers fail.

The CTS data stores are configured for affinity load balancing for optimal performance:



The AM configuration, policies, application data, and identities reside in the `idrepo` directory service. The connection between each AM pod and the `ds-idrepo-0` pod is the primary connection. Connections between the AM pods and the other `idrepo` pods are secondary:



DS replication

All DS instances are configured for full replication of identities, configuration data, and session tokens.

Backup and restore

The CDM is ready to back up directory data, but backups are not scheduled by default. To schedule backups, see "*Backing Up and Restoring Directory Data*" in the *Cloud Deployment Guide*.

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.

Begin at the Beginning

Using the *CDM Cookbook* simplifies deploying ForgeRock Identity Platform in the cloud. But if you deviate from the sequence of steps or customize any of the artifacts, you may not attain the same benchmark results the Cloud Deployment Team documented.

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.

Engage a ForgeRock Cloud Expert

Use the CDM to familiarize yourself with the steps you perform to deploy the platform in the cloud. When you're ready to implement a production-quality deployment, engage a ForgeRock technical consultant or partner. If you're not yet 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, configure an AKS environment, and create an AKS cluster before you install the CDM.

+ *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.

The chapter covers the following topics:

Installing Required Third-Party Software

Before installing the CDM, you must obtain non-ForgeRock software and install it 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 deploying the CDM on Microsoft Azure. 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
Pulumi	1.11.1	https://www.pulumi.com/docs/
Helm	3.0.3	https://github.com/helm/helm
Gradle	6.1.1_1	https://gradle.org
Node.js	13.8.0	https://nodejs.org/en/docs/
Kubernetes log display utility (**stern**)	1.11.0	https://github.com/wercker/stern
Azure Command Line Interface	2.1.0	https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest

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

Setting Up an Azure Subscription for the CDM

The CDM runs in a Kubernetes cluster in an Azure subscription.

This section outlines how the Cloud Deployment Team created and configured our Azure subscription before we created our cluster.

To replicate Azure subscription creation and configuration, follow this procedure:

To Configure an Azure Subscription for the CDM

1. Assign the following roles to users who will deploy the CDM:
   - Azure Kubernetes Service Cluster Admin Role
   - Azure Kubernetes Service Cluster User Role
   - Contributor
   - User Access Administrator

   Remember, the CDM is a reference implementation, and is not for production use. The roles you assign in this step are suitable for the CDM. When you plan your production deployment, you'll need to determine which Azure roles are required.

2. Log in to Azure services as a user with the roles you assigned in the previous step:

```
$ az login --username my-user-name
```

3. View your current subscription ID:

```
$ az account show
```

4. If necessary, set the current subscription ID to the one you will use to deploy the CDM:

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

5. 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.
   - The subscription, resource groups, and resources you create for your AKS cluster must reside in the same region.

6. As of this writing, the CDM uses Standard FSv2 Family vCPUs for the DS node pool. Make sure that your subscription has an adequate quota for this vCPU type in the region where you'll deploy the CDM. If the quota is lower than 192 CPUs, request a quota increase to 192 CPUs (or higher) before you create the cluster for the CDM.

   When you plan your production deployment, you'll need to determine which CPU types are needed, and, possibly, increase quotas.

7. The CDM uses Azure Container Registry (ACR) for storing Docker images.

   If you do not have a container registry in your subscription, create one.

Creating and Setting up a Kubernetes Cluster

Now that you've installed third-party software on your local computer and set up an Azure subscription, you're ready to create a Kubernetes cluster for the CDM in your project.

The Cloud Deployment Team used Pulumi software to create the CDM cluster. This section describes how the team used Pulumi to create and set up a Kubernetes cluster that can run the CDM. It covers the following topics:

- "Obtaining the forgeops Repository"
- "Installing Node.js Dependencies"
- "Creating the Cluster"

- "Setting up Your Local Computer to Push Docker Images"

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 Node.js Dependencies

The `cluster` directory in the `forgeops` repository contains Pulumi scripts for creating the CDM cluster.

The Pulumi scripts are written in TypeScript and run in the Node.js environment. Before running the scripts, you'll need to install the Node.js dependencies listed in the `/path/to/forgeops/cluster/pulumi/package.json` file as follows:

To Install Node.js Dependencies

1. Change to the `/path/to/forgeops/cluster/pulumi` directory.

2. Remove any previously installed Node.js dependencies:

   ```
   $ rm -rf node_modules
   ```

3. Install dependencies:

   ```
   $ npm install
   > deasync@0.1.15 install /Users/someuser/Repositories/forgeops/cluster/pulumi/node_modules/deasync >
    node ./build.js
   . . .
   added 415 packages from 508 contributors and audited 4469 packages in 22.036s
   found 0 vulnerabilities
   ```

[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*.

Creating the Cluster

After cloning the `forgeops` repository and installing Node.js dependencies, you're ready to create the Kubernetes cluster for the CDM.

This section outlines how the Cloud Deployment Team created our cluster. The cluster has the following characteristics:

- `prod`, `nginx`, and `cert-manager` namespaces created
- NGINX ingress controller deployed
- Certificate Manager deployed
- Prometheus and Grafana monitoring tools deployed

Perform the following procedures to replicate CDM cluster creation:

- "To Create a Kubernetes Cluster for CDM"
- "To Set up Helm"
- "To Deploy an NGINX Ingress Controller"
- "To Deploy Certificate Manager"
- "To Deploy Prometheus, Grafana, and Alertmanager"

To Create a Kubernetes Cluster for CDM

1. Obtain the following information from your AKS administrator:
   - The Azure region and zones in which you will create the cluster. The CDM is deployed within a single region.

     The Cloud Deployment Team deployed the CDM in the `eastus` region. If you want to validate your deployment against the benchmarks in "*Benchmarking CDM Performance*", use this region when you deploy the CDM, regardless of your actual location.

     However, if you would like to deploy the CDM in a different Azure region, you may do so. The region must support Standard FSv2 Family vCPUs.
   - The name of your Azure Container Registry.
   - The name of the resource group associated with your Azure Container Registry.
2. ForgeRock provides Pulumi scripts to use for cluster creation. Use them when you deploy the CDM. After you've finished going through this *CDM Cookbook*, you can use the CDM as a

sandbox to explore a different infrastructure-as-code solution, if you like. When you plan your production deployment, you'll need to identify your organization's preferred infrastructure-as-code solution, and create your own cluster creation automation scripts, if necessary.

Store your Pulumi passphrase in an environment variable:

```
$ export PULUMI_CONFIG_PASSPHRASE=my-passphrase
```

The default Pulumi passphrase is `password`.

3. Log in to Pulumi using the local option or the Pulumi service.

   For example, to log in using the local option:

   ```
   $ pulumi login -l
   ```

   As of this writing, issues have been encountered when using cloud provider backends for storing Pulumi stacks, a preview feature. Because of this, do not specify a cloud provider backend when logging in to Pulumi.

4. Create infrastructure components to support your cluster:

   a. Change to the directory that contains the Azure infrastructure stack configuration files:

      ```
      $ cd /path/to/forgeops/cluster/pulumi/azure/infra
      ```

   b. Verify that your current working directory is `/path/to/forgeops/cluster/pulumi/azure/infra`. If you are not in this directory, Pulumi will create the infrastructure stack incorrectly.

   c. Initialize the infrastructure stack:

      ```
      $ pulumi stack init azure-infra
      ```

      Note that initializing a Pulumi stack also selects the stack, so you don't need to explicitly execute the **pulumi stack select** command.

   d. (Optional) If you're deploying the CDM in a region other than `eastus`, configure your infrastructure stack with your region. Use the region you obtained in Step 1:

      ```
      $ pulumi config set azure-infra:location my-region
      ```

   e. Configure the Azure resource group for Azure Container Registry. Use the resource group you obtained in Step 1:

      ```
      $ pulumi config set azure-infra:acrResourceGroupName my-resource-group
      ```

   f. Create the infrastructure components:

      ```
      $ pulumi up
      ```

      Pulumi provides a preview of the operation and issues the following prompt:

      ```
      Do you want to perform this update?
      ```

Review the operation, and then select `yes` if you want to proceed.

g. To verify that Pulumi created the infrastructure components, log in to the Azure console. Display the resource groups. You should see a new resource group named `azure-infra-ip-resource-group`.

5. Create your cluster:

a. Change to the directory that contains the cluster configuration files:

```
$ cd /path/to/forgeops/cluster/pulumi/azure/aks
```

b. Verify that your current working directory is `/path/to/forgeops/cluster/pulumi/azure/aks`. If you are not in this directory, Pulumi will create the CDM stack incorrectly.

c. Initialize the CDM stack:

```
$ pulumi stack init aks-medium
```

d. (Optional) If you're deploying the CDM in a region other than `eastus`, configure your infrastructure stack with your region. Use the region you obtained in Step 1:

```
$ pulumi config set aks:location my-aks-region
```

e. Configure the Azure resource group for Azure Container Registry. Use the resource group you obtained in Step 1:

```
$ pulumi config set aks:acrResourceGroupName my-resource-group
```

f. Create the cluster:

```
$ pulumi up
```

Pulumi provides a preview of the operation and issues the following prompt:

```
Do you want to perform this update?
```

Review the operation, and then select `yes` if you want to proceed.

g. Make a note of the static IP address that Pulumi reserved. The address appears in the output from the **pulumi up** command. Look for output similar to:

```
staticIpAddress    : "152.52.292.141"
```

You'll need the IP address when you deploy the NGINX ingress controller.

h. Verify that Pulumi created the cluster using the Azure console.

6. After creating a Kubernetes cluster, Pulumi does not write cluster configuration information to the default Kubernetes configuration file, `$HOME/.kube/config`. Configure your local computer's Kubernetes settings so that the **kubectl** command can access your new cluster:

a. Verify that the `/path/to/forgeops/cluster/pulumi/azure/aks` directory is still your current working directory.

b. Create a `kubeconfig` file with your new cluster's configuration in the current working directory:

```
$ pulumi stack output kubeconfig > kubeconfig
```

c. Configure Kubernetes to get cluster information from the union of the new `kubeconfig` file and the default Kubernetes configuration file:

```
$ export KUBECONFIG=$PWD/kubeconfig:$HOME/.kube/config
```

d. Run the **kubectx** command.

The output should contain your newly created cluster and any existing clusters.

The current context should be set to the context for your new cluster.

7. Check the status of the pods in the your cluster until all the pods are ready:

a. List all the pods in the cluster:

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE
kube-system   azure-cni-networkmonitor-gsbkg    1/1     Running   0          3m38s
kube-system   azure-cni-networkmonitor-k26mc    1/1     Running   0          3m40s
kube-system   azure-cni-networkmonitor-ng4qn    1/1     Running   0          8m40s
. . .
kube-system   azure-ip-masq-agent-4kkpg         1/1     Running   0          8m40s
kube-system   azure-ip-masq-agent-6r699         1/1     Running   0          8m40s
. . .
kube-system   kube-proxy-5ztxd                  1/1     Running   0          8m23s
kube-system   kube-proxy-6th8b                  1/1     Running   0          9m6s
. . .
kube-system   coredns-698c77c5d7-k6q9h          1/1     Running   0          9m
kube-system   coredns-698c77c5d7-knwwm          1/1     Running   0          9m
kube-system   coredns-autoscaler-...            1/1     Running   0          9m
kube-system   metrics-server-69df9f75bf-fc4pn   1/1     Running   1          20m
kube-system   tunnelfront-5b56b76594-6wzps      1/1     Running   1          20m
```

b. Review the output. Deployment is complete when:

- The `READY` column indicates all running 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` or `Completed`.

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

d. Get a list of nodes:

```
$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-dsnodes-11254213-vmss000000     Ready    agent   17m   v1.15.10
aks-dsnodes-11254213-vmss000001     Ready    agent   17m   v1.15.10
aks-dsnodes-11254213-vmss000002     Ready    agent   16m   v1.15.10
aks-dsnodes-11254213-vmss000003     Ready    agent   17m   v1.15.10
aks-dsnodes-11254213-vmss000004     Ready    agent   18m   v1.15.10
aks-dsnodes-11254213-vmss000005     Ready    agent   18m   v1.15.10
aks-workernodes-11254213-vmss000000 Ready    agent   23m   v1.15.10
aks-workernodes-11254213-vmss000001 Ready    agent   22m   v1.15.10
aks-workernodes-11254213-vmss000002 Ready    agent   22m   v1.15.10
aks-workernodes-11254213-vmss000003 Ready    agent   23m   v1.15.10
aks-workernodes-11254213-vmss000004 Ready    agent   23m   v1.15.10
aks-workernodes-11254213-vmss000005 Ready    agent   22m   v1.15.10
```

e. Apply the `"forgerock.io/role=ds"` label to **all six** DS nodes, to enable communication with them:

```
$ kubectl label node ds-node-name "forgerock.io/role=ds"
```

For example, to label the `aks-dsnodes-11254213-vmss000000` node:

```
$ kubectl label node aks-dsnodes-11254213-vmss000000 "forgerock.io/role=ds"
node/aks-dsnodes-11254213-vmss000000 labeled
```

To Set up Helm

Because the NGINX Ingress Controller, Prometheus, and Grafana are distributed as Helm charts, you must set up Helm on your local computer before you can deploy the other components:

1. List your working set of chart repositories:

```
$ helm repo list
```

2. If your working set of Helm chart repositories does not have the `stable` chart repository, add it:

```
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

3. Refresh your local Helm repository cache:

```
$ helm repo update
```

To Deploy an NGINX Ingress Controller

Before you perform this procedure, you must have initialized your CDM cluster by performing the steps in "To Create a Kubernetes Cluster for CDM". If you did not set up your cluster using this technique, the cluster might be missing some required configuration.

Also, remember, the CDM is a reference implementation, and is not for production use. Use the NGINX ingress controller when you deploy the CDM. After you've finished going through this *CDM Cookbook*, you can use the CDM as a sandbox to explore deploying a different ingress controller. When you plan your production deployment, you'll need to determine which ingress controller to use in production.

1. Deploy the NGINX ingress controller in your cluster. For `static-ip-address`, specify the IP address obtained when you performed Step 5.g of "To Create a Kubernetes Cluster for CDM":

```
$ /path/to/forgeops/bin/ingress-controller-deploy.sh \
 -a -i static-ip-address -r azure-infra-ip-resource-group
namespace/nginx created
Release "nginx-ingress" does not exist. Installing it now.
NAME: nginx-ingress
. . .
```

2. Check the status of the services in the `nginx` namespace to note the external IP address for the ingress controller:

```
$ kubectl get services --namespace nginx
NAME                            TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)  AGE
nginx-ingress-controller        LoadBalancer   10.0.131.150  52.152.192.41  80...    23s
nginx-ingress-default-backend   ClusterIP      10.0.146.216  none           80...    23s
```

3. You'll access ForgeRock Identity Platform services through the ingress controller. The URLs you'll use must be resolvable from your local computer.

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

```
ingress-ip-address prod.iam.example.com
```

   For *`ingress-ip-address`*, specify the external IP of the ingress controller service in the previous command.

To Deploy Certificate Manager

The CDM is a reference implementation, and is not for production use. Use cert-manager when you deploy the CDM. After you've finished going through this *CDM Cookbook*, you can use the CDM as a sandbox to explore different certificate management tooling, if you like. When you plan your production deployment, you'll need to determine how you want to manage certificates in production.

1. Deploy the Certificate Manager in your cluster:

```
$ /path/to/forgeops/bin/certmanager-deploy.sh
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
namespace/cert-manager created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
. . .
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
deployment.apps/cert-manager created
deployment.apps/cert-manager-webhook created
mutatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/cert-manager-webhook created
deployment.extensions/cert-manager-webhook condition met
clusterissuer.cert-manager.io/default-issuer created
secret/certmanager-ca-secret created
```

2. Check the status of the pods in the `cert-manager` namespace until all the pods are ready:

```
$ kubectl get pods --namespace cert-manager
NAME                                       READY STATUS    RESTARTS AGE
cert-manager-6d5fd89bdf-khj5w              1/1   Running   0        3m57s
cert-manager-cainjector-7d47d59998-h5b48   1/1   Running   0        3m57s
cert-manager-webhook-6559cc8549-8vdtp      1/1   Running   0        3m56s
```

To Deploy Prometheus, Grafana, and Alertmanager

The CDM is a reference implementation, and is not for production use. Use Prometheus, Grafana, and Alertmanager when you deploy the CDM. After you've finished going through this *CDM Cookbook*, you can use the CDM as a sandbox to explore different monitoring, reporting, and alerting tooling, if you like. When you plan your production deployment, you'll need to determine how you want to implement monitoring, alerts, and reporting in your environment.

1. The Helm charts for Prometheus, Grafana, and Alertmanager, formerly located in the `coreos-charts` repository, now reside in Helm's stable chart repository. If a chart repository with the URL https://s3-eu-west-1.amazonaws.com/coreos-charts/stable is in your working set, remove it.

2. Deploy Prometheus, Grafana, and Alertmanager in your cluster. You can safely ignore `info: skipping unknown hook: "crd-install"` messages:

```
$ /path/to/forgeops/bin/prometheus-deploy.sh
namespace/monitoring created
"stable" has been added to your repositories
Release "prometheus-operator" does not exist. Installing it now.
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
. . .
NAME: prometheus-operator
LAST DEPLOYED: Mon Feb 10 16:47:45 2020
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
. . .
customresourcedefinition.apiextensions.k8s.io/prometheuses.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/servicemonitors.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/servicemonitors.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/podmonitors.monitoring.coreos.com condition met
customresourcedefinition.apiextensions.k8s.io/alertmanagers.monitoring.coreos.com condition met
Release "forgerock-metrics" does not exist. Installing it now.
NAME: forgerock-metrics
LAST DEPLOYED: Mon Feb 10 16:48:27 2020
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

3. Check the status of the pods in the `monitoring` namespace until all the pods are ready:

```
$ kubectl get pods --namespace monitoring
NAME                                                     READY STATUS    RESTARTS AGE
alertmanager-prometheus-operator-alertmanager-0          2/2   Running   0        5m8s
prometheus-operator-grafana-7b8598c98f-glhmn             2/2   Running   0        5m16s
prometheus-operator-kube-state-metrics-...               1/1   Running   0        5m16s
prometheus-operator-operator-55966c69dd-76v46            2/2   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-82r4b       1/1   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-85ns8       1/1   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-kgwln       1/1   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-rrwrx       1/1   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-vl8f9       1/1   Running   0        5m16s
prometheus-operator-prometheus-node-exporter-xmjrf       1/1   Running   0        5m16s
prometheus-prometheus-operator-prometheus-0              3/3   Running   1        4m57s
```

Setting up Your Local Computer to Push Docker Images

In the deployment environment you're setting up, Skaffold builds Docker images using the Docker software you've installed on your local computer. After it builds the images, Skaffold pushes them to a Docker registry available to your AKS cluster. With the images on the remote Docker registry, Skaffold can orchestrate the ForgeRock Identity Platform, creating containers from the Docker images.

For Skaffold to push the Docker images:

- Docker must be running on your local computer.

- Your local computer needs credentials that let Skaffold push the images to the Docker repository available to your cluster.
- Skaffold needs to know the location of the Docker repository.

Perform the following procedure to let Skaffold to push Docker images to a registry accessible to your cluster:

To Set up Your Local Computer to Push Docker Images

1. If it's not already running, start Docker on your local computer. For more information, see the Docker documentation.
2. If you don't already have the name of the container registry that will hold ForgeRock Docker images, obtain it from your Azure administrator.
3. Log in to your container registry:

   ```
   $ az acr login --name registry-name
   ```

   Azure repository logins expire after 4 hours. Because of this, you'll need to log in to ACR whenever your login session expires. [2]
4. Run the **kubectx** command to obtain the Kubernetes context.
5. Configure Skaffold with your Docker repository location and Kubernetes context:

   ```
   $ skaffold config \
    set default-repo registry-name.azurecr.io/cdm -k my-kubernetes-context
   ```

   For example:

   ```
   $ skaffold config set default-repo my-container-registry.azurecr.io/cdm -k aks
   ```

You're now ready to deploy the CDM.

[2] You can automate logging in to ACR every 4 hours by using the **cron** utility.

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 procedure:

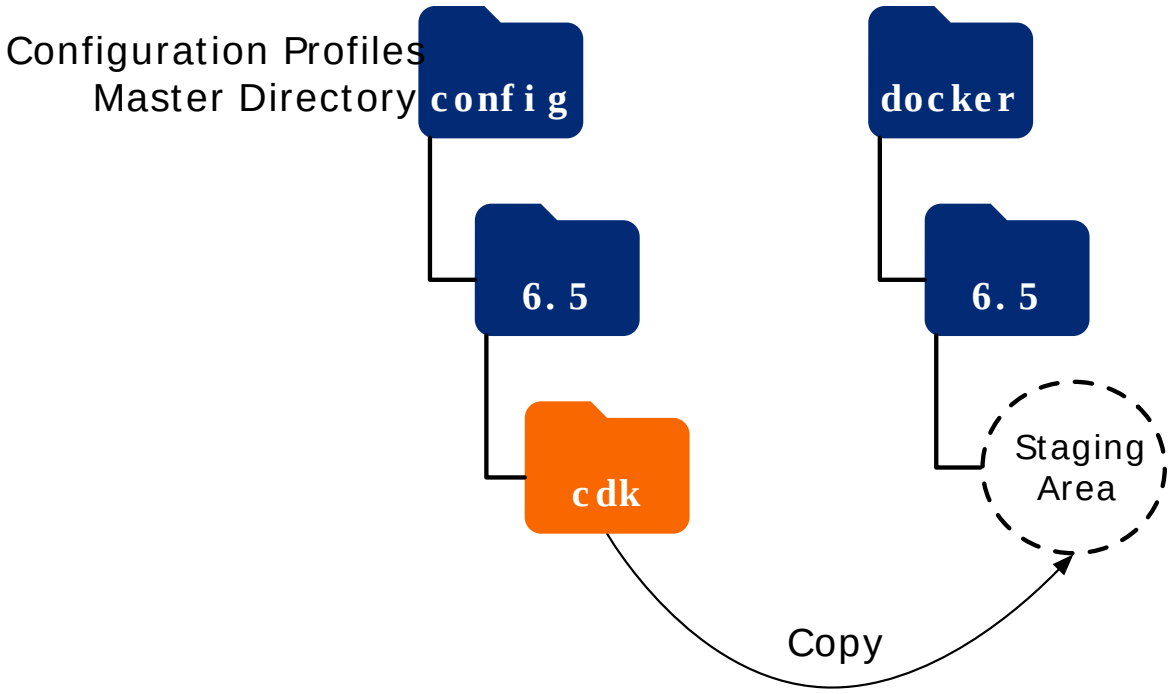
To Deploy the CDM

1. Initialize the staging area for configuration profiles with the canonical CDK configuration profile[1] 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:

[1] The CDM and the CDK both use the CDK canonical configuration profile.



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*.

2. Change to the `/path/to/forgeops` directory and execute the **skaffold run** command. Specify `skaffold-6.5.yaml` as the Skaffold pipeline file:

```
$ cd /path/to/forgeops
$ skaffold run -f skaffold-6.5.yaml -p medium
```

3. Make the `prod` namespace your current namespace:

```
$ kubens prod
```

4. Check the status of the pods in the `prod` namespace until all the pods are ready:

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

```
$ kubectl get pods
NAME                          READY   STATUS     RESTARTS   AGE
am-9758bc5fd-hndsg            1/1     Running    0          2m37s
amster-55c76dc4cb-njvdj       1/1     Running    0          2m37s
ds-cts-0                      1/1     Running    0          2m36s
ds-cts-1                      1/1     Running    0          114s
ds-cts-2                      1/1     Running    0          70s
ds-idrepo-0                   1/1     Running    0          2m36s
ds-idrepo-1                   1/1     Running    0          112s
ds-idrepo-2                   1/1     Running    0          74s
forgeops-secrets-k82w7        0/1     Completed  0          2m35s
idm-0                         1/1     Running    0          2m28s
idm-1                         1/1     Running    0          2m30s
postgres-idm-0                1/1     Running    0          2m36s
```

b. Review the output. Deployment is complete when:

- The `READY` column indicates all running 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` or `Completed`.

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

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.

   b. Delete the pod:

   ```
   $ kubectl delete pod am-pod
   ```

6. Scale the number of AM pods to three:

   ```
   $ kubectl scale --replicas=3 deployment am
   statefulset.extensions/idm scaled
   ```

7. Verify that three AM pods are available:

```
$ kubectl get pods
NAME                          READY   STATUS     RESTARTS   AGE
am-9758bc5fd-hndsg            1/1     Running    0          5m37s
am-9758bc5fd-qr124            1/1     Running    0          2m55s
am-9758bc5fd-xb294            1/1     Running    0          5m36s
amster-55c76dc4cb-njvdj       1/1     Running    0          5m37s
ds-cts-0                      1/1     Running    0          5m36s
ds-cts-1                      1/1     Running    0          290s
ds-cts-2                      1/1     Running    0          250s
ds-idrepo-0                   1/1     Running    0          5m36s
ds-idrepo-1                   1/1     Running    0          272s
ds-idrepo-2                   1/1     Running    0          224s
forgeops-secrets-k82w7        0/1     Completed  0          5m35s
idm-0                         1/1     Running    0          2m28s
idm-1                         1/1     Running    0          2m30s
postgres-idm-0                1/1     Running    0          5m36s
```

Chapter 4

Using the CDM

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

Accessing ForgeRock Identity Platform Services

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

AM and IDM 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.

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 and IDM have been configured in the CDM, see Configuration in the `forgeops` repository's top-level README file for more information about the configurations.

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://prod.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://prod.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://prod.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://prod.iam.example.com/openidm/info/ping
{
    "_id":" ",
    "_rev":"",
    " shortDesc":"OpenIDM ready",
    "state":"ACTIVE_READY"
}
```

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

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

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

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

```
$ kubectl \
 port-forward \
 $(kubectl get pods --selector=app.kubernetes.io/name=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.
3. Log in to Grafana as the `admin` user. The password is `password`.
4. 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

For information about the Prometheus web UI, see the Prometheus documentation.

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

For a description of the CDM monitoring architecture and information about how to customize CDM monitoring, see "*Monitoring Your Deployment*" in the *Cloud Deployment Guide*.

Chapter 5

Benchmarking CDM Performance

This chapter provides steps you can take to replicate the Cloud Deployment Team benchmarking process. Topics include the following:

About CDM Benchmarking

We've published benchmarking instructions in this guide, and a spreadsheet with our results, to give you a means for validating your own CDM deployment. If you follow the benchmarking instructions in this guide *with no configuration modifications*, you should attain similar throughput and latency results.

Benchmark Sizing and Cost Estimates

We conducted the tests using the specifications detailed in the Configuration tab of the benchmarking results spreadsheet. Factors beyond the scope of the CDM, or a failure to use our documented sizing and configuration, may affect your benchmark test results. These factors might include (but are not limited to): updates to cloud platform SDKs; changes to third-party software required for Kubernetes; changes you have made to sizing or configuration to suit your business needs.

To view the cost estimates for our benchmarked configuration, click the link for your platform in the Sample Pricing column in the Configuration tab of the benchmarking results spreadsheet.

How CDM Benchmarking Relates to Your Deployment

After you've successfully deployed the CDM, run the tests in this chapter. Once you've validated your own results against the benchmarks reported here, you can be sure that ForgeRock Identity Platform is successfully running in the cloud to CDM specifications.

The CDM is designed to:

- Conform to DevOps best practices.

- Facilitate continuous integration and continuous deployment.
- Scale and deploy on any Kubernetes environment in the cloud.

If you require higher performance than the benchmarks reported here, you can scale your deployment horizontally and vertically. Vertically scaling ForgeRock Identity Platform works particularly well in the cloud. For more information about scaling your deployment, contact your qualified ForgeRock partner or technical consultant.

Running DS and AM Benchmark Tests

This section details how the Cloud Deployment Team benchmarked DS and AM performance in the CDM.

Generating Test Users for DS and AM Benchmark Testing

Before you can test DS and AM performance in a CDM deployment, you must generate test users, provision the CDM userstores, and prime the directory servers.

Complete the following steps:

To Provision the Userstores with Test Users

1. Run the `make-users.sh` script in each of the three `ds-idrepo` pods:

   ```
   $ kubectl exec ds-idrepo-x -it bin/make-users.sh 10000000
   ```

   where *x* is 0, 1, or 2.

   Be sure to run the script three times, once for each `ds-idrepo` pod in the CDM deployment. To save time, run the script simultaneously in three separate terminal windows or tabs.

   When the Cloud Deployment Team ran the `make-users.sh` script, it took approximately 15 minutes to run.

2. Prime the directory servers:

   a. Prime the directory server running in the `ds-idrepo-0` pod:

      i. Open a shell in the `id-dsrepo-0` pod:

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

      ii. If you have not already done so, obtain the directory superuser's password by running the `print-secrets.sh dsadmin` command.

      iii. Run the following command:

```
$ ldapsearch -D "cn=Directory Manager" -w directory-superuser-password \
 -p 1389 -b "ou=identities"  uid=user.*  | grep uid: | wc -l
10000000
```

iv. Exit from the `id-dsrepo-0` pod's shell:

```
$ exit
```

b. Prime the directory server running in the `ds-idrepo-1` pod.

c. Prime the directory server running in the `ds-idrepo-2` pod.

Running DS Benchmark Tests

The Cloud Deployment Team conducted DS benchmarking using the `searchrate`, `modrate`, and `addrate` tests. These tests are packaged with DS. See the `ds-bench.sh` script for details such as number of client threads, duration, and filters.

Searchrate

The DS `searchrate` test measures the search throughput and response time of a directory service. Before running this test, the userstore must be provisioned with user entries. See "Generating Test Users for DS and AM Benchmark Testing" for more information.

To run the `searchrate` test, open a shell in the `ds-idrepo-0` pod and run the `ds-bench.sh` script with the `srch` option. Example:

```
$ scripts/ds-bench.sh srch 60 localhost 10000000
```

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the Searchrate column on the DS tab.

Modrate

The DS `modrate` test measures the modify throughput and response time of a directory service. Before running this test, the userstore must be provisioned with user entries. See "Generating Test Users for DS and AM Benchmark Testing" for more information.

To run the `modrate` test, open a shell in the `ds-idrepo-0` pod and run the `ds-bench.sh` script with the `mod` option. Example:

```
$ scripts/ds-bench.sh mod 60 localhost 10000000
```

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the Modrate column on the DS tab.

Addrate

The DS `addrate` test measures the add throughput and response time of a directory server.

To run the `addrate` test, open a shell in the `ds-idrepo-0` pod and run the `ds-bench.sh` script with the `add` option. Example:

```
$ scripts/ds-bench.sh add 60 localhost 10000000
```

When the `addrate` test has completed, it purges entries that it previously created. Be sure to let the `addrate` test's purge phase finish to completion.

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the Addrate column on the DS tab.

Running AM Benchmark Tests

The Cloud Deployment Team developed Gatling simulation scripts for benchmarking AM performance.

The `AMRestAuthNSim.scala` simulation tests authentication rates using the REST API.

The `AMAccessTokenSim.scala` simulation tests OAuth 2.0 authorization code flow performance.

Before You Begin

The following are prerequisites for running the AM benchmark tests:

1. Make sure the userstore is provisioned, and the Directory Services cache is primed.

   See "To Provision the Userstores with Test Users".

2. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the number of users for the test:

   ```
   $ export TARGET_HOST=prod.iam.example.com
   $ export CONCURRENCY=300
   $ export DURATION=60
   $ export USER_PREFIX=user.
   $ export PASSWORD=password
   $ export USER_POOL=10000000
   $ export oauth2_client_id="clientOIDC_0"
   $ export oauth2_redirect_uri="http://fake.com"
   ```

REST Login

This benchmark test measures throughput and response times of an AM server performing REST authentications.

To run the benchmark test:

1. Make sure that you have fulfilled the prerequisites specified in "Before You Begin".
2. Configure AM for stateful sessions:
   a. Log in to the AM console as the `amadmin` user. For details, see "Accessing AM Services".
   b. Select the top-level realm.
   c. Select Properties.
   d. Disable the Use Client-based Sessions option.
   e. Select Save Changes.
3. Change to the `/path/to/forgeops/docker/gatling` directory.
4. Run the simulation:
   ```
   $ gradle clean; gradle gatlingRun-am.AMRestAuthNSim
   ```
5. When the simulation is complete, the name of the file containing the test results appears near the end of the output. Open the file in a browser to review the results.

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the REST Authentication columns on the AM tab.

OAuth 2.0 - Authorization Code Grant Flow

This benchmark test measures throughput and response time of an AM server performing authentication, authorization, and session token management using stateful tokens. In this test, one transaction includes all three operations.

To run the benchmark test:

1. Make sure that you have fulfilled the prerequisites specified in "Before You Begin".
2. Configure AM for stateful sessions:
   a. Log in to the AM console as the `amadmin` user. For details, see "Accessing AM Services".
   b. Select the top-level realm.
   c. Select Properties.
   d. Disable the Use Client-based Sessions option.
   e. Select Save Changes.
3. Configure AM for stateful OAuth2 tokens:
   a. Select Services > OAuth2 Provider.

b. Disable the Use Client-based Access & Refresh Tokens option.

c. Select Save Changes.

4. Change to the `/path/to/forgeops/docker/gatling` directory.

5. Run the simulation:

```
$ gradle clean; gradle gatlingRun-am.AMAccessTokenSim
```

6. When the simulation is complete, the name of the file containing the test results appears near the end of the output. Open the file in a browser to review the results.

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the OAuth 2.0 Authorization Code Grant Flow columns on the AM tab.

Running IDM Benchmark Tests

The IDM tests examine query, read, and delete performance obtained when using the IDM API. The Cloud Deployment Team generated load using the following simulations:

- `IDMReadCreateUsersSim65.scala`
- `IDMDeleteUsersSim65.scala`

The IDM tests do not use the identities used for the DS and AM tests. Instead, you generate a different set of identities when you run the `IDMReadCreateUsersSim65` simulation. This additional set of identities is also used by the `IDMDeleteUsersSim65` simulation.

Before You Begin

Perform the following prerequisite step before running the IDM benchmark tests:

1. Set environment variables that specify the host on which to run the test, the number of concurrent threads to spawn when running the test, the duration of the test (in seconds), the first part of the user ID, and the number of users for the test:

```
$ export TARGET_HOST=prod.iam.example.com
$ export CONCURRENCY=300
$ export DURATION=60
$ export USER_PREFIX=usertest
$ export PASSWORD=Passw0rd
$ export USER_POOL=10000
```

Query and Create

This benchmark test measures throughput and response time of an IDM server performing create operations.

To run the benchmark test:

1. Make sure that you have fulfilled the prerequisites specified in "Before You Begin".
2. Change to the `/path/to/forgeops/docker/gatling` directory.
3. Run the simulation:
   ```
   $ gradle clean; gradle gatlingRun-idm.IDMReadCreateUsersSim65
   ```
4. When the simulation is complete, the name of the file containing the test results appears near the end of the output. Open the file in a browser to review the results.

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the Query and Create columns on the IDM tab.

Query and Delete

This benchmark test measures throughput and response time of an IDM server performing delete operations.

1. Make sure that you have fulfilled the prerequisites specified in "Before You Begin".
2. Change to the `/path/to/forgeops/docker/gatling` directory.
3. Run the simulation:
   ```
   $ gradle clean; gradle gatlingRun-idm.IDMDeleteUsersSim65
   ```
4. When the simulation is complete, the name of the file containing the test results appears near the end of the output. Open the file in a browser to review the results.

You can compare your results with the Cloud Deployment Team's results here. You'll find the team's results in the Query and Delete columns on the IDM tab.

Chapter 6
Removing the CDM

When you're done working with the CDM, you can remove it from Azure by following this procedure:

To Remove the CDM

1. Log in to Pulumi using the local option or the Pulumi service. Be sure to log in the same way that you logged in when you created your cluster in Step 3 of "To Create a Kubernetes Cluster for CDM".
2. Remove your cluster:
   a. Change to the directory that contains the cluster configuration files:
      ```
      $ cd /path/to/forgeops/cluster/pulumi/azure/aks
      ```
   b. Select the Pulumi stack that you used when you created your cluster:
      ```
      $ pulumi stack select aks-medium
      ```
   c. Delete your cluster:
      ```
      $ pulumi destroy
      ```
      Pulumi provides a preview of the operation and issues the following prompt:
      ```
      Do you want to perform this destroy?
      ```
      Review the operation, and then select `yes` if you want to proceed.
   d. To verify that Pulumi removed the cluster, log in to the Azure console and select Kubernetes services.

      You should not see the CDM cluster in the list of Kubernetes clusters.
3. Remove infrastructure components:
   a. Change to the directory that contains the Azure infrastructure stack configuration files:
      ```
      $ cd /path/to/forgeops/cluster/pulumi/azure/infra
      ```
   b. Select the `azure-infra` Pulumi stack:
      ```
      $ pulumi stack select azure-infra
      ```

c. Delete the infrastructure components:

```
$ pulumi destroy
```

Pulumi provides a preview of the operation and issues the following prompt:

```
Do you want to perform this destroy?
```

Review the operation, and then select `yes` if you want to proceed.

d. To verify that Pulumi removed the infrastructure components, log in to the Azure console. Display the resource groups.

The resource group named `azure-infra-ip-resource-group` should not be present.

4. Remove the CDM cluster from your local computer's Kubernetes settings:

a. Unset the `KUBECONFIG` environment variable:

```
$ unset KUBECONFIG
```

b. Run the **kubectx** command.

The Kubernetes context for the CDM cluster should not appear in the **kubectx** command output.

Chapter 7
Taking the Next Steps

If you've followed the instructions in this cookbook *without modifying configurations*, then the following indicates that you've successfully deployed the CDM:

- The Kubernetes cluster and pods are up and running.
- DS, AM, and IDM are installed and running. You can access each ForgeRock component.
- DS is provisioned with sample users. Replication and failover work as expected.
- Monitoring tools are installed and running. You can access a monitoring console for DS, AM, and IDM.
- You can run the benchmarking tests in this cookbook without errors.
- Your benchmarking test results are similar to the benchmarks reported in this cookbook.

When you're satisfied that all of these conditions are met, then you've successfully taken the first steps to deploy the ForgeRock Identity Platform in the cloud. Congratulations!

Now that you're familiar with the CDM—ForgeRock's reference implementation—you're ready to work with a project team to plan and configure your production deployment. You'll need a team with expertise in the ForgeRock Identity Platform, in your cloud provider, and in Kubernetes on your cloud provider. We strongly recommend that you engage a ForgeRock technical consultant or partner to assist you with deploying the platform in production.

You'll perform these major activities:

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

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

Site reliability engineering. Site reliability engineers monitor the ForgeRock Identity Platform deployment, and keep the deployment up and running based on your business requirements. These

might include use cases, service-level agreements, thresholds, and load test profiles. The Cloud Deployment Guide, and READMEs in the `forgeops` repository, provide information about site reliability.

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" in the *DevOps Developer's Guide: Using Minikube*.
- Description of the environment, including the following information:
  - Environment type: Minikube, GKE, EKS, AKS, or OpenShift.
  - Software versions of supporting components:
    - Oracle VirtualBox (Minikube environments only).

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

Getting Support and Contacting ForgeRock

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

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

Appendix B. Homebrew Package Names

The following table lists the Homebrew package names for third-party software used in AKS deployment 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`	
Pulumi	`pulumi`	`pulumi`	
Helm	`kubernetes-helm`	`kubernetes-helm`	
Node.js	`node`	`node`	
Gradle	`gradle`	`gradle`	
Kubernetes log display utility (`stern`)	`stern`	`stern`	
Azure Command Line Interface	`azure-cli`	`azure-cli`	

[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.