Cloud Deployment Model Cookbook for GKE

/ ForgeRock Identity Platform 6.5

Latest update: 6.5.2

Gina Cariaga
David Goldsmith
Shankar Raman

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

Abstract

Step-by-step instructions for getting the CDM up and running on Google Kubernetes Engine (GKE).

Table of Contents

Preface

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

For information about how to customize the CDM after you've deployed it, and how to maintain the deployment, see the Site Reliability Guide for GKE.

Before You Begin

Before deploying the ForgeRock Identity Platform in a DevOps environment, read the important information in Start Here.

About ForgeRock Identity Platform Software

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

The platform includes the following components:

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

Chapter 1

About the ForgeRock Cloud Deployment Model

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

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

1.1. How the CDM Relates to Your Deployment

Using the CDM artifacts and *CDM Cookbook* instructions, you can quickly get the ForgeRock Identity Platform running in a Kubernetes cloud environment. The CDM is not a deployment template. But you can deploy the CDM to validate your pre-production deployment against the benchmark results reported in this guide. Then you can customize or scale your deployment environment to meet your specific business requirements. From there, you can develop your own custom template, and write scripts to automate your production deployment.

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

Simplifies deployment. We then developed artifacts such as Helm charts and configuration files to simplify the deployment process. We deployed small-sized, medium-sized, and large-sized production-quality Kubernetes clusters. 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 have 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 your cloud environment. It bypasses the deploy-test-integrate-test-repeat cycle many customers struggle through when spinning up the ForgeRock Identity Platform in the cloud for the first time.

1.2. CDM Overview

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

Multi-zone Kubernetes cluster

ForgeRock Identity Platform is deployed in a Kubernetes cluster. You specify one of three cluster sizes:

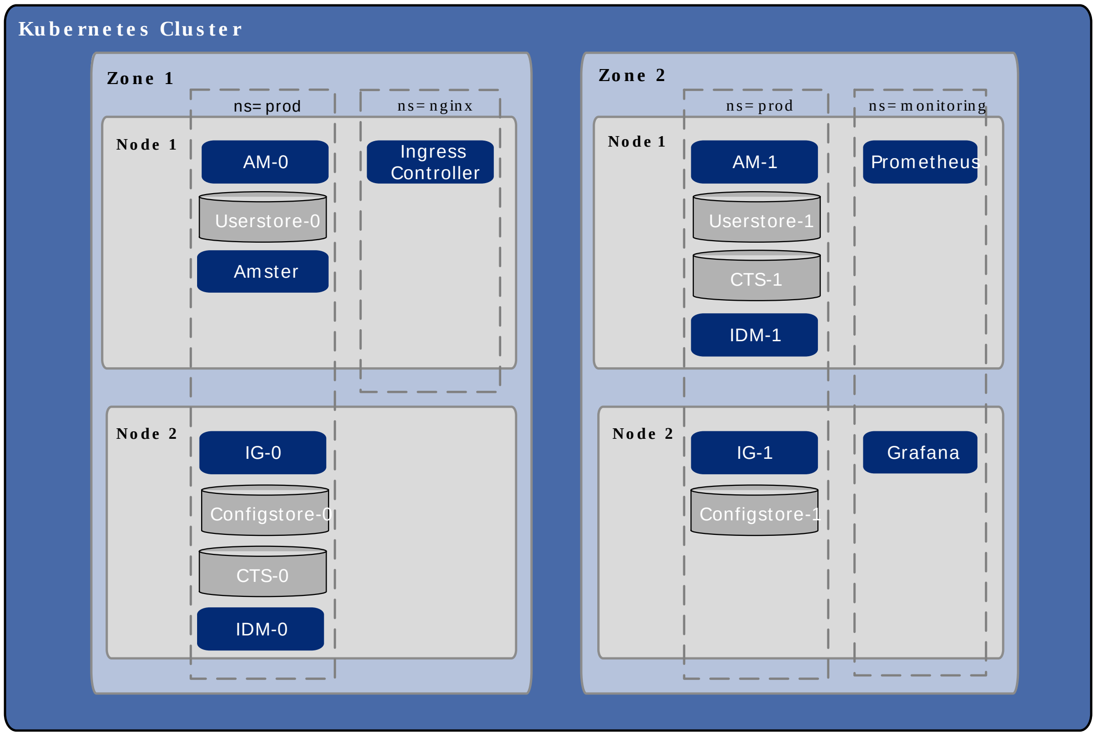
- A small cluster with capacity to handle 1,000,000 test users
- A medium cluster with capacity to handle 10,000,000 test users
- A large cluster with capacity to handle 100,000,000 test users

Two nodes are deployed in the primary zone, and two in an additional zone. This multi-zone, multi-node configuration facilitates scaling and high availability within the cluster.

Ready-to-use ForgeRock Identity Platform components

- DS instances are deployed with redundancy. Each instance uses separate data stores for users, configuration, and session tokens.
- The `amster` pod facilitates AM deployment by:
  - Obtaining the AM post-installation configuration.
  - Installing the AM server based on configuration from Helm charts in the `forgeops` repository.
  - Importing the post-installation configuration into the AM configuration store.
- AM instances are deployed with redundancy and configured to access the DS data stores.
- IDM instances are deployed with redundancy and configured to work with the DS identity store.
- IG instances are deployed with redundancy.

CDM Kubernetes Architecture



Secured communication

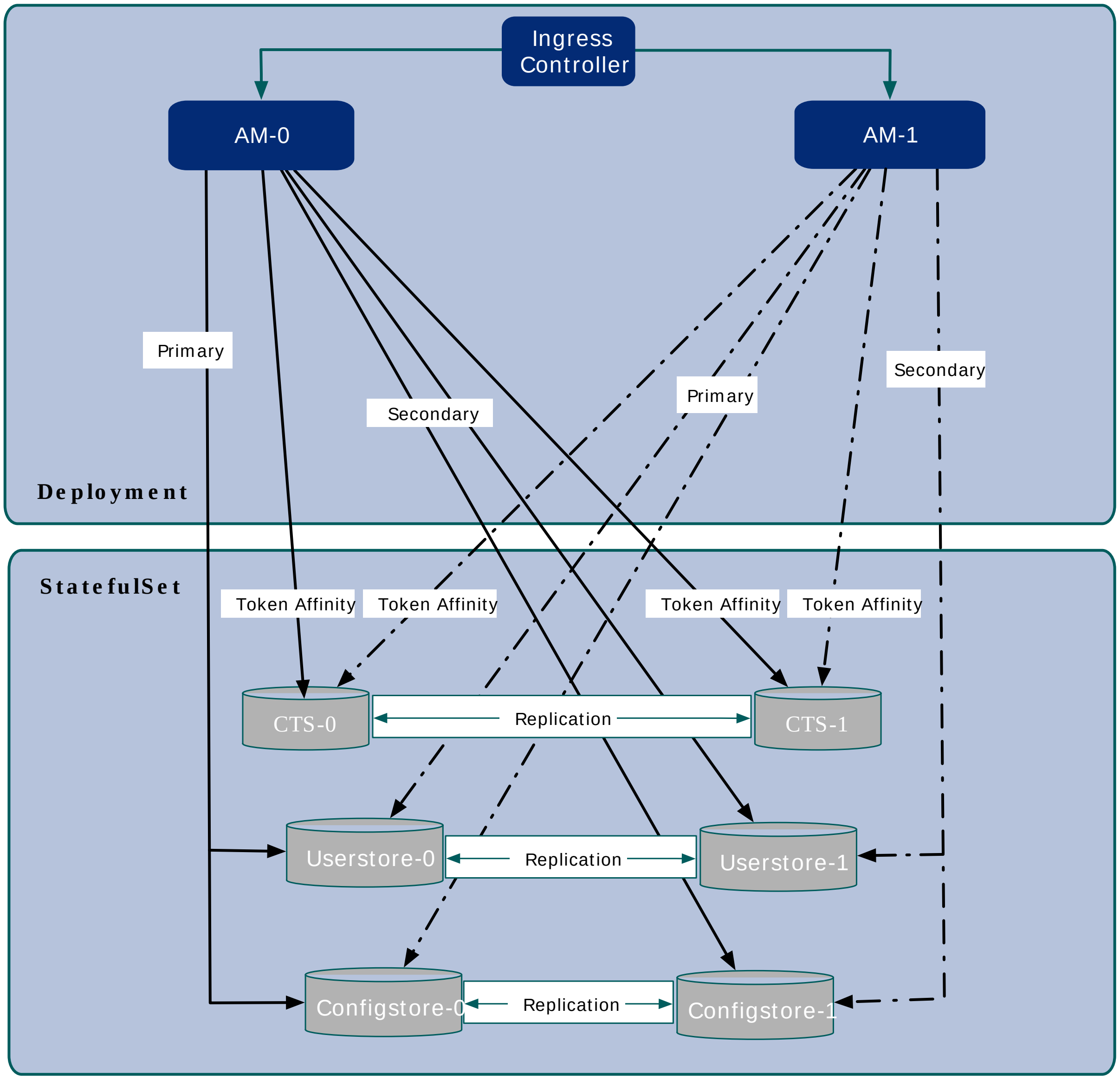
The ingress controller is SSL-enabled. SSL is terminated at the ingress controller. Incoming requests and outgoing responses are encrypted. For more information, see:

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

StatefulSets

The CDM uses Kubernetes stateful sets to manage the CDM pods. Stateful sets protect against data loss if Kubernetes client containers fail. Primary communication takes place between AM and configuration stores. Secondary communication takes place among AM, configuration stores, and userstores. The CTS data stores are configured for affinity load balancing for optimal performance. The AM configuration, policies, and application data reside in the configuration store.

Process Flows in the CDM



Authentication and authorization

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

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 *Site Reliability Guide for GKE*
- "*Backing Up and Restoring Directory Data*" in the *Site Reliability Guide for Amazon EKS*

1.3. Third-Party Software Deployed With the CDM

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

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

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

1.4. Best Practices for Implementing the CDM

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

1.4.1. 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 results the Cloud Deployment Team documented.

1.4.2. Provide End-to-End Project Management

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

1.4.3. Engage a ForgeRock Cloud Expert

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

Chapter 2
Setting Up the Deployment Environment

This chapter describes how to set up your local computer, how to configure a Google Cloud Platform (GCP) project, and how to create a GKE cluster before you install the CDM.

The chapter covers the following topics:

- "Installing Required Third-Party Software"
- "Setting up a GCP Project for the CDM"
- "Creating and Setting up a Kubernetes Cluster"

2.1. Installing Required Third-Party Software

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

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

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

2.2. Setting up a GCP Project for the CDM

The CDM runs in a Kubernetes cluster in a GCP project.

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

To replicate GCP project creation and configuration, follow this procedure:

To Configure a GCP Project for the CDM

1. Log in to the Google Cloud Console and create a new GCP project.
2. If necessary, set the Google Cloud SDK configuration to reference your new project with the **gcloud config set project** command.

3. Create and configure the `cert-manager` service account in your project:

   a. Create a service account named `cert-manager`. When you create the service account, be sure to download its private key in JSON format.

   b. Assign the Kubernetes Engine Developer and DNS Administrator permissions to the service account.

4. Assign the following roles to users who will be deploying CDM:

   - Cloud KMS CryptoKey Encrypter/Decrypter
   - Editor
   - Kubernetes Engine Admin
   - Kubernetes Engine Cluster Admin

5. Choose the GCP region in which you will deploy the CDM.

   The Cloud Deployment Team deployed the CDM in the `us-east1` region. Use this region when deploying the CDM, regardless of your actual location, if you want to validate your deployment against the benchmarks in "*Benchmarking the CDM Performance*".

   To use any other region, note the following:

   - Objects required for your GKE cluster must reside in the same region.
   - You must change the `--zone` and `--node-locations` arguments when you run the **gcloud container clusters create** command.

6. (Optional) Encrypt the service account private key, and then delete the unencrypted key to ensure that you don't leave a cleartext key on disk. See "To Encrypt the Service Account Private Key" for the steps that the Cloud Deployment Team used to encrypt the key.

7. Reserve a static IP address for your project in the region in which you will deploy the CDM.

8. Create a cloud DNS zone in the GCP project. Configure the zone's A records to point to your static IP address. Do *not* change the DNS zone's name servers.

9. At your domain registrar, configure your DNS domain's name servers to match the name servers from the cloud DNS zone.

10. Create a Cloud Filestore instance that resides in the region (`us-east1`) and zone (`us-east1-d`) in which you will deploy the CDM. Configure the Cloud Filestore with the fileshare name `export`.

    After the Cloud Filestore has been created, note its IP address. You'll need this IP address for "To Customize the Deployment YAML Files".

To Encrypt the Service Account Private Key

There are many ways to encrypt the service account private key. The following is an example of the technique used in the CDM:

1. Configure key management in your project:

    a. Enable the Cloud KMS API.

    b. Create the `forgeops-build` key ring.

    c. Create a symmetric encryption/decryption key named `cert-manager` in the `forgeops-build` key ring.

    The key is used:

    - To encrypt the `cert-manager` service account private key in the next step.

    - To decrypt the service account private key in "Deploying the Certificate Manager".

2. Encrypt the service account private key with the `cert-manager` key from the `forgeops-build` key ring:

```
$ gcloud kms encrypt \
 --plaintext-file=/path/to/downloaded/input/private/key \
 --ciphertext-file=/path/to/output/encrypted/file \
 --keyring=forgeops-build --key=cert-manager --location=global
```

3. Delete the unencrypted private key of the service account:

```
$ rm /path/to/downloaded/private/key
```

2.3. Creating and Setting up a Kubernetes Cluster

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

- "Cloning the forgeops Repository"
- "Creating the Cluster"
- "Creating Namespaces"
- "Creating a Storage Class"
- "Configuring RBAC for the Cluster"
- "Deploying an Ingress Controller"

- "Deploying the Certificate Manager"
- "Deploying Monitoring Infrastructure"

2.3.1. Cloning the forgeops Repository

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

To Obtain the forgeops Repository

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

1. Clone the `forgeops` repository:

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

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

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

2.3.2. Creating the Cluster

This section outlines how the Cloud Deployment Team created our cluster.

To Create a Kubernetes Cluster for CDM

1. If necessary, set the Google Cloud SDK configuration to reference your new project with the **gcloud config set project** command.

2. Obtain the latest patch version of Kubernetes 1.14 supported in your GKE zone:

   ```
   $ gcloud container get-server-config --zone=us-east1-d | grep 1.14
   ```

   The command returns the Kubernetes release information as *x*.*y*.*z*-gke.*N*, where *x* is the major version, *y* is the minor version, *z* is the patch version, and *N* is the security updates and bug fix number.

   When creating the cluster you don't need to specify the bug fix number of the kubernetes version.

3. Identify values that you will need when you run the command to create a Kubernetes cluster in the next step:

   - **Kubernetes Cluster Name**. Any string that you want to use as the name of your Kubernetes cluster.
   - **GCP Project ID**. The ID of the GCP project that you set up when you performed "To Configure a GCP Project for the CDM".

- **Kubernetes Version**. The latest patch version of Kubernetes 1.14 supported in your region. Use the latest patch version you obtained in Step 2 of this procedure, for example, 1.14.6.

4. Using the **gcloud container clusters create** command, create a small, medium, or large cluster for your deployment. In the example commands, replace `my-cluster` with your Kubernetes cluster name, and replace `my-project` with your GCP project ID.

   - Use the following example command to create a small cluster (1,000,000 users):

```
$ gcloud container clusters create "my-cluster" \
 --project="my-project" \
 --zone="us-east1-d" \
 --node-locations="us-east1-d,us-east1-c" \
 --cluster-version="kubernetes-version" \
 --machine-type="custom-4-16384" \
 --min-cpu-platform="Intel Skylake" \
 --image-type=COS \
 --disk-size=80 \
 --disk-type=pd-ssd \
 --network="default" \
 --subnetwork="default" \
 --num-nodes=2 \
 --min-nodes=2 \
 --max-nodes=4 \
 --labels="createdby=cdm-user" \
 --scopes "https://www.googleapis.com/auth/cloud-platform" \
 --addons=HorizontalPodAutoscaling \
 --enable-autoscaling \
 --enable-autorepair \
 --enable-cloud-logging \
 --enable-cloud-monitoring \
 --enable-ip-alias
```

   - Use the following example command to create a medium cluster (10,000,000 users):

```
$ gcloud container clusters create "my-cluster" \
 --project="my-project" \
 --zone="us-east1-d" \
 --node-locations="us-east1-d,us-east1-c" \
 --cluster-version="kubernetes-version" \
 --machine-type="custom-16-65536" \
 --min-cpu-platform="Intel Skylake" \
 --image-type=COS \
 --disk-size=80 \
 --disk-type=pd-ssd \
 --network="default" \
 --num-nodes=2 \
 --min-nodes=2 \
 --max-nodes=4 \
 --labels="createdby=cdm-user" \
 --scopes "https://www.googleapis.com/auth/cloud-platform" \
 --addons=HorizontalPodAutoscaling \
 --enable-autoscaling \
 --enable-autorepair \
 --enable-cloud-logging \
 --enable-cloud-monitoring \
 --enable-ip-alias
```

- Use the following example command to create a large cluster (100,000,000 users):

```
$ gcloud container clusters create "my-cluster" \
 --project="my-project" \
 --zone="us-east1-d" \
 --node-locations="us-east1-d,us-east1-c" \
 --cluster-version="kubernetes-version" \
 --machine-type="custom-32-73728" \
 --min-cpu-platform="Intel Skylake" \
 --image-type=COS \
 --disk-size=80 \
 --disk-type=pd-ssd \
 --network="default" \
 --num-nodes=2 \
 --min-nodes=2 \
 --max-nodes=4 \
 --labels="createdby=cdm-user" \
 --scopes "https://www.googleapis.com/auth/cloud-platform" \
 --addons=HorizontalPodAutoscaling \
 --enable-autoscaling \
 --enable-autorepair \
 --enable-cloud-logging \
 --enable-cloud-monitoring \
 --enable-ip-alias
```

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

2.3.3. Creating Namespaces

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

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

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

To Create Namespaces for CDM

1. Create the `monitoring` namespace:

   ```
   $ kubectl create namespace monitoring
   ```

2. Create the `prod` namespace:

   ```
   $ kubectl create namespace prod
   ```

3. Set context to the `prod` namespace:

   ```
   $ kubens prod
   ```

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

2.3.4. Creating a Storage Class

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

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

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

To Create a Storage Class

- Run the following **kubectl** command:

```
$ kubectl create --filename - <<EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-
ssd
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-nvme
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
EOF
```

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

2.3.5. Configuring RBAC for the Cluster

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

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

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

To Configure RBAC

1. Create the `tiller` service account:

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

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

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

3. Initialize the `tiller` service account:

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

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

2.3.6. Deploying an Ingress Controller

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

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

When specifying the IP address of the ingress controller, use the static IP address you reserved when you performed "To Configure a GCP Project for the CDM". While the ingress controller can be configured with or without a static IP, for performance and ease of administration, the CDM uses a reserved static IP address for Cloud DNS and for its ingress controller. Note that the static IP address resides in the same region as the CDM deployment.

For more information about IP addresses in GCP, see IP addresses in the Google Cloud documentation.

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

To Deploy the Ingress Controller

- Run the following command:

  ```
  $ helm install --namespace nginx --name nginx  \
   --set rbac.create=true \
   --set controller.publishService.enabled=true \
   --set controller.stats.enabled=true \
   --set controller.service.externalTrafficPolicy=Local \
   --set controller.service.type=LoadBalancer \
   --set controller.image.tag="0.21.0" \
   --set controller.service.loadBalancerIP=IP address of ingress controller \
   stable/nginx-ingress
  ```

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

2.3.7. Deploying the Certificate Manager

The CDM uses the `cert-manager` Kubernetes add-on to automate certificate issuance and management. The certificate manager uses the Let's Encrypt certificate authority.

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

To Deploy the Certificate Manager

1. If you encrypted the private key for your project's certificate management service account, decrypt it. For example:

   ```
   $ gcloud kms decrypt \
    --ciphertext-file=/path/to/encrypted/input/file \
    --plaintext-file=/path/to/decrypted/output/key \
    --keyring=forgeops-build --key=cert-manager --location=global
   ```

For more information about the certificate management service account, see "Setting up a GCP Project for the CDM".

2. Copy the file containing the service account's decrypted private key into your `forgeops` repository clone, at the path `etc/cert-manager/cert-manager.json`:

   ```
   $ cp my-decrypted-key.json /path/to/forgeops/etc/cert-manager/cert-manager.json
   ```

3. Create the `clouddns` secret. The certificate manager needs this secret to access your project's Cloud DNS zone:

   ```
   $ kubectl create secret generic clouddns \
    --from-file=/path/to/forgeops/etc/cert-manager/cert-manager.json -n kube-system
   secret "clouddns" created
   ```

4. Delete the decrypted key file after you have created the secret:

   ```
   $ rm my-decrypted-key.json /path/to/forgeops/etc/cert-manager/cert-manager.json
   ```

5. Change to the directory that contains configuration values for the `cert-manager` Helm chart:

   ```
   $ cd /path/to/forgeops/etc/cert-manager
   ```

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

   ```
   $ helm install stable/cert-manager --namespace kube-system --version v0.5.0
   NAME:   cert-manager
   LAST DEPLOYED: Fri Aug  2 16:12:17 2019
   NAMESPACE: kube-system
   STATUS: DEPLOYED
   . . .
   ```

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

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

   After the certificate manager is available, proceed to the next step.

8. Create a cluster issuer resource:

   a. Edit the `/path/to/forgeops/etc/cert-manager/cluster-issuer.yaml` file. Change the values for the following two keys:

      - For `email`, specify an e-mail address for receiving notifications.
      - For `project`, specify the name of the GCP project where your Kubernetes cluster resides.

   b. Create the cluster issuer:

```
$ kubectl create -f /path/to/forgeops/etc/cert-manager/cluster-issuer.yaml \
 --namespace kube-system
clusterissuer.certmanager.k8s.io/letsencrypt-prod created
```

You can write your own script to automate certificate manager deployment. For an example script, see `deploy-cert-manager.sh`, which is called by the example environment setup script, `gke-up.sh`.

2.3.8. Deploying Monitoring Infrastructure

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

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

To Deploy CDM Monitoring Tools

1. Refresh your local Helm repository cache:

   ```
   $ helm repo update
   ```

2. Install the `prometheus-operator` chart:

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

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

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

4. Install the `forgerock-metrics` chart:

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

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

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

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

Chapter 3 Deploying the CDM

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

Perform the following procedures:

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

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

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

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

To Customize the Deployment YAML Files

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

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. In your clone of the `forgeops` repository, change to the directory containing the appropriate artifacts:

- If you're deploying the small cluster: `/path/to/forgeops/samples/config/prod/s-cluster`
- If you're deploying the medium cluster: `/path/to/forgeops/samples/config/prod/m-cluster`
- If you're deploying the large cluster: `/path/to/forgeops/samples/config/prod/l-cluster`

3. Make the following changes to the `common.yaml` file:

   - Change the value of the `domain` key to your own domain, making sure you put a period before the domain name.
   - Change the value of the `fqdn` key to `login.prod.`*`domain`*, where *`domain`* is your own domain.

   For example:

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

4. In the `nfs` section of the `dsadmin.yaml` file, change these key values:

   - For GKE, set these key values:

     - For `server`, specify the Cloud Filestore's IP address.
     - For `path`, specify `/export`.

     In this example, the Cloud Filestore's IP address is `10.198.53.26`:

     ```
     . . .
     nfs:
       server: 10.198.53.26
       path: /export
     . . .
     ```

     For more information about CDM's Cloud Filestore requirement, see "Setting up a GCP Project for the CDM".

   - For Amazon EKS, set these key values:

     - For `server`, specify the file system ID of your Amazon EFS file system.
     - For `path`, specify `/export`.

     In this example, the domain qualified file system ID of Amazon EFS storage `fs-dcdb9c96.efs.us-east-1.amazonaws.com`:

     ```
     . . .
     nfs:
       server: fs-dcdb9c96.efs.us-east-1.amazonaws.com
       path: /export
     . . .
     ```

For more information about how to set up an Amazon EFS storage for CDM, see "Creating an Amazon EFS File System" in the *Cloud Deployment Model Cookbook for Amazon EKS*.

To Set Your Kubernetes Context

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

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

   ```
   $ kubectx
   ```

   a. Examine the **kubectx** command output. Highlighted text indicates the Kubernetes cluster in your current context.

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

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

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

   a. Examine the **kubens** command output. Highlighted output indicates the Kubernetes namespace set in your current context.

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

      ```
      $ kubens prod
      ```

To Deploy Supporting Components

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.

2. In your clone of the `forgeops` repository, change to the directory containing the appropriate artifacts:

   - If you're deploying the small cluster: `/path/to/forgeops/samples/config/prod/s-cluster`
   - If you're deploying the medium cluster: `/path/to/forgeops/samples/config/prod/m-cluster`
   - If you're deploying the large cluster: `/path/to/forgeops/samples/config/prod/l-cluster`

3. Install the `frconfig` chart, which creates Kubernetes objects used by other ForgeRock pods:

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

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

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

4. Install the `dsadmin` chart, which creates a persistent volume claim for the NFS storage class, mounts the shared-access file storage service that holds directory data backups, and archives backups to cloud storage:

```
$ helm install --name prod-dsadmin --namespace prod \
 --values common.yaml --values dsadmin.yaml /path/to/forgeops/helm/dsadmin
NAME:   prod-dsadmin
LAST DEPLOYED: Wed Jul 31 16:10:16 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/PersistentVolumeClaim
NAME       AGE
ds-backup  1s

==> v1/Deployment
dsadmin  0s

==> v1beta1/CronJob
gcs-sync  0s

==> v1/Pod(related)

NAME                      READY  STATUS   RESTARTS  AGE
dsadmin-69fb874f74-ncgz4  0/1    Pending  0         0s

==> v1/PersistentVolume

NAME       AGE
ds-backup  1s
```

To Deploy DS

Install the `ds` chart three times to create DS servers for the configuration, user, and CTS stores:

1. Install the AM configuration store:

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

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

==> v1/Service
configstore  0s

==> v1beta1/StatefulSet
configstore  0s

==> v1/Pod(related)

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

==> v1/Secret

NAME         AGE
configstore  0s
```

2. Install the AM user store:

```
$ helm install --name prod-userstore --namespace prod \
 --values common.yaml --values userstore.yaml /path/to/forgeops/helm/ds
NAME:   prod-userstore
LAST DEPLOYED: Wed Jul 31 16:14:56 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
NAME       AGE
userstore  0s

==> v1beta1/StatefulSet
userstore  0s

==> v1/Pod(related)

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

==> v1/Secret

NAME       AGE
userstore  0s

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

Note

For information about provisioning the user store, see:

- "Before You Begin"
- "Before You Begin" in the *Cloud Deployment Model Cookbook for Amazon EKS*

3. Install the AM CTS store, which holds AM session tokens, OAuth 2.0 tokens, and session blacklists and whitelists:

```
$ helm install --name prod-ctsstore --namespace prod \
 --values common.yaml --values ctsstore.yaml /path/to/forgeops/helm/ds
NAME:   prod-ctsstore
LAST DEPLOYED: Wed Jul 31 16:15:53 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/StatefulSet
NAME      AGE
ctsstore  0s

==> v1/Pod(related)

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

==> v1/Secret

NAME      AGE
ctsstore  0s

==> v1/ConfigMap
ctsstore  0s

==> v1/Service
ctsstore  0s
```

4. Review the logs from the directory server pods to verify that the following DS pods started up successfully:

   - `configstore-0`
   - `configstore-1`
   - `userstore-0`
   - `userstore-1`
   - `ctsstore-0`

- `ctsstore-1`

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

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

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

To Deploy AM

1. Install the `openam` chart:

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

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

==> v1/Service
openam  1s

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

==> v1beta1/Ingress
openam  1s

==> v1/Pod(related)

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

==> v1/Secret

NAME            AGE
openam-secrets  1s
```

2. Install the `amster` chart:

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

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

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

==> v1beta1/Deployment
amster  0s

==> v1/Pod(related)

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

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

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

   ```
   $ kubectl get pods
   NAME                                            READY     STATUS    RESTARTS   AGE
   amster-7b8b6d9667-hxq57                         2/2       Running   0          1m
   configstore-0                                   1/1       Running   0          10m
   configstore-1                                   1/1       Running   0          10m
   ctsstore-0                                      1/1       Running   0          9m
   ctsstore-1                                      1/1       Running   0          9m
   dsadmin-69fb874f74-dln6b                        1/1       Running   0          11m
   prod-openam-openam-6f846bbf69-gt82c             1/1       Running   0          2m
   userstore-0                                     1/1       Running   0          10m
   userstore-1                                     1/1       Running   0          9m
   ```

   b. Review the output. Deployment is complete when:

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

      - All entries in the `STATUS` column indicate `Running`.

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

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

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

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

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

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

```
. . .
Got Response code 200
AM web app is up and ready to be configured
About to begin configuration
Executing Amster to configure AM
Executing Amster script /opt/amster/scripts/00_install.amster
Jul 31, 2019 4:51:29 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
-------------------------------------------------------------------------------
am> :load /opt/amster/scripts/00_install.amster
07/31/2019 04:51:32:699 PM GMT: Checking license acceptance...
07/31/2019 04:51:32:700 PM GMT: License terms accepted.
07/31/2019 04:51:32:706 PM GMT: Checking configuration directory /home/forgerock/openam.
07/31/2019 04:51:32:707 PM GMT: ...Success.
07/31/2019 04:51:32:712 PM GMT: Tag swapping schema files.
07/31/2019 04:51:32:747 PM GMT: ...Success.
07/31/2019 04:51:32:747 PM GMT: Loading Schema odsee_config_schema.ldif
07/31/2019 04:51:32:825 PM GMT: ...Success.
07/31/2019 04:51:32:825 PM GMT: Loading Schema odsee_config_index.ldif
07/31/2019 04:51:32:855 PM GMT: ...Success.
07/31/2019 04:51:32:855 PM GMT: Loading Schema cts-container.ldif
07/31/2019 04:51:32:945 PM GMT: ...Success
.
. . .
```

The following output indicates that deployment is complete:

```
. . .
07/31/2019 04:51:53:215 PM GMT: Setting up monitoring authentication file.
Configuration complete!
Executing Amster script /opt/amster/scripts/01_import.amster
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
------------------------------------------------------------------------------
am> :load /opt/amster/scripts/01_import.amster
Importing directory /git/config/default/am/empty-import
Import completed successfully
Configuration script finished
+ pause
+ echo Args are 0
+ echo Container will now pause. You can use kubectl exec to run export.sh
+ true
+ sleep 1000000
Args are 0
Container will now pause. You can use kubectl exec to run export.sh
```

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

   After the restart, AM uses the appropriate CTS configuration.

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

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

   b. Delete the pod:

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

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

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

To Deploy IDM

1. Install the `postgres-openidm` chart:

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

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

```
postgres-openidm  1s

==> v1/Service
postgresql  1s

==> v1beta1/Deployment
postgres-openidm  1s

==> v1/Pod(related)

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

==> v1/Secret

NAME              AGE
postgres-openidm  1s

==> v1/ConfigMap
openidm-sql  1s


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

To get your user password run:

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

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

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

2. Install the `openidm` chart:

```
$ helm install --name prod-openidm --namespace prod \
 --values common.yaml --values openidm.yaml /path/to/forgeops/helm/openidm
NAME:   prod-openidm
LAST DEPLOYED: Thu Aug  1 12:46:24 2019
NAMESPACE: prod
STATUS: DEPLOYED

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

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

==> v1/Service
```

```
openidm  1s

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

==> v1beta1/Ingress
openidm  1s

==> v1/Pod(related)

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


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

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

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

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

   ```
   $ kubectl get pods
   NAME                                          READY     STATUS    RESTARTS   AGE
   amster-7b8b6d9667-hxq57                       2/2       Running   0          31m
   configstore-0                                 1/1       Running   0          40m
   configstore-1                                 1/1       Running   0          40m
   ctsstore-0                                    1/1       Running   0          40m
   ctsstore-1                                    1/1       Running   0          39m
   dsadmin-69fb874f74-dln6b                      1/1       Running   0          41m
   postgres-openidm-6b6fb898cb-mthrj             1/1       Running   0          17m
   prod-openam-openam-6f846bbf69-867rc           1/1       Running   0          24m
   prod-openidm-openidm-696685d8cb-pt5x4         2/2       Running   0          12m
   userstore-0                                   1/1       Running   0          40m
   userstore-1                                   1/1       Running   0          39m
   ```

   b. Review the output. Deployment is complete when:

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

      - All entries in the `STATUS` column indicate `Running`.

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

To Deploy IG

1. Install the `openig` chart:

```
$ helm install --name prod-openig --namespace prod \
 --values common.yaml --values openig.yaml /path/to/forgeops/helm/openig
NAME:   prod-openig
LAST DEPLOYED: Thu Aug  1 12:46:24 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
openig  1s

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

==> v1beta1/Ingress
openig  1s

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


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


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

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

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

```
$ kubectl get pods
NAME                                          READY     STATUS    RESTARTS   AGE
amster-7b8b6d9667-hxq57                       2/2       Running   0          31m
configstore-0                                 1/1       Running   0          40m
configstore-1                                 1/1       Running   0          40m
ctsstore-0                                    1/1       Running   0          40m
ctsstore-1                                    1/1       Running   0          39m
dsadmin-69fb874f74-dln6b                      1/1       Running   0          41m
postgres-openidm-6b6fb898cb-mthrj             1/1       Running   0          17m
prod-openam-openam-6f846bbf69-867rc           1/1       Running   0          24m
prod-openidm-openidm-696685d8cb-pt5x4         2/2       Running   0          12m
prod-openig-openig-9b4dfb574-jdp4n            1/1       Running   0          1m
userstore-0                                   1/1       Running   0          40m
userstore-1                                   1/1       Running   0          39m
```

   b. Review the output. Deployment is complete when:

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

- All entries in the `STATUS` column indicate `Running`.

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

To Deploy the web Application

The `web` application is required for running the IG reverse proxy benchmark.

1. Install the `web` chart:

```
$ helm install --name prod-web --namespace prod \
 --values common.yaml /path/to/forgeops/helm/web
Release "prod-web" does not exist. Installing it now.
NAME:   prod-web
LAST DEPLOYED: Tue Jul 30 16:33:45 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                       READY  STATUS   RESTARTS  AGE
prod-web-747bdd74c7-zz8wd  0/1    Pending  0         0s

==> v1/ConfigMap

NAME      AGE
prod-web  0s

==> v1/Service
prod-web  0s

==> v1beta2/Deployment
prod-web  0s

==> v1beta1/Ingress
prod-web  0s


NOTES:
1. Get the application URL by running these commands:
```

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

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

```
$ kubectl get pods
NAME                                       READY     STATUS    RESTARTS   AGE
amster-7b8b6d9667-hxq57                    2/2       Running   0          31m
configstore-0                              1/1       Running   0          40m
configstore-1                              1/1       Running   0          40m
ctsstore-0                                 1/1       Running   0          40m
ctsstore-1                                 1/1       Running   0          39m
dsadmin-69fb874f74-dln6b                   1/1       Running   0          41m
postgres-openidm-6b6fb898cb-mthrj          1/1       Running   0          17m
prod-openam-openam-6f846bbf69-867rc        1/1       Running   0          24m
prod-openidm-openidm-696685d8cb-pt5x4      2/2       Running   0          12m
prod-openig-openig-9b4dfb574-jdp4n         1/1       Running   0          1m
prod-web-747bdd74c7-zz8wd                  1/1       Running   0          5s
userstore-0                                1/1       Running   0          40m
userstore-1                                1/1       Running   0          39m
```

b. Review the output. Deployment is complete when:

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

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

To Scale the Deployment

The ForgeRock Identity Platform Helm charts create one AM pod, one IDM pod, and two IG pods. To create a second pod for AM and IDM, run the **kubectl scale replicas=2** command.

1. Get the AM deployment name:

```
$ kubectl get deployments
NAME                        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
amster                      1         1         1            1           34m
dsadmin                     1         1         1            1           45m
postgres-openidm            1         1         1            1           20m
prod-openam-openam          1         1         1            1           35m
prod-openidm-openig         1         1         1            1           16m
```

2. Scale the number of AM pods to two:

```
$ kubectl scale --replicas=2 deployment prod-openam-openam
deployment.extensions/prod-openam-openam scaled
```

3. Get the IDM stateful set name:

```
$ kubectl get statefulsets
NAME                        DESIRED   CURRENT   AGE
prod-openam-openidm         1         1         35m
userstore                   1         1         54m
configstore                 1         1         55m
ctsstore                    1         1         56m
```

4. Scale the number of IDM pods to two:

```
$ kubectl scale --replicas=2 statefulset prod-openidm-openidm
statefulset.extensions/prod-openidm-openidm scaled
```

5. Verify that two `openam`, `openidm`, and `openig` pods are available in your cluster:

```
$ kubectl get pods
NAME                                        READY     STATUS    RESTARTS   AGE
amster-7b8b6d9667-hxq57                     2/2       Running   0          37m
configstore-0                               1/1       Running   0          47m
configstore-1                               1/1       Running   0          46m
ctsstore-0                                  1/1       Running   0          46m
ctsstore-1                                  1/1       Running   0          46m
dsadmin-69fb874f74-dln6b                    1/1       Running   0          48m
postgres-openidm-6b6fb898cb-mthrj           1/1       Running   0          23m
prod-openam-openam-6f846bbf69-867rc         1/1       Running   0          30m
prod-openam-openam-6f846bbf69-bbpfg         1/1       Running   0          1m
prod-openidm-openidm-696685d8cb-nbd2n       2/2       Running   0          27s
prod-openidm-openidm-696685d8cb-pt5x4       2/2       Running   0          19m
prod-openig-openig-6a12fe4501-n1402         2/2       Running   0          50s
prod-openig-openig-6a12fe4501-g2x01         2/2       Running   0          48m
userstore-0                                 1/1       Running   0          46m
userstore-1                                 1/1       Running   0          46m
```

Chapter 4 Using the CDM

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

4.1. Accessing ForgeRock Identity Platform Services

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

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

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

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

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

4.1.1. Before You Begin

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

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

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

4.1.2. Accessing AM Services

Access the AM console and REST APIs as follows:

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

To Access the AM Console

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

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

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

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

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

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

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

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

   AM prompts you to log in.

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

To Access the AM REST APIs

1. Start a terminal window session.

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

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

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

4.1.3. Accessing IDM Services

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

Access the IDM Admin UI and REST APIs as follows:

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

To Access the IDM Admin UI Console

1. Open a new window or tab in a web browser.
2. Navigate to the IDM Admin UI URL, https://openidm.prod.*my-domain*/admin.

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

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

To Access the IDM REST APIs

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

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

4.1.4. Accessing IG Services

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

To Verify IG Is Operational

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

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

   You should see a message similar to the following:

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

4.1.5. Accessing DS

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

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

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

4.2. Monitoring the CDM

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

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

To Access Grafana Dashboards

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

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

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

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

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

To Access the Prometheus Web UI

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

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

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

   The Prometheus web UI appears in the browser.

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

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

To enable monitoring in the CDM, see:

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

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

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

Chapter 5
Benchmarking the CDM Performance

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

- "About CDM Benchmarking"
- "Before You Begin"
- "Running Directory Services Benchmark Tests"
- "Running AM, IDM, or IG Benchmark Tests"

5.1. About CDM Benchmarking

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

Important

These results are based on Cloud Deployment Model (CDM) benchmark tests we conducted through January, 2019. We conducted the tests using Kubernetes version 1.11, and default sizing and configurations described in this guide. Factors beyond the scope of the CDM or its default sizing and configuration may affect your benchmark test results. These factors might include (but are not limited to): updates to the cloud provider's SDK; changes to third-party software required for Kubernetes; changes you have made to sizing or configuration to suit your business needs.

5.1.1. Default Sizing and Configuration

By default, the CDM is sized to strike a balance between optimal performance and low production cost. After you've successfully deployed the CDM, your deployment is sized and configured to the following specifications.

Description	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Kubernetes version	1.11.2	1.11.2	1.11.2
Nodes	4	4	4
vCPU	4	16	32

Description	Small Cluster	Medium Cluster	Large Cluster
Memory	16 GB	64 GB	72 GB
Storage	PD-SSD 100GB	PD-SSD 500GB and 850GB	PD-SSD 500GB and 1TB
Virtual Machine Size	Custom 4x16 Skylake	Custom 16x64 Skylake	Custom 32x72 Skylake
Average DS user entry size	1K	1K	1K
AM CTS storage scheme	`GrantSet`	`GrantSet`	`GrantSet`
AM CTS reaper	Disabled on AM Enabled TTL on CTS DS	Disabled on AM Enabled TTL on CTS DS	Disabled on AM Enabled TTL on CTS DS

5.1.2. Cost Estimates for Running CDM on Google Cloud Platform

The following table compares approximate monthly costs of running the CDM using the default small, medium, and large cluster configurations. These approximations were derived using GCP cost calculators and analyzing actual GCP billings for resource usage.

	Small Cluster	Medium Cluster	Large Cluster
Number of Users	1,000,000	10,000,000	100,000,000
Benchmark Tests Date	January, 2019	January, 2019	January, 2019
Virtual Machines Cost per VM with 1-year commitment.	4x16 = $104/month 104 x 4 nodes = $416/month	16x64 = $358/month 358 x 4 nodes = $1430/month	32x72 = $706/month 706 x 4 nodes = $2824/month
Network SSD Storage For databases.	400 GB = $68month	3000 GB = $510/month	4000 GB = $680/month
Google Cloud Filestore For backup.	$200/month	$200/month	$400/month
Google Cloud Storage For backup archival.	1 TB = $20/month	2 TB = $40/month	4 TB = $80/month
Networking Includes load balancing, inter-zone egress, and other resources based on network traffic volume.	$20/month	$40/month	$80/month
Cloud SQL This is optional if IDM is deployed.	1x4 - HA - 100GB storage 200 GB backup $100/month	2x8 - HA - 200GB storage 400 GB backup $300/month	4x16 - HA - 300GB storage 600 GB backup $500/month
GCP Logging and Monitoring Optional Stackdriver dashboard.	$100/month	$200/month	$300/month

	Small Cluster	Medium Cluster	Large Cluster
Estimated Total Cost Per Month	$524 $944 with all options.	$2020 $2500 with all options.	$3584 $4864 with all options.

5.1.3. 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.

5.2. Before You Begin

Before you can test CDM performance, you must generate test users, provision the CDM userstores, and prime the Directory Servers. Provision the first userstore by creating and importing an LDIF file. Provision the second userstore using the DS backup and restore capabilities and setting the generation ID. This ensures that the data is synchronized for subsequent replication.

Complete the following steps:

To Provision the Userstores with Test Users

1. Open a shell in the `userstore-0` pod. Example:

   ```
   $ kubectl exec userstore-0 -it bash
   ```

2. Edit the `/opt/opendj/config/MakeLDIF/example.template` file:

   a. Change the `suffix` definition on the first line to `define suffix=ou=identities`.

   b. Change the `numusers` definition:

      - For a small cluster, specify `define numusers=1000000`.
      - For a medium cluster, specify `define numusers=10000000`.

- For a large cluster, specify `define numusers=100000000`.

c. In the `branch: [suffix]` section, change the line `objectClass: domain` to `objectClass: organizationalUnit`.

An example of the edited file snippet for a small cluster:

```
define suffix=ou=identities
define maildomain=example.com
define numusers=1000000

branch: [suffix]
objectClass: top
objectClass: organizationalUnit
...
```

3. Provision the `userstore-0` directory with test users:

a. Generate test users using the **makeldif** command:

```
$ /opt/opendj/bin/makeldif -o /tmp/1m.ldif \
 /opt/opendj/config/MakeLDIF/example.template
Processed 1000 entries
Processed 2000 entries
Processed 3000 entries
Processed 4000 entries
...
LDIF processing complete. 1000002 entries written
```

b. Zip the resulting LDIF file, and copy the zipped archive to the `/opt/opendj/bak` directory:

```
$ gzip /tmp/1m.ldif
$ cp /tmp/1m.ldif.gz /opt/opendj/bak
```

c. Import the test users into the `userstore-0` directory:

```
$ /opt/opendj/bin/import-ldif -F -c -l /opt/opendj/bak/1m.ldif.gz \
 -p 4444 -D "cn=Directory Manager" -w password --trustAll -n amIdentityStore
Import task 20190731145319759 scheduled to start immediately
[31/Jul/2019:14:53:19 +0000] severity="NOTICE" msgCount=0 msgID=org.opends
.messages.backend-413 message="Import task 20190731145319759 started
 execution"
...
[31/Jul/2019:14:53:43 +0000] severity="NOTICE" msgCount=14 msgID=org.opends.messages.backend-531
 message="Total import time was 22 seconds. Phase one processing completed in 14 seconds, phase
 two processing completed in 8 seconds"
[31/Jul/2019:14:53:43 +0000] severity="NOTICE" msgCount=15 msgID=org.opends
.messages.backend-519 message="Processed 100002 entries, imported 100002,
 skipped 0, rejected 0 and migrated 0 in 23 seconds (average rate 4288.4/
sec)"
...
Import task 20190731145319759 has been successfully completed
```

4. For small, medium, and large clusters, prime the `userstore-0` directory. Run the following LDAP search:

```
$ ldapsearch -p 1389 -D "cn=Directory Manager" \
 -w password -b "ou=identities" objectclass=* > /dev/null
```

It could take as long as ten minutes or more to complete the search operation.

5. Make a backup of the `userstore-0` directory containing the imported test users:

```
$ /opt/opendj/scripts/backup.sh
Backup task 20190731145523468 scheduled to start immediately
[31/Jul/2019:14:55:23 +0000] severity="NOTICE" msgCount=0 msgID=org.opends.messages.backend-413
 message="Backup task 20190731145523468 started execution"
[31/Jul/2019:14:55:23 +0000] severity="NOTICE" msgCount=1 msgID=org.opends.messages.tool-280
 message="Starting backup for backend amIdentityStore"
[31/Jul/2019:14:55:30 +0000] severity="NOTICE" msgCount=2 msgID=org.opends.messages.utility-321
 message="Archived backup file: 00000000.jdb"
[31/Jul/2019:14:55:30 +0000] severity="NOTICE" msgCount=3 msgID=org.opends.messages.tool-283
 message="The backup process completed successfully"
[31/Jul/2019:14:55:30 +0000] severity="NOTICE" msgCount=4 msgID=org.opends.messages.backend-414
 message="Backup task 20190731145523468 finished execution in the state Completed successfully"
Backup task 20190731145523468 has been successfully completed
```

6. Close the shell in the `userstore=0` pod.

```
$ exit
```

7. Provision the `userstore-1` directory with test users:

   a. Open a shell in the `userstore-1` pod:

   ```
   $ kubectl exec userstore-1 -it bash
   ```

   b. Restore the test users into the `userstore-1` directory from the `userstore-0` directory backup:

```
$ /opt/opendj/bin/restore \
 --hostname localhost --port 4444 --bindDN cn="Directory Manager" \
 --bindPasswordFile /var/run/secrets/opendj/dirmanager.pw --trustAll \
 --backupDirectory /opt/opendj/bak/default/userstore-prod/amIdentityStore
 Restore task 20190731150155326 scheduled to start immediately
[31/Jul/2019:15:01:55 +0000] severity="NOTICE" msgCount=0 msgID=org.opends.messages.backend-413
 message="Restore task 20190731150155326 started execution"
[31/Jul/2019:15:01:55 +0000] severity="NOTICE" msgCount=1 msgID=org.opends.messages.backend-370
 message="The backend amIdentityStore is now taken offline"
[31/Jul/2019:15:01:56 +0000] severity="NOTICE" msgCount=2 msgID=org.opends.messages.utility-320
 message="Restored backup file: 00000000.jdb (size 139284948)"
[31/Jul/2019:15:01:56 +0000] severity="INFORMATION" msgCount=3 msgID=org.opends.messages.backend
-410 message="JE backend 'amIdentityStore' does not specify the number of cleaner threads:
 defaulting to 2 threads"
[31/Jul/2019:15:01:56 +0000] severity="INFORMATION" msgCount=4 msgID=org.opends.messages.backend
-411 message="JE backend 'amIdentityStore' does not specify the number of lock tables: defaulting
 to 13"
[31/Jul/2019:15:01:58 +0000] severity="NOTICE" msgCount=5 msgID=org.opends.messages.backend-513
 message="The database backend amIdentityStore containing 100002 entries has started"
[31/Jul/2019:15:01:58 +0000] severity="WARNING" msgCount=6 msgID=org.opends.messages.replication
-96 message="Directory server DS(11) has connected to replication server RS(11) for domain
 "ou=identities" at 10.32.3.11:8989, but the generation IDs do not match, indicating that a full
 re-initialization is required. The local (DS) generation ID is 19094517 and the remote (RS)
 generation ID is 65525"
[31/Jul/2019:15:01:58 +0000] severity="NOTICE" msgCount=7 msgID=org.opends.messages.backend-414
 message="Restore task 20190731150155326 finished execution in the state Completed successfully"
Restore task 20190731150155326 has been successfully completed
```

8. Set the generation ID on the `userstore-1` server so that replication between the `userstore-0` and `userstore-1` servers works:

   a. Review the standard output from the **restore** command that you ran in the preceding step. Locate the message similar to this message:

   ```
   ...the generation IDs do not match, indicating that a full re-initialization is required. The
    local (DS) generation ID is 19094517 and the remote (RS) generation ID is 65525"
   ```

   b. Obtain the local generation ID from the message. In the example, the local generation ID is `19094517`.

   c. Run the following command to set the local generation ID of the `userstore-1` server to the generation ID of the `userstore-1` server:

   ```
   $ /opt/opendj/bin/ldapmodify -p 1389 -D "cn=Directory Manager" -w password <<EOF
   dn: ds-task-id=mytask-1234,cn=Scheduled Tasks,cn=Tasks
   objectClass: ds-task-reset-generation-id
   ds-task-class-name: org.opends.server.tasks.SetGenerationIdTask
   ds-task-reset-generation-id-domain-base-dn: ou=identities
   ds-task-reset-generation-id-new-value: 19094517
   EOF
   ```

   In the preceding command:

   - The value for `ds-task-id` can be any unique value.

- The value for `ds-task-reset-generation-id-new-value` must be the local generation ID that you obtained from the **restore** command output.

d. Review the `/opt/opendj/logs/errors` file. You should see a message similar to the following:

```
[02/Aug/2019:18:08:54 +0100] category=SYNC severity=NOTICE msgID=78 msg=The generation ID for
 domain "dc=example,dc=com" has been reset to 65525
```

e. Continue examining the `/opt/opendj/logs/errors` file. After a short wait, you should see a message similar to the following:

```
[31/Jul/2019:17:56:12 +0100] category=SYNC severity=NOTICE msgID=62 msg=Directory server DS(7775)
 has connected to replication server RS(20607) for domain "dc=example,dc=com" at 127.0.0.1:8990
 with generation ID 65525
```

9. Prime the `userstore-1` directory by running the following LDAP search:

```
$ ldapsearch -p 1389 -D "cn=Directory Manager" \
 -w password -b "ou=identities" objectclass=* > /dev/null
```

It could take as long as ten minutes or more to complete the search operation.

After both of the DS servers have been primed, you can run any of the benchmark tests in this guide.

5.3. Running Directory Services Benchmark Tests

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

To Run DS Benchmark Tests

1. The userstore must be provisioned, and the Directory Server must be primed.

   See "To Provision the Userstores with Test Users".

2. Open a shell in the `userstore-1` pod, and run the `ds-bench.sh` script. Example:

```
$ scripts/ds-bench.sh srch 600 localhost 1000000
```

In this example:

- `srch` specifies the searchrate tool.
- `600` specifies the duration (in seconds) of the load.
- `localhost`specifies the target host.

- `1000000` specifies the number of users.

See the `ds-bench` script in Git at `forgeops/docker/ds/scripts/ds-bench.sh`.

5.3.1. Searchrate

The DS `searchrate` utility measures the search throughput and response time of a directory service using user-defined searches. Before running this test, the userstore must be provisioned with users. See "Before You Begin" for more information.

To run the `searchrate` utility, open a shell in the `userstore-1` pod and run the `ds-bench` script. Example:

```
$ /opt/opendj/scripts/ds-bench.sh srch 600 localhost 1000000
```

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Throughput	~13 K searches/second	~50 K searches/second	~36 K searches/second
Response times	Mean: 25 ms	Mean:10 ms	Mean:13 ms
Resource requests	Memory: 3 Gi CPU: 2000 m	Memory: 8 Gi CPU: 2000 m	Memory: 20 Gi CPU: 4000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 8000 m	Memory: 27 Gi CPU: 10000 m
Pod usage	Memory: 3928 Mi CPU: 2834 m	Memory: 8587 Mi CPU: 7602 m	Memory: 27276 Mi CPU: 9893 m
Node usage	Memory: 9085 Mi CPU: 2883 m	Memory: 28145 Mi CPU: 8820 m	Memory: 47709 Mi CPU: 12348 m

5.3.2. Modrate

The DS `modrate` utility measures the modify throughput and response time of a directory service using user-defined modifications. Before running this test, the userstore must be provisioned with users. See "Before You Begin" for more information.

To run the `modrate` utility, open a shell in the `userstore-1` pod and run the `ds-bench` script. Example:

```
$ /opt/opendj/scripts/ds-bench.sh mod 600 localhost 1000000
```

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Throughput	~3 K mods/second	~6.4 K mods/second	~4 K mods/second

Metric	Small Cluster	Medium Cluster	Large Cluster
Response times	Mean: 3 ms	Mean: 1.2 ms	Mean: 2 ms
Resource requests	Memory: 3 Gi CPU: 2000 m	Memory: 8 Gi CPU: 2000 m	Memory: 20 Gi CPU: 4000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 8000 m	Memory: 27 Gi CPU: 10000 m
Pod usage	Memory: 4355 Mi CPU: 2966 m	Memory: 10768 Mi CPU: 7882 m	Memory: 27276 Mi CPU: 9893 m
Node usage	Memory: 9563 Mi CPU: 3129 m	Memory: 20706 Mi CPU: 9137 m	Memory: 47709 Mi CPU: 12348 m

5.3.3. Addrate

The DS `addrate` utility measures the add throughput and response time of a directory server using user-defined entries. Before running this test, the userstore must be provisioned with users. See "Before You Begin" for more information.

To run the `addrate` utility, open a shell in the `userstore-1` pod and run the `ds-bench` script. Example:

```
$ /opt/opendj/scripts/ds-bench.sh add 600 localhost 1000000
```

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Throughput	~300 adds/second	~4.5 K adds/second	~200 K adds/second
Response times	Mean: 25 ms	Mean: 1.6 ms	Mean: 42 ms
Resource requests	Memory: 3 Gi CPU: 2000 m	Memory: 8 Gi CPU: 2000 m	Memory: 20 Gi CPU: 4000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 8000 m	Memory: 27 Gi CPU: 10000 m
Pod usage	Memory: 3784 Mi CPU: 1470 m	Memory: 10815 Mi CPU: 7923 m	Memory: 27276 Mi CPU: 9893 m
Node usage	Memory: 8896 Mi CPU: 2841 m	Memory: 20727 Mi CPU: 10265 m	Memory: 47709 Mi CPU: 12348 m

5.4. Running AM, IDM, or IG Benchmark Tests

The Cloud Deployment Team developed the `gatling-benchmark` Helm chart for benchmarking AM, IDM, and IG performance. First the Helm chart runs one of a number of Scala-based use case simulations. Then the Helm chart exposes test results that you can view in a browser or download in a zipped file.

To Run AM, IDM, or IG Benchmark Tests

1. In the `/path/to/forgeops/helm/gatling-benchmark` directory, edit the `values.yaml` file.

   a. In the `testname` field, specify the name of the simulation you want to run.

      See "Access Management Benchmark Tests", "Identity Management Benchmark Tests", or "Identity Gateway Benchmark Tests" for more information.

   b. In the `users` field, specify **`1000000`** for a small cluster, or **`10000000`** for a medium cluster. You can specify **`10000000`** for a large AM or a large IG cluster.

   c. In the `concurrency` field, specify **`50`**.

   d. In the `duration` field, specify **`600`**.

2. The userstores must be provisioned, and the Directory Server must be primed.

   See "To Provision the Userstores with Test Users".

3. To start the benchmark test, install the `gatling-benchmark` Helm chart. By default, the test begins immediately. Example:

```
$ helm install --name benchmark helm/gatling-benchmark
 NAME:   benchmark
 LAST DEPLOYED: Tue Jul 30 14:56:40 2019
 NAMESPACE: prod
 STATUS: DEPLOYED

 RESOURCES:
 ==> v1beta1/Ingress
 NAME     AGE
 gatling  1s

 ==> v1/Pod(related)

 NAME                                READY  STATUS   RESTARTS  AGE
 forgeops-benchmark-5957478bb4-qd9d5  0/1    Pending  0         0s

 ==> v1/ConfigMap

 NAME        AGE
 nginx-conf  1s

 ==> v1/PersistentVolumeClaim
 forgeops-benchmark-pvc  1s

 ==> v1/Service
 benchmark-gatling-benchmark  1s

 ==> v1beta1/Deployment
 forgeops-benchmark  1s
```

4. To verify that the `gatling-benchmark` Helm chart is installed, run the `kubectl get pods` command. For example:

   ```
   $ kubectl get pods
   NAME                                  READY   STATUS    RESTARTS   AGE
   amster-689d795777-59p67               2/2     Running   0          1d
   configstore-0                         1/1     Running   0          8h
   configstore-1                         1/1     Running   0          9h
   ctsstore-0                            1/1     Running   0          8h
   ctsstore-1                            1/1     Running   0          1d
   dsadmin-784fd647b4-pzbcs              1/1     Running   0          1d
   forgeops-benchmark-5957478bb4-qd9d5   0/1     Init:0/2  0          3h
   postgres-openidm-6fbf98d5bc-v2hhm     1/1     Running   0          1d
   prod-openam-openam-5cf6b9498-c4sgh    1/1     Running   0          8h
   prod-openam-openam-5cf6b9498-cp247    1/1     Running   1          9h
   prod-openidm-openidm-0                2/2     Running   1          9h
   prod-openidm-openidm-1                2/2     Running   1          1d
   userstore-0                           1/1     Running   0          9h
   userstore-1                           1/1     Running   0          8h]
   ```

   In this example, the `forgeops-benchmark-5957478bb4-qd9d5` pod is created and initializing.

5. In the `/path/to/forgeops/helm/gatling-benchmark` directory, run the **get-logs.sh** script.

   Gatling output is sent directly to the console. For example:

   ```
   $ ./get-logs.sh
   [Loop n.0
    READY to run gatling.sh

    ----------------------------
    Provided script parameters:
    java_options: -Dam_host=login.prod.perf.forgerock-qa.com -Dam_port=443 -Dam_protocol=https -
   Didm_host=openidm.prod.perf.forgerock-qa.com -Didm_port=443 -Didm_protocol=https -Dig_host=openig
   -Dig_port=80 -Dig_protocol=http -DlogoutPercent=0 -Dusers=1000000 -Dconcurrency=50 -Dduration=30 -
   Dwarmup=60 -Dissue_token_info=false -Doauth2_client_id=clientOIDC_0 -Doauth2_client_pw=password
    cmd_options:
    gatling_args:-m -s am.AMAccessTokenSim -rd am.AMAccessTokenSim
    ----------------------------

    Starting gatling simulation

    ----------------------------
    GATLING COMMAND: gatling.sh -m -s am.AMAccessTokenSim -rd am.AMAccessTokenSim
    ----------------------------
    GATLING_HOME is set to /opt/gatling
    20:11:29.149 [WARN ] i.g.c.ZincCompiler$ - Pruning sources from previous analysis, due to
    incompatible CompileSetup.
    Simulation am.AMAccessTokenSim started...

    ================================================================================
    2018-11-27 20:11:42                                           5s elapsed
    ---- Requests ------------------------------------------------------------------
    > Global                                                   (OK=389    KO=0     )
    > Rest login stage                                         (OK=131    KO=0     )
    > Authorize stage                                          (OK=130    KO=0     )
    > AccessToken stage                                        (OK=128    KO=0     )
   ```

```
---- OAuth2 Auth code flow -----------------------------------------------------
[------                                                                    ]  0%
waiting: 46     / active: 4      / done:0
================================================================================



================================================================================
2018-11-27 20:11:47                                           10s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                    (OK=2178   KO=0     )
> Rest login stage                                          (OK=727    KO=0     )
> Authorize stage                                           (OK=726    KO=0     )
> AccessToken stage                                         (OK=725    KO=0     )

---- OAuth2 Auth code flow -----------------------------------------------------
[------------                                                              ]  0%
waiting: 42     / active: 8      / done:0
================================================================================



================================================================================
2018-11-27 20:11:52                                           15s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                    (OK=5506   KO=0     )
> Rest login stage                                          (OK=1840   KO=0     )
> Authorize stage                                           (OK=1834   KO=0     )
> AccessToken stage                                         (OK=1832   KO=0     )

---- OAuth2 Auth code flow -----------------------------------------------------
[------------------                                                        ]  0%
waiting: 38     / active: 12     / done:0
================================================================================



================================================================================
2018-11-27 20:11:57                                           20s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                    (OK=10577  KO=0     )
> Rest login stage                                          (OK=3532   KO=0     )
> Authorize stage                                           (OK=3525   KO=0     )
> AccessToken stage                                         (OK=3520   KO=0     )

---- OAuth2 Auth code flow -----------------------------------------------------
[------------------------                                                  ]  0%
waiting: 34     / active: 16     / done:0
================================================================================
```

The test will continue to run for the duration you specified in the `values.yaml` file.

6. (Optional) To stop a test, run the **helm del --purge** command. Example:

```
$ helm del --purge benchmark
```

To View Test Results in a Browser

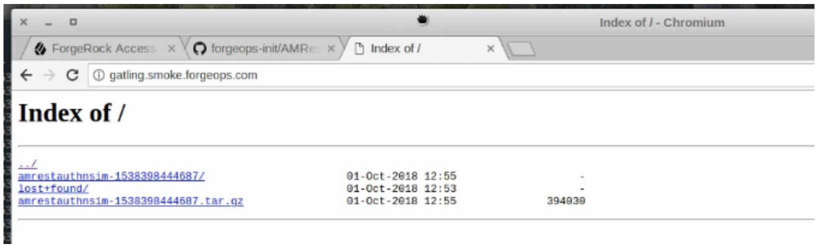
1. Confirm that an ingress address with an FQDN exists in the `/etc/hosts` file. Example:

```
$ kubectl get ingress
 NAME      HOSTS                          ADDRESS         PORTS     AGE
 gatling   gatling.prod.forgeops.com      35.231.229.20   80        4h
 openam    login.prod.forgeops.com        35.231.229.20   80, 443   1d
 openidm   openidm.prod.forgeops.com      35.231.229.20   80, 443   1d
```

2. In a browser go to the URL of the Gatling host listed in the previous step.

   In this example, the Gatling host URL is http://gatling.prod.forgeops.com.

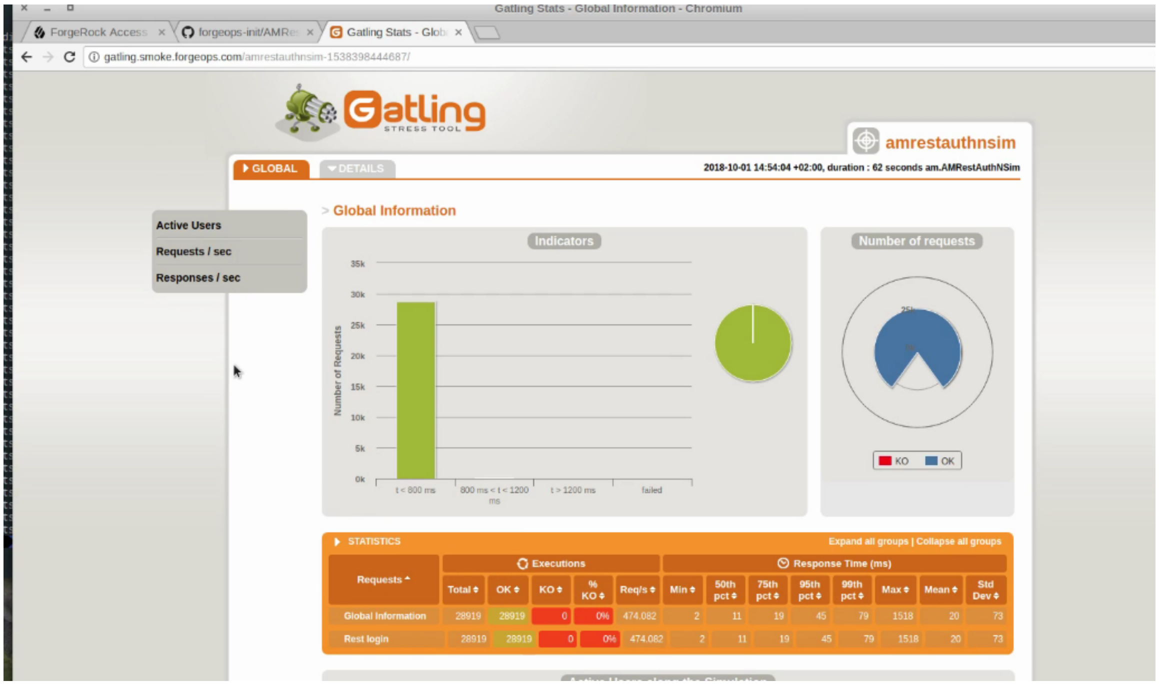
   a. You can download the HTML report in the `*.tar.gz` file for future use. Once the `gatling-benchmark` Helm chart is deleted, you can no longer access these files.



Index of /

```
../
amrestauthnsim-1538398444687/                     01-Oct-2018 12:55                   -
lost+found/                                       01-Oct-2018 12:53                   -
amrestauthnsim-1538398444687.tar.gz               01-Oct-2018 12:55              394030
```

   b. Click the link to go to the Gatling dashboard.



Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	28919	28919	0	0%	474.082	2	11	19	45	79	1518	20	73
Rest login	28919	28919	0	0%	474.082	2	11	19	45	79	1518	20	73

5.4.1. Access Management Benchmark Tests

The AMRestAuthNSim.scala simulation tests the authentication rates using the REST API. This simulation was run for stateless and stateful session tokens. A stateful session has an opaque token that persists in the CTS server so that the session token can be invalidated immediately if needed.

The AMAccessTokenSim.scala simulation tests stateless and stateful tokens in the OAuth 2.0 authorization code flow. A stateless token is a JWT that carries its state as a cookie set in the browser; the JWT has no corresponding server component.

5.4.1.1. REST Login - Stateful Authentication

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

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. Click Save Changes.

3. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `am.AMRestAuthNSim`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	288 authNs/second	1640 authNs/second	3380 authNs/second
Peak response times	Mean: 86 ms 95th percentile: 185 ms	Mean: 30 ms 95th percentile: 55 ms	Mean: 59 ms 95th percentile: 177 ms

Metric	Small Cluster	Medium Cluster	Large Cluster
Resource requests	Memory: 4 Gi CPU: 2000 m	Memory: 8 Gi CPU: 11000 m	Memory: 20 Gi CPU: 10000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 12000 m	Memory: 26 Gi CPU: 1600 m
Peak pod usage	Memory: 3093 Mi CPU: 1890 m	Memory: 5904 Mi CPU: 8053 m	Memory: 11495 Mi CPU: 14900 m
Peak node usage	Memory: 7732 Mi CPU: 3213 m	Memory: 29111 Mi CPU: 11108 m	Memory: 50448 Mi CPU: 23312 m

5.4.1.2. REST Login - Stateless Authentication

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure AM for stateless 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. Enable the Use Client-based Sessions option.

   e. Click Save Changes.

3. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `am.AMRestAuthNSim`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

```
$ helm install --name benchmark helm/gatling-benchmark
```

2. Send the logs to the terminal window:

```
$ ./get-logs.sh
```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

```
$ helm del --purge benchmark
```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	292 authNs/second	1900 authNs/second	3424 authNs/second
Peak response times	Mean: 85 ms 95th percentile: 168 ms	Mean: 26 ms 95th percentile: 39 ms	Mean: 58 ms 95th percentile: 105 ms
Resource requests	Memory: 4 Gi CPU: 2000 m	Memory: 8 Gi CPU: 11000 m	Memory: 20 Gi CPU: 10000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 12000 m	Memory: 26 Gi CPU: 1600 m
Peak pod usage	Memory: 4401 Mi CPU: 1814 m	Memory: 5768 Mi CPU: 8637 m	Memory: 11495 Mi CPU: 14900 m
Peak node usage	Memory: 9336 Mi CPU: 3265 m	Memory: 28606 Mi CPU: 11982 m	Memory: 50448 Mi CPU: 23312 m

5.4.1.3. OAuth 2.0 - Authorization Code Grant Flow - Stateful Tokens

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.

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

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. Click 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. Click Save Changes.

4. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `am.AMAccessTokenSim`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	427 transactions/second	1751 transactions/second	3803 transactions/second

Metric	Small Cluster	Medium Cluster	Large Cluster
Peak response times	Mean: 58 ms 95th percentile: 180 ms	Mean: 28 ms 95th percentile: 64 ms	Mean: 52 ms 95th percentile: 142 ms
Resource requests	Memory: 4 Gi CPU: 2000 m	Memory: 8 Gi CPU: 11000 m	Memory: 20 Gi CPU: 10000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 12000 m	Memory: 26 Gi CPU: 1600 m
Peak pod usage	Memory: 4478 Mi CPU: 2588 m		Memory: 11495 Mi CPU: 14900 m
Peak node usage	Memory: 9440 Mi CPU: 3460 m		Memory: 50448 Mi CPU: 23312 m

5.4.1.4. OAuth 2.0 - Authorization Code Grant Flow - Stateless Tokens

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure AM for stateless 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. Enable the Use Client-based Sessions option.

   e. Click Save Changes.

3. Configure AM for stateless OAuth2 tokens:

   a. Select Services > OAuth2 Provider

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

   c. Click Save Changes.

4. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   • For `testname`, specify the value `am.AMAccessTokenSim`.

- For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
- For `concurrency`, specify the value `50`.
- For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	352 transactions/second	1544 transactions/second	3179 transactions/second
Peak response times	Mean: 70 ms 95th percentile: 192 ms	Mean: 32 ms 95th percentile: 61 ms	Mean: 62 ms 95th percentile: 148 ms
Resource requests	Memory: 4 Gi CPU: 2000 m	Memory: 8 Gi CPU: 11000 m	Memory: 20 Gi CPU: 10000 m
Resource limits	Memory: 5 Gi CPU: 3000 m	Memory: 11 Gi CPU: 12000 m	Memory: 26 Gi CPU: 1600 m
Peak pod usage	Memory: 4484 Mi CPU: 2796 m		Memory: 11495 Mi CPU: 14900 m
Peak node usage	Memory: 9495 Mi CPU: 3525 m		Memory: 50448 Mi CPU: 23312 m

5.4.2. Identity Management Benchmark Tests

The IDM tests examine the create, read, update, and delete (CRUD) performance through the IDM API. The Cloud Deployment Team generated CRUD load using the following simulations:

- `IDMCreateManagedUsers.scala`
- `IDMReadManagedUsers.scala`
- `IDMUpdateManagedUsers.scala`
- `IDMDeleteManagedUsers.scala`

You can specify a duration for the IDMCreateManagedUsers.scala simulation. The other simulations run for the required duration to complete read, update and delete the users.

5.4.2.1. Create

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `idm.IDMCreateManagedUsers`.
   - For `users`, specify the value `1000000` for a small cluster, or `10000000` for a medium cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster
Number of users	1,000,000	10,000,000
Peak throughput	77 creates/second	430 creates/second
Peak response times	Mean: 193 ms 95th percentile: 266 ms	Mean: 69 ms
Resource requests	Memory: 2 Gi CPU: 1500 m	Memory: 4 Gi CPU: 2000 m
Resource limits	Memory: 3 Gi CPU: 3000 m	Memory: 6 Gi CPU: 10000 m
Peak pod usage	Memory: 1723 Mi CPU: 2021 m	Memory: 3445 Mi CPU: 9950 m
Peak node usage	Memory: 10280 Mi CPU: 2159 m	

5.4.2.2. Read

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

Before running the benchmark test

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `idm.IDMReadManagedUsers`.
   - For `users`, specify the value `1000000` for a small cluster, or `10000000` for a medium cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

```
$ helm del --purge benchmark
```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster
Number of users	1,000,000	10,000,000
Peak throughput	217 reads/second	997 reads/second
Peak response times	Mean: 68 ms 95th percentile: 104 ms	Mean: 30 ms
Resource requests	Memory: 2 Gi CPU: 1500 m	Memory: 4 Gi CPU: 2000 m
Resource limits	Memory: 3 Gi CPU: 3000 m	Memory: 6 Gi CPU: 10000 m
Peak pod usage	Memory: 1728 Mi CPU: 2009 m	
Peak node usage	Memory: 10337 Mi CPU: 1678 m	

5.4.2.3. Update

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `idm.IDMUpdateManagedUsers`.
   - For `users`, specify the value `1000000` for a small cluster, or `10000000` for a medium cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster
Number of users	1,000,000	10,000,000
Peak throughput	78 updates/second	378 updates/second
Peak response times	Mean: 190 ms 95th percentile: 259 ms	Mean: 79 ms
Resource requests	Memory: 2 Gi CPU: 1500 m	Memory: 4 Gi CPU: 2000 m
Resource limits	Memory: 3 Gi CPU: 3000 m	Memory: 6 Gi CPU: 10000 m
Peak pod usage	Memory: 2031 Mi CPU: 2013 m	
Peak node usage	Memory: 11039 Mi CPU: 2754 m	

5.4.2.4. Delete

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

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `idm.IDMDeleteManagedUsers`.

- For `users`, specify the value `1000000` for a small cluster, or `10000000` for a medium cluster.
- For `concurrency`, specify the value `50`.
- For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster
Number of users	1,000,000	10,000,000
Peak throughput	149 deletes/second	810 deletes/second
Peak response times	Mean: 99 ms 95th percentile: 134 ms	Mean: 37 ms
Resource requests	Memory: 2 Gi CPU: 1500 m	Memory: 4 Gi CPU: 2000 m
Resource limits	Memory: 3 Gi CPU: 3000 m	Memory: 6 Gi CPU: 10000 m
Peak pod usage	Memory: 2497 Mi CPU: 2020 m	
Peak node usage	Memory: 11061 Mi CPU: 2120 m	

5.4.3. Identity Gateway Benchmark Tests

The IG tests measure throughput and response time of IG acting as a resource server and as a reverse proxy server. The Cloud Deployment Team generated load using the following simulations:

- Simulations that test IG acting as a resource server:
  - The `IGAccessTokensSim.scala` simulation, which tests a resource server that caches OAuth 2.0 tokens. With caching enabled, verification by an AM server is triggered after the first request only. Subsequent requests from the same user don't require verification.
  - The `IGAccessTokensNoCacheSim.scala` simulation, which tests a resource server with no cache. With no cache, each request must be verified by an AM server.

  Both simulations require the availability of OAuth 2.0 tokens. Prior to running either simulation, you run the `IGGenerateTokensSim.scala` simulation to generate the tokens.
- The `IGReverseProxyWebSim.scala` simulation, which tests IG acting as a reverse proxy server in front of an NGINX web server. The NGINX web server returns a static web page.

5.4.3.1. Resource Server - Cached

This benchmark test measures throughput and response time of an IG resource server that caches tokens.

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".
2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:
   - For `testname`, specify the value `"ig.IGGenerateTokensSim ig.IGAccessTokensSim"`.
   - For `oauth2_client`, specify the value `client-application`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:
   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```
2. Send the logs to the terminal window:
   ```
   $ ./get-logs.sh
   ```
3. (Optional) View the results in the Gatling UI.

See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

```
$ helm del --purge benchmark
```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	2428 transactions/second	4441 transactions/second	10863 transactions/second
Peak response times	Mean: 10 ms 95th percentile: 36 ms	Mean: 11 ms 95th percentile: 33 ms	Mean: 9 ms 95th percentile: 25 ms
Resource requests	Memory: 2 Gi CPU: 1000 m	Memory: 2 Gi CPU: 2000 m	Memory: 2 Gi CPU: 2000 m
Resource limits	Memory: 1 Gi CPU: 1000 m	Memory: 4 Gi CPU: 4000 m	Memory: 4 Gi CPU: 4000 m
Peak pod usage	Memory: 1200 Mi CPU: 86 m	Memory: 2281 Mi CPU: 1921 m	Memory: 1524 Mi CPU: 2431 m
Peak node usage			

5.4.3.2. Resource Server - No Cache

This benchmark test measures throughput and response time of an IG resource server with no cache. With no cache, all requests go directly to AM.

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `"ig.IGGenerateTokensSim ig.IGAccessTokensNoCacheSim"`.
   - For `oauth2_client`, specify the value `client-application`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.

- For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	1030 transactions/second	3225/second	10571/second
Peak response times	Mean: 24 ms 95th percentile: 39 ms	Mean: 15 ms 95th percentile: 36 ms	Mean: 9 ms 95th percentile: 25 ms
Resource requests	Memory: 2 Gi CPU: 1000 m	Memory: 2 Gi CPU: 2000 m	Memory: 2 Gi CPU: 2000 m
Resource limits	Memory: 1 Gi CPU: 1000 m	Memory: 4 Gi CPU: 4000 m	Memory: 4 Gi CPU: 4000 m
Peak pod usage	Memory: 955 Mi CPU: 585 m	Memory: 1778 Mi CPU: 3067 m	Memory: 1536 Mi CPU: 2232 m
Peak node usage			

5.4.3.3. Reverse Proxy

This benchmark test measures throughput and response time of an IG server as a reverse proxy in front of an nginx web server. The nginx web server returns a static web page.

Before running the benchmark test:

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

   See "To Provision the Userstores with Test Users".

2. Configure the `/path/to/forgeops/helm/gatling-benchmark/values.yaml` file:

   - For `testname`, specify the value `ig.IGReverseProxyWebSim`.
   - For `users`, specify the value `1000000` for a small cluster, `10000000` for a medium cluster, or `100000000` for a large cluster.
   - For `concurrency`, specify the value `50`.
   - For `duration`, specify the value `600`.

To run the benchmark test:

1. Install the `gatling-benchmark` Helm chart:

   ```
   $ helm install --name benchmark helm/gatling-benchmark
   ```

2. Send the logs to the terminal window:

   ```
   $ ./get-logs.sh
   ```

3. (Optional) View the results in the Gatling UI.

   See "To View Test Results in a Browser".

4. (Optional) Stop the benchmark test:

   ```
   $ helm del --purge benchmark
   ```

See also "Running AM, IDM, or IG Benchmark Tests".

You can compare your results with the Cloud Deployment Team results here:

Metric	Small Cluster	Medium Cluster	Large Cluster
Number of users	1,000,000	10,000,000	100,000,000
Peak throughput	3739 transactions/second	6915 transactions/second	12210 transactions/second
Peak response times	Mean: 7 ms 95th percentile: 51 ms	Mean: 7 ms 95th percentile: 24 ms	Mean: 8 ms 95th percentile: 23 ms
Resource requests	Memory: 2 Gi CPU: 1000 m	Memory: 2 Gi CPU: 2000 m	Memory: 2 Gi CPU: 2000 m
Resource limits	Memory: 1 Gi CPU: 1000 m	Memory: 4 Gi CPU: 4000 m	Memory: 4 Gi CPU: 4000 m
Peak pod usage	Memory: 954 Mi CPU: 997 m	Memory: 1741 Mi CPU: 2850 m	Memory: 1545 Mi CPU: 2780 m

Chapter 6 Taking the Next Steps

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

- The Kubernetes cluster and pods are up and running.
- DS, AM, IDM, and IG are installed and running. You can access each ForgeRock component.
- DS is provisioned with users. Replication and failover work as expected.
- Monitoring tools are installed and running. You can access a monitoring console for DS, AM, IDM, and IG.
- 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 completed your initial deployment phase. Congratulations!

Now you're ready to engineer your ForgeRock Identity Platform site reliability. See the ForgeRock Site Reliability Guides for information about:

- Modifying the CDM configuration to suit your business needs.
- Monitoring site health and performance.
- Backing up configuration and user data for disaster preparedness.
- Securing your site.

The CDM and its documentation continue to evolve. As they evolve, the *ForgeRock Site Reliability Guides* will become your resources for information about CDM continuous delivery and continuous integration.

Appendix A. Getting Support

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

A.1. ForgeRock DevOps Support

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

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

A.1.1. Commercial Support

ForgeRock provides commercial support for the following DevOps resources:

- Dockerfiles and Helm charts in the `forgeops` Git repository
- ForgeRock DevOps guides.

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

- *ForgeRock Access Management Release Notes*
- *ForgeRock Identity Management Release Notes*

- *ForgeRock Directory Services Release Notes*
- *ForgeRock Identity Message Broker Release Notes*
- *ForgeRock Identity Gateway Release Notes*

A.1.2. Support Limitations

ForgeRock provides no commercial support for the following:

- Artifacts other than Dockerfiles or Helm charts in the `forgeops` and `forgeops-init` repositories. Examples include scripts, example configurations, and so forth.
- Non-ForgeRock infrastructure. Examples include Docker, Kubernetes, Google Cloud Platform, Amazon Web Services, and so forth.
- Non-ForgeRock software. Examples include Java, Apache Tomcat, NGINX, Apache HTTP Server, and so forth.
- Production deployments that use the DevOps evaluation-only Docker images. When deploying the ForgeRock Identity Platform using Docker images, you must build and use your own images for production deployments. For information about how to build Docker images for the ForgeRock Identity Platform, see "*Building and Pushing Docker Images*" in the *DevOps Developer's Guide*.

A.1.3. Third-Party Kubernetes Services

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

Red Hat OpenShift is a tested and supported platform using Kubernetes for deployment. ForgeRock uses OpenShift tools such as Minishift, as well as other representative environments such as Amazon AWS for the testing. We do not test using bare metal due to the many customer permutations of deployment and configuration that may exist, and therefore cannot guarantee that we have tested in the same way a customer chooses to deploy. We will make commercially reasonable efforts to provide first-line support for any reported issue. In the case we are unable to reproduce a reported issue internally, we will request the customer engage OpenShift support to collaborate on problem identification and remediation. Customers deploying on OpenShift are expected to have a support contract in place with IBM/Red Hat that ensures support resources can be engaged if this situation may occur.

A.2. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

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

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

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

A.3. How to Report Problems or Provide Feedback

If you are a named customer Support Contact, contact ForgeRock using the Customer Support Portal to request information or report a problem with Dockerfiles or Helm charts in the DevOps Examples or the CDM.

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

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

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

  If the problem occurs on a Kubernetes system other than Minikube, GKE, EKS, OpenShift, or AKS, we might ask you to reproduce the problem on one of those.
- HTML output from the **debug-logs.sh** script. For more information, see "Running the debug-logs.sh Script" in the *DevOps Developer's Guide*.
- Description of the environment, including the following information:
  - Environment type: Minikube, GKE, EKS, AKS, or OpenShift.
  - Software versions of supporting components:
    - Oracle VirtualBox (Minikube environments only).
    - Docker client (all environments).
    - Minikube (all environments).
    - **kubectl** command (all environments).
    - Kubernetes Helm (all environments).
    - Google Cloud SDK (GKE environments only).
    - Amazon AWS Command Line Interface (EKS environments only).

  - Azure Command Line Interface (AKS environments only).
- `forgeops` repository branch.
- Any patches or other software that might be affecting the problem.

A.4. Getting Support and Contacting ForgeRock

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

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

Glossary

affinity (AM)	AM affinity based load balancing ensures that the CTS token creation load is spread over multiple server instances (the token origin servers). Once a CTS token is created and assigned to a session, all subsequent token operations are sent to the same token origin server from any AM node. This ensures that the load of CTS token management is spread across directory servers. 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`.

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 Helm charts, Docker images, and other artifacts expressly to build the CDM.
CloudFormation (AWS)	CloudFormation is a service that helps you model and set up your Amazon Web Services (AWS) resources. You create a template that describes all the AWS resources that you want. AWS CloudFormation takes care of provisioning and configuring those resources for you. 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.
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.
replication controller	A replication controller ensures that a specified number of Kubernetes pod replicas are running at any one time. The `replication controller` ensures that a pod or a homogeneous set of pods is always up and available. Source: *ReplicationController* in the Kubernetes Concepts documentation.
secret (Kubernetes)	A Kubernetes secret is a secure object that stores sensitive data, such as passwords, OAuth 2.0 tokens, and SSH keys in your clusters. 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.
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.
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.