Site Reliability Guide for Amazon EKS

/ 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

A companion guide to the *ForgeRock Cloud Deployment Model Cookbook for Amazon EKS*. Provides guidance for customizing the Cloud Deployment Model (CDM) in a production deployment on Amazon EKS.

Table of Contents

Preface

The *Site Reliability Guide for Amazon EKS* provides instructions to help you keep ForgeRock Identity Platform™ running smoothly on EKS. The guide contains information about monitoring, securing, backing up, and restoring the CDM.

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

Engineering ForgeRock Site Reliability

In any ForgeRock Identity Platform cloud deployment, the overall site reliability goal is the same:

> Deliver the optimal identity and access management (IAM) service level required to meet the current IAM demand.

Use this guide to customize and scale your ForgeRock cloud deployment to meet your specific IAM service requirements. This guide also provides guidance on monitoring and tuning your ForgeRock site to meet real-world demands. The information and examples in this guide are especially relatable if you deploy the ForgeRock Cloud Deployment Model (CDM) to get started.

1.1. Deploying the ForgeRock Cloud Deployment Model

This *ForgeRock Site Reliability Guide* supplements the *ForgeRock Cloud Deployment Model (CDM) Cookbook.* The cookbook provides steps for a simplified and repeatable deployment process:

1. To get started, see the cookbook for your cloud platform:
   - *ForgeRock Cloud Deployment Model Cookbook for GKE.*
   - *ForgeRock Cloud Deployment Model Cookbook for Amazon EKS.*
2. After you've deployed the CDM, you'll be able to validate the base deployment against these criteria:
   - The Kubernetes cluster and pods are up and running.
   - DS, AM, IDM, and IG are installed and running. You can access each ForgeRock component.
   - DS is provisioned with 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.
   - The benchmarking tests in the cookbook run without errors.
   - Your benchmarking test results are similar to the benchmarks reported in this cookbook.
3. After you've validated your CDM deployment, use this *Site Reliability Guide* to implement a production-quality deployment that meets your specific IAM requirements.

1.2. Using This Guide

ForgeRock Identity Platform deployment work is never really finished. It's likely you'll want to customize the CDM base configurations to meet your service level requirements. Even after you've modified the configurations, you'll need to monitor the system for changes in real-world performance and availability. You should also periodically review changes to your IAM needs. Understanding that your ForgeRock site reliability can change rapidly, you'll always be prepared to make incremental changes to the system.

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

Chapter 2

Monitoring Your Deployment

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

2.1. About CDM Monitoring

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

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

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

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

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

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

To enable monitoring in the CDM, see

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

To access CDM monitoring dashboards, see:

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

Note

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

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

2.2. Importing Custom Grafana Dashboards

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

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

2.3. Modifying the Prometheus Operator Configuration

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

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

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

2.4. Configuring Additional Alerts

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

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

Chapter 3

Backing Up and Restoring Directory Data

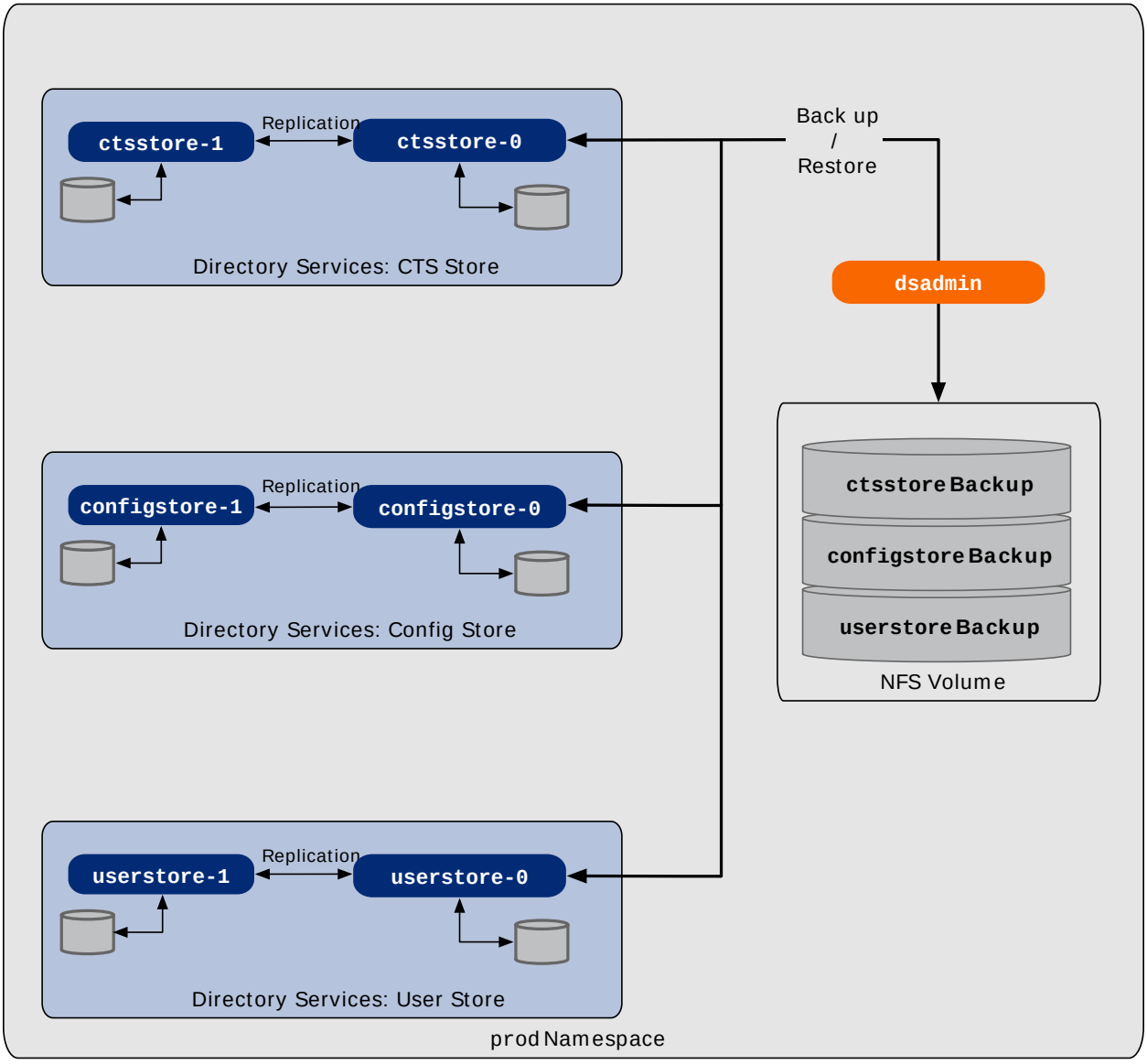
This chapter describes the backup and restore functionality in CDM. It explains how to customize backup and restore operations in your deployment.

This chapter includes the following sections:

3.1. About Backup and Restore

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

Backup and Restore Overview



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

- Three DS instances are deployed using Kubernetes stateful sets. Each stateful set is associated with a dedicated, persistent disk volume with a persistent volume claim (PVC). Each DS instance is a combined directory and replication server.
- A shared-access file storage service, such as Google Cloud Filestore or Amazon EFS, is used to hold backups. The backup volume is mounted as a shared NFS volume using a PVC, and is used for storing backups of directory data. The mount point for the backup volume is the `/opt/opendj/bak` directory in each DS instance.
- The `dsadmin` pod stores and retrieves the DS backups.
- The scripts necessary for performing backup and restore operations are available in the `scripts` folder of each directory server pod.
- You can use one of the DS pods to schedule backups using the **kubectl exec** command. See "Performing User-Initiated Backups" for more information on how to enable and schedule backups.
- The backup schedule is configured based on the load and on a purge delay of 8 hours for DS instances. The default backup schedule is configured for the following:
  - An incremental backup is performed every hour.
  - A full backup is performed once a day.
- You can customize the schedule of backups based on your volume of activity and recovery objectives. To customize the backup schedule see "Customizing the Backup Schedule".
- When you perform a restore operation on a CDM instance, the latest backup available in the `/opt/opendj/bak` is used for restoring the CDM instance.
- Optionally, you can archive backup to cloud storage.

3.2. Enabling CDM-Scheduled Backups

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

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

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

- For the `userstore` instance:

```
$ kubectl exec userstore-0 -it schedule-backup.sh
```

- For the `ctsstore` instance:

```
$ kubectl exec ctsstore-0 -it schedule-backup.sh
```

- For the `configstore` instance:

```
$ kubectl exec configstore-0 -it schedule-backup.sh
```

3.3. Customizing the Backup Schedule

You can alter the backup schedule using the following procedure:

To Customize the Backup Schedule for a DS Instance

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

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

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

   - `FULL_CRON="0 0 * * *"`
   - `INCREMENTAL_CRON="0 * * * *"`

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

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

3. Log out of the DS instance and run the **kubectl exec *DS* -it schedule-backup.sh** command. For example:

   ```
   $ kubectl exec userstore-0 -it schedule-backup.sh
   ```

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

3.4. Performing User-Initiated Backups

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

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

To Initiate a Full Backup

1. To make a full backup of user store directory data on demand, use the following **kubectl exec** command:

```
$ kubectl exec userstore-0 -it -- scripts/backup.sh --start 0
Starting backup to /opt/opendj/bak/default/userstore-prod
Backup task 20190801191506671 scheduled to start immediately
. . .
[01/Aug/2019:19:15:06 +0000] severity="NOTICE" msgCount=56 msgID=org.opends.messages.tool-283
 message="The backup process completed successfully"
[01/Aug/2019:19:15:06 +0000] severity="NOTICE" msgCount=57 msgID=org.opends.messages.backend-414
 message="Backup task 20181127191506671 finished execution in the state Completed successfully"
Backup task 20181127191506671 has been successfully completed
```

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

```
$ kubectl exec userstore-0 -it -- scripts/list-backup.sh
Listing backups in /opt/opendj/bak/default/userstore-prod
adminRoot backups
Backup ID:          20190801191506Z
Backup Date:        01/Aug/2019:19:15:06 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none

ads-truststore backups
Backup ID:          2019080171915O6Z
Backup Date:        01/Aug/2019:19:15:06 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none

amIdentityStore backups
Backup ID:          20190801191506Z
Backup Date:        01/Aug/2019:19:15:06 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none
...
tasks backups
Backup ID:          20190801191506Z
Backup Date:        01/Aug/2019:19:15:06 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
```

```
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none
```

3.5. Exporting User Data

Exporting directory data to LDIF is very useful to:

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

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

To Export User Data

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

```
$ kubectl exec userstore-0 -it -- scripts/export-ldif.sh
Exporting LDIF
Exporting ldif of adminRoot to /opt/opendj/bak/default/userstore-prod/adminRoot-080119262019.04.ldif
Export task 20190801192605510 scheduled to start immediately
[01/Aug/2019:19:26:05 +0000] severity="NOTICE" msgCount=0 msgID=org.opends.messages.backend-413
 message="Export task 20190801192605510 started execution"
[01/Aug/2019:19:26:05 +0000] severity="INFORMATION" msgCount=1 msgID=org.opends.messages.tool-1662
 message="Exporting to /opt/opendj/bak/default/userstore-prod/adminRoot-112719262018.04.ldif"
[01/Aug/2019:19:26:05 +0000] severity="NOTICE" msgCount=2 msgID=org.opends.messages.backend-414
 message="Export task 20190801192605510 finished execution in the state Completed successfully"
Export task 20190801192605510 has been successfully completed
```

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

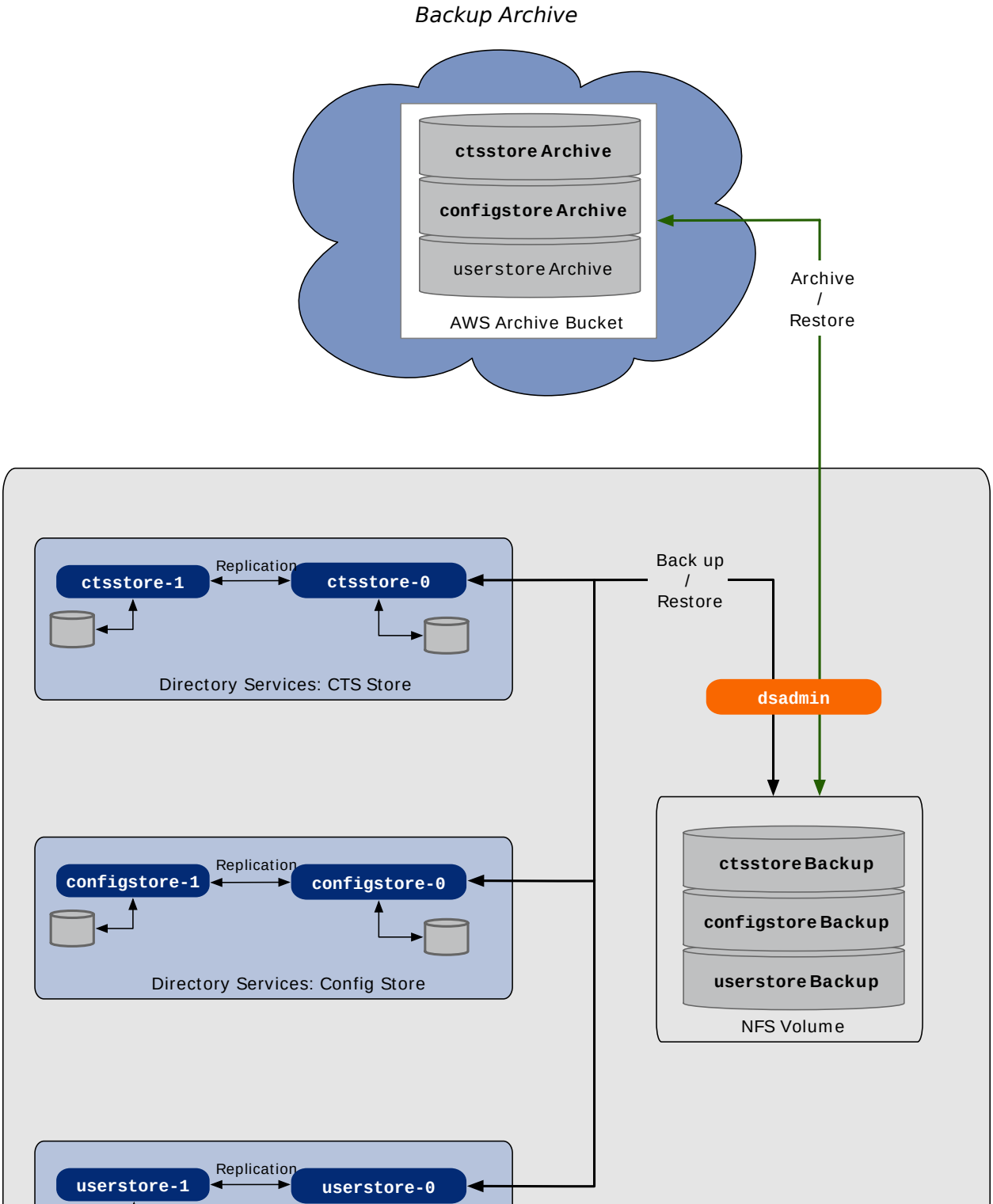
```
$ kubectl exec userstore-0 -it -- ls -l ./bak/default/userstore-prod/
     total 48
      drwxr-xr-x 2 forgerock forgerock 4096 Aug  1 19:15 adminRoot
      -rw------- 1 forgerock forgerock 2802 Aug  1 19:26 adminRoot-080119262019.04.ldif
      drwxr-xr-x 2 forgerock forgerock 4096 Aug  1 19:15 ads-truststore
      ...
```

3.6. Archiving the CDM Backup

To avoid losing backup files when a disaster occurs, archive the backup files to an Amazon S3 file store.

First, configure an Amazon S3 file store in your deployment. Then, edit the `dsadmin.yaml` file to enable archiving backup files to your Amazon S3 file store.

The following diagram shows how backup files are archived to Amazon S3 Filestore:


Backup Archive
ctsstore Archive
configstore Archive
userstore Archive
AWS Archive Bucket
Archive
/
Restore
ctsstore-1
Replication
ctsstore-0
Directory Services: CTS Store
Back up
/
Restore
dsadmin
configstore-1
Replication
configstore-0
Directory Services: Config Store
ctsstore Backup
configstore Backup
userstore Backup
NFS Volume
userstore-1
Replication
userstore-0

To Enable Archiving Backup Files

The following procedure enables archiving all the backup files that have been made for the `userstore`, `ctsstore`, and `configstore` instances:

1. Set the following values in the YAML file that you use to install the `dsadmin` Helm chart:

   - `s3.enabled` to `true`
   - `s3.bucket` to the URL for your storage bucket
   - `createPVC` to `false`

   Here is a sample `dsadmin.yaml` file snippet in which archive is enabled, and the backup is archived to an Amazon S3 bucket at the location *s3://my-br-bucket*:

   ```
   ...
   s3:
     enabled: true
     bucket:  s3://my-br-bucket
   ...
   createPVC: false
   ...
   ```

2. Delete the `prod-dsadmin` Helm release.

   ```
   $ helm delete --purge prod-dsadmin
   These resources were kept due to the resource policy:
   [PersistentVolumeClaim] ds-backup
   [PersistentVolume] ds-backup

   release "prod-dsadmin" deleted
   ```

3. Reinstall the `dsadmin` Helm chart:

```
$ helm install --name prod-dsadmin --namespace prod \
 --values common.yaml --values dsadmin.yaml /path/to/forgeops/helm/dsadmin
NAME:   prod-dsadmin
LAST DEPLOYED: Fri Aug  2 11:13:10 2019
NAMESPACE: prod
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/CronJob
NAME      AGE
gcs-sync  1s

==> v1/Pod(related)

NAME                       READY  STATUS    RESTARTS  AGE
dsadmin-784fd647b4-lcgjg   0/1    Init:0/1  0         1s

==> v1/Deployment

NAME     AGE
dsadmin  1s
```

You can view the list of backups archived to your Amazon S3 bucket by using the AWS UI.

For more information about the backup archive process, see the README.md file for the `dsadmin` Helm chart.

3.7. Using CDM Restore

You can enable the restore capability in the CDM, and redeploy DS pods, to restore data from the latest available backup. This is useful when a system disaster occurs, or when directory services are lost. This is also useful when you want to port test environment data to a production deployment. Note that when you redeploy DS data, the entire DS environment is restored. To restore specific portions of the directory data without having to redeploy the DS environment, see "User-Initiated Restore".

To Redeploy DS From a Backup

Use these steps to redeploy the user store from a backup:

1. If either the `userstore-0` or `userstore-1` pod is running, list the backups available. For example, if the `userstore-0` pod is running, execute the following command:

```
$ kubectl exec userstore-0 -it -- scripts/list-backup.sh
Listing backups in /opt/opendj/bak/default/userstore-prod
adminRoot backups
Backup ID:          20190801202519Z
Backup Date:        01/Aug/2019:20:25:27 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none

ads-truststore backups
Backup ID:          20190801202519Z
Backup Date:        01/Aug/2019:20:25:27 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none
...
tasks backups
Backup ID:          20190801202519Z
Backup Date:        01/Aug/2019:20:25:27 +0000
Is Incremental:     false
Is Compressed:      true
Is Encrypted:       false
Has Unsigned Hash:  false
Has Signed Hash:    false
Dependent Upon:     none
```

2. Delete the `prod-userstore` Helm release:

```
$ helm delete --purge prod-userstore
release "prod-userstore" deleted
```

3. List the PVCs in your namespace using the **kubectl get pvc** command:

```
$ kubectl get pvc
NAME               STATUS   VOLUME                                     CAPACITY   ACCESS MODES
 STORAGECLASS   AGE
db-configstore-0   Bound    pvc-2a1caac1-f3c8-11e8-8de8-42010a8e005c   10Gi       RWO
 standard       9h
db-configstore-1   Bound    pvc-5f517dab-f3c8-11e8-8de8-42010a8e005c   10Gi       RWO
 standard       9h
db-ctsstore-0      Bound    pvc-65f4af80-f3c8-11e8-8de8-42010a8e005c   100Gi      RWO            fast
          9h
db-ctsstore-1      Bound    pvc-82c993f2-f3c8-11e8-8de8-42010a8e005c   100Gi      RWO            fast
          9h
db-userstore-0     Bound    pvc-478ef730-f3c8-11e8-8de8-42010a8e005c   100Gi      RWO            fast
          9h
db-userstore-1     Bound    pvc-6dc544a6-f3c8-11e8-8de8-42010a8e005c   100Gi      RWO            fast
          9h
ds-backup          Bound    ds-backup                                  1Ti        RWX            nfs
          9h
```

4. Delete the PVCs for userstore using the **kubectl delete pvc** command:

```
$ kubectl delete pvc db-userstore-1
persistentvolumeclaim "db-userstore-1" deleted
$ kubectl delete pvc db-userstore-0
persistentvolumeclaim "db-userstore-0" deleted
```

Important

Do not delete the `ds-backup` PVC. If you delete the backup PVC, you cannot restore data.

5. Edit the `userstore.yaml` file, and set the `restore.enabled` key to `true`.

```
...
# Restore parameters.
restore:
 enabled: true
...
```

6. Redeploy the `userstore` directory. During deployment, the directory data is restored from the backup:

```
$ helm install --name prod-userstore --namespace prod  \
 --values common.yaml --values userstore.yaml /path/to/forgeops/helm/ds
NAME:   prod-userstore
LAST DEPLOYED: Thu Aug  1 13:04:00 2019
NAMESPACE: prod
STATUS: DEPLOYED

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

==> v1/ConfigMap
userstore  0s

==> v1/Service
userstore  0s

==> v1beta1/StatefulSet
userstore  0s

==> v1/Pod(related)

NAME         READY  STATUS    RESTARTS  AGE
userstore-0  0/1    Init:0/1  0         0s
```

7. Verify that the DS pods are all running:

```
$ kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
...
userstore-0                       1/1       Running   0          11m
userstore-1                       0/1       Running   0          4m
```

3.8. User-Initiated Restore

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

Consider following these best practices:

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

Chapter 4

Securing Your Deployment

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

4.1. Changing Default Secrets

The `forgeops` repository contains hardcoded passwords and keys. The Cloud Deployment Team used these default secrets when we deployed and benchmarked the CDM. If you use the CDM as a starting point for your production deployment, you must change the default secrets to secure your deployment.

This section contains the default values for each secret, and describes how to change each secret's value.

4.1.1. Changing Default DS Secrets

This section covers the DS secrets in the `forgeops` repository.

Directory Manager

The DS directory superuser in the CDM is named `cn=Directory Manager`.

The default value for the directory manager password is `password`.

To change the directory manager user's password, modify the value in the `/path/to/forgeops/helm/ds/secrets/dirmanager.pw` file before installing the `ds` Helm chart.

Monitor User Password

The DS monitor user is a special user named `uid=monitor`. This user has privileges to read monitoring data.

The default value for the monitor user's password is `password`.

To change the monitor user's password, modify the value in the `/path/to/forgeops/helm/ds/secrets/monitor.pw` file before installing the `ds` Helm chart.

For more information about the monitor user, see Directory Server Setup Parameters in the *DS Installation Guide*.

AM Profile Passwords

In the CDM, DS instances are installed with setup profiles. Each profile generates a password-protected administrative account:

Profile	DN of Administrative Account
`am-cts`	`uid=openam_cts,ou=admins,ou=famrecords,ou=openam-session,ou=tokens`
`am-identity-store`	`uid=am-identity-bind-account,ou=admins,ou=identities`
`am-config`	`uid=am-config,ou=admins,ou=am-config`

The default password for all of the administrative accounts is `password`.

To change the administrative account passwords on a running DS service, use the **ldappasswordmodify** utility or call the DS REST API to change the password. Note that immediately after DS starts, the default password will be in use for a short period.

For more information about DS setup profiles, see Using Directory Server Setup Profiles in the *DS Installation Guide*.

4.1.2. Changing Default AM and Amster Secrets

This section covers the AM and Amster secrets in the `forgeops` repository.

`amadmin` User's Password

The `amadmin` user is the top-level AM administrator.

The `amster` Helm chart generates a random value for the `amadmin` user's password when it installs AM. To obtain the generated password, see "To Access the AM Console" in the *DevOps Developer's Guide*.

To set the `amadmin` user's password to a specific value, set the `amadminPassword` value when installing the `amster` Helm chart. For example, **helm install --set amadminPassword=my-amadmin-password amster**.

To change the way the password is generated, modify the `amster` Helm chart configuration map, located at `/path/to/forgeops/helm/amster/templates/config-map.yaml`.

Password Encryption Key

The password encryption key encrypts passwords in the AM configuration repository.

The `amster` Helm chart generates a random value for the password encryption key when it installs AM.

To set the password encryption key to a specific value, set the `encryptionKey` value when installing the `amster` Helm chart. For example, **helm install --set encryptionKey=my-encryption-key amster**.

To change the way the password encryption key is generated, modify the Amster Helm chart configuration map, located at `/path/to/forgeops/helm/amster/templates/config-map.yaml`.

For more information about the AM password encryption key, see the description of the `--pwdEncKey` parameter of the **install-openam** command in the *Amster User Guide*.

AM Keystore

The AM keystore contains keys and certificates used for federation, token signatures, and other purposes. It also contains passwords required for AM to start.

The `forgeops` repository contains a default AM keystore at the path `/path/to/forgeops/helm/openam/secrets/keystore.jceks`.

To change keys and passwords in the default keystore, first modify the default keystore as necessary. Then replace the default keystore with the modified version.

For more information about the contents of the AM keystore, and how to modify it, see Setting Up Keys and Keystores in the *AM Setup and Maintenance Guide*.

Storepass

The AM keystore's password is called the *storepass*.

To obtain the default storepass value, run the command **cat /path/to/forgeops/helm/openam/secrets/.storepass**.

To change the storepass value, follow the instructions in To Change the Keystore Password in the *AM Setup and Maintenance Guide*. Then replace the values in the `/path/to/forgeops/helm/openam/secrets/.storepass` and `/path/to/forgeops/helm/openam/secrets/storepass` files with the new storepass.

Keypass

AM uses an unencrypted password to protect private keys in the AM keystore. This password is known as the *keypass*.

To obtain the keypass' default value, run the command **cat /path/to/forgeops/helm/openam/secrets/.keypass**.

To change the keypass value, follow the instructions in To Change Key Alias Passwords in the *AM Setup and Maintenance Guide*. Then replace the values in the `/path/to/forgeops/helm/openam/secrets/.keypass` and `/path/to/forgeops/helm/openam/secrets/keypass` files with the new keypass.

Amster Key

Amster uses a key pair to authenticate client connections. The `amster` pod needs the private key, and clients need the public key. In the CDM, the `openam` pod is an Amster client that therefore needs the Amster public key.

Default values for the key pair are available in the `forgeops` repository at the following locations. Note that the public key must be present in both the `amster` and openam Helm charts:

- Amster private key: `/path/to/forgeops/helm/amster/secrets/id_rsa`
- Amster public key: `/path/to/forgeops/helm/amster/secrets/authorized_keys`
- Amster public key: `/path/to/forgeops/helm/openam/secrets/authorized_keys`

To change the Amster key, follow the instructions in To Create and Configure a Private Key Pair in the *Amster User Guide*. Then replace the values in the `/path/to/forgeops/helm/amster/secrets/id_rsa`, `/path/to/forgeops/helm/amster/secrets/authorized_keys`, and `/path/to/forgeops/helm/openam/secrets/authorized_keys` files.

4.1.3. Changing Default IDM Secrets

This section covers the IDM secrets in the `forgeops` repository.

IDM Administrator Password

The password for the IDM administrator is set to the default password when IDM starts.

To change the IDM administrator's password on a running IDM service, patch the `openidm-admin` user object with the new password. Note that immediately after IDM starts, the default password will be in use for a short period.

For more information about patching the `openidm-admin` user object, see Replacing Default Security Settings in the *IDM Integrator's Guide*.

IDM Keystore

The IDM keystore contains keys used to encrypt configuration information, user passwords, and JWT tokens. It also contains keys used to sign JWT tokens.

The `forgeops` repository contains a default IDM keystore at the path `/path/to/forgeops/helm/openidm/secrets/keystore.jceks`.

To change keys in the default keystore, first modify the default keystore as necessary. Then replace the default keystore with the modified version.

For more information about identifying and modifying IDM keys, see the following sections in the *IDM Integrator's Guide*:

- Displaying the Contents of the Keystore
- Updating Encryption Keys

IDM Keystore Password

The default IDM keystore password is stored in the `forgeops` repository.

This password is stored as one of the IDM boot properties in the `openidm` Helm chart's configmap, at the path `/path/to/forgeops/helm/openidm/templates/configmap.yaml`.

To modify the default keystore password, follow the instructions in To Change the Default Keystore Password in the *IDM Integrator's Guide*. Be sure to replace the value of the `openidm.keystore.password` key with the new password in the `/path/to/forgeops/helm/openidm/templates/configmap.yaml` file.

4.2. Granting Access to Multiple Users

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

To Grant Users Access to the EKS Cluster

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

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

4. Under the `mapRoles` section, insert the `mapUser` section. An example is shown here with the following parameters:
   - The user ARN is `arn:aws:iam::012345678901:user/new.user`.
   - The user name registered in AWS is *new.user*.

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

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

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

6. Save the configuration map.

4.3. Controlling Access by Configuring a CIDR Block

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

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

For example:

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

4.4. Securing Communication With ForgeRock Identity Platform Servers

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

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

- **Over HTTPS using a self-signed certificate**. Communication is encrypted, but users will receive warnings about insecure communication from some browsers. For configuration steps, see "Configuring and Installing the frconfig Helm Chart" in the *DevOps Developer's Guide*.
- **Over HTTPS using a certificate with a trust chain that starts at a trusted root certificate**. Communication is encrypted, and users will not receive warnings from their browsers. For configuration steps, see "Configuring and Installing the frconfig Helm Chart" in the *DevOps Developer's Guide*.
- **Over HTTPS using a dynamically obtained certificate from Let's Encrypt**. Communication is encrypted and users will not receive warnings from their browsers. A `cert-manager` pod installed in

your Kubernetes cluster calls Let's Encrypt to obtain a certificate, and then automatically installs a Kubernetes secret. This option is covered in the next section.

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

4.4.1. Automating Certificate Management

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

For the steps the Cloud Deployment Team used to deploy the `cert-manager` add-on in the CDM, see "Deploying the Certificate Manager" in the *Cloud Deployment Model Cookbook for Amazon EKS*.

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

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

Appendix A. Getting Support

This appendix contains information about support options for the ForgeRock 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.