DevOps Guide

ForgeRock Identity Platform 5.0.0

David Goldsmith

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

Abstract

Guide to ForgeRock Identity Platform™ deployment using DevOps techniques.

Table of Contents

Preface v
1. Introducing DevOps for the ForgeRock Identity Platform 1
    1.1. Approaches to Deploying Software Releases 1
    1.2. Automating Deployments Using DevOps Practices 2
    1.3. Limitations 4
2. Implementing DevOps Environments 6
    2.1. Kubernetes Running on a Minikube Virtual Machine 6
    2.2. Kubernetes Running on Google Container Engine 12
3. Deploying the OpenAM and OpenDJ Example 20
    3.1. About the Example 20
    3.2. Working With the OpenAM and OpenDJ Example 22
    3.3. Preparing the Environment 23
    3.4. Creating Docker Images 26
    3.5. Orchestrating the Deployment 34
    3.6. Modifying and Saving the OpenAM Configuration 42
4. Deploying the OpenIDM Example 45
    4.1. About the Example 45
    4.2. Working With the OpenIDM Example 47
    4.3. Preparing the Environment 49
    4.4. Creating Docker Images 52
    4.5. Orchestrating the Deployment 59
    4.6. Modifying and Saving the OpenIDM Configuration 63
5. Deploying the OpenIG Example 64
    5.1. About the Example 64
    5.2. Working With the OpenIG Example 65
    5.3. Preparing the Environment 67
    5.4. Creating the Docker Image 69
    5.5. Orchestrating the Deployment 74
    5.6. Modifying and Saving the OpenIG Configuration 77
6. Troubleshooting DevOps Deployments 78
    6.1. Troubleshooting the Environment 79
    6.2. Troubleshooting Containerization 80
    6.3. Troubleshooting Orchestration 81
7. Reference 92
    7.1. Git Repositories Used by the DevOps Examples 92
    7.2. Naming Docker Images 93
    7.3. Using the build.sh Script to Create Docker Images 94
    7.4. Specifying Deployment Options in the custom.yaml File 95
A. Getting Support 97
    A.1. Accessing Documentation Online 97
    A.2. Joining the ForgeRock Community 98
    A.3. How to Report Problems or Provide Feedback 98
    A.4. Getting Support and Contacting ForgeRock 99
B. Change Log 100

B.1. Deprecated Features 100
B.2. Bug Fixes 101
B.3. Documentation Updates 101

Preface

This guide covers installation, configuration, and deployment of the ForgeRock Identity Platform using DevOps techniques.

This guide provides a general introduction to DevOps deployment of ForgeRock® software and an overview of DevOps deployment strategies. It also includes several deployment examples that illustrate best practices to help you get started with your own DevOps deployments.

About ForgeRock Identity Platform Software

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

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

Changes Since the Original Release

See the Appendix B, "*Change Log*" for descriptions of bug fixes and documentation changes made since ForgeRock DevOps Examples 5.0.0 was originally released.

Getting Started With DevOps Deployments

Use this guide to help you get started with DevOps deployments of the ForgeRock Identity Platform as follows:

1. Familiarize yourself with the overview of DevOps concepts by reading Chapter 1, "*Introducing DevOps for the ForgeRock Identity Platform*".

2. Determine which environment(s) you intend to use for DevOps deployments, and then implement the environment(s). Follow the instructions in Chapter 2, "*Implementing DevOps Environments*".

3. Deploy one or more of the examples described in this guide. Each example has its own chapter.

   While deploying the example, refer to the material in Chapter 7, "*Reference*", which might be useful.

The deployment environments supported for the ForgeRock Identity Platform DevOps Examples require you to install software products that are *not* part of the ForgeRock Identity Platform. We strongly recommend that you become familiar with basic concepts for the following software before attempting to use it even in your initial experiments with DevOps deployments:

Table 1. DevOps Environments Prerequisite Software

Software	Recommended Level of Familiarity	Links to Introductory Material
Oracle VirtualBox	Install, start, and stop VirtualBox software; understand virtual machine settings; create snapshots	First Steps chapter in the VirtualBox documentation
Docker Client	Build, list, and remove images; understand the Docker client-server architecture; understand Docker registry concepts	Get Started With Docker tutorial
Kubernetes	Identify Kubernetes entities such as pods and clusters; understand the Kubernetes client-server architecture	Kubernetes tutorials Scalable Microservices with Kubernetes on Udacity The Illustrated Children's Guide to Kubernetes
Minikube	Understand what Minikube is; create and start a Minikube virtual machine; run **docker** and **kubectl** commands that access the Docker Engine and Kubernetes cluster running in the Minikube virtual machine	Running Kubernetes Locally via Minikube Hello Minikube tutorial
kubectl (Kubernetes client)	Run **kubectl** commands on a Kubernetes cluster	**kubectl** command overview
Kubernetes Helm	Understand what a Helm chart is; understand the Helm client-server architecture; run the **helm** command to install, list, and delete Helm charts in a Kubernetes cluster	Blog entry describing Helm charts
Google Container Engine (GKE)	Create a Google Cloud Platform account and project, and make GKE available in the project	GKE Quickstart
Google Cloud SDK	Run the **gcloud** command to access GKE components in a Google Cloud Platform project	Google Cloud SDK documentation

Chapter 1

Introducing DevOps for the ForgeRock Identity Platform

You can deploy the ForgeRock Identity Platform using DevOps practices.

This chapter introduces concepts that are relevant to DevOps deployments of the ForgeRock Identity Platform:

- Traditional and cloud automation deployment. See Section 1.1, "Approaches to Deploying Software Releases".
- Containers. See Section 1.2.1, "Introducing Containerization".
- Orchestration. See Section 1.2.2, "Introducing Container Orchestration".

1.1. Approaches to Deploying Software Releases

This section explores two approaches to software deployment: *traditional deployment* and *deployment using DevOps practices*.

Traditional deployment of software systems has the following characteristics:

- Failover and scalability are achievable, but systems are often brittle and require significant design and testing when implementing failover or when scaling deployments up and down.
- After deployment, it is common practice to keep a software release static for months, or even years, without changing its configuration because of the complexity of deploying a new release.
- Changes to software configuration require extensive testing and validation before deployment of a new service release.

DevOps practices apply the principle of encapsulation to software deployment by using techniques such as virtualization, continuous integration, and automated deployment. DevOps practices are especially suitable for elastic *cloud automation deployment*, in which the number of servers on which software is deployed varies depending on system demand.

An analogy that has helped many people understand the rationale for using DevOps practices is *pets vs. cattle*. [1] You might think of servers in traditional deployments as pets. You likely know the server

[1] The first known usage of this analogy was by Glenn Berry in his presentation, Scaling SQL Software , when describing the difference between scaling up and scaling out.

by name, for example, `ldap.mycompany.com`. If the server fails, it might need to be "nursed" to be brought back to life. If the server runs out of capacity, it might not be easy to replace it with a bigger server, or with an additional server, because changing a single server can affect the behavior of the whole deployment.

Servers in DevOps deployments are more like cattle. Individual servers are more likely to be numbered than named. If a server goes down, it is simply removed from the deployment, and the functionality that it used to perform is then performed by other cattle in the "herd." If more servers are needed to achieve a higher level of performance than was initially anticipated when your software release was rolled out, they can be easily added to the deployment. Servers can be easily added to and removed from the deployment at any time to accommodate spikes in usage.

The ForgeRock DevOps Examples are available with ForgeRock Identity Platform 5.0.0. These examples provide reference implementations that you can use to deploy the ForgeRock Identity Platform using DevOps practices.

1.2. Automating Deployments Using DevOps Practices

The ForgeRock DevOps Examples implement two DevOps practices: containerization and orchestration. This section provides a conceptual introduction to these two practices and introduces you to the DevOps implementations supported by the the DevOps Examples.

1.2.1. Introducing Containerization

Containerization is a technique for virtualizing software applications. Containerization differs from operating system-level virtualization in that one or more containers run on an existing operating system.

There are multiple implementations of containerization, including chroot jails, FreeBSD jails, Solaris containers, rkt app container images, and Docker containers.

The ForgeRock DevOps Examples support Docker for containerization, taking advantage of the following capabilities:

- **File-Based Representation of Containers**. Docker *images* contain a file system and run-time configuration information. Docker *containers* are running instances of Docker images.
- **Modularization**. Docker images are based on other Docker images. For example, an OpenAM image is based on a Tomcat image that is itself based on an OpenJDK JRE image. In this example, the OpenAM container has OpenAM software, Tomcat software, and the OpenJDK JRE.
- **Collaboration**. Public and private Docker registries let users collaborate by providing cloud-based access to Docker images. Continuing with the example, the public Docker registry at https://hub.docker.com/ has Docker images for Tomcat and the OpenJDK JRE that any user can download. You build Docker images for the ForgeRock Identity Platform based on the Tomcat and OpenJDK JRE images in the public Docker registry. You can then push the Docker images to a private Docker registry that other users in your organization can access.

The ForgeRock DevOps Examples include scripts and descriptor files, such as Dockerfiles, that you can use to build reference Docker images for the ForgeRock Identity Platform. These files are available for download with a ForgeRock BackStage account.

To obtain these files, clone the Git repository at https://stash.forgerock.org/projects/DOCKER/repos/docker.

You can either build the reference Docker images or create customized images based on the reference images, and then upload the images to your own Docker registry.

1.2.2. Introducing Container Orchestration

After software containers have been created, they can be deployed for use. The term *software orchestration* refers to the deployment and management of software systems. *Orchestration frameworks*, which enable automated, repeatable, managed deployments, are commonly associated with DevOps practices. *Container orchestration frameworks* are orchestration frameworks that deploy and manage container-based software.

Many software orchestration frameworks provide deployment and management capabilities for Docker containers. For example:

- Amazon EC2 Container Service
- Docker Swarm
- Kubernetes
- Mesosphere Marathon

The ForgeRock DevOps Examples support the Kubernetes orchestration framework. Kubernetes lets users take advantage of built-in features, such as automated best-effort container placement, monitoring, elastic scaling, storage orchestration, self-healing, service discovery, load balancing, secret management, and configuration management.

There are many Kubernetes implementations. The ForgeRock DevOps Examples support deployment to the following implementations:

- Google Container Engine (GKE) , Google's cloud-based Kubernetes orchestration framework for Docker containers. GKE is suitable for production deployments of the ForgeRock Identity Platform.
- Minikube , a single-node Kubernetes cluster running inside a virtual machine. Minikube provides a single-system deployment environment suitable for proofs of concept and development.

The Kubernetes framework uses `.json` and/or `.yaml` format *manifests*—configuration files—to specify deployment artifacts. Kubernetes Helm is a tool that lets you specify *charts* to package Kubernetes manifests together.

The ForgeRock DevOps Examples include the following Kubernetes support files, available for download with a ForgeRock BackStage account:

- Helm charts to deploy the ForgeRock Identity Platform on Minikube and GKE.

- Kubernetes manifests to deploy a load balancer—referred to as an *ingress* in Kubernetes nomenclature—on GKE. (Minikube deployments use a built-in ingress.)
- Deployment scripts to deploy Docker containers using the Helm charts and Kubernetes manifests on Minikube and GKE.

To obtain these files, clone the Git repository at https://stash.forgerock.org/projects/DOCKER/repos/fretes.

You can either use the reference Helm charts available in the `fretes` repository when deploying the ForgeRock Identity Platform, or you can customize the charts as needed before deploying the ForgeRock Identity Platform to a Kubernetes cluster. Deployment to a Kubernetes implementation other than Minikube or GKE is possible although significant customization might be required.

You can initialize the configuration of the OpenAM, OpenIDM, and OpenIG components of the ForgeRock Identity Platform from JSON files. The Git repository at `https://stash.forgerock.org/scm/cloud/forgeops-init.git`, available as part of the DevOps Examples, contains example JSON files that you can use to configure these ForgeRock components.

Note

ForgeRock also provides a service broker for applications orchestrated in the Cloud Foundry framework (which is *not* a Kubernetes orchestration framework). The service broker lets Cloud Foundry applications access OAuth 2.0 features provided by the ForgeRock Identity Platform. For more information, see the ForgeRock Service Broker Guide .

1.3. Limitations

The following are known limitations of DevOps deployments on the ForgeRock Identity Platform 5.0.0:

- OpenAM browser-based authentication is stateful. Therefore, users performing authentication against an OpenAM server must always return to the same server instance. As a result, when an authenticating user accesses OpenAM through a load balancer, the load balancer must be configured to use sticky sessions.
- The OpenAM and OpenDJ deployment example does not support exporting configuration directly to a Git repository. See Section 3.6, "Modifying and Saving the OpenAM Configuration" for details.
- Changing some OpenAM configuration properties requires a server restart. For OpenAM deployments with mutable configuration, modifications to these properties do not take effect until the containers running OpenAM are redeployed.
- OpenIDM servers are unable to start if the OpenIDM JDBC repository is unavailable. Therefore, it is imperative that the JDBC repository is up and running before you attempt to start OpenIDM in a DevOps deployment.

- Clustered OpenIDM servers are not removed from the cluster node list when they are brought down or when they fail. In elastic deployments, with servers frequently added to and removed from clusters, the cluster node list can grow to be quite large.

The following are known limitations of the ForgeRock DevOps Examples:

- The DevOps Examples do not include examples of OpenAM Web Policy Agent and OpenAM Java EE Policy Agent.
- The DevOps Examples do not include example deployments of the OpenAM **ssoadm** command. However, you can use the OpenAM REST API and the **amster** command with the OpenAM and OpenDJ deployment example.
- The DevOps Examples do not include a deployment example of the entire ForgeRock Identity Platform. Examples are available for ForgeRock Identity Platform components only.
- Example commands in this guide have been tested on the macOS operating system only. Modifications might be required when running the example commands on other operating systems.

Chapter 2

Implementing DevOps Environments

This chapter provides instructions for setting up environments to run DevOps deployment examples in this guide.

2.1. Kubernetes Running on a Minikube Virtual Machine

This section specifies requirements for an environment in which you can deploy examples that run in a Minikube virtual machine. Perform the following steps to set up a Minikube environment:

1. Review Section 2.1.1, "Introducing the Minikube Environment".
2. Install all of the required software listed in Section 2.1.1.1, "Software Requirements".
3. Create a Minikube virtual machine. See Section 2.1.2, "Creating and Initializing a Minikube Virtual Machine".

After completing these steps, you are ready to deploy any examples in this guide that use a Minikube environment.

2.1.1. Introducing the Minikube Environment

Minikube lets you run Kubernetes locally by providing a single-node Kubernetes cluster inside a virtual machine.

The following components are required for a Minikube environment:

- **Hypervisor**. A hypervisor is required to run the Minikube virtual machine. The ForgeRock DevOps Examples have been tested with Oracle VirtualBox.
- **Minikube**. Minikube software includes the **minikube** client, Docker and rkt containerization run-time environments, and tools for creating and operating a single-system virtual machine running a Kubernetes cluster. The ForgeRock DevOps Examples have been tested with Docker containerization only.
- **Docker**. Minikube deployments require the **docker** client included with Docker software in addition to the Docker containerization run-time environment included with Minikube. In a Minikube deployment, use the **docker** client to:
  - Push reference or customized images to the Docker Engine in Minikube

  • Push reference or customized images to a private Docker registry
  • Pull reference or customized images from a private Docker registry to the Docker Engine in Minikube

- **Kubernetes**. Minikube deployments require the **kubectl** client in addition to the Kubernetes runtime environment included with Minikube software. Use the **kubectl** client to perform various operations on the Kubernetes cluster.

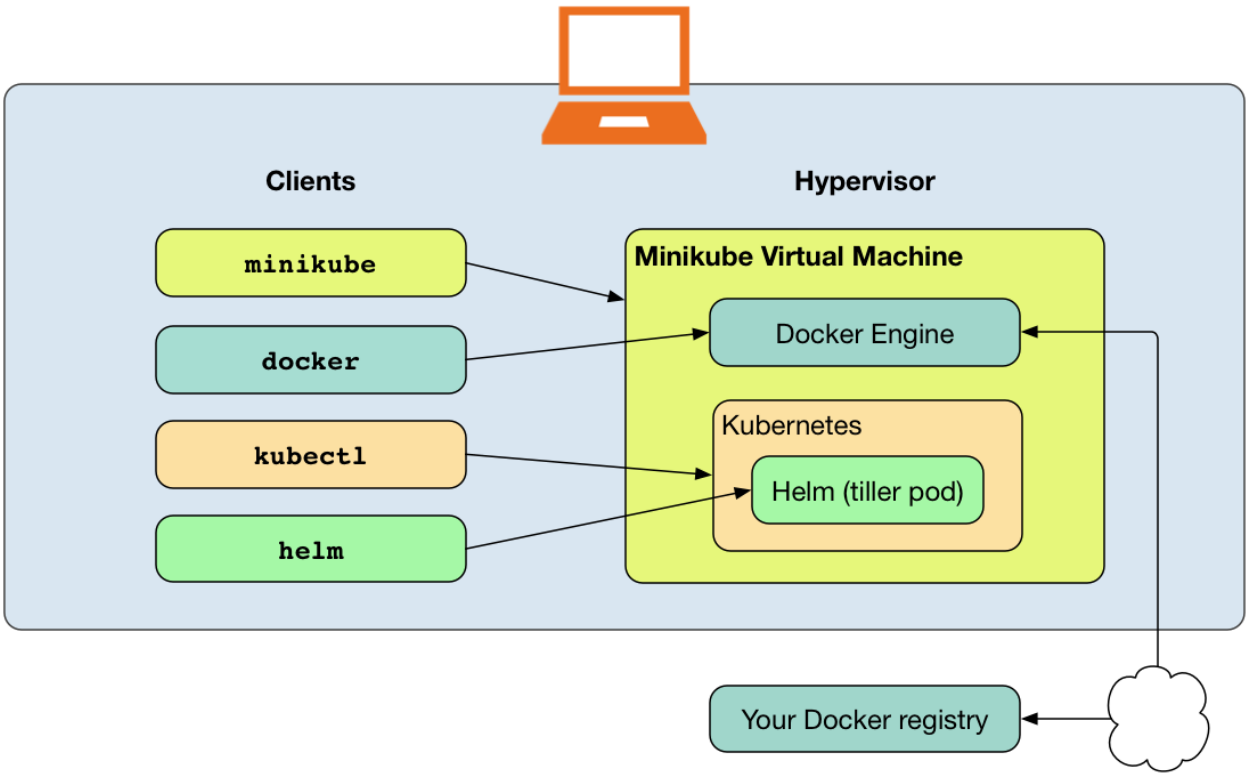
  Supported Kubernetes cluster versions for this release of the DevOps Examples are listed in Table 2.2, "Software Versions for Minikube Deployment Environments". [1]

- **Helm**. The ForgeRock DevOps Examples use Helm charts to orchestrate containerized applications within Kubernetes. Helm software includes the **helm** client and the Helm application that runs in Kubernetes, named `tiller`.

The following diagram illustrates a Minikube environment.

[1] Note that you can specify a Kubernetes cluster version other than Minikube's default version with the `--kubernetes-version` option of the **minikube start** command.

Figure 2.1. Minikube Environment



2.1.1.1. Software Requirements

To create an environment for examples that run in a Minikube virtual machine, install the following software in your local environment.

Table 2.1. Software Requirements, Minikube Deployment Environment

Software	URL for More Information
Oracle VirtualBox	https://www.virtualbox.org/wiki/downloads
Docker Client	https://www.docker.com/community-edition
Minikube	http://kubernetes.io/docs/getting-started-guides/minikube

Software	URL for More Information
kubectl (Kubernetes client)	https://kubernetes.io/docs/tasks/kubectl/install
Kubernetes Helm	https://github.com/kubernetes/helm/blob/master/docs/install.md

The DevOps Examples have been tested with a combination of software versions listed in the following table. Although using older or newer versions of the software *might* work, we recommend using the versions listed in the table when running the examples.

Table 2.2. Software Versions for Minikube Deployment Environments

Oracle VirtualBox	Docker Client	Minikube	Kubernetes Cluster	kubectl	Kubernetes Helm
5.1.20	17.03.1-ce-*platform*	0.18.0	1.6.0	1.6.2	2.3.1

2.1.2. Creating and Initializing a Minikube Virtual Machine

Perform the following procedure to create and initialize a Minikube virtual machine:

Procedure 2.1. To Create and Initialize a Minikube Virtual Machine

1. Run the following command to create the Minikube virtual machine:

```
$ minikube start --memory=8192 --disk-size=30g --vm-driver=virtualbox
Starting local Kubernetes cluster...
Kubectl is now configured to use the cluster.
```

   When you create a Minikube VM with a version of Minikube, the `Downloading Minikube ISO` message might also appear in the **minikube start** command output.

   The **minikube start** command takes several minutes to run. The command creates a VirtualBox virtual machine with 8 GB RAM and a 30 GB virtual disk. This amount of RAM and disk space is adequate for running the reference deployments in the ForgeRock DevOps Examples. You might need to increase the RAM and disk space for customized deployments or if you plan to use the environment for other purposes.

   If you do not have 8 GB of free memory, you can change the virtual machine memory usage to 4 GB by specifying `--memory=4096` as a **minikube start** command option. Although this amount of RAM is smaller than the requirement for testing or production systems, it might be enough for evaluation.

2. Run the following command to initialize Helm:

```
$ helm init
$HELM_HOME has been configured at $HOME/.helm.

Tiller (the helm server side component) has been installed into your Kubernetes Cluster.
Happy Helming!
```

3. (Optional) Run tests to verify that your environment is operational:

   a. Deploy the sample `hello-minikube` pod on port 8000 on the Kubernetes cluster running in Minikube:

   ```
   $ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --hostport=8000 --
   port=8080
   deployment "hello-minikube" created
   ```

   b. Run the **kubectl get pods** command, which lists the pods running on the Kubernetes cluster:

   ```
   $ kubectl get pods --all-namespaces
   NAMESPACE     NAME                                    READY     STATUS    RESTARTS   AGE
   default       hello-minikube-55824521-wgz88           1/1       Running   0          1m
   kube-system   default-http-backend-74vjw              1/1       Running   0          31m
   kube-system   kube-addon-manager-minikube             1/1       Running   0          31m
   kube-system   kube-dns-v20-txn3t                      3/3       Running   0          31m
   kube-system   kubernetes-dashboard-73s9j              1/1       Running   0          31m
   kube-system   nginx-ingress-controller-6320p          1/1       Running   0          31m
   kube-system   tiller-deploy-2885612843-4mp1l          1/1       Running   0          9m
   ```

   The pods in the `kube-system` namespace are deployed automatically with Minikube except for the `tiller-deploy` pod, which was deployed when you ran the **helm init** command.

   Verify that all the Kubernetes pods have reached Running status before proceeding. With a slow network connection, it might take several minutes before all the pods reach Running status.

   c. Access the `hello-minikube` pod. Run a **curl** command to port 8000 on the Minikube IP address:

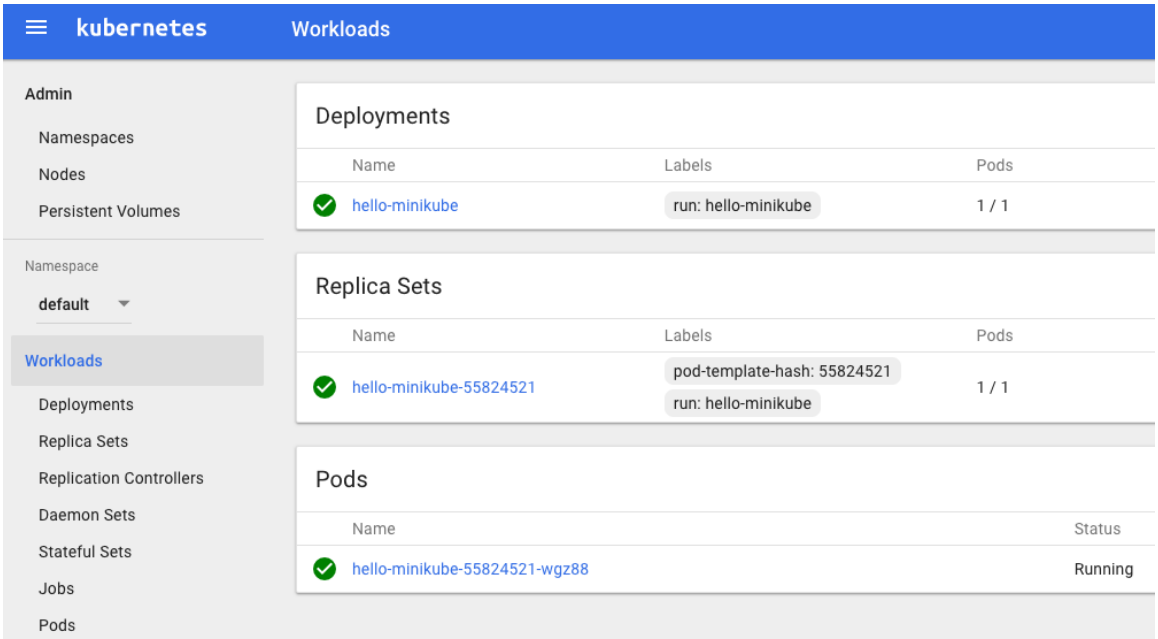
   ```
   $ curl $(minikube ip):8000
   CLIENT VALUES:
   client_address=192.168.99.1
   command=GET
   real path=/
   query=nil
   request_version=1.1
   request_uri=http://192.168.99.100:8080/

   SERVER VALUES:
   server_version=nginx: 1.10.0 - lua: 10001

   HEADERS RECEIVED:
   accept=*/*
   host=192.168.99.100:8000
   user-agent=curl/7.43.0
   BODY:
   -no body in request-
   ```

d. Start the Kubernetes dashboard, a web UI for managing a Kubernetes cluster:

```
$ minikube dashboard
```

A page similar to the following appears in your browser.

Figure 2.2. Kubernetes Dashboard for a Minikube Deployment



You are now ready to deploy any examples in this guide that use a Minikube environment.

2.1.3. Deleting a Minikube Virtual Machine

If you no longer want to use a Minikube environment, execute the following commands to remove the Minikube virtual machine from your system:

```
$ minikube stop
Stopping local Kubernetes cluster...
Machine stopped.
$ minikube delete
Deleting local Kubernetes cluster...
Machine deleted.
```

2.2. Kubernetes Running on Google Container Engine

This section specifies requirements for an environment in which you can deploy examples that run on Google Container Engine (GKE). Perform the following steps to set up a GKE environment:

1. Review Section 2.2.1, "Introducing the GKE Environment".
2. Install all of the required software listed in Section 2.2.1.1, "Software Requirements".
3. Perform required post-installation activities outlined in Section 2.2.2, "Performing One-Time Post-Installation Activities".
4. Create a GKE cluster. See Section 2.2.3, "Creating and Initializing a GKE Kubernetes Cluster".

After completing these steps, you are ready to deploy any examples in this guide that use a GKE environment.

2.2.1. Introducing the GKE Environment

GKE provides container management and orchestration, enabling you to run Kubernetes clusters on Google Cloud Platform.

The following components are required for a GKE environment:

- **Google Cloud Platform Project**. GKE resources belong to a Google Cloud Platform project. Projects enable billing, allow administrators to specify collaborators, and support other Google services, such as GKE deployment. In order to work with a project, your Google account must be added to the project with a suitable role.

  The **gcloud** client provides command-line access to Google Cloud Platform projects.

- **GKE Cluster**. A GKE cluster is a group of resources managed by Kubernetes within Google Cloud Platform.

  The **gcloud** client provides command-line access to GKE clusters.

- **Docker**. GKE deployments require the **docker** client included with Docker software in addition to the Docker containerization run-time environment included with GKE. In a GKE deployment, use the **docker** client to:

  - Push reference or customized images to the Docker Engine in your GKE cluster
  - Push reference or customized images to a private Docker registry

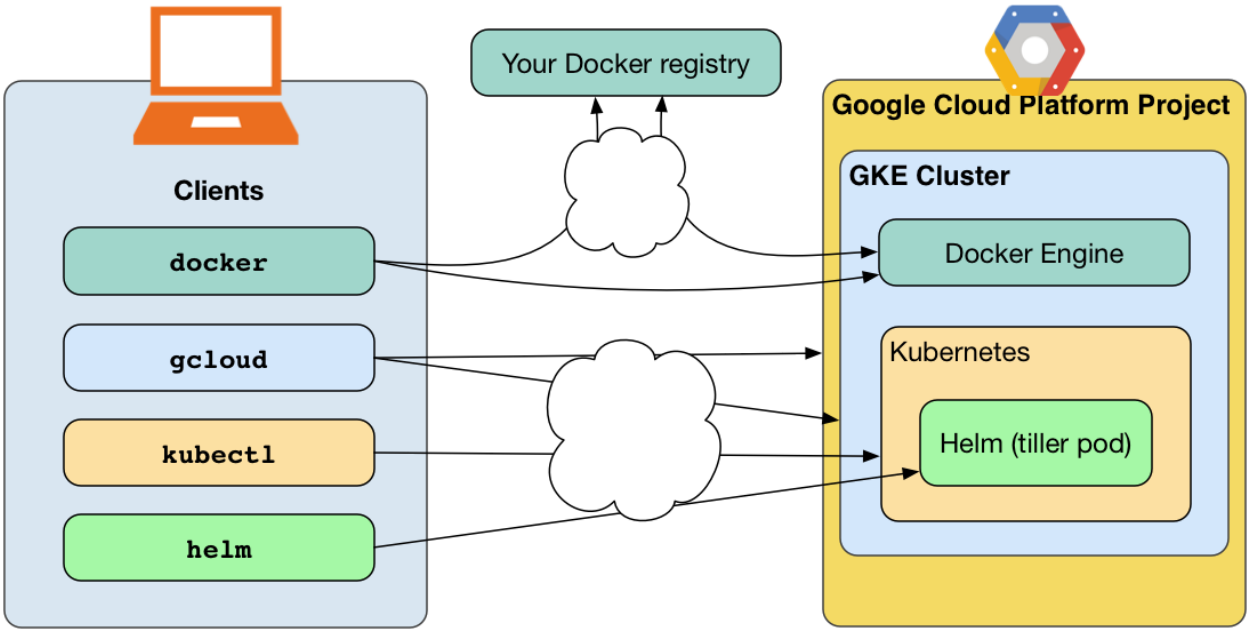
- **Kubernetes**. GKE deployments require the **kubectl** client to perform various operations on the Kubernetes cluster.

  Supported Kubernetes cluster versions for this release of the DevOps Examples are listed in Table 2.4, "Software Versions for GKE Deployment Environments".

- **Helm**. The ForgeRock DevOps Examples use Helm charts to orchestrate containerized applications within Kubernetes. Helm software includes the **helm** client and the Helm application that runs in Kubernetes, named `tiller`.

The following diagram illustrates a GKE environment.

Figure 2.3. GKE Environment



2.2.1.1. Software Requirements

To create an environment for examples that run in GKE, install the following software in your local environment.

Table 2.3. Software Requirements, GKE Deployment Environment

Software	URL for More Information
Google Cloud SDK	https://cloud.google.com/sdk/downloads
Docker Client	https://www.docker.com/community-edition

Software	URL for More Information
kubectl (Kubernetes client)	https://kubernetes.io/docs/tasks/kubectl/install
Kubernetes Helm	https://github.com/kubernetes/helm/blob/master/docs/install.md

The DevOps Examples have been tested with a combination of software versions listed in the following table. Although using older or newer versions of the software *might* work, we recommend using the versions listed in the table when running the examples.

Table 2.4. Software Versions for GKE Deployment Environments

Google Cloud SDK	Docker Client	Kubernetes Cluster	kubectl	Kubernetes Helm
152.0.0	17.03.1-ce-*platform*	1.5.6, 1.6.1	1.6.2	2.3.1

2.2.2. Performing One-Time Post-Installation Activities

After installing all the software specified in Table 2.3, "Software Requirements, GKE Deployment Environment", perform the following procedure:

Procedure 2.2. To Perform Post-Installation Actions

1. Create a new Google Cloud Platform project, or gain access to an existing project, with the Google Container Engine API enabled. See Quickstart for Google Container Engine for more information about project requirements for GKE.

   If you are given access to an existing project, request the `owner` or `editor` project role. One of these two roles is required for GKE cluster creation.

2. If you did not run the optional `install.sh` script when you installed Google Cloud SDK, add the directory where Google Cloud SDK binaries are installed, `/path/to/google-cloud-sdk/bin`, to your path.

3. Install the alpha and beta components to the Google Cloud SDK:

```
$ gcloud components install --quiet alpha beta
Your current Cloud SDK version is: 140.0.0
Installing components from version: 140.0.0

┌──────────────────────────────────────────────┐
│     These components will be installed.      │
├──────────────────────┬──────────┬────────────┤
│         Name         │  Version │    Size    │
├──────────────────────┼──────────┼────────────┤
│ gcloud Alpha Commands │ 2016.01.12 │ < 1 MiB │
│ gcloud Beta Commands  │ 2016.01.12 │ < 1 MiB │
└──────────────────────┴──────────┴────────────┘

For the latest full release notes, please visit:
  https://cloud.google.com/sdk/release_notes

╔══════════════════════════════════════════════════════════════╗
╠═ Creating update staging area                               ═╣
╠══════════════════════════════════════════════════════════════╣
╠═ Installing: gcloud Alpha Commands                          ═╣
╠══════════════════════════════════════════════════════════════╣
╠═ Installing: gcloud Beta Commands                           ═╣
╠══════════════════════════════════════════════════════════════╣
╠═ Creating backup and activating new installation            ═╣
╚══════════════════════════════════════════════════════════════╝

Performing post processing steps...done.

Update done!
```

4. Configure the Google Cloud SDK standard and beta components to use your Google account: [2]

   a. Configure the Google Cloud SDK standard component:

   ```
   $ gcloud auth login
   ```

   A Google screen appears in your browser, prompting you to authenticate to Google. Authenticate using your Google account with an `owner` or `editor` role in the project in which you will create a GKE cluster.

   b. Configure the Google Cloud SDK beta component:

   ```
   $ gcloud beta auth application-default login
   ```

   A Google screen appears in your browser, prompting you to authenticate to Google. Authenticate using the same account you used to configure the standard component.

5. Configure the Google Cloud SDK to use your Google Cloud Platform project:

   a. List Google Cloud Platform projects associated with your Google account:

[2] No account configuration is required for the Google SDK alpha component.

```
$ gcloud projects list
PROJECT_ID          NAME                PROJECT_NUMBER
my-project          My Project          12345767890123
```

b. Configure the Google Cloud SDK for your project:

```
$ gcloud config set project my-project
Updated property [core/project].
```

You have now completed the post-installation activities required for the GKE environment.

2.2.3. Creating and Initializing a GKE Kubernetes Cluster

After completing the steps outlined in Section 2.2.2, "Performing One-Time Post-Installation Activities", perform the following procedure to create and initialize a Kubernetes cluster on GKE:

Procedure 2.3. To Create and Initialize a GKE Cluster

1. Change to the `gke` directory within the clone of the `fretes` repository.
2. Review the `create-cluster.sh` script, which creates a GKE cluster.

   By default, this script creates a cluster named `openam`, with a 50 GB disk on an `n1-standard-2` class host. The cluster is single-node and can expand to four nodes.

   Modify the **gcloud alpha container clusters** command in the `create-cluster.sh` script if you want to change any of the cluster defaults.
3. Create the GKE cluster:

```
$ ./create-cluster.sh --cluster-name my-cluster
This will create a cluster with all Kubernetes Alpha features enabled
.
- This cluster will not covered by the Container Engine SLA and should
 not be used for production workloads
.
- You will not be able to upgrade the master or nodes
.
- The cluster will be deleted after 30 days.

Do you want to continue (Y/n)?  Y

Creating cluster my-cluster...done.
Created [https://container.googleapis.com/v1/projects/engineering-devops/zones/us-central1-f/clusters/
my-cluster].
kubeconfig entry generated for my-cluster.
NAME        ZONE           MASTER_VERSION              MASTER_IP        MACHINE_TYPE   NODE_VERSION
 NUM_NODES  STATUS
my-cluster  us-central1-f  1.5.2 ALPHA (29 days left)  104.154.104.103  n1-standard-2  1.5.2         1
          RUNNING
```

   GKE cluster creation takes several minutes.

4. If you intend to work with reference or customized Docker images from your own Docker registry, set up your GKE cluster to access the registry.

   If your private registry is a Google Container Registry, see https://cloud.google.com/container-registry/ for more information.

   For private registries hosted by other vendors, refer to your vendor's documentation.

5. Run the following command to initialize Helm:

```
$ helm init
$HELM_HOME has been configured at $HOME/.helm.

Tiller (the helm server side component) has been installed into your Kubernetes Cluster.
Happy Helming!
```

6. (Optional) Run tests to verify that your environment is operational:

   a. Run the **kubectl get pods** command, which lists the pods running on the Kubernetes cluster. Output should be similar to the following:

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                                    READY     STATUS
 RESTARTS   AGE
kube-system   fluentd-cloud-logging-gke-my-....-default-pool-90228dc6-b5p1 1/1       Running   0
        24m
kube-system   heapster-v1.2.0-3455740371-hdkj0                         2/2       Running   0
        23m
kube-system   kube-dns-4101612645-620f9                               4/4       Running   0
        25m
kube-system   kube-dns-autoscaler-2715466192-rqcs7                    1/1       Running   0
        25m
kube-system   kube-proxy-gke-my-cluster-default-pool-90228dc6-b5p1     1/1       Running   0
        24m
kube-system   kubernetes-dashboard-3543765157-9sgxg                   1/1       Running   0
        25m
kube-system   tiller-deploy-2885612843-5jcsp                          1/1       Running   0
        4m
```

      The pods in the `kube-system` namespace are deployed automatically during cluster creation except for the `tiller-deploy` pod, which was deployed when you ran the **helm init** command.

   b. Start a Kubernetes proxy and access the Kubernetes dashboard:

      i. Open a separate terminal window.

      ii. Run the following command in the separate terminal window:

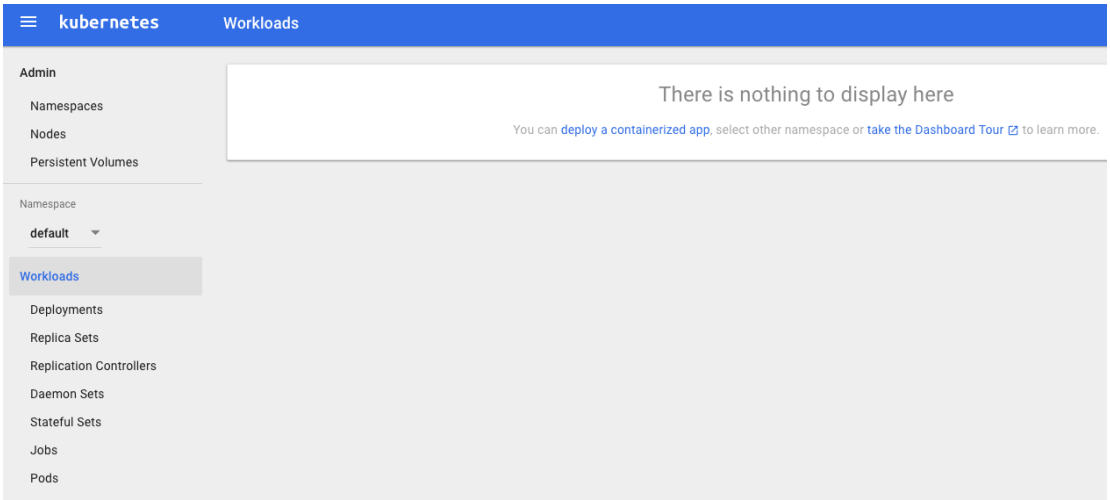
```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

      Do not close the terminal window unless you no longer want to access the Kubernetes dashboard.

iii. Navigate to http://localhost:8001/ui in a browser.

A page similar to the following appears.

Figure 2.4. Kubernetes Dashboard for a GKE Deployment



2.2.4. Deleting a GKE Kubernetes Cluster

If you no longer want to use a GKE environment, perform the following procedure to remove the cluster from your Google Cloud Platform project:

Procedure 2.4. To Delete a GKE Cluster

1. Change to the `helm/bin` directory within the clone of the `fretes` repository.
2. Run the **remove-all.sh** script to remove all components from the cluster. [3]
3. If you do not need persistent volumes (PVs) used by the cluster to remain available after cluster deletion, delete them by using the Google Compute Engine console.

[3] The **remove-all.sh** explicitly removes persistent volume claims (PVCs) from the cluster, thus making it possible to remove persistent volumes (PVs) from Kubernetes in a subsequent step.

Warning

Do *not* perform this step unless you are certain that you do not intend to use the PVs at a later time.

4. Change to the `gke` directory within the clone of the `fretes` repository.
5. Review the `delete-cluster.sh` script, which deletes a GKE cluster.

   Modify the **gcloud container clusters** command in the `delete-cluster.sh` script if you want to change any of the script's defaults.
6. Delete the GKE cluster:

```
$ ./delete-cluster.sh --cluster-name my-cluster
The following clusters will be deleted.
 - [my-cluster] in [us-central1-f]

Do you want to continue (Y/n)?  Y

Deleting cluster my-cluster...done.
Deleted [https://container.googleapis.com/v1/projects/engineering-devops/zones/us-central1-f/clusters/
my-cluster].
```

Chapter 3

Deploying the OpenAM and OpenDJ Example

This chapter provides instructions for deploying the reference implementation of the OpenAM and OpenDJ DevOps example.

The following is a high-level overview of the steps to deploy this example:

- Familiarize yourself with the example deployment. See the deployment diagram and explanation in Section 3.1, "About the Example".
- Prepare an environment for running the example, including verifying that you have a supported environment, removing objects from previous deployments from the environment, and deploying a Kubernetes ingress (load balancer). See Section 3.3, "Preparing the Environment".
- Download ForgeRock software and create Docker images for OpenAM, Amster, and OpenDJ. See Section 3.4, "Creating Docker Images".
- Orchestrate the example in a Kubernetes cluster and verify the deployment. See Section 3.5, "Orchestrating the Deployment".

3.1. About the Example

The reference deployment of the OpenAM and OpenDJ DevOps example has the following architectural characteristics:

- **Kubernetes ingress**. From outside the deployment, OpenAM is accessed through a Kubernetes ingress (load balancer) configured with session stickiness.
- **Installation-only `openam` and `amster` pods**. These two pods are created when you run the `openam.sh` script that installs the Helm chart that initializes OpenAM. [1] After installation and configuration completes, these pods are no longer needed and are deleted for the deployment.
- **Run-time `openam-xxxxxxxxxx-yyyyy` pod(s)**. This pod, created elastically by Kubernetes[1], runs the OpenAM server. Multiple instances can be started if required. The Kubernetes ingress redirects requests to one of these pods.
- **Run-time `amster-aaaaaaaaaa-bbbbb` pod(s)**. This pod, also created elastically by Kubernetes[1], lets users run the **amster** command after OpenAM installation has completed. For example, if you were using the `openam` run-time to modify the OpenAM configuration, you might create a **cron** job in this pod

[1] Pods created statically, such as the `openam` and `amster` pods, can have fixed names. Run-time pods created elastically by Kubernetes have variable names.

that runs an **amster** command to export the configuration to a Git repository or to flat files. Multiple instances can be started if required, although a single instance is usually sufficient.

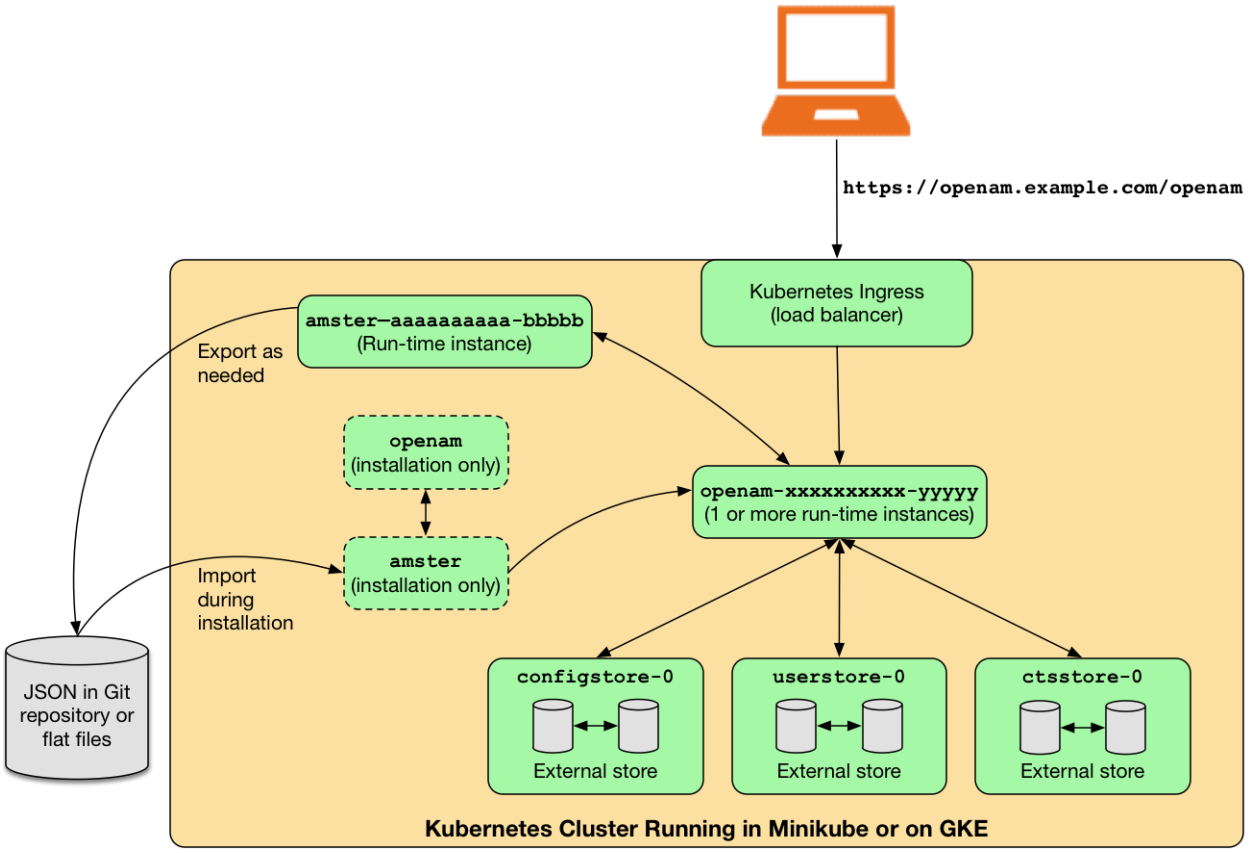
- **JSON configuration stored in a Git repository or flat files**. After installing OpenAM, the deployment imports configuration stored in JSON files that it obtains from either flat files or a Git repository. The additional configuration is accessible to, but not inside of the Kubernetes cluster. Because configuration is stored outside of the cluster, it can persist if the cluster is deleted.

  You can initiate a job in the `amster-aaaaaaaaaa-bbbbb` pod that exports the OpenAM configuration, and then use the resulting output as input to a new OpenAM deployment.

- **External OpenDJ stores for OpenAM configuration, users, and CTS tokens**. All of the stores that OpenAM uses are created as external stores that can optionally be replicated.

The following diagram illustrates the example.

Figure 3.1. OpenAM and OpenDJ DevOps Deployment Example



3.2. Working With the OpenAM and OpenDJ Example

This section presents an example workflow to set up a development environment in which you configure OpenAM, iteratively modify and save the OpenAM configuration, and then migrate the configuration to a test or production environment.

This workflow illustrates many of the capabilities available in the DevOps Examples. It is only one way of working with the example deployment. Use this workflow to help you better understand the DevOps Examples, and as a starting point for your own DevOps deployments.

Note that this workflow is an overview of how you might work with the DevOps Examples and does not provide step-by-step instructions. It does provide links to subsequent sections in this chapter that include detailed procedures you can follow when deploying the DevOps Examples.

Table 3.1. Example Workflow, OpenAM and OpenDJ DevOps Deployment

Step	Details
Implement a Minikube environment	Set up a Minikube environment for developing the OpenAM configuration. See Section 2.1, "Kubernetes Running on a Minikube Virtual Machine".
Get up-to-date versions of required Git repositories	Make sure you have up-to-date versions of the `release/5.0.0` branch of the Git repositories that contain Docker and Kubernetes artifacts and initial configuration files. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".
Deploy the OpenAM and OpenDJ example in Minikube	Follow the procedures in Section 3.3, "Preparing the Environment", Section 3.4, "Creating Docker Images", and Section 3.5, "Orchestrating the Deployment". Specify the following value in the `custom.yaml` file described in Section 3.5.1, "Specifying Deployment Options": • `hostPath: /path/containing/forgeops-init-clone` The `hostPath` value *must* have a subdirectory named `forgeops-init`. The `forgeops-init` repository initializes OpenAM by creating the following: • A sample realm • A sample OIDC client Remove the sample configuration if you do not want it.
Modify and save the OpenAM configuration	Iterate through the following steps as many times as you need to: • Modify the OpenAM configuration using the OpenAM console, the REST API, or the **amster** command. See Section 3.5.3, "Verifying the Deployment" for details about how to access the deployed OpenAM server. • Export the OpenAM configuration as described in Section 3.6, "Modifying and Saving the OpenAM Configuration". OpenAM configuration files will be exported

Step	Details
	to the directory you defined as the `hostPath` in the `custom.yaml` file when you deployed the example. • Manage the OpenAM configuration in a Git repository. When you move to a test or production deployment, you will use the configuration stored in the Git repository to initialize OpenAM.
Implement a GKE environment	Set up a GKE environment for test and production deployments. See Section 2.2, "Kubernetes Running on Google Container Engine".
Deploy the OpenAM and OpenDJ example in GKE	Follow the procedures in Section 3.3, "Preparing the Environment", Section 3.4, "Creating Docker Images", and Section 3.5, "Orchestrating the Deployment". Be sure to specify the following values in the `custom.yaml` file described in Section 3.5.1, "Specifying Deployment Options": • `registry: gcr.io/` • `repo: myGKEProject` • `stackConfigSource: gitRepo: repository: myConfigGitRepo` • `stackConfigSource: gitRepo: revision: myConfigRevision` For `myConfigGitRepo`, specify the Git repository in which you saved the OpenAM configuration while you were developing it. For `myConfigRevision`, specify the Git revision that contains the desired version of the OpenAM configuration.

After you have deployed a test or production OpenAM server, you can continue to update the OpenAM configuration in your development environment, and then redeploy OpenAM with the updated configuration. Reiterate the development/deployment cycle as follows:

- Modify the OpenAM configuration on the Minikube deployment and commit the changes in a Git repository.
- Redeploy the OpenAM and OpenDJ example in GKE based on the updated configuration, specifying the desired revision in the `stackConfigSource: gitRepo: revision:` property in the `custom.yaml` file.

3.3. Preparing the Environment

The OpenAM and OpenDJ DevOps example can be run in the Minikube and GKE environments.

Before deploying the example, be sure you have a working environment as described in one of the following sections:

- Section 2.1, "Kubernetes Running on a Minikube Virtual Machine"
- Section 2.2, "Kubernetes Running on Google Container Engine"

Most of the steps for deploying the example are identical for the two test environments. Environment-specific differences are called out in the deployment procedures in this chapter.

To prepare your environment:

- Make sure you have up-to-date versions of the `release/5.0.0` branch of the `docker`, `fretes`, and `forgeops-init` repositories. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".
- Remove any objects left over from previous deployments to ensure you are deploying the example in a clean environment. See Section 3.3.1, "Removing Existing Deployment Objects".
- Deploy an ingress in your environment. See Section 3.3.2, "Deploying a Kubernetes Ingress".

3.3.1. Removing Existing Deployment Objects

Before deploying the example, ensure that any objects remaining from previous deployments have been removed from your environment. Perform the following procedure:

Procedure 3.1. To Remove Existing Deployment Artifacts

1. Verify that Helm is running in your environment:

   ```
   $ kubectl get pods --all-namespaces | grep tiller-deploy
   kube-system   tiller-deploy-2779452559-3bznh              1/1       Running   1          13d
   ```

   If the **kubectl** command returns no output, restart Helm by running the **helm init** command.

   Note that the **helm init** command starts a Kubernetes pod with a name starting with `tiller-deploy`.

2. Run the **remove-all.sh** script to remove any Kubernetes objects left over from previous ForgeRock deployments:

   ```
   $ cd /path/to/fretes/helm/bin
   $ ./remove-all.sh
   ```

   Output from the **remove-all.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. `Error: release: not found` messages *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the **kubectl get pods** command to verify that no pods that run ForgeRock software [2] in the `default` namespace are active in your test environment.

   If Kubernetes pods running ForgeRock software are still active, wait several seconds, and then run the **kubectl get pods** command again. You might need to run the command several times before all the pods running ForgeRock software are terminated.

[2] See the deployment diagrams in the introductory sections for each DevOps example for the names of pods that run ForgeRock software. For example, see Section 3.1, "About the Example" for the names of pods deployed for the OpenAM and OpenDJ example.

If all the pods in the cluster were running ForgeRock software, the procedure is complete when the `No resources found` message appears:

```
$ kubectl get pods
No resources found.
```

If some pods in the cluster were running non-ForgeRock software, the procedure is complete when only pods running non-ForgeRock software appear in response to the **kubectl get pods** command. For example:

```
$ kubectl get pods
hello-minikube-55824521-b0qmb   1/1       Running   0          2m
```

3.3.2. Deploying a Kubernetes Ingress

The OpenAM and OpenDJ DevOps example is accessed through a Kubernetes ingress.

Ingress deployment differs on Minikube and GKE environments. Perform one of the following procedures, depending on your environment:

- Procedure 3.2, "To Deploy and Access an Ingress on Minikube"
- Procedure 3.3, "To Deploy an Ingress on GKE"

Procedure 3.2. To Deploy and Access an Ingress on Minikube

Minikube users only. GKE users should perform Procedure 3.3, "To Deploy an Ingress on GKE" instead.

- Enable the default ingress built into Minikube:

```
$ minikube addons enable ingress
ingress was successfully enabled
```

Procedure 3.3. To Deploy an Ingress on GKE

GKE users only. Minikube users should perform Procedure 3.2, "To Deploy and Access an Ingress on Minikube" instead.

The GKE deployment uses the nginx ingress controller, which is suitable for development and testing. When deploying in production, use the Google Load Balancer ingress controller. Refer to the GKE documentation for more information.

Perform the following steps to deploy the nginx ingress controller:

1. Change to the directory containing Kubernetes manifests to deploy an nginx ingress:

```
$ cd /path/to/fretes/ingress
```

2. Run the `delete-ingress.sh` script to remove a leftover ingress deployment, if it exists:

```
$ ./delete-ingress.sh
```

Output from the **delete-ingress.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. Error messages indicating that components were not found *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the `create-nginx-ingress.sh` script to deploy an nginx ingress:

```
$ ./create-nginx-ingress.sh
deployment "default-http-backend" created
service "default-http-backend" created
configmap "nginx-load-balancer-conf" created
configmap "tcp-configmap" created
deployment "nginx-ingress-controller" created
```

3.4. Creating Docker Images

This section covers how to work with Docker images needed to deploy the OpenAM and OpenDJ example:

- Review which Docker images are needed to run the example, and when they need to be created, removed, and rebuilt. See Section 3.4.1, "About Docker Images for the Example".
- Remove existing Docker images from a registry or from Docker cache. See Section 3.4.2, "Removing Existing Docker Images".
- Download ForgeRock software binary files and copy them to the `docker` repository. See Section 3.4.3, "Obtaining ForgeRock Software Binary Files".
- Build Docker images based on the reference Dockerfiles provided in the `docker` repository. See Section 3.4.4, "Building Docker Images".

Note

If you need customized Docker images, refer to the `README.md` files and the Dockerfile comments in the `docker` repository.

3.4.1. About Docker Images for the Example

The OpenAM and OpenDJ example requires the following Docker images for ForgeRock components:

- `openam`
- `amster`
- `opendj`

Once created, a Docker image's contents are static. Remove and rebuild images when:

- You want to upgrade them to use newer versions of OpenAM, Amster, or OpenDJ software.
- You changed files that impact image content, and you want to redeploy modified images. Common modifications include (but are not limited to) the following:
  - Changes to security files, such as passwords and keystores.
  - Changes to file locations or other bootstrap configuration in the OpenAM `boot.json` file.
  - Changes to the Tomcat web container's configuration in the `server.xml` file.
  - Changes to the OpenAM web application configuration in the `context.xml` file.
  - Dockerfile changes to install additional software on base images.

3.4.2. Removing Existing Docker Images

If the `openam`, `amster`, and `opendj` images are present in your environment, remove them before creating new images.

Perform the following procedure to remove existing Docker images from your environment:

Procedure 3.4. To Remove Existing Docker Images

Because Docker image names can vary depending on organizations' requirements, the image names shown in the example commands in this procedure might not match your image names. For information about the naming conventions used for Docker images in the DevOps Examples, see Section 7.2, "Naming Docker Images".

Perform the following steps to remove Docker images:

1. **Minikube users only.** Set up your shell to use the Docker environment within Minikube:

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

   This command sets environment variables that enable the Docker client running on your laptop to access the Docker server running in the Minikube virtual machine.

2. Run the **docker images** command to determine whether `openam`, `amster`, and `opendj` Docker images are present in your test environment.

3. If the output from the **docker images** showed that `openam`, `amster`, and `opendj` images were present in your environment, remove them.

   If you are not familiar with removing Docker images, run the **docker rmi --help** command for more information about command-line options. For more information about ForgeRock Docker image names, see Section 7.2, "Naming Docker Images".

   The following example commands remove images from the local Docker cache in a Minikube deployment:

```
$ docker rmi -f forgerock/openam:14.0.0
Untagged: forgerock/openam:14.0.0
Deleted: sha256:7a3336f64975ee9f7b11ce77f8fa010545f05b10beb1b60e2dac306a68764ed3
Deleted: sha256:1ce5401fe3f6dfb0650447c1b825c2fae86eaa0fe5c7fccf87e6a70aed1d571d
. . .
Deleted: sha256:59701800a35ab4d112539cf958d84a6d663b31ad495992c0ff3806259df93f5d
Deleted: sha256:018353c2861979a296b60e975cb69b9f366397fe3ac30cd3fe629124c55fae8c

$ docker rmi -f forgerock/amster:14.0.0
Untagged: forgerock/amster:14.0.0
Deleted: sha256:25f5c8b9fb214e91a36f5ff7b3c286221f61ded610660902fa0bcdaef018dba6
Deleted: sha256:38fc379ca54c183bc93a16d1b824139a70ccb59cacc8f859a10e12744a593680
. . .
Deleted: sha256:b739a5393d7da17c76eb52dec786007e0394410c248fdffcc21b761054d653cb
Deleted: sha256:ba5f78fdc7d19a4e051e19bfc10c170ff869f487a74808ac5e003a268f72d34f

$ docker rmi -f forgerock/opendj:4.0.0
Untagged: forgerock/opendj:4.0.0
Deleted: sha256:65f9f4f7374a43552c4a09f9828bde618aa22e3e504e97274ca691867c1c357b
Deleted: sha256:cf7698333e0d64b25f25d270cb8facd8f8cc77c18e809580bb0978e9cb73aded
. . .
Deleted: sha256:deba2feeaea378fa7d35fc87778d3d58af287efeca288b630b4660fc9dc76435
Deleted: sha256:dcbe724b0c75a5e75b28f23a3e1550e4b1201dc37ef5158d181fc6ab3ae83271
```

4. Run the **docker images** command to verify that you removed the `openam`, `amster`, and `opendj` images.

3.4.3. Obtaining ForgeRock Software Binary Files

Perform the following procedure if:

- You have not yet obtained ForgeRock software binary files for the OpenAM and OpenDJ example.
- You want to obtain newer versions of ForgeRock software that versions you previously downloaded.

Skip this step if you want to build Docker images based on versions of ForgeRock software you previously downloaded and copied into the `docker` repository.

Procedure 3.5. To Obtain ForgeRock Binary Files

Perform the steps in the following procedure to obtain ForgeRock software for the OpenAM and OpenDJ example, and to copy it to the required locations for building the `openam`, `amster`, and `opendj` Docker images:

1. Download the following binary files from the ForgeRock BackStage download site :
   - `AM-5.0.0.war`
   - `Amster-5.0.0.zip`
   - `DS-5.0.0.zip`
2. Copy (or move) and rename the downloaded binary files as follows:

Table 3.2. Binary File Locations, OpenAM and OpenDJ Example

Binary File	Location
`AM-5.0.0.war`	`/path/to/docker/openam/openam.war`
`Amster-5.0.0.zip`	`/path/to/docker/amster/amster.zip`
`DS-5.0.0.zip`	`/path/to/docker/opendj/opendj.zip`

3.4.4. Building Docker Images

Perform one of the following procedures to build the `openam`, `amster`, and `opendj` Docker images:

- **Minikube users**, perform Procedure 3.6, "To Build Docker Images for Minikube".
- **GKE users**, perform Procedure 3.7, "To Build Docker Images for GKE".

Procedure 3.6. To Build Docker Images for Minikube

Minikube users only. GKE users should perform Procedure 3.7, "To Build Docker Images for GKE" instead.

Perform the following steps:

1. If you have not already done so, set up your shell to use the Docker environment within Minikube:
   ```
   $ eval $(minikube docker-env)
   ```
2. Change to the directory that contains the clone of the ForgeRock `docker` repository:
   ```
   $ cd /path/to/docker
   ```
3. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a Minikube deployment:
   ```
   $ ./build.sh -R forgerock -t 14.0.0 openam
   ```
   This command builds a Docker image with the repository name `forgerock/openam`, tags the image with `14.0.0`, and writes the image in the local Docker cache.
4. Build the `openam`, `amster`, and `opendj` images using the **build.sh** script:
   a. Build the `openam` image:
      ```
      $ ./build.sh -R forgerock -t 14.0.0 openam
      Building openam
      Sending build context to Docker daemon 144.8 MB
      Step 1 : FROM tomcat:8.5-jre8
      ```

```
 ---> 7f855aeeaebf
Step 2 : ENV CATALINA_HOME /usr/local/tomcat
 ---> Running in f613161e1b06
 ---> 4199c6ae2c5a
Removing intermediate container f613161e1b06
Step 3 : ENV PATH $CATALINA_HOME/bin:$PATH
 ---> Running in aa131ff5d44a
 ---> cff5da14da9f
Removing intermediate container aa131ff5d44a
Step 4 : WORKDIR $CATALINA_HOME
 ---> Running in 8fad09628c4a
 ---> 2f9cb34ac2c7
Removing intermediate container 8fad09628c4a
Step 5 : EXPOSE 8080 8443
 ---> Running in 8038c3c9281f
 ---> 54c200f4813d
Removing intermediate container 8038c3c9281f
Step 6 : ENV OPENAM_VERSION 14.0.0
 ---> Running in 9dd825024400
 ---> c9586820aa82
Removing intermediate container 9dd825024400
Step 7 : ADD openam.war /tmp/openam.war
 ---> 0ac70fd8fb81
Removing intermediate container 1c83327ff0c1
Step 8 : RUN rm -fr /usr/local/tomcat/webapps/*   && unzip -q /tmp/openam.war -d /usr/local/tomcat
/webapps/openam   && rm /tmp/openam.war
 ---> Running in c05cf8197dde
 ---> 9e6cdd4ff296
Removing intermediate container c05cf8197dde
Step 9 : RUN mkdir -p /root/openam/openam && mkdir -p /root/.openamcfg   && echo "/root/openam" >
/root/.openamcfg/AMConfig_usr_local_tomcat_webapps_openam_
 ---> Running in 6c0851d44e7f
 ---> 482d64b26dff
Removing intermediate container 6c0851d44e7f
Step 10 : ADD server.xml /usr/local/tomcat/conf/server.xml
 ---> 64a47ce7463c
Removing intermediate container 3bb4dfad036f
Step 11 : ADD context.xml /usr/local/tomcat/conf/context.xml
 ---> 256025cfb766
Removing intermediate container 30ec1806087f
Successfully built 256025cfb766
```

b. Build the `amster` image:

```
$ ./build.sh -R forgerock -t 14.0.0 amster
Building amster
Sending build context to Docker daemon 25.99 MB
Step 1 : FROM openjdk:8-jre-alpine
 ---> c017141bdaa8
Step 2 : ADD *.zip /tmp/
 ---> 7fce2b86e189
Removing intermediate container 8ab7f85cd047
Step 3 : RUN apk add --no-cache su-exec unzip curl git     && unzip -q /tmp/amster.zip -d /var/tmp
/amster     && rm /tmp/*.zip
 ---> Running in 6a13f123abab
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/community/x86_64/APKINDEX.tar.gz
(1/8) Installing libssh2 (1.7.0-r2)
```

```
(2/8) Installing libcurl (7.52.1-r2)
(3/8) Installing curl (7.52.1-r2)
(4/8) Installing expat (2.2.0-r0)
(5/8) Installing pcre (8.39-r0)
(6/8) Installing git (2.11.1-r0)
(7/8) Installing su-exec (0.2-r0)
(8/8) Installing unzip (6.0-r2)
Executing busybox-1.25.1-r0.trigger
OK: 99 MiB in 57 packages
 ---> 9b1d59b625ce
Removing intermediate container 6a13f123abab
Step 4 : WORKDIR /var/tmp
 ---> Running in 4bedfd5f647f
 ---> 6f1697b77ac9
Removing intermediate container 4bedfd5f647f
Step 5 : ADD *.sh /var/tmp/
 ---> aa1b30563b1c
Removing intermediate container 3d175a90afc9
Step 6 : ENTRYPOINT /var/tmp/docker-entrypoint.sh
 ---> Running in 5e3b6c098118
 ---> 58c21c403e38
Removing intermediate container 5e3b6c098118
Step 7 : CMD configure
 ---> Running in ce00d8de0e65
 ---> 724fd4476171
Removing intermediate container ce00d8de0e65
Successfully built 724fd4476171
```

c. Build the `opendj` image:

```
$ ./build.sh -R forgerock -t 4.0.0 opendj
Building opendj
Sending build context to Docker daemon 36.72 MB
Step 1 : FROM openjdk:8-jre
 ---> a4d689e63201
Step 2 : WORKDIR /opt
 ---> Running in ffc8232c7e52
 ---> b6550cc28c42
Removing intermediate container ffc8232c7e52
Step 3 : ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/
 ---> Running in 71a836937fad
 ---> 6367adbb58b6
Removing intermediate container 71a836937fad
Step 4 : ENV OPENDJ_JAVA_ARGS -server -Xmx512m -XX:+UseG1GC
 ---> Running in 96a918edef8e
 ---> 8fdc128681e5
Removing intermediate container 96a918edef8e
Step 5 : ENV DIR_MANAGER_PW_FILE /var/secrets/opendj/dirmanager.pw
 ---> Running in 751f210697d0
 ---> 246c03259df2
Removing intermediate container 751f210697d0
Step 6 : ENV BASE_DN dc=example,dc=com
 ---> Running in d303c1514118
 ---> de39441e1c8a
Removing intermediate container d303c1514118
Step 7 : ADD opendj.zip /tmp/
 ---> 94b6ee9fdb0c
Removing intermediate container 66aef4aa1c77
```

```
Step 8 : RUN apt-get update && apt-get install -y ldap-utils &&     unzip -q /tmp/opendj.zip
-d /opt  && rm /tmp/opendj.zip &&     echo "/opt/opendj/data" > /opt/opendj/instance.loc  &&
   mkdir -p /opt/opendj/data/lib/extensions &&     mkdir -p /var/secrets/opendj &&     echo -n
 "password"  > ${DIR_MANAGER_PW_FILE}
 ---> Running in 3a00d1d90ad7
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Get:2 http://security.debian.org jessie/updates/main amd64 Packages [458 kB]
Ign http://deb.debian.org jessie InRelease
Get:3 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:4 http://deb.debian.org jessie-backports InRelease [166 kB]
Get:5 http://deb.debian.org jessie Release.gpg [2373 B]
Get:6 http://deb.debian.org jessie Release [148 kB]
Get:7 http://deb.debian.org jessie-updates/main amd64 Packages [17.6 kB]
Get:8 http://deb.debian.org jessie-backports/main amd64 Packages [1119 kB]
Get:9 http://deb.debian.org jessie/main amd64 Packages [9049 kB]
Fetched 11.2 MB in 3s (3683 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following extra packages will be installed:
  libsasl2-modules
Suggested packages:
  libsasl2-modules-gssapi-mit libsasl2-modules-gssapi-heimdal
  libsasl2-modules-otp libsasl2-modules-ldap libsasl2-modules-sql
The following NEW packages will be installed:
  ldap-utils libsasl2-modules
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 289 kB of archives.
After this operation, 943 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian/ jessie/main ldap-utils amd64 2.4.40+dfsg-1+deb8u2 [188 kB]
Get:2 http://deb.debian.org/debian/ jessie/main libsasl2-modules amd64 2.1.26.dfsg1-13+deb8u1 [101
 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 289 kB in 0s (1912 kB/s)
Selecting previously unselected package ldap-utils.
(Reading database ... 9186 files and directories currently installed.)
Preparing to unpack .../ldap-utils_2.4.40+dfsg-1+deb8u2_amd64.deb ...
Unpacking ldap-utils (2.4.40+dfsg-1+deb8u2) ...
Selecting previously unselected package libsasl2-modules:amd64.
Preparing to unpack .../libsasl2-modules_2.1.26.dfsg1-13+deb8u1_amd64.deb ...
Unpacking libsasl2-modules:amd64 (2.1.26.dfsg1-13+deb8u1) ...
Setting up ldap-utils (2.4.40+dfsg-1+deb8u2) ...
Setting up libsasl2-modules:amd64 (2.1.26.dfsg1-13+deb8u1) ...
 ---> 2b2d3c313012
Removing intermediate container 3a00d1d90ad7
Step 9 : WORKDIR /opt/opendj
 ---> Running in b8669c69d1d9
 ---> 2198186af3ab
Removing intermediate container b8669c69d1d9
Step 10 : ADD Dockerfile /
 ---> 8d25c0a60b5a
Removing intermediate container 87285951f0bb
Step 11 : ADD bootstrap/ /opt/opendj/bootstrap/
 ---> 4c15398f6d98
Removing intermediate container 5d80afc80145
Step 12 : ADD *.sh /opt/opendj/
 ---> 016a130cce40
Removing intermediate container 592c9795ee1a
```

```
Step 13 : EXPOSE 389 636 4444 8989
 ---> Running in cc4d52ed967f
 ---> 66add8ed6024
Removing intermediate container cc4d52ed967f
Step 14 : ADD run.sh /opt/opendj/run.sh
 ---> 7a8606b4ed07
Removing intermediate container 0b4606ea8db1
Step 15 : CMD /opt/opendj/run.sh
 ---> Running in 4637ed1dd28c
 ---> 2b9f33380b5a
Removing intermediate container 4637ed1dd28c
Successfully built 2b9f33380b5a
```

5. Run the **docker images** command to verify that the `openam`, `amster`, and `opendj` images are now available.

Procedure 3.7. To Build Docker Images for GKE

GKE users only. Minikube users should perform Procedure 3.6, "To Build Docker Images for Minikube" instead.

Perform the following steps:

1. Change to the directory that contains the clone of the ForgeRock `docker` repository:

   ```
   $ cd /path/to/docker
   ```

2. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a GKE deployment:

   ```
   $ ./build.sh -r gcr.io -R myProject -t 14.0.0 -g openam
   ```

   This command builds a Docker image for deployment on GKE with the repository name `myProject/openam`, tags the image with `14.0.0`, and pushes the image to the `gcr.io` registry. Note that for Docker images deployed on GKE, the first part of the repository component of the image name *must* be your Google Cloud Platform project name.

3. Build the `openam`, `amster`, and `opendj` images using the **build.sh** script:

   a. Build the `openam` image. For example:

      ```
      $ ./build.sh -R gcr.io -R myProject -t 14.0.0 -g openam
      Building openam
      . . .
      ```

   b. Build the `amster` image. For example:

      ```
      $ ./build.sh -R gcr.io -R myProject -t 14.0.0 -g amster
      Building amster
      . . .
      ```

c. Build the `opendj` image. For example:

```
$ ./build.sh -R gcr.io -R myProject -t 14.0.0 -g opendj
Building opendj
. . .
```

4. Run the **docker images** command to verify that the `openam`, `amster`, and `opendj` images are now available.

3.5. Orchestrating the Deployment

This section covers how to orchestrate the Docker containers for this deployment example into your Kubernetes environment.

3.5.1. Specifying Deployment Options

Kubernetes options specified in the `/path/to/fretes/helm/custom.yaml` file override default options specified in Helm charts in the reference deployment.

Before deploying this example, you must edit this file and specify options pertinent to your deployment.

For information about specifying deployment options in the `custom.yaml` file, see Section 7.4, "Specifying Deployment Options in the custom.yaml File". For a commented example, see the `/path/to/fretes/helm/custom-template.yaml` file in the `fretes` repository.

3.5.1.1. custom.yaml File Examples

This section provides several examples of `custom.yaml` files that could be used with the OpenAM and OpenDJ DevOps example.

Example 3.1. Minikube Deployment, Configuration Imported From Flat Files

The following is an example of a `custom.yaml` file for a deployment on Minikube in which OpenAM configuration is imported from flat files after OpenAM installation:

```
cookieDomain: .example.com
registry: ""
stackConfigSource:
  hostPath:
    path: /path/to/additional-config
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys Docker images from the local cache.
- After installation, OpenAM configuration is imported from the `/path/to/additional-config/forgeops-init/amster` directory.

- After deployment, OpenAM uses `example.com` as its cookie domain.

Example 3.2. GKE Deployment, Configuration Imported From a Git Repository

The following is an example of a `custom.yaml` file for a deployment on GKE in which OpenAM configuration is imported from a Git repository after OpenAM installation:

```
cookieDomain: .example.com
registry: gcr.io/
repo: engineering-devops
stackConfigSource:
  gitRepo:
    repository: https://stash.forgerock.org/scm/cloud/forgeops-init.git
    revision: HEAD
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys Docker images from the `engineering-devops` repository in the `gcr.io` Docker registry.
- After installation, additional OpenAM configuration is imported from the `forgeops-init/openam` directory of the `HEAD` revision of the `https://stash.forgerock.org/scm/cloud/forgeops-init.git` Git repository.
- After deployment, OpenAM uses `example.com` as its cookie domain.

Example 3.3. Minikube Deployment, Default OpenAM Configuration

The following is an example of a `custom.yaml` file for a deployment on Minikube in which no OpenAM configuration is imported after OpenAM installation:

```
cookieDomain: .example.com
amster:
  skipImport: true
registry: ""
stackConfigSource:
  emptyDir: {}
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys Docker images from the local cache.
- The default OpenAM configuration is applied during installation. No other configuration is applied. This option is valid only for Minikube deployments.
- After deployment, OpenAM uses `example.com` as its cookie domain.

Note

When you specify the `amster: skipImport` option with the value `true`:

- The `stackConfigSource` option is still required, as shown in the preceding example. Specify the value `emptyDir: {}` to indicate that no OpenAM configuration should be applied after installation.

- External CTS and user stores are ready and available for use, but are *not* used unless you explicitly configure OpenAM to use them.

3.5.2. Deploying Helm Charts

Perform the steps in the following procedure to deploy the Helm charts for the OpenAM and OpenDJ DevOps example to the Kubernetes cluster in your environment:

Procedure 3.8. To Deploy Helm Charts for the OpenAM and OpenDJ Example

1. Change to the `/path/to/fretes/helm/bin` directory.
2. Run the **openam.sh** script, which deploys OpenDJ instances, calls Amster to install and configure OpenAM, and then brings up OpenAM:

   ```
   $ ./openam.sh
   ```

   Output similar to the following appears in the terminal window:

   ```
   Creating OpenDJ configuration store
   Configuring instance configstore
   NAME:   configstore
   LAST DEPLOYED: Wed Mar 29 09:59:26 2017
   NAMESPACE: default
   STATUS: DEPLOYED

   RESOURCES:
   ==> v1/Service
   NAME         CLUSTER-IP  EXTERNAL-IP  PORT(S)           AGE
   configstore  None        <none>       389/TCP,4444/TCP  1s

   ==> apps/v1beta1/StatefulSet
   NAME         DESIRED  CURRENT  AGE
   configstore  1        1        1s

   ==> v1/Secret
   NAME         TYPE    DATA  AGE
   configstore  Opaque  1     1s


   Waiting for pod configstore-0 to be
    ready
   ..done
   Creating OpenDJ user store
   Configuring instance userstore
   NAME:   userstore
   LAST DEPLOYED: Wed Mar 29 09:59:47 2017
   NAMESPACE: default
   STATUS: DEPLOYED

   RESOURCES:
   ==> v1/Secret
   NAME       TYPE    DATA  AGE
   userstore  Opaque  1     0s
   ```

```
==> v1/Service
NAME       CLUSTER-IP  EXTERNAL-IP  PORT(S)           AGE
userstore  None        <none>       389/TCP,4444/TCP  0s

==> apps/v1beta1/StatefulSet
NAME       DESIRED  CURRENT  AGE
userstore  1        1        0s


Waiting for pod userstore-0 to be
 ready
.....done
Creating OpenDJ CTS store
Configuring instance ctsstore
NAME:   ctsstore
LAST DEPLOYED: Wed Mar 29 10:00:39 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME      TYPE    DATA  AGE
ctsstore  Opaque  1     2s

==> v1/Service
NAME      CLUSTER-IP  EXTERNAL-IP  PORT(S)           AGE
ctsstore  None        <none>       389/TCP,4444/TCP  2s

==> apps/v1beta1/StatefulSet
NAME      DESIRED  CURRENT  AGE
ctsstore  1        1        2s


Waiting for pod ctsstore-0 to be
 ready
....done
Installing amster chart
NAME:   amster
LAST DEPLOYED: Wed Mar 29 10:01:23 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME            TYPE    DATA  AGE
amster-secrets  Opaque  8     2s

==> v1/ConfigMap
NAME           DATA  AGE
amster-config  1     2s

==> v1/Service
NAME    CLUSTER-IP  EXTERNAL-IP  PORT(S)       AGE
openam  10.0.0.85   <nodes>      80:31741/TCP  2s

==> v1/Pod
NAME    READY  STATUS             RESTARTS  AGE
amster  0/1    ContainerCreating  0         2s
```

```
openam  0/1    Init:0/1           0         2s


Waiting for pod amster to be ready
Waiting for OpenAM server at http://openam:80/openam/config/options
.htm
...Got Response code 000
response code 000. Will continue to
 wait
.Waiting for OpenAM server at http://openam:80/openam/config/options.htm
Got Response code 200
OpenAM web app is up and ready to be configured
About to begin configuration
Executing Amster to configure OpenAM
Executing Amster script /amster/00_install.amster
Mar 29, 2017 5:02:44 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Amster OpenAM Shell (14.0.0 build 24b5258daa, JVM: 1.8.0_121)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /amster/00_install
.amster
.03/29/2017 05:02:49:619 PM UTC: Checking license acceptance...
03/29/2017 05:02:49:619 PM UTC: License terms accepted.
03/29/2017 05:02:49:627 PM UTC: Checking configuration directory /root/openam.
03/29/2017 05:02:49:628 PM UTC: ...Success.
03/29/2017 05:02:49:633 PM UTC: Tag swapping schema files.
03/29/2017 05:02:49:710 PM UTC: ...Success.
03/29/2017 05:02:49:714 PM UTC: Loading Schema odsee_config_schema.ldif
03/29/2017 05:02:49:844 PM UTC: ...Success
.
. . .
03/29/2017 05:03:11:444 PM UTC: Setting up monitoring authentication file.
Configuration complete!
Executing Amster script /amster/01_import.amster
Amster OpenAM Shell (14.0.0 build 24b5258daa, JVM: 1.8.0_121)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /amster/01_import
.amster
.Importing directory /amster-config/forgeops-init/amster
Imported /amster-config/forgeops-init/amster/global/SessionPropertyWhiteList.json
Imported /amster-config/forgeops-init/amster/global/User.json
Imported /amster-config/forgeops-init/amster/global/Naming.json
Imported /amster-config/forgeops-init/amster/global/PushNotification
.json
. . .
Import completed successfully

Configuration script finished
.Removing the amster installation chart
Starting openam runtime
NAME:   openam
LAST DEPLOYED: Wed Mar 29 10:03:28 2017
NAMESPACE: default
STATUS: DEPLOYED
```

```
RESOURCES:
==> v1/Secret
NAME            TYPE    DATA  AGE
openam-secrets  Opaque  8     1s

==> v1/ConfigMap
NAME       DATA  AGE
boot-json  1     1s

==> v1/Service
NAME    CLUSTER-IP  EXTERNAL-IP  PORT(S)                      AGE
openam  10.0.0.7    <nodes>      80:30080/TCP,443:30443/TCP  1s

==> extensions/v1beta1/Deployment
NAME    DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
openam  1        1        1           0          1s
amster  1        1        1           0          1s

==> extensions/v1beta1/Ingress
NAME    HOSTS  ADDRESS         PORTS  AGE
openam  *      192.168.99.100  80     1s


bin/openam.sh: line 44: 79530 Terminated: 15          kubectl logs amster -f
You will see a Terminated: message from the kubectl logs command. It is OK to ignore this.
Done
```

3. Review the output from the OpenAM and OpenDJ deployment and verify that no errors occurred during OpenAM installation.

   The start of the output for OpenAM installation is similar to the following:

```
Amster OpenAM Shell (14.0.0 build 24b5258daa, JVM: 1.8.0_121)
Type ':help' or ':h' for help
.
--------------------------------------------------------------------------------
am> :load /amster/00_install
.amster
.03/29/2017 05:02:49:619 PM UTC: Checking license acceptance...
03/29/2017 05:02:49:619 PM UTC: License terms accepted.
03/29/2017 05:02:49:627 PM UTC: Checking configuration directory /root/openam.
03/29/2017 05:02:49:628 PM UTC: ...Success.
```

4. If you imported configuration after OpenAM installation, review the output from the OpenAM and OpenDJ deployment and verify that no errors occurred during the import.

   The start of the output for importing the OpenAM configuration is similar to the following:

```
Executing amster script /amster/01_import
.amster
.Amster OpenAM Shell (14.0.0 build 24b5258daa, JVM: 1.8.0_121)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /amster/01_import.amster
Importing directory /amster-config/forgeops-init/amster
Imported /amster-config/forgeops-init/amster/global/AgentService.json
Imported /amster-config/forgeops-init/amster/global/AuditLogging.json
Imported /amster-config/forgeops-init/amster/global/AuthenticatorOath.json
```

5. Query the status of pods in your deployment until all pods are ready:

   a. Run the **kubectl get pods** command to query your deployment's status. For example:

   ```
   $ kubectl get pods
   NAME                     READY     STATUS              RESTARTS   AGE
   amster-498943944-jn84f   0/1       Running             0          32s
   configstore-0            1/1       Running             0          2m
   ctsstore-0               1/1       Running             0          2m
   openam-23824049-14mqh    0/1       Running             0          32s
   userstore-0              1/1       Running             0          2m
   ```

   b. Review the output from the **kubectl get pods** command. Deployment is complete when:

   - All pods are completely ready. For example, a pod with the value `1/1` in the `READY` column of the output is completely ready, while a pod with the value `0/1` is not completely ready.
   - All pods have attained `Running` status.

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

6. Get the ingress' IP address:

   ```
   $ kubectl get ingresses
   kubectl get ingresses
   NAME             HOSTS     ADDRESS          PORTS     AGE
   openam-ingress   *         104.154.45.166   80        15m
   ```

7. Add an entry similar to the following to your `/etc/hosts` file to enable access to the cluster through the ingress:

   ```
   104.154.46.166 openam.example.com
   ```

   In this example, `104.154.46.166` is the IP address returned from the **kubectl get ingresses** command.

3.5.3. Verifying the Deployment

After you have deployed the Helm charts for the example, verify that the deployment is active and available by accessing the OpenAM console and upgrading OpenAM if necessary:

Procedure 3.9. To Verify the Deployment

1. If necessary, start a web browser.
2. Navigate to the OpenAM deployment URL, for example, `https://openam.example.com/openam`.

   The Kubernetes ingress handles the request and routes you to a running OpenAM instance.
3. OpenAM prompts you to log in or upgrade depending on how you initialized the OpenAM configuration:

   - If you configured Helm to import the configuration, and if the version of OpenAM from which the configuration was exported matches the version of OpenAM you just installed, then OpenAM prompts you to log in.
   - If you configured Helm to import the configuration and, if the version of OpenAM from which the configuration was exported is older than the version of OpenAM you just installed, then OpenAM prompts you to upgrade.
   - If you configured Helm to skip importing OpenAM configuration (`amster: skipImport: true` option in the `custom.yaml` file), then OpenAM prompts you to log in.

   If the login page appears, deployment of the example was successful. Log in to OpenAM as the `amadmin` user with password `password`. *Skip the rest of the steps in this procedure*.

   If a page prompting you to upgrade OpenAM appears, then you must do so before you can verify the deployment. *Perform the remaining steps in this procedure*.
4. Click the link to upgrade OpenAM.

   A window with the license agreement appears.
5. Scroll to the bottom of the license agreement.

   Click the check box to accept the license agreement, and then click Continue.

   A dialog box identifying the current and new versions of OpenAM appears:

   - The current version is the version of OpenAM from which the configuration used to initialize OpenAM was exported.
   - The new version is the version of OpenAM installed into Kubernetes.
6. Click Upgrade.

   Progress messages appear while OpenAM is upgraded. When the upgrade finishes, the Upgrade Complete dialog box appears.
7. Restart the pod that runs the OpenAM server:

a. Query Kubernetes for the pod with a name that includes the string `openam`. For example:

```
$ kubectl get pods | grep openam
openam-23824049-14mqh    1/1       Running             0          1h
```

b. Delete the pod that runs the OpenAM server. For example:

```
$ kubectl delete pod openam-23824049-14mqh
pod "openam-23824049-14mqh" deleted
```

After deletion, Kubernetes automatically starts a new OpenAM pod that accesses the upgraded configuration store.

c. Run the **kubectl get pods** command to locate a pod with a name that includes the string `openam`. For example:

```
$ kubectl get pods | grep openam
openam-23824049-lkvp5    0/1       Running             0          2m
```

Observe that Kubernetes started a new `openam` pod with a name that differs from the original OpenAM pod name.

d. If necessary, continue to query your deployment's status until the new pod is completely deployed, as follows:

- The new pod is completely ready. For example, a pod with the value `1/1` in the `READY` column of the output is completely ready, while a pod with the value `0/1` is not completely ready.
- The new pod has attained `Running` status.

8. Navigate to the URL, `https://openam.example.com/openam`. The login page should appear now, and you should be able to log in to OpenAM as the `amadmin` user with password `password`.

3.6. Modifying and Saving the OpenAM Configuration

Important

Configuration management capabilities described in this section are available only for deployments running on Minikube for which the `stackConfigSource` option in the `custom.yaml` file is set to `hostPath`.

You can use these configuration management techniques as you develop the OpenAM configuration, and then import the configuration to a production environment running on GKE.

After you have successfully orchestrated an OpenAM and OpenDJ deployment as described in this chapter, you can modify the OpenAM configuration, save it, and use the revised configuration to initialize a subsequent OpenAM deployment.

Storing the configuration in a version control system like a Git repository lets you take advantage of capabilities such as version control, difference analysis, and branches when managing the OpenAM

configuration. Configuration management enables migration from a development environment to a test environment and then to a production environment. Deployment migration is one of the primary objectives of DevOps techniques.

To modify the OpenAM configuration, use any OpenAM management tool:

- The OpenAM console
- The OpenAM REST API
- Amster

Once you are ready to save your configuration, perform the following procedure:

Procedure 3.10. To Save the OpenAM Configuration

1. Verify that when you deployed the OpenAM and OpenDJ example, the value of the `stackConfigSource` option in the `custom.yaml` file was set to `hostPath`.

2. Query Kubernetes for the pod with a name that includes the string `amster`. For example:

```
$ kubectl get pods | grep amster
amster-498943944-jn84f    1/1       Running             0          2m
```

3. Run the **export.sh** script in the `amster` pod you identified in the previous step:

```
$ kubectl exec amster-498943944-jn84f -it ./export.sh
```

   The script exports the OpenAM configuration to the location on your local file system identified by the `hostPath: path` option in the `custom.yaml` file.

   Output similar to the following appears in the terminal window while the **export.sh** script runs:

```
Mar 01, 2017 9:53:12 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Amster OpenAM Shell (14.0.0 build 24b5258daa, JVM: 1.8.0_121)
Type ':help' or ':h' for help
.
--------------------------------------------------------------------------------
am> :load /tmp/export.amster
Export completed successfully
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

 modified:   global/Authentication.json
 modified:   global/AuthenticatorOath.json
 modified:   global/AuthenticatorPush.json

 . . .

 modified:   realms/root/Scripts/9de3eb62-f131-4fac-a294-7bd170fd4acb.json
 modified:   realms/root/Scripts/c827d2b4-3608-4693-868e-bbcf86bd87c7.json
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)

 global/Realms/root-realm2.json
 global/Realms/root.json
 global/Servers/01/DirectoryConfiguration.json
 global/Sites/
 realms/root-realm2/

no changes added to commit (use "git add" and/or "git commit -a")
```

4. If you are managing the OpenAM configuration using Git, commit changed files and add new files to the repository.

Saving the updated configuration is complete. You can redeploy the OpenAM and OpenDJ example using the updated configuration at any time.

Chapter 4

Deploying the OpenIDM Example

This chapter provides instructions for deploying the reference implementation of the OpenIDM DevOps example.

The following is a high-level overview of the steps to deploy this example:

- Familiarize yourself with the example deployment. See the deployment diagram and explanation in Section 4.1, "About the Example".
- Prepare an environment for running the example, including verifying that you have a supported environment, removing objects from previous deployments from the environment, and deploying a Kubernetes ingress (load balancer). See Section 4.3, "Preparing the Environment".
- Download ForgeRock software and create Docker images for OpenIDM and OpenDJ. See Section 4.4, "Creating Docker Images".
- Orchestrate the example in a Kubernetes cluster and verify the deployment. See Section 4.5, "Orchestrating the Deployment".

4.1. About the Example

The reference deployment of the OpenIDM DevOps example has the following architectural characteristics:

- **Kubernetes ingress**. From outside the deployment, OpenIDM is accessed through a Kubernetes ingress (load balancer).
- **Run-time `openidm-xxxxxxxxxx-yyyyy` pod(s)**. This pod, created elastically by Kubernetes[1], runs the OpenIDM server. Multiple instances can be started if required. The Kubernetes ingress redirects requests to one of these pods.
- **Run-time `openidm-postgres-aaaaaaaaaa-bbbbb` pod(s)**. This pod, created elastically by Kubernetes[1], runs the OpenIDM repository as a PostgreSQL database.

  The PostgreSQL pod is for development use only. When deploying OpenIDM in production, configure your JDBC repository to support clustered, highly-available operations.

[1] Pods created statically, such as the `userstore-0` pod, can have fixed names. Run-time pods created elastically by Kubernetes have variable names.

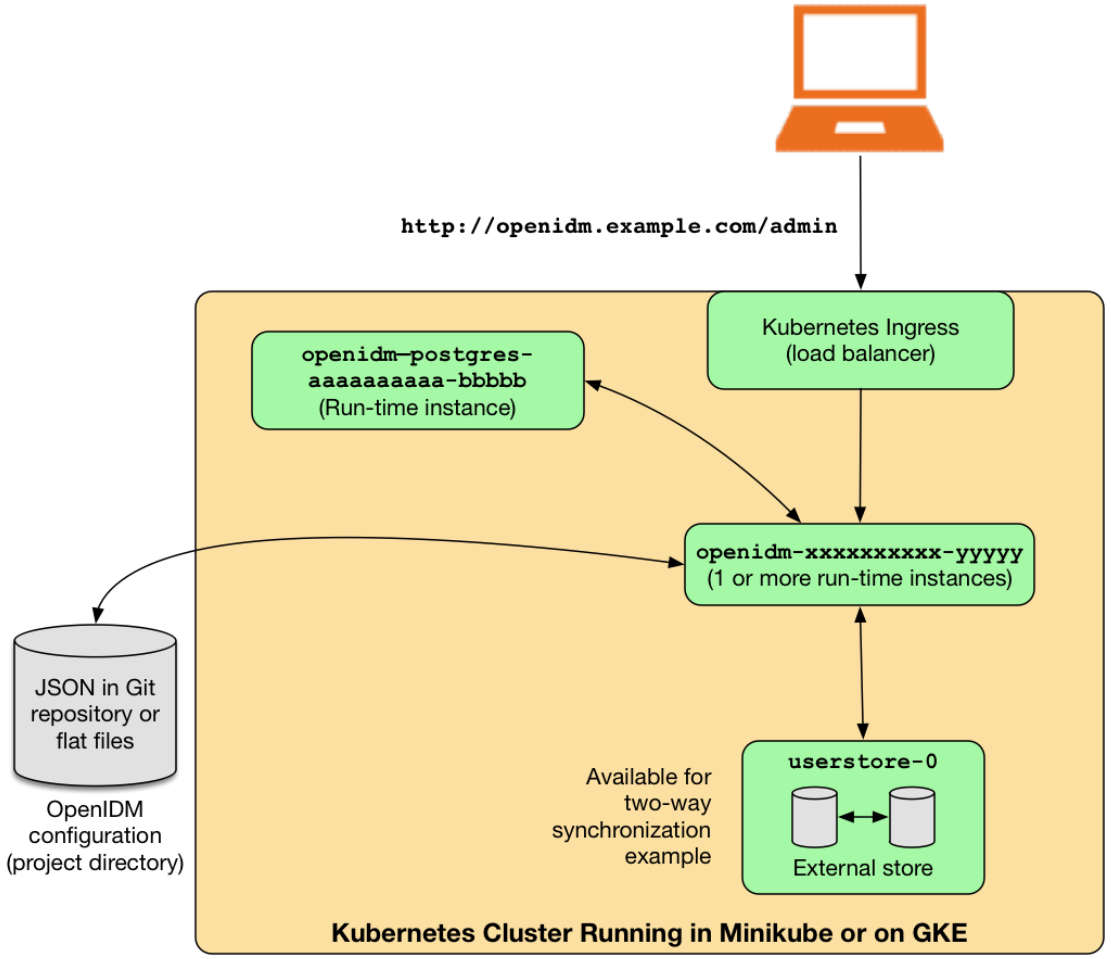
- **JSON configuration stored in a Git repository or flat files**. After installation, the deployment starts OpenIDM, referencing configuration stored in JSON files that it obtains from either flat files or a Git repository. The OpenIDM configuration is accessible to, but not inside of the Kubernetes cluster. Because configuration is stored outside of the cluster, it can persist if the cluster is deleted.

  If you use flat files for the OpenIDM JSON configuration, OpenIDM updates the files if you modify the configuration from the Admin UI or by using REST API calls.

- **OpenDJ user store**. The reference deployment implements bidirectional data synchronization between OpenIDM and LDAP described in Synchronizing Data Between LDAP and OpenIDM in the *Samples Guide*. The OpenDJ user store contains the LDAP entries that are synchronized.

The following diagram illustrates the example.

Figure 4.1. OpenIDM DevOps Deployment Example



4.2. Working With the OpenIDM Example

This section presents an example workflow to set up a development environment in which you configure OpenIDM, iteratively modify and save the OpenIDM configuration, and then migrate the configuration to a test or production environment.

This workflow illustrates many of the capabilities available in the DevOps Examples. It is only one way of working with the example deployment. Use this workflow to help you better understand the DevOps Examples, and as a starting point for your own DevOps deployments.

Note that this workflow is an overview of how you might work with the DevOps Examples and does not provide step-by-step instructions. It does provide links to subsequent sections in this chapter that include detailed procedures you can follow when deploying the DevOps Examples.

Table 4.1. Example Workflow, OpenIDM DevOps Deployment

Step	Details
Implement a Minikube environment	Set up a Minikube environment for developing the OpenIDM configuration. See Section 2.1, "Kubernetes Running on a Minikube Virtual Machine".
Get up-to-date versions of required Git repositories	Make sure you have up-to-date versions of the `release/5.0.0` branch of the Git repositories that contain Docker and Kubernetes artifacts and initial configuration files. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".
Deploy the OpenIDM example in Minikube	Follow the procedures in Section 4.3, "Preparing the Environment", Section 4.4, "Creating Docker Images", and Section 4.5, "Orchestrating the Deployment". Specify the following value in the `custom.yaml` file described in Section 4.5.1, "Specifying Deployment Options": • `hostPath: /path/containing/forgeops-init-clone` The `hostPath` value *must* have a subdirectory named `forgeops-init`. The `forgeops-init` repository initializes OpenIDM with configuration for running bidirectional data synchronization between OpenIDM and LDAP, as described in Synchronizing Data Between LDAP and OpenIDM in the *Samples Guide*. Remove the configuration for this sample if you do not want it as part of your OpenIDM configuration.
Modify and save the OpenIDM configuration	Iterate through the following steps as many times as you need to: • Modify the OpenIDM configuration using the OpenIDM Admin UI or the REST API. See Section 4.5.3, "Verifying the Deployment" for details about how to access the deployed OpenIDM server. • Save the OpenIDM configuration as described in Section 4.6, "Modifying and Saving the OpenIDM Configuration". • Manage the OpenIDM configuration in a Git repository. When you move to a test or production deployment, you will use the configuration stored in the Git repository to initialize OpenIDM.
Implement a GKE environment	Set up a GKE environment to use for test and production deployments. See Section 2.2, "Kubernetes Running on Google Container Engine".
Deploy the OpenIDM example in GKE	Follow the procedures in Section 4.3, "Preparing the Environment", Section 4.4, "Creating Docker Images", and Section 4.5, "Orchestrating the Deployment".

Step	Details
	Be sure to specify the following values in the `custom.yaml` file described in Section 4.5.1, "Specifying Deployment Options": • `registry: gcr.io/` • `repo: myGKEProject` • `stackConfigSource: gitRepo: repository: myConfigGitRepo` • `stackConfigSource: gitRepo: revision: myConfigRevision` For `myConfigGitRepo`, specify the Git repository in which you saved the OpenIDM configuration while you were developing it. For `myConfigRevision`, specify the Git revision that contains the desired version of the OpenIDM configuration.

After you have deployed a test or production OpenIDM server, you can continue to update the OpenIDM configuration in your development environment, and then redeploy OpenIDM with the updated configuration. Reiterate the development/deployment cycle as follows:

- Modify the OpenIDM configuration on the Minikube deployment and commit the changes in a Git repository.
- Redeploy the OpenIDM example in GKE based on the updated configuration, specifying the desired revision in the `stackConfigSource: gitRepo: revision:` property in the `custom.yaml` file.

4.3. Preparing the Environment

The OpenIDM DevOps example can be run in the Minikube and GKE environments.

Before deploying the example, be sure you have a working environment as described in one of the following sections:

- Section 2.1, "Kubernetes Running on a Minikube Virtual Machine"
- Section 2.2, "Kubernetes Running on Google Container Engine"

Most of the steps for deploying the example are identical for the two test environments. Environment-specific differences are called out in the deployment procedures in this chapter.

To prepare your environment:

- Make sure you have up-to-date versions of the `release/5.0.0` branch of the `docker`, `fretes`, and `forgeops-init` repositories. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".
- Remove any objects left over from previous deployments to ensure you are deploying the example in a clean environment. See Section 4.3.1, "Removing Existing Deployment Objects".

- Deploy an ingress in your environment. See Section 4.3.2, "Deploying a Kubernetes Ingress".

4.3.1. Removing Existing Deployment Objects

Before deploying the example, ensure that any objects remaining from previous deployments have been removed from your environment. Perform the following procedure:

Procedure 4.1. To Remove Existing Deployment Artifacts

1. Verify that Helm is running in your environment:

   ```
   $ kubectl get pods --all-namespaces | grep tiller-deploy
   kube-system   tiller-deploy-2779452559-3bznh              1/1       Running   1          13d
   ```

   If the **kubectl** command returns no output, restart Helm by running the **helm init** command.

   Note that the **helm init** command starts a Kubernetes pod with a name starting with `tiller-deploy`.

2. Run the **remove-all.sh** script to remove any Kubernetes objects left over from previous ForgeRock deployments:

   ```
   $ cd /path/to/fretes/helm/bin
   $ ./remove-all.sh
   ```

   Output from the **remove-all.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. `Error: release: not found` messages *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the **kubectl get pods** command to verify that no pods that run ForgeRock software [2] in the `default` namespace are active in your test environment.

   If Kubernetes pods running ForgeRock software are still active, wait several seconds, and then run the **kubectl get pods** command again. You might need to run the command several times before all the pods running ForgeRock software are terminated.

   If all the pods in the cluster were running ForgeRock software, the procedure is complete when the `No resources found` message appears:

   ```
   $ kubectl get pods
   No resources found.
   ```

   If some pods in the cluster were running non-ForgeRock software, the procedure is complete when only pods running non-ForgeRock software appear in response to the **kubectl get pods** command. For example:

[2] See the deployment diagrams in the introductory sections for each DevOps example for the names of pods that run ForgeRock software. For example, see Section 4.1, "About the Example" for the names of pods deployed for the OpenIDM example.

```
$ kubectl get pods
hello-minikube-55824521-b0qmb   1/1       Running   0          2m
```

4.3.2. Deploying a Kubernetes Ingress

The OpenIDM DevOps example is accessed through a Kubernetes ingress.

Ingress deployment differs on Minikube and GKE environments. Perform one of the following procedures, depending on your environment:

- Procedure 4.2, "To Deploy and Access an Ingress on Minikube"
- Procedure 4.3, "To Deploy an Ingress on GKE"

Procedure 4.2. To Deploy and Access an Ingress on Minikube

Minikube users only. GKE users should perform Procedure 4.3, "To Deploy an Ingress on GKE" instead.

- Enable the default ingress built into Minikube:

```
$ minikube addons enable ingress
ingress was successfully enabled
```

Procedure 4.3. To Deploy an Ingress on GKE

GKE users only. Minikube users should perform Procedure 4.2, "To Deploy and Access an Ingress on Minikube" instead.

The GKE deployment uses the nginx ingress controller, which is suitable for development and testing. When deploying in production, use the Google Load Balancer ingress controller. Refer to the GKE documentation for more information.

Perform the following steps to deploy the nginx ingress controller:

1. Change to the directory containing Kubernetes manifests to deploy an nginx ingress:

```
$ cd /path/to/fretes/ingress
```

2. Run the `delete-ingress.sh` script to remove a leftover ingress deployment, if it exists:

```
$ ./delete-ingress.sh
```

   Output from the **delete-ingress.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. Error messages indicating that components were not found *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the `create-nginx-ingress.sh` script to deploy an nginx ingress:

```
$ ./create-nginx-ingress.sh
deployment "default-http-backend" created
service "default-http-backend" created
configmap "nginx-load-balancer-conf" created
configmap "tcp-configmap" created
deployment "nginx-ingress-controller" created
```

4.4. Creating Docker Images

This section covers how to work with Docker images needed to deploy the OpenIDM example:

- Review which Docker images are needed to run the example, and when they need to be created, removed, and rebuilt. See Section 4.4.1, "About Docker Images for the Example".
- Remove existing Docker images from a registry or from Docker cache. See Section 4.4.2, "Removing Existing Docker Images".
- Download ForgeRock software binary files and copy them to the `docker` repository. See Section 4.4.3, "Obtaining ForgeRock Software Binary Files".
- Build Docker images based on the reference Dockerfiles provided in the `docker` repository. See Section 4.4.4, "Building Docker Images".

Note

If you need customized Docker images, refer to the `README.md` files and the Dockerfile comments in the `docker` repository.

4.4.1. About Docker Images for the Example

The OpenIDM example requires the following Docker images for ForgeRock components:

- `openidm`
- `opendj`

Once created, a Docker image's contents are static. Remove and rebuild images when:

- You want to update them to use newer versions of OpenIDM or OpenDJ software.
- You changed files that impact image content, and you want to redeploy modified images. Common modifications include (but are not limited to) the following:
  - Changes to security files, such as passwords and keystores.
  - Dockerfile changes to install additional software on base images.

4.4.2. Removing Existing Docker Images

If the `openidm` and `opendj` images are present in your environment, remove them before creating new images.

Perform the following procedure to remove existing Docker images from your environment:

Procedure 4.4. To Remove Existing Docker Images

Because Docker image names can vary depending on organizations' requirements, the image names shown in the example commands in this procedure might not match your image names. For information about the naming conventions used for Docker images in the DevOps Examples, see Section 7.2, "Naming Docker Images".

Perform the following steps to remove Docker images:

1. **Minikube users only.** Set up your shell to use the Docker environment within Minikube:

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

   This command sets environment variables that enable the Docker client running on your laptop to access the Docker server running in the Minikube virtual machine.

2. Run the **docker images** command to determine whether `openidm` and `opendj` Docker images are present in your test environment.

3. If the output from the **docker images** showed that `openidm` and `opendj` images were present in your environment, remove them.

   If you are not familiar with removing Docker images, run the **docker rmi --help** command for more information about command-line options. For more information about ForgeRock Docker image names, see Section 7.2, "Naming Docker Images".

   The following example commands remove images from the local Docker cache in a Minikube deployment:

   ```
   $ docker rmi -f forgerock/openidm:5.0.0
   Untagged: forgerock/openidm:5.0.0
   Deleted: sha256:7a3336f64975ee9f7b11ce77f8fa010545f05b10beb1b60e2dac306a68764ed3
   Deleted: sha256:1ce5401fe3f6dfb0650447c1b825c2fae86eaa0fe5c7fccf87e6a70aed1d571d
   . . .
   Deleted: sha256:59701800a35ab4d112539cf958d84a6d663b31ad495992c0ff3806259df93f5d
   Deleted: sha256:018353c2861979a296b60e975cb69b9f366397fe3ac30cd3fe629124c55fae8c

   $ docker rmi -f forgerock/opendj:4.0.0
   Untagged: forgerock/opendj:4.0.0
   Deleted: sha256:65f9f4f7374a43552c4a09f9828bde618aa22e3e504e97274ca691867c1c357b
   Deleted: sha256:cf7698333e0d64b25f25d270cb8facd8f8cc77c18e809580bb0978e9cb73aded
   . . .
   Deleted: sha256:deba2feeaea378fa7d35fc87778d3d58af287efeca288b630b4660fc9dc76435
   Deleted: sha256:dcbe724b0c75a5e75b28f23a3e1550e4b1201dc37ef5158d181fc6ab3ae83271
   ```

4. Run the **docker images** command to verify that you removed the `openidm` and `opendj` images.

4.4.3. Obtaining ForgeRock Software Binary Files

Perform the following procedure if:

- You have not yet obtained ForgeRock software binary files for the OpenIDM example.
- You want to obtain newer versions of ForgeRock software that versions you previously downloaded.

Skip this step if you want to build Docker images based on versions of ForgeRock software you previously downloaded and copied into the `docker` repository.

Procedure 4.5. To Obtain ForgeRock Binary Files

Perform the steps in the following procedure to obtain ForgeRock software for the OpenIDM example, and to copy it to the required locations for building the `openidm` and `opendj` Docker images:

1. Download the following binary files from the ForgeRock BackStage download site :
   - `IDM-5.0.0.zip`
   - `DS-5.0.0.zip`
2. Copy (or move) and rename the downloaded binary files as follows:

Table 4.2. Binary File Locations, OpenIDM Example

Binary File	Location
`IDM-5.0.0.zip`	`/path/to/docker/openidm/openidm.zip`
`DS-5.0.0.zip`	`/path/to/docker/opendj/opendj.zip`

4.4.4. Building Docker Images

Perform one of the following procedures to build the `openidm` and `opendj` Docker images:

- **Minikube users**, perform Procedure 4.6, "To Build Docker Images for Minikube".
- **GKE users**, perform Procedure 4.7, "To Build Docker Images for GKE".

Procedure 4.6. To Build Docker Images for Minikube

Minikube users only. GKE users should perform Procedure 4.7, "To Build Docker Images for GKE" instead.

Perform the following steps:

1. If you have not already done so, set up your shell to use the Docker environment within Minikube:

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

2. Change to the directory that contains the clone of the ForgeRock `docker` repository:

```
$ cd /path/to/docker
```

3. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a Minikube deployment:

```
$ ./build.sh -R forgerock -t 5.0.0 openidm
```

   This command builds a Docker image with the repository name `forgerock/openidm`, tags the image with `5.0.0`, and writes the image in the local Docker cache.

4. Build the `openidm` and `opendj` images using the **build.sh** script:

   a. Build the `openidm` image:

```
$ ./build.sh -R forgerock -t 5.0.0 openidm
Building openidm
Sending build context to Docker daemon 84.19 MB
Step 1 : FROM openjdk:8-jre-alpine
 ---> c017141bdaa8
Step 2 : WORKDIR /opt
 ---> Running in 938adb90c271
 ---> c1b0f5bd064b
Removing intermediate container 938adb90c271
Step 3 : EXPOSE 8080
 ---> Running in 788584c198e8
 ---> 26045be5a073
Removing intermediate container 788584c198e8
Step 4 : ADD Dockerfile /
 ---> 5a2e45827c15
Removing intermediate container a39c87858ff3
Step 5 : ENV JAVA_OPTS -Xmx1024m -server -XX:+UseG1GC
 ---> Running in 776b1d915dbc
 ---> b8a0d492a2ad
Removing intermediate container 776b1d915dbc
Step 6 : ADD openidm.zip /var/tmp/openidm.zip
 ---> 739cefe51d40
Removing intermediate container 2a16b3ea506f
Step 7 : RUN apk add --no-cache su-exec libc6-compat postgresql-client &&    adduser -D -h  /opt
/openidm openidm openidm &&    unzip -q /var/tmp/openidm.zip &&    chown -R openidm:openidm /opt/
openidm &&    rm -f /var/tmp/openidm.zip  && rm -fr /opt/openidm/samples
 ---> Running in 66c7f3dccf30
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/community/x86_64/APKINDEX.tar.gz
(1/11) Installing libc6-compat (1.1.15-r6)
(2/11) Installing ncurses-terminfo-base (6.0-r7)
(3/11) Installing ncurses-terminfo (6.0-r7)
(4/11) Installing ncurses-libs (6.0-r7)
(5/11) Installing libedit (20150325.3.1-r3)
(6/11) Installing db (5.3.28-r0)
(7/11) Installing libsasl (2.1.26-r8)
(8/11) Installing libldap (2.4.44-r3)
(9/11) Installing libpq (9.6.2-r0)
```

```
(10/11) Installing postgresql-client (9.6.2-r0)
(11/11) Installing su-exec (0.2-r0)
Executing busybox-1.25.1-r0.trigger
OK: 90 MiB in 60 packages
 ---> 49cf81a1769c
Removing intermediate container 66c7f3dccf30
Step 8 : ADD docker-entrypoint.sh /opt/openidm/docker-entrypoint.sh
 ---> 94d8bf502cbe
Removing intermediate container 6220dd662290
Step 9 : ADD logging.properties /opt/openidm/logging.properties
 ---> bfef94e170d7
Removing intermediate container 15cb01ed4027
Step 10 : WORKDIR /opt/openidm
 ---> Running in 990af608c48e
 ---> a6a1c057fe2d
Removing intermediate container 990af608c48e
Step 11 : ENTRYPOINT /opt/openidm/docker-entrypoint.sh
 ---> Running in 35bea01a0f8d
 ---> 8f5917ce6724
Removing intermediate container 35bea01a0f8d
Step 12 : CMD openidm
 ---> Running in 88d9063b1713
 ---> ec99066fef12
Removing intermediate container 88d9063b1713
Successfully built ec99066fef12
```

b. Build the `opendj` image:

```
$ ./build.sh -R forgerock -t 4.0.0 opendj
Building opendj
Sending build context to Docker daemon 36.72 MB
Step 1 : FROM openjdk:8-jre
 ---> a4d689e63201
Step 2 : WORKDIR /opt
 ---> Running in 7d1398a2d999
 ---> 24ade2fe377d
Removing intermediate container 7d1398a2d999
Step 3 : ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/
 ---> Running in f5334ed37403
 ---> a01f00196aac
Removing intermediate container f5334ed37403
Step 4 : ENV OPENDJ_JAVA_ARGS -server -Xmx512m -XX:+UseG1GC
 ---> Running in ba1e2d9df634
 ---> 123a79b7f06a
Removing intermediate container ba1e2d9df634
Step 5 : ENV DIR_MANAGER_PW_FILE /var/secrets/opendj/dirmanager.pw
 ---> Running in 25a827aa5814
 ---> 259b706465c5
Removing intermediate container 25a827aa5814
Step 6 : ENV BASE_DN dc=example,dc=com
 ---> Running in 0a8f8d69dc44
 ---> 29fac8af3b4a
Removing intermediate container 0a8f8d69dc44
Step 7 : ADD opendj.zip /tmp/
 ---> 0cc8e3168dd7
Removing intermediate container 4ddb526cd284
Step 8 : RUN apt-get update && apt-get install -y ldap-utils &&     unzip -q /tmp/opendj.zip
-d /opt  && rm /tmp/opendj.zip &&     echo "/opt/opendj/data" > /opt/opendj/instance.loc  &&
```

```
   mkdir -p /opt/opendj/data/lib/extensions &&     mkdir -p /var/secrets/opendj &&     echo -n
 "password"  > ${DIR_MANAGER_PW_FILE}
 ---> Running in 2223add79916
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://deb.debian.org jessie InRelease
Get:2 http://security.debian.org jessie/updates/main amd64 Packages [461 kB]
Get:3 http://deb.debian.org jessie-updates InRelease [145 kB]
Get:4 http://deb.debian.org jessie-backports InRelease [166 kB]
Get:5 http://deb.debian.org jessie Release.gpg [2373 B]
Get:6 http://deb.debian.org jessie-updates/main amd64 Packages [17.6 kB]
Get:7 http://deb.debian.org jessie Release [148 kB]
Get:8 http://deb.debian.org jessie-backports/main amd64 Packages [1119 kB]
Get:9 http://deb.debian.org jessie/main amd64 Packages [9049 kB]
Fetched 11.2 MB in 3s (3360 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following extra packages will be installed:
  libsasl2-modules
Suggested packages:
  libsasl2-modules-gssapi-mit libsasl2-modules-gssapi-heimdal
  libsasl2-modules-otp libsasl2-modules-ldap libsasl2-modules-sql
The following NEW packages will be installed:
  ldap-utils libsasl2-modules
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 289 kB of archives.
After this operation, 943 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian/ jessie/main ldap-utils amd64 2.4.40+dfsg-1+deb8u2 [188 kB]
Get:2 http://deb.debian.org/debian/ jessie/main libsasl2-modules amd64 2.1.26.dfsg1-13+deb8u1 [101
 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 289 kB in 0s (1590 kB/s)
Selecting previously unselected package ldap-utils.
(Reading database ... 9186 files and directories currently installed.)
Preparing to unpack .../ldap-utils_2.4.40+dfsg-1+deb8u2_amd64.deb ...
Unpacking ldap-utils (2.4.40+dfsg-1+deb8u2) ...
Selecting previously unselected package libsasl2-modules:amd64.
Preparing to unpack .../libsasl2-modules_2.1.26.dfsg1-13+deb8u1_amd64.deb ...
Unpacking libsasl2-modules:amd64 (2.1.26.dfsg1-13+deb8u1) ...
Setting up ldap-utils (2.4.40+dfsg-1+deb8u2) ...
Setting up libsasl2-modules:amd64 (2.1.26.dfsg1-13+deb8u1) ...
 ---> 877dcc588f6a
Removing intermediate container 2223add79916
Step 9 : WORKDIR /opt/opendj
 ---> Running in 94176cfe7d1c
 ---> 59c01ff28034
Removing intermediate container 94176cfe7d1c
Step 10 : ADD Dockerfile /
 ---> d3b991f735cc
Removing intermediate container ad1b5f084d3d
Step 11 : ADD bootstrap/ /opt/opendj/bootstrap/
 ---> edea71955062
Removing intermediate container d9146b09a871
Step 12 : ADD *.sh /opt/opendj/
 ---> cb5e0b2eea42
Removing intermediate container 7a62c73a2686
Step 13 : EXPOSE 389 636 4444 8989
 ---> Running in 3f468d048ca0
```

```
 ---> 55496df498ca
Removing intermediate container 3f468d048ca0
Step 14 : ADD run.sh /opt/opendj/run.sh
 ---> 9804d24a5d37
Removing intermediate container 00e488840c69
Step 15 : CMD /opt/opendj/run.sh
 ---> Running in bf1380d6557b
 ---> 01ebb522a71a
Removing intermediate container bf1380d6557b
Successfully built 01ebb522a71a
```

5. Run the **docker images** command to verify that the `openidm` and `opendj` images are now available.

Procedure 4.7. To Build Docker Images for GKE

GKE users only. Minikube users should perform Procedure 4.6, "To Build Docker Images for Minikube" instead.

Perform the following steps:

1. Change to the directory that contains the clone of the ForgeRock `docker` repository:

   ```
   $ cd /path/to/docker
   ```

2. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a GKE deployment:

   ```
   $ ./build.sh -r gcr.io -R myProject -t 5.0.0 -g openidm
   ```

   This command builds a Docker image for deployment on GKE with the repository name `myProject/openidm`, tags the image with `5.0.0`, and pushes the image to the `gcr.io` registry. Note that for Docker images deployed on GKE, the first part of the repository component of the image name *must* be your Google Cloud Platform project name.

3. Build the `openidm` and `opendj` images using the **build.sh** script:

   a. Build the `openidm` image. For example:

   ```
   $ ./build.sh -R gcr.io -R myProject -t 5.0.0 -g openidm
   Building openidm
   . . .
   ```

   b. Build the `opendj` image. For example:

   ```
   $ ./build.sh -R gcr.io -R myProject -t 4.0.0 -g opendj
   Building opendj
   . . .
   ```

4. Run the **docker images** command to verify that the `openidm` and `opendj` images are now available.

4.5. Orchestrating the Deployment

This section covers how to orchestrate the Docker containers for this deployment example into your Kubernetes environment.

4.5.1. Specifying Deployment Options

Kubernetes options specified in the `/path/to/fretes/helm/custom.yaml` file override default options specified in Helm charts in the reference deployment.

Before deploying this example, you must edit this file and specify options pertinent to your deployment.

For information about specifying deployment options in the `custom.yaml` file, see Section 7.4, "Specifying Deployment Options in the custom.yaml File". For a commented example, see the `/path/to/fretes/helm/custom-template.yaml` file in the `fretes` repository.

4.5.1.1. custom.yaml File Examples

This section provides several examples of `custom.yaml` files that could be used with the OpenIDM DevOps example.

Example 4.1. Minikube Deployment, Configuration Read From Flat Files

The following is an example of a `custom.yaml` file for a deployment on Minikube in which the OpenIDM configuration is read from flat files:

```
cookieDomain: .example.com
registry: ""
stackConfigSource:
  hostPath:
    path: /path/to/config
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys Docker images from the local cache.
- The OpenIDM configuration is read from the `/path/to/config/forgeops-init/openidm` directory.
- After deployment, the Kubernetes ingress uses the `cookieDomain` value, `example.com`, as the domain portion of the fully-qualified domain name (FQDN) to which it routes requests: `openidm.example.com`. Despite the property's name, the deployment does not use host cookies.

Example 4.2. GKE Deployment, Configuration Read from a Git Repository

The following is an example of a `custom.yaml` file for a deployment on GKE in which the OpenIDM configuration is read from a Git repository:

```
cookieDomain: .example.com
registry: gcr.io/
repo: engineering-devops
stackConfigSource:
  gitRepo:
    repository: https://stash.forgerock.org/scm/cloud/forgeops-init.git
    revision: HEAD
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys Docker images from the `engineering-devops` repository in the `gcr.io` Docker registry.
- The OpenIDM configuration is read from the `forgeops-init/openidm` directory of the `HEAD` revision of the `https://stash.forgerock.org/scm/cloud/forgeops-init.git` Git repository.
- After deployment, the Kubernetes ingress uses the `cookieDomain` value, `example.com`, as the domain portion of the FQDN to which it routes requests: `openidm.example.com`. Despite the property's name, the deployment does not use host cookies.

4.5.2. Deploying Helm Charts

Perform the steps in the following procedure to deploy the Helm charts for the OpenIDM DevOps example to the Kubernetes cluster in your environment:

Procedure 4.8. To Deploy Helm Charts for the OpenIDM Example

1. Change to the `/path/to/fretes/helm/bin` directory.
2. Run the **openidm.sh** script, which deploys a PostgreSQL database and an OpenDJ instance, and then brings up OpenIDM:

   ```
   $ ./openidm.sh
   ```

   Output similar to the following appears in the terminal window:

   ```
   NAME:   postgres
   LAST DEPLOYED: Wed Mar 29 10:21:50 2017
   NAMESPACE: default
   STATUS: DEPLOYED

   RESOURCES:
   ==> v1/Service
   NAME        CLUSTER-IP  EXTERNAL-IP  PORT(S)   AGE
   postgresql  10.0.0.76   <none>       5432/TCP  1s

   ==> extensions/v1beta1/Deployment
   NAME              DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
   openidm-postgres  1        1        1           0          0s

   ==> v1/Secret
   NAME                      TYPE    DATA  AGE
   postgres-postgres-openid  Opaque  1     1s
   ```

```
==> v1/ConfigMap
NAME         DATA  AGE
openidm-sql  5     1s

==> v1/PersistentVolumeClaim
NAME                       STATUS  VOLUME                                     CAPACITY  ACCESSMODES  AGE
postgres-postgres-openid   Bound   pvc-3253cbc2-14a4-11e7-aa8a-0800278581ae   8Gi       RWO          1s


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

To get your user password run:

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

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

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


Starting an OpenDJ user store instance
Configuring instance userstore
NAME:   userstore
LAST DEPLOYED: Wed Mar 29 10:21:52 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME       TYPE    DATA  AGE
userstore  Opaque  1     1s

==> v1/Service
NAME       CLUSTER-IP  EXTERNAL-IP  PORT(S)           AGE
userstore  None        <none>       389/TCP,4444/TCP  1s

==> apps/v1beta1/StatefulSet
NAME       DESIRED  CURRENT  AGE
userstore  1        1        1s


Waiting for pod userstore-0 to be
 ready
...done
Waiting for Postgres to start
Waiting for pod openidm-postgres-2803803828-h6t3c to be
 ready
.Starting OpenIDM
NAME:   openidm
LAST DEPLOYED: Wed Mar 29 10:22:34 2017
NAMESPACE: default
STATUS: DEPLOYED
```

```
RESOURCES:
==> v1/Service
NAME              CLUSTER-IP  EXTERNAL-IP  PORT(S)       AGE
openidm-openidm   10.0.0.182  <nodes>      80:30629/TCP  1s

==> extensions/v1beta1/Deployment
NAME     DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
openidm  1        1        1           0          1s

==> extensions/v1beta1/Ingress
NAME             HOSTS                ADDRESS         PORTS  AGE
openidm-ingress  openidm.example.com  192.168.99.100  80     1s

==> v1/Secret
NAME             TYPE    DATA  AGE
openidm-secrets  Opaque  2     1s


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

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

You can also connect using Kubernetes services:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services
 openidm-openidm)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0]
.address}")
  echo http://$NODE_IP:$NODE_PORT/#login/


If you are running Minikube you can also use:

minikube service  openidm-openidm
```

3. Add an entry similar to the following to your `/etc/hosts` file to enable access to the cluster through the ingress:

   ```
   104.154.46.166 openidm.example.com
   ```

   In this example, `104.154.46.166` is the IP address returned from the **kubectl get ingresses** command.

4.5.3. Verifying the Deployment

After you have deployed the Helm charts for the example, verify that the deployment is active and available by accessing the OpenIDM Admin UI:

Procedure 4.9. To Verify the Deployment

1. In a web browser, navigate to the OpenIDM Admin UI's deployment URL, for example, `http://openidm.example.com/admin`.

The Kubernetes ingress handles the request and routes you to a running OpenIDM instance.

2. Log in to OpenIDM as the `openidm-admin` user with password `openidm-admin`.

4.6. Modifying and Saving the OpenIDM Configuration

Important

Configuration management capabilities described in this section are available only for deployments running on Minikube for which the `stackConfigSource` option in the `custom.yaml` file is set to `hostPath`.

You can use these configuration management techniques as you develop the OpenIDM configuration, and then import the configuration to a production environment running on GKE.

After you have successfully orchestrated an OpenIDM deployment as described in this chapter, you can modify the OpenIDM configuration, save it, and use the revised configuration to initialize a subsequent OpenIDM deployment.

Storing the configuration in a version control system like a Git repository lets you take advantage of capabilities such as version control, difference analysis, and branches when managing the OpenIDM configuration. Configuration management enables migration from a development environment to a test environment and then to a production environment. Deployment migration is one of the primary objectives of DevOps techniques.

To modify the OpenIDM configuration, use one of the OpenIDM management tools:

- The OpenIDM Admin UI
- The OpenIDM REST API

Once you are ready to save your configuration, perform the following procedure:

Procedure 4.10. To Save the OpenIDM Configuration

1. Verify that when you deployed the OpenIDM example, the value of the `stackConfigSource` option in the `custom.yaml` file was set to `hostPath`.
2. If you are managing the OpenIDM configuration using Git, commit changed files and add new files to the repository.

Saving the updated configuration is complete. You can redeploy the OpenIDM example using the updated configuration at any time.

Chapter 5

Deploying the OpenIG Example

This chapter provides instructions for deploying the reference implementation of the OpenIG DevOps example.

The following is a high-level overview of the steps to deploy this example:

- Familiarize yourself with the example deployment. See the deployment diagram and explanation in Section 5.1, "About the Example".
- Prepare an environment for running the example, including verifying that you have a supported environment, removing objects from previous deployments from the environment, and deploying a Kubernetes ingress (load balancer). See Section 5.3, "Preparing the Environment".
- Download ForgeRock software and create a Docker image for OpenIG. See Section 5.4, "Creating the Docker Image".
- Orchestrate the example in a Kubernetes cluster and verify the deployment. See Section 5.5, "Orchestrating the Deployment".

5.1. About the Example

The reference deployment of the OpenIG DevOps example has the following architectural characteristics:

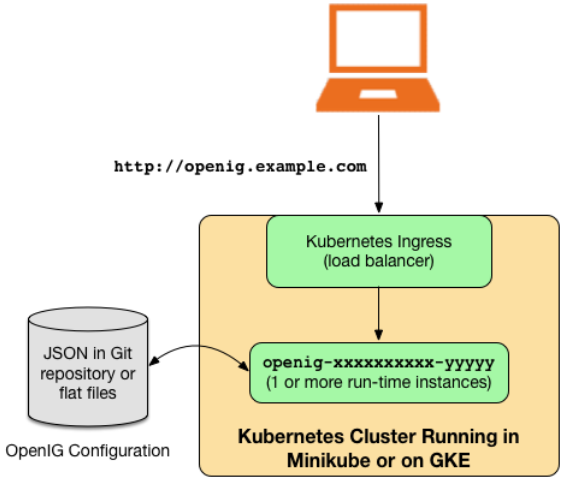
- **Kubernetes ingress**. From outside the deployment, OpenIG is accessed through a Kubernetes ingress (load balancer).
- **Run-time `openig-xxxxxxxxxx-yyyyy` pod(s)**. This pod, created elastically by Kubernetes[1], runs the OpenIG server. Multiple instances can be started if required. The Kubernetes ingress redirects requests to one of these pods.
- **JSON configuration stored in a Git repository or flat files**. After installation, the deployment starts OpenIG, referencing configuration stored in JSON files that it obtains from either flat files or a Git repository. The OpenIG configuration is accessible to, but not inside of the Kubernetes cluster. Because configuration is stored outside of the cluster, it can persist if the cluster is deleted.

  If you use flat files for the OpenIG JSON configuration, the files are updated if you modify the configuration while the server is running.

[1] Pods created statically can have fixed names. Run-time pods created elastically by Kubernetes have variable names.

The following diagram illustrates the example.

Figure 5.1. OpenIG DevOps Deployment Example



5.2. Working With the OpenIG Example

This section presents an example workflow to set up a development environment in which you configure OpenIG, iteratively modify and save the OpenIG configuration, and then migrate the configuration to a test or production environment.

This workflow illustrates many of the capabilities available in the DevOps Examples. It is only one way of working with the example deployment. Use this workflow to help you better understand the DevOps Examples, and as a starting point for your own DevOps deployments.

Note that this workflow is an overview of how you might work with the DevOps Examples and does not provide step-by-step instructions. It does provide links to subsequent sections in this chapter that include detailed procedures you can follow when deploying the DevOps Examples.

Table 5.1. Example Workflow, OpenIG DevOps Deployment

Step	Details
Implement a Minikube environment	Set up a Minikube environment for developing the OpenIG configuration. See Section 2.1, "Kubernetes Running on a Minikube Virtual Machine".
Get up-to-date versions of required Git repositories	Make sure you have up-to-date versions of the `release/5.0.0` branch of the Git repositories that contain Docker and Kubernetes artifacts and initial configuration files. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".

Step	Details
Deploy the OpenIG example in Minikube	Follow the procedures in Section 5.3, "Preparing the Environment", Section 5.4, "Creating the Docker Image", and Section 5.5, "Orchestrating the Deployment". Specify the following value in the `custom.yaml` file described in Section 5.5.1, "Specifying Deployment Options": • `/path/containing/forgeops-init-clone` The `hostPath` value *must* have a subdirectory named `forgeops-init`. The `forgeops-init` repository initializes OpenIG with configuration for the following routes: • A sample OpenAM policy agent (`01-pep.json`) • A simple throttling example (`20-simplethrottle.json`) Remove the configuration for these routes if you do not want them as part of your OpenIG configuration.
Modify and save the OpenIG configuration	Iterate through the following steps as many times as you need to: • Modify the OpenIG configuration by using OpenIG Studio or by manually editing JSON files in the `hostPath` directory. See Section 5.5.3, "Verifying the Deployment" for details about how to access the deployed OpenIG server if you want to use OpenIG Studio. • Save the OpenIG configuration as described in Section 5.6, "Modifying and Saving the OpenIG Configuration". • Manage the OpenIG configuration in a Git repository. When you move to a test or production deployment, you will use the configuration stored in the Git repository to initialize OpenIG.
Implement a GKE environment	Set up a GKE environment to use for test and production deployments. See Section 2.2, "Kubernetes Running on Google Container Engine".
Deploy the OpenIG example in GKE	Follow the procedures in Section 5.3, "Preparing the Environment", Section 5.4, "Creating the Docker Image", and Section 5.5, "Orchestrating the Deployment". Be sure to specify the following values in the `custom.yaml` file described in Section 5.5.1, "Specifying Deployment Options": • `registry: gcr.io/` • `repo: myGKEProject` • `stackConfigSource: gitRepo: repository: myConfigGitRepo` • `stackConfigSource: gitRepo: revision: myConfigRevision`

Step	Details
	For `myConfigGitRepo`, specify the Git repository in which you saved the OpenIG configuration while you were developing it. For `myConfigRevision`, specify the Git revision that contains the desired version of the OpenIG configuration.

After you have deployed a test or production OpenIG server, you can continue to update the OpenIG configuration in your development environment, and then redeploy OpenIG with the updated configuration. Reiterate the development/deployment cycle as follows:

- Modify the OpenIG configuration on the Minikube deployment and commit the changes in a Git repository.
- Redeploy the OpenIG example in GKE based on the updated configuration, specifying the desired revision in the `stackConfigSource: gitRepo: revision:` property in the `custom.yaml` file.

5.3. Preparing the Environment

The OpenIG DevOps example can be run in the Minikube and GKE environments.

Before deploying the example, be sure you have a working environment as described in one of the following sections:

- Section 2.1, "Kubernetes Running on a Minikube Virtual Machine"
- Section 2.2, "Kubernetes Running on Google Container Engine"

Most of the steps for deploying the example are identical for the two test environments. Environment-specific differences are called out in the deployment procedures in this chapter.

To prepare your environment:

- Make sure you have up-to-date versions of the `release/5.0.0` branch of the `docker`, `fretes`, and `forgeops-init` repositories. For more information, see Section 7.1, "Git Repositories Used by the DevOps Examples".
- Remove any objects left over from previous deployments to ensure you are deploying the example in a clean environment. See Section 5.3.1, "Removing Existing Deployment Objects".
- Deploy an ingress in your environment. See Section 5.3.2, "Deploying a Kubernetes Ingress".

5.3.1. Removing Existing Deployment Objects

Before deploying the example, ensure that any objects remaining from previous deployments have been removed from your environment. Perform the following procedure:

Procedure 5.1. To Remove Existing Deployment Artifacts

1. Verify that Helm is running in your environment:

```
$ kubectl get pods --all-namespaces | grep tiller-deploy
kube-system   tiller-deploy-2779452559-3bznh              1/1       Running   1          13d
```

If the **kubectl** command returns no output, restart Helm by running the **helm init** command.

Note that the **helm init** command starts a Kubernetes pod with a name starting with `tiller-deploy`.

2. Run the **remove-all.sh** script to remove any Kubernetes objects left over from previous ForgeRock deployments:

```
$ cd /path/to/fretes/helm/bin
$ ./remove-all.sh
```

Output from the **remove-all.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. `Error: release: not found` messages *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the **kubectl get pods** command to verify that no pods that run ForgeRock software [2] in the `default` namespace are active in your test environment.

If Kubernetes pods running ForgeRock software are still active, wait several seconds, and then run the **kubectl get pods** command again. You might need to run the command several times before all the pods running ForgeRock software are terminated.

If all the pods in the cluster were running ForgeRock software, the procedure is complete when the `No resources found` message appears:

```
$ kubectl get pods
No resources found.
```

If some pods in the cluster were running non-ForgeRock software, the procedure is complete when only pods running non-ForgeRock software appear in response to the **kubectl get pods** command. For example:

```
$ kubectl get pods
hello-minikube-55824521-b0qmb   1/1       Running   0          2m
```

5.3.2. Deploying a Kubernetes Ingress

The OpenIG DevOps example is accessed through a Kubernetes ingress.

Ingress deployment differs on Minikube and GKE environments. Perform one of the following procedures, depending on your environment:

- Procedure 5.2, "To Deploy and Access an Ingress on Minikube"

[2] See the deployment diagrams in the introductory sections for each DevOps example for the names of pods that run ForgeRock software. For example, see Section 5.1, "About the Example" for the names of pods deployed for the OpenIG example.

- Procedure 5.3, "To Deploy an Ingress on GKE"

Procedure 5.2. To Deploy and Access an Ingress on Minikube

Minikube users only. GKE users should perform Procedure 5.3, "To Deploy an Ingress on GKE" instead.

- Enable the default ingress built into Minikube:

```
$ minikube addons enable ingress
ingress was successfully enabled
```

Procedure 5.3. To Deploy an Ingress on GKE

GKE users only. Minikube users should perform Procedure 5.2, "To Deploy and Access an Ingress on Minikube" instead.

The GKE deployment uses the nginx ingress controller, which is suitable for development and testing. When deploying in production, use the Google Load Balancer ingress controller. Refer to the GKE documentation for more information.

Perform the following steps to deploy the nginx ingress controller:

1. Change to the directory containing Kubernetes manifests to deploy an nginx ingress:

```
$ cd /path/to/fretes/ingress
```

2. Run the `delete-ingress.sh` script to remove a leftover ingress deployment, if it exists:

```
$ ./delete-ingress.sh
```

   Output from the **delete-ingress.sh** script varies, depending on what was deployed to the Kubernetes cluster before the command ran. Error messages indicating that components were not found *do not* indicate actual errors—they simply indicate that the script attempted to delete Kubernetes objects that did not exist in the cluster.

3. Run the `create-nginx-ingress.sh` script to deploy an nginx ingress:

```
$ ./create-nginx-ingress.sh
deployment "default-http-backend" created
service "default-http-backend" created
configmap "nginx-load-balancer-conf" created
configmap "tcp-configmap" created
deployment "nginx-ingress-controller" created
```

5.4. Creating the Docker Image

This section covers how to work with the Docker image needed to deploy the OpenIG example:

- Review when the Docker image needed to run the example needs to be created, removed, and rebuilt. See Section 5.4.1, "About the Docker Image for the Example".

- Remove an existing Docker image from a registry or from Docker cache. See Section 5.4.2, "Removing an Existing Docker Image".
- Download the ForgeRock software binary file and copy it to the `docker` repository. See Section 5.4.3, "Obtaining ForgeRock Software Binary Files".
- Build the Docker image based on the reference Dockerfile provided in the `docker` repository. See Section 5.4.4, "Building the Docker Image".

Note

If you need to customize the Docker image, refer to the `README.md` files and the Dockerfile comments in the `docker` repository.

5.4.1. About the Docker Image for the Example

The example requires a Docker image for OpenIG.

Once created, a Docker image's contents are static. Remove and rebuild the image when:

- You want to update it to use a newer version of OpenIG software.
- You changed files that impact image content, and you want to redeploy a modified image. Common modifications include (but are not limited to) the following:
  - Changes to security files, such as passwords and keystores.
  - Dockerfile changes to install additional software on base images.

5.4.2. Removing an Existing Docker Image

If the `openig` image is present in your environment, remove it before creating a new image.

Perform the following procedure to remove an existing Docker image from your environment:

Procedure 5.4. To Remove an Existing Docker Image

Because Docker image names can vary depending on organizations' requirements, the image names shown in the example commands in this procedure might not match your image names. For information about the naming conventions used for Docker images in the DevOps Examples, see Section 7.2, "Naming Docker Images".

1. **Minikube users only.** Set up your shell to use the Docker environment within Minikube:

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

   This command sets environment variables that enable the Docker client running on your laptop to access the Docker server running in the Minikube virtual machine.

2. Run the **docker images** command to determine whether the `openig` Docker image is present in your test environment.

3. If the output from the **docker images** showed that the `openig` image was present in your environment, remove it.

   If you are not familiar with removing Docker images, run the **docker rmi --help** command for more information about command-line options. For more information about ForgeRock Docker image names, see Section 7.2, "Naming Docker Images".

   The following example command removes an image from the local Docker cache in a Minikube deployment:

```
$ docker rmi -f forgerock/openig:5.0.0
Untagged: forgerock/openig:5.0.0
Deleted: sha256:7a3336f64975ee9f7b11ce77f8fa010545f05b10beb1b60e2dac306a68764ed3
Deleted: sha256:1ce5401fe3f6dfb0650447c1b825c2fae86eaa0fe5c7fccf87e6a70aed1d571d
. . .
Deleted: sha256:59701800a35ab4d112539cf958d84a6d663b31ad495992c0ff3806259df93f5d
Deleted: sha256:018353c2861979a296b60e975cb69b9f366397fe3ac30cd3fe629124c55fae8c
```

4. Run the **docker images** command to verify that you removed the `openig` image.

5.4.3. Obtaining ForgeRock Software Binary Files

Perform the following procedure if:

- You have not yet obtained the ForgeRock software binary file for the OpenIG example.
- You want to obtain a newer version of ForgeRock software than the version you previously downloaded.

Skip this step if you want to build a Docker image based on a version of ForgeRock software you previously downloaded and copied into the `docker` repository.

Procedure 5.5. To Obtain the ForgeRock Binary File

Perform the steps in the following procedure to obtain ForgeRock software for the OpenIG example, and to copy it to the required location for building the `openig` Docker image:

1. Download the `IG-5.0.0.war` binary file from the ForgeRock BackStage download site .

2. Copy (or move) and rename the downloaded binary file as follows:

Table 5.2. Binary File Locations, OpenIG Example

Binary File	Location
`IG-5.0.0.war`	`/path/to/docker/openig/openig.war`

5.4.4. Building the Docker Image

Perform one of the following procedures to build the `openig` Docker image:

- **Minikube users**, perform Procedure 5.6, "To Build the Docker Image for Minikube".
- **GKE users**, perform Procedure 5.7, "To Build the Docker Image for GKE".

Procedure 5.6. To Build the Docker Image for Minikube

Minikube users only. GKE users should perform Procedure 5.7, "To Build the Docker Image for GKE" instead.

Perform the following steps:

1. If you have not already done so, set up your shell to use the Docker environment within Minikube:

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

2. Change to the directory that contains the clone of the ForgeRock `docker` repository:

   ```
   $ cd /path/to/docker
   ```

3. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a Minikube deployment:

   ```
   $ ./build.sh -R forgerock -t 5.0.0 openig
   ```

   This command builds a Docker image with the repository name `forgerock/openig`, tags the image with `5.0.0`, and writes the image in the local Docker cache.

4. Build the `openig` image using the **build.sh** script:

   ```
   $ ./build.sh -R forgerock -t 5.0.0 openig
   Building openig
   Sending build context to Docker daemon 41.55 MB
   Step 1 : FROM tomcat:8.5-jre8
    ---> 7f855aeeaebf
   Step 2 : ENV CATALINA_HOME /usr/local/tomcat
    ---> Running in 625fe328ee48
    ---> 6e5a2b35013d
   Removing intermediate container 625fe328ee48
   Step 3 : ENV PATH $CATALINA_HOME/bin:$PATH
    ---> Running in 8559f6cd3a05
    ---> 53b864f03ebf
   Removing intermediate container 8559f6cd3a05
   Step 4 : ENV OPENIG_BASE /var/openig
    ---> Running in 90805760cb4e
    ---> 3e50016cd8bd
   Removing intermediate container 90805760cb4e
   Step 5 : WORKDIR $CATALINA_HOME
    ---> Running in 0608844ca6ce
   ```

```
 ---> 518df390c41d
Removing intermediate container 0608844ca6ce
Step 6 : RUN rm -fr webapps/*
 ---> Running in 5db9174643d8
 ---> 39de45b0839c
Removing intermediate container 5db9174643d8
Step 7 : EXPOSE 8080 8443
 ---> Running in cae5d59ee13c
 ---> 896c0dd96949
Removing intermediate container cae5d59ee13c
Step 8 : ADD openig.war /tmp/openig.war
 ---> ed108c64457b
Removing intermediate container 093e28c8b9bd
Step 9 : RUN unzip -q /tmp/openig.war -d /usr/local/tomcat/webapps/ROOT && rm -f /tmp/openig.war
 ---> Running in 9ecc683f7c91
 ---> 0964dbf05b03
Removing intermediate container 9ecc683f7c91
Step 10 : ADD sample-config/* /var/openig/config/
 ---> 9423ced73820
Removing intermediate container 1dd329a8cea0
Successfully built 9423ced73820
```

5. Run the **docker images** command to verify that the `openig` image is now available.

Procedure 5.7. To Build the Docker Image for GKE

GKE users only. Minikube users should perform Procedure 5.6, "To Build the Docker Image for Minikube" instead.

Perform the following steps:

1. Change to the directory that contains the clone of the ForgeRock `docker` repository:

   ```
   $ cd /path/to/docker
   ```

2. To prepare for building Docker images, review the **build.sh** command options described in Section 7.3, "Using the build.sh Script to Create Docker Images" and determine which options to specify for your deployment.

   For example, the following is a typical **build.sh** command for a GKE deployment:

   ```
   $ ./build.sh -r gcr.io -R myProject -t 5.0.0 -g openig
   ```

   This command builds a Docker image for deployment on GKE with the repository name `myProject/openig`, tags the image with `5.0.0`, and pushes the image to the `gcr.io` registry. Note that for Docker images deployed on GKE, the first part of the repository component of the image name *must* be your Google Cloud Platform project name.

3. Build the `openig` image using the **build.sh** script:

   ```
   $ ./build.sh -R gcr.io -R myProject -t 5.0.0 -g openig
   Building openig
   . . .
   ```

4. Run the **docker images** command to verify that the `openig` image is now available.

5.5. Orchestrating the Deployment

This section covers how to orchestrate the Docker containers for this deployment example into your Kubernetes environment.

5.5.1. Specifying Deployment Options

Kubernetes options specified in the `/path/to/fretes/helm/custom.yaml` file override default options specified in Helm charts in the reference deployment.

Before deploying this example, you must edit this file and specify options pertinent to your deployment.

For information about specifying deployment options in the `custom.yaml` file, see Section 7.4, "Specifying Deployment Options in the custom.yaml File". For a commented example, see the `/path/to/fretes/helm/custom-template.yaml` file in the `fretes` repository.

5.5.1.1. custom.yaml File Examples

This section provides several examples of `custom.yaml` files that could be used with the OpenIG DevOps example.

Example 5.1. Minikube Deployment, Configuration Read From Flat Files

The following is an example of a `custom.yaml` file for a deployment on Minikube in which the OpenIG configuration is read from flat files:

```
cookieDomain: .example.com
registry: ""
stackConfigSource:
  hostPath:
    path: /path/to/config
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys the Docker image from the local cache.
- The OpenIG configuration is read from the `/path/to/config/forgeops-init/openig` directory.
- After deployment, the Kubernetes ingress uses the `cookieDomain` value, `example.com`, as the domain portion of the fully-qualified domain name (FQDN) to which it routes requests: `openig.example.com`. Despite the property's name, the deployment does not use host cookies.

Example 5.2. GKE Deployment, Configuration Read from a Git Repository

The following is an example of a `custom.yaml` file for a deployment on GKE in which the OpenIG configuration is read from a Git repository:

```
cookieDomain: .example.com
registry: gcr.io/
repo: engineering-devops
stackConfigSource:
  gitRepo:
    repository: https://stash.forgerock.org/scm/cloud/forgeops-init.git
    revision: HEAD
```

As a result of the options specified in the preceding `custom.yaml`:

- Kubernetes deploys the Docker image from the `engineering-devops` repository in the `gcr.io` Docker registry.
- The OpenIG configuration is read from the `forgeops-init/openig` directory of the `HEAD` revision of the `https://stash.forgerock.org/scm/cloud/forgeops-init.git` Git repository.
- After deployment, the Kubernetes ingress uses the `cookieDomain` value, `example.com`, as the domain portion of the FQDN to which it routes requests: `openig.example.com`. Despite the property's name, the deployment does not use host cookies.

5.5.2. Deploying Helm Charts

Perform the steps in the following procedure to deploy the Helm charts for the OpenIG DevOps example to the Kubernetes cluster in your environment:

Procedure 5.8. To Deploy Helm Charts for the OpenIG Example

1. Change to the `/path/to/fretes/helm/bin` directory.
2. Run the **openig.sh** script, which deploys an OpenIG server and then brings it up:

   ```
   $ ./openig.sh
   ```

   Output similar to the following appears in the terminal window:

```
NAME:   openig
LAST DEPLOYED: Wed Mar 29 10:28:11 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
NAME           CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
openig-openig  10.0.0.177  <none>       80/TCP   0s

==> extensions/v1beta1/Deployment
NAME           DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
openig-openig  1        1        1           0          0s

==> extensions/v1beta1/Ingress
NAME            HOSTS               ADDRESS         PORTS  AGE
openig-ingress  openig.example.com  192.168.99.100  80     0s


NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=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
```

3. Add an entry similar to the following to your `/etc/hosts` file to enable access to the cluster through the ingress:

   ```
   104.154.46.166 openig.example.com
   ```

   In this example, `104.154.46.166` is the IP address returned from the **kubectl get ingresses** command.

5.5.3. Verifying the Deployment

After you have deployed the Helm charts for the example, verify that the deployment is active and available by connecting to the OpenIG server:

Procedure 5.9. To Verify the Deployment

1. In a web browser, access the OpenIG server through the Kubernetes ingress, for example, `http://openig.example.com`.

   The Kubernetes ingress handles the request and routes you to a running OpenIG instance.

   The following message appears in the browser:

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

2. If the OpenIG server is running in development mode, access OpenIG Studio. For example, `http://openig.example.com/openig/studio`.

5.6. Modifying and Saving the OpenIG Configuration

Important

Configuration management capabilities described in this section are available only for development mode (mutable mode) deployments running on Minikube for which the `stackConfigSource` option in the `custom.yaml` file is set to `hostPath`.

You can use these configuration management techniques as you develop the OpenIG configuration of a server running in development mode, and then import the configuration to a production server running in production mode (immutable mode) on GKE.

After you have successfully orchestrated an OpenIG deployment as described in this chapter, you can modify the OpenIG configuration, save it, and use the revised configuration to initialize a subsequent OpenIG deployment.

Storing the configuration in a version control system like a Git repository lets you take advantage of capabilities such as version control, difference analysis, and branches when managing the OpenIG configuration. Configuration management enables migration from a development environment to a test environment and then to a production environment. Deployment migration is one of the primary objectives of DevOps techniques.

You can modify the OpenIG configuration as follows:

- Edit the configuration in the `hostPath` directory.
- Add, modify, or delete routes using OpenIG Studio.

Once you are ready to save your configuration, perform the following procedure:

Procedure 5.10. To Save the OpenIG Configuration

1. Verify that when you deployed the OpenIG example, the value of the `stackConfigSource` option in the `custom.yaml` file was set to `hostPath`.
2. If you are managing the OpenIG configuration using Git, commit changed files and add new files to the repository.

Saving the updated configuration is complete. You can redeploy the OpenIG example using the updated configuration at any time.

Chapter 6
Troubleshooting DevOps Deployments

DevOps cloud deployments are multi-layered and often complex.

Errors and misconfigurations can crop up in a variety of places. Performing a logical, systematic search for the source of a problem can be daunting. This chapter provides information and tips that can help you troubleshoot deployment issues in a DevOps environment.

The following table provides an overview of steps to follow and information to collect when attempting to resolve an issue.

Table 6.1. Troubleshooting Overview for DevOps Deployments

Step	More Information
Verify that you installed supported software versions in your environment.	Section 6.1.1, "Verifying Versions of Required Software"
If you are using Minikube, verify that the Minikube VM is configured adequately.	Section 6.1.2, "Verifying the Minikube VM's Configuration (Minikube Only)"
If you are using Minikube, verify that the Minikube VM has sufficient disk space.	Section 6.1.3, "Checking for Sufficient Disk Space (Minikube Only)"
Review the names of your Docker images.	Section 6.2.1, "Reviewing Docker Image Names"
Enable bash completion for the **kubectl** command to make running the command easier.	Section 6.3.1, "Enabling kubectl bash Tab Completion"
Review information about Kubernetes pods.	Section 6.3.2, "Fetching Details About Kubernetes Pods"
Review the Kubernetes cluster's event log.	Section 6.3.3, "Accessing the Kubernetes Cluster's Event Log"
Review each Kubernetes pod's log.	Section 6.3.5, "Obtaining Kubernetes Pod Logs"
View ForgeRock-specific files, such as audit, debug, and application logs, and other files.	Section 6.3.6, "Accessing Files in Kubernetes Pods"
Perform a dry run of Helm chart creation and examine the YAML that Helm sends to Kubernetes.	Section 6.3.7, "Performing a Dry Run of Helm Chart Installation"
Review logs of system components such as Docker and Kubernetes.	Section 6.3.8, "Accessing the Kubelet Log"

6.1. Troubleshooting the Environment

This section provides tips and techniques to troubleshoot problems with a Minikube or GKE environment.

6.1.1. Verifying Versions of Required Software

Environments in which you run the DevOps Examples must be based on supported versions of software, documented in the following tables:

- Table 2.2, "Software Versions for Minikube Deployment Environments"
- Table 2.4, "Software Versions for GKE Deployment Environments"

Use the following commands to determine software versions:

Table 6.2. Determining Software Versions

Software	Command
Oracle VirtualBox	**VBoxManage --version**
Docker Client	**docker version**
Minikube	**minikube version**
kubectl (Kubernetes client)	**kubectl version**
Kubernetes Helm	**helm version**
Google Cloud SDK	**gcloud version**

6.1.2. Verifying the Minikube VM's Configuration (Minikube Only)

The **minikube start** command example in Procedure 2.1, "To Create and Initialize a Minikube Virtual Machine" specifies the virtual hardware requirements for a Minikube VM.

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

6.1.3. Checking for Sufficient Disk Space (Minikube Only)

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

Verify that adequate disk space remains by logging into the Minikube VM and running a command to display free disk space:

```
$ minikube ssh
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           3.9G  375M  3.6G  10% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
tmpfs           3.9G   64K  3.9G   1% /
tmp
/dev/sda1        28G  3.6G   23G  14% /mnt/
sda1
/Users          931G  208G  723G  23% /Users
$ exit
logout
```

In the preceding example, 23 GB of free disk space is available on the Minikube VM.

6.2. Troubleshooting Containerization

This section provides tips and techniques to troubleshoot problems creating or accessing Docker containers.

6.2.1. Reviewing Docker Image Names

The components that comprise Docker image names are properties in the DevOps Examples Helm charts. You can either use the default image names hardcoded in the Helm charts or override the defaults, but in either case, Docker images must have the names expected by Helm, or deployment of one or more Kubernetes pods will fail.

A very common error when deploying the DevOps Examples is a mismatch between the names of one or more Docker images and the names of the Docker images expected by the Helm charts. See Procedure 6.1, "To Diagnose and Correct Docker Name Mismatches" for troubleshooting a Docker image name mismatch.

To verify that your Docker image names match the image names expected by the DevOps Examples Helm charts:

- If you are using the Minikube environment, run the **eval $(minikube docker-env)** command, and then run the **docker images** command. Review the names of Docker images available for deployment by Helm. The following output shows Docker images used by the OpenAM and OpenDJ example:

```
$ eval $(minikube docker-env)
$ docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
forgerock/opendj                    4.0.0      1bd60c97874b        7 minutes ago       404 MB
forgerock/amster                    14.0.0     5a3de8e2b00d        8 minutes ago       164 MB
forgerock/openam                    14.0.0     81305fa2eef2        8 minutes ago       700
 MB
. . .
```

- Compare the Docker image names to the image names expected by Helm. The default image names hardcoded in the DevOps Examples Helm charts are as follows:

Table 6.3. Default Image Names Expected by Helm

Repository (Minikube Images)	Registry + Repository (GKE Images)	Tag
`forgerock/openam`	`gcr.io/Google Cloud Platform project/openam`	`14.0.0`
`forgerock/amster`	`gcr.io/Google Cloud Platform project/amster`	`14.0.0`
`forgerock/opendj`	`gcr.io/Google Cloud Platform project/opendj`	`4.0.0`
`forgerock/openidm`	`gcr.io/Google Cloud Platform project/openidm`	`5.0.0`
`forgerock/openig`	`gcr.io/Google Cloud Platform project/openig`	`5.0.0`

Chapter 7, "*Reference*" describes Docker image naming conventions and requirements for the DevOps Examples:

- For information about naming conventions used by the DevOps Examples for Docker images, see Section 7.2, "Naming Docker Images".
- For information about how to specify Docker image names when building Docker images with the **build.sh** command, see Section 7.3, "Using the build.sh Script to Create Docker Images".
- For information about overriding the default Docker image names expected by the Helm charts, see Section 7.4, "Specifying Deployment Options in the custom.yaml File".

6.3. Troubleshooting Orchestration

This section provides tips and techniques to help you troubleshoot problems related to Docker container orchestration in Kubernetes.

6.3.1. Enabling kubectl bash Tab Completion

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

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

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

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

6.3.2. Fetching Details About Kubernetes Pods

The **kubectl describe pod** command provides detailed information about the status of a running Kubernetes pod, including the following:

- Configuration information
- Pod status
- Volume mounts
- Log of events related to the pod

To fetch details about a pod, obtain the pod's name using the **kubectl get pods** command, and then run the **kubectl describe pod** command, supplying the name of the pod to describe:

```
$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
amster-3035894593-k17mc  1/1       Running   0          10m
configstore-0            1/1       Running   0          13m
ctsstore-0               1/1       Running   0          11m
openam-306579823-3brgj   1/1       Running   0          10m
userstore-0              1/1       Running   0          12m

$ kubectl describe pod openam-306579823-3brgj
Name:  openam-306579823-3brgj
Namespace: default
Node:  minikube/192.168.99.100
Start Time: Mon, 24 Apr 2017 15:28:27 -0700
Labels:  app=rousing-liger-openam
  component=openam
  pod-template-hash=306579823
  vendor=forgerock
Annotations: kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":
{"kind":"ReplicaSet","namespace":"default","name":"openam-306579823","uid":"55bde159-293d-11e7-8bd3
-080027298fc3","...
Status:  Running
IP:  172.17.0.11
Controllers: ReplicaSet/openam-306579823
Init Containers:
  copy:
    Container ID: docker://7dafd0050ee28316a71cb16064aa9637d2b04ab419c03567e31a2be86dd21053
    Image:  alpine
    Image ID:  docker://sha256:4a415e3663882fbc554ee830889c68a33b3585503892cc718a4698e91ef2a526
    Port:
    Command:
      /bin/sh
      -c
      mkdir -p /root/openam/openam/debug; umask u=rwx,g=,o= ; cd /root/openam; cp -L /var/boot/boot.json
.; cp  -rL /var/secrets/openam/.?* openam; cp -L /var/secrets/openam/authorized_keys .
    State:  Terminated
      Reason:  Completed
      Exit Code: 0
      Started:  Mon, 01 Jan 0001 00:00:00 +0000
      Finished:  Mon, 24 Apr 2017 15:28:27 -0700
    Ready:  True
    Restart Count: 0
    Environment: <none>
```

```
    Mounts:
      /root/openam from openam-root (rw)
      /var/boot from openam-boot (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-9p07d (ro)
      /var/secrets/openam from openam-secrets (rw)
Containers:
  openam:
    Container ID: docker://f5d5ae7037ee776b8b417af98df42d7501e18be80836b85a08eaed6d8cec124d
    Image:  forgerock/openam:14.0.0
    Image ID:  docker://sha256:f14a94c0078b4c41cab986f0cdd059c967dc9e47a267d6379c7060472f738891
    Port:  8080/TCP
    State:  Running
      Started:  Mon, 24 Apr 2017 15:28:28 -0700
    Ready:  True
    Restart Count: 0
    Liveness:  http-get http://:8080/openam/isAlive.jsp delay=60s timeout=10s period=30s #success=1
 #failure=3
    Readiness:  http-get http://:8080/openam/isAlive.jsp delay=40s timeout=5s period=20s #success=1
 #failure=3
    Environment:
      JAVA_OPTS: -Xmx1g
    Mounts:
      /root/openam from openam-root (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-9p07d (ro)
Conditions:
  Type  Status
  Initialized  True
  Ready  True
  PodScheduled  True
Volumes:
  openam-root:
    Type: EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
  openam-secrets:
    Type: Secret (a volume populated by a Secret)
    SecretName: openam-secrets
    Optional: false
  openam-boot:
    Type: ConfigMap (a volume populated by a ConfigMap)
    Name: boot-json
    Optional: false
  log-config:
    Type: ConfigMap (a volume populated by a ConfigMap)
    Name: am-log-config
    Optional: false
  default-token-9p07d:
    Type: Secret (a volume populated by a Secret)
    SecretName: default-token-9p07d
    Optional: false
QoS Class: BestEffort
Node-Selectors: <none>
Tolerations: <none>
Events:
  FirstSeen LastSeen Count From   SubObjectPath   Type  Reason  Message
  --------- -------- ----- ----   -------------   -------- ------  -------
  9m  9m  1 default-scheduler     Normal  Scheduled Successfully assigned openam-306579823-3brgj to
 minikube
  9m  9m  1 kubelet, minikube spec.initContainers{copy} Normal  Pulled  Container image "alpine" already
 present on machine
```

```
  9m  9m  1 kubelet, minikube spec.initContainers{copy} Normal  Created  Created container with id
 7dafd0050ee28316a71cb16064aa9637d2b04ab419c03567e31a2be86dd21053
  9m  9m  1 kubelet, minikube spec.initContainers{copy} Normal  Started  Started container with id
 7dafd0050ee28316a71cb16064aa9637d2b04ab419c03567e31a2be86dd21053
  9m  9m  1 kubelet, minikube spec.containers{openam}  Normal  Pulled  Container image "forgerock/
openam:14.0.0" already present on machine
  9m  9m  1 kubelet, minikube spec.containers{openam}  Normal  Created  Created container with id
 f5d5ae7037ee776b8b417af98df42d7501e18be80836b85a08eaed6d8cec124d
  9m  9m  1 kubelet, minikube spec.containers{openam}  Normal  Started  Started container with id
 f5d5ae7037ee776b8b417af98df42d7501e18be80836b85a08eaed6d8cec124d
```

6.3.3. Accessing the Kubernetes Cluster's Event Log

The **kubectl describe pod** command, described in the previous section, lists Kubernetes events for a single pod. While reviewing the events for a pod can be useful when troubleshooting, it is often helpful to obtain the cluster-wide event log.

The **kubectl get events** command returns the event log for the cluster's entire lifetime. You might want to redirect the output of the **kubectl get events** command to a file—clusters that have been running for a long time can have very large event logs.

A common troubleshooting technique is to run a Kubernetes operation, such as installing a Helm chart in one terminal window, and to simultaneously run the **kubectl get events** command with the `--watch` argument in a second terminal window. New Kubernetes events appear in the second terminal window as the Kubernetes operation proceeds in the first window.

The following is an extract of the Kubernetes event log from deployment of the OpenAM and OpenDJ example:

```
LASTSEEN                      FIRSTSEEN                     COUNT     NAME                      KIND
    SUBOBJECT                TYPE      REASON    SOURCE              MESSAGE
2017-04-24 15:26:33 -0700 PDT   2017-04-24 15:26:33 -0700 PDT   1         ctsstore-0   Pod
   spec.containers{opendj}   Normal    Created   kubelet, minikube   Created container with id
 009e31e75e052324e29f2a9ad475655ea4564c575539a76c70ec2f335651053c
2017-04-24 15:26:33 -0700 PDT   2017-04-24 15:26:33 -0700 PDT   1         ctsstore-0   Pod
   spec.containers{opendj}   Normal    Started   kubelet, minikube   Started container with id
 009e31e75e052324e29f2a9ad475655ea4564c575539a76c70ec2f335651053c
2017-04-24 15:27:03 -0700 PDT   2017-04-24 15:27:03 -0700 PDT   1         openam    Pod
 Normal    Scheduled   default-scheduler   Successfully assigned openam to minikube
2017-04-24 15:27:03 -0700 PDT   2017-04-24 15:27:03 -0700 PDT   1         amster    Pod
 Normal    Scheduled   default-scheduler   Successfully assigned amster to minikube
2017-04-24 15:27:03 -0700 PDT   2017-04-24 15:27:03 -0700 PDT   1         amster    Pod       spec
.containers{amster}   Normal    Pulled    kubelet, minikube   Container image "forgerock/amster:14.0.0"
 already present on machine
2017-04-24 15:27:03 -0700 PDT   2017-04-24 15:27:03 -0700 PDT   1         openam    Pod       spec
.initContainers{copy-secrets}   Normal    Pulled    kubelet, minikube   Container image "alpine" already
 present on machine
2017-04-24 15:27:03 -0700 PDT   2017-04-24 15:27:03 -0700 PDT   1         amster    Pod
  spec.containers{amster}   Normal    Created   kubelet, minikube   Created container with id
 b83d408129bba96aa3cc815d27eea363b2f024b9525b6f15373f2fc7f930804b
2017-04-24 15:27:04 -0700 PDT   2017-04-24 15:27:04 -0700 PDT   1         openam    Pod       spec
.initContainers{copy-secrets}   Normal    Created   kubelet, minikube   Created container with id
 ef8b07fa52cd3a2c152920ea5a213f70d166812c1a8c132c7069c1b1f9cc7a47
2017-04-24 15:27:04 -0700 PDT   2017-04-24 15:27:04 -0700 PDT   1         amster    Pod
  spec.containers{amster}   Normal    Started   kubelet, minikube   Started container with id
 b83d408129bba96aa3cc815d27eea363b2f024b9525b6f15373f2fc7f930804b
2017-04-24 15:27:04 -0700 PDT   2017-04-24 15:27:04 -0700 PDT   1         openam    Pod       spec
.initContainers{copy-secrets}   Normal    Started   kubelet, minikube   Started container with id
 ef8b07fa52cd3a2c152920ea5a213f70d166812c1a8c132c7069c1b1f9cc7a47
2017-04-24 15:27:04 -0700 PDT   2017-04-24 15:27:04 -0700 PDT   1         openam    Pod       spec
.containers{openam}   Normal    Pulled    kubelet, minikube   Container image "forgerock/openam:14.0.0"
 already present on machine
```

6.3.4. Troubleshooting Pods That Will not Start

When starting, Kubernetes pods obtain Docker images. In the DevOps Examples, the names of the Docker images are defined in Helm charts. If a Docker image configured in one of the Helm charts is not available, the pod will not start.

The most common reason for pod startup failure is a Docker image name mismatch. An image name mismatch occurs when a Docker image name configured in a Helm chart does not match any available Docker images. Troubleshoot and fix Docker image name mismatches as follows:

Procedure 6.1. To Diagnose and Correct Docker Name Mismatches

1. Review the default Docker image names expected by the DevOps Examples Helm charts covered in Section 6.2.1, "Reviewing Docker Image Names".

2. Run the **kubectl get pods** command. Any pods with the `ImagePullBackOff` or `ErrImagePull` status are unable to start. For example:

```
$ kubectl get pods
NAME            READY     STATUS             RESTARTS   AGE
configstore-0   0/1       ImagePullBackOff   0          11m
```

3. Run the **kubectl describe pod** on the pod that won't start and review the Events section at the bottom of the output:

```
$ kubectl describe pod configstore-0
. . .
Events:
  FirstSeen LastSeen Count From   SubObjectPath  Type  Reason   Message
  --------- -------- ----- ----   -------------  -------- ------   -------
  13m  13m  2 default-scheduler    Warning  FailedScheduling SchedulerPredicates failed due to
 PersistentVolumeClaim is not bound: "data-configstore-0", which is unexpected.
  13m  13m  1 default-scheduler    Normal  Scheduled  Successfully assigned configstore-0 to minikube
  13m  2m  7 kubelet, minikube spec.containers{opendj} Normal  Pulling   pulling image "forgerock/
opendj:4.0.0"
  13m  2m  7 kubelet, minikube spec.containers{opendj} Warning  Failed   Failed to pull image
 "forgerock/opendj:4.0.0": rpc error: code = 2 desc = Error: image forgerock/opendj not found
  13m  2m  7 kubelet, minikube    Warning  FailedSync  Error syncing pod, skipping: failed to
 "StartContainer" for "opendj" with ErrImagePull: "rpc error: code = 2 desc = Error: image forgerock/
opendj not found"

  13m 9s 53 kubelet, minikube spec.containers{opendj} Normal BackOff  Back-off pulling image
 "forgerock/opendj:4.0.0"
  13m 9s 53 kubelet, minikube    Warning FailedSync Error syncing pod, skipping: failed to
 "StartContainer" for "opendj" with ImagePullBackOff: "Back-off pulling image \"forgerock/opendj:4.0
.0\""
```

Look for events with the text `Failed to pull image` and `Back-off pulling image`. These events indicate the name of the Docker image that Kubernetes is trying to retrieve to create a running pod.

Note that the cluster-wide event log also contains these events, so you can see them in the **kubectl get events** command ouptut.

4. Run the **docker images** command to list available Docker images:

```
$ docker images
REPOSITORY                                          TAG                 IMAGE ID            CREATED
          SIZE
forgerock/amster                                    14.0.0     d736b9f77cd2        19 minutes ago
   164 MB
forgerock/openam                                    14.0.0     e4d4f80b5ec8        19 minutes ago
   700 MB
forgerock/opendj                                    4.0.0X     835c336abd36        2 hours ago
   400
 MB
. . .
```

A Docker image name mismatch occurs when the Docker image that Kubernetes was attempting to retrieve is not available in your Docker registry.

In the preceding example, observe that Kubernetes attempts to access the `forgerock/opendj:4.0.0` image, which does not exist among the available local Docker images. The **docker images** output

shows that there is a `forgerock/opendj` image, which was probably tagged incorrectly with `4.0.0X` instead of `4.0.0`.

5. If a Docker name mismatch is the reason for the pod not starting, terminate the deployment, recreate the Docker image with the correct name, and redeploy.

6.3.5. Obtaining Kubernetes Pod Logs

In addition to Kubernetes clusters' event logs, each Kubernetes pod has its own log that contains output written to `stdout` by applications running in the pod.

To obtain a Kubernetes pod's log, run the **kubectl logs** command.

To follow changes to a pod's Kubernetes log, you can run a Kubernetes operation such as deploying OpenAM in one terminal window, and simultaneously run the **kubectl logs -f** command in a second terminal window. New entries written to `stdout` in the pod appear in the second terminal window as the Kubernetes operation proceeds in the first window.

The following is an example of `stdout` entries in a pod running OpenAM. The output comprises messages from Tomcat startup:

```
$ kubectl logs openam-306579823-3brgj
24-Apr-2017 22:28:29.064 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:
        Apache Tomcat/8.5.13
24-Apr-2017 22:28:29.066 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:
        Mar 27 2017 14:25:04 UTC
24-Apr-2017 22:28:29.067 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:
        8.5.13.0
24-Apr-2017 22:28:29.067 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:
        Linux
24-Apr-2017 22:28:29.067 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:
        4.7.2
24-Apr-2017 22:28:29.067 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:
        amd64
24-Apr-2017 22:28:29.067 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:
        /usr/lib/jvm/java-8-openjdk-amd64/jre
24-Apr-2017 22:28:29.068 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:
        1.8.0_121-8u121-b13-1~bpo8+1-b13
24-Apr-2017 22:28:29.068 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:
        Oracle Corporation
24-Apr-2017 22:28:29.068 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE:
        /usr/local/tomcat
24-Apr-2017 22:28:29.069 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME:
        /usr/local/tomcat
24-Apr-2017 22:28:29.069 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Djava.util.logging.config.file=/usr/local/tomcat/conf/logging.properties
24-Apr-2017 22:28:29.070 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
24-Apr-2017 22:28:29.071 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Xmx1g
24-Apr-2017 22:28:29.073 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Djdk.tls.ephemeralDHKeySize=2048
24-Apr-2017 22:28:29.074 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Djava.protocol.handler.pkgs=org.apache.catalina.webresources
```

```
24-Apr-2017 22:28:29.074 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Dcatalina.base=/usr/local/tomcat
24-Apr-2017 22:28:29.074 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Dcatalina.home=/usr/local/tomcat
24-Apr-2017 22:28:29.075 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line
 argument: -Djava.io.tmpdir=/usr/local/tomcat/temp
24-Apr-2017 22:28:29.075 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent Loaded
 APR based Apache Tomcat Native library 1.2.12 using APR version 1.5.1.
24-Apr-2017 22:28:29.077 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent APR
 capabilities: IPv6 [true], sendfile [true], accept filters [false], random [true].
24-Apr-2017 22:28:29.077 INFO [main] org.apache.catalina.core.AprLifecycleListener.lifecycleEvent APR/
OpenSSL configuration: useAprConnector [false], useOpenSSL [true]
24-Apr-2017 22:28:29.091 INFO [main] org.apache.catalina.core.AprLifecycleListener.initializeSSL OpenSSL
 successfully initialized (OpenSSL 1.1.0e  16 Feb 2017)
24-Apr-2017 22:28:29.225 INFO [main] org.apache.coyote.AbstractProtocol.init Initializing ProtocolHandler
 ["http-nio-8080"]
24-Apr-2017 22:28:29.256 INFO [main] org.apache.tomcat.util.net.NioSelectorPool.getSharedSelector Using a
 shared selector for servlet write/read
24-Apr-2017 22:28:29.258 INFO [main] org.apache.catalina.startup.Catalina.load Initialization processed in
 695 ms
24-Apr-2017 22:28:29.306 INFO [main] org.apache.catalina.core.StandardService.startInternal Starting
 service Catalina
24-Apr-2017 22:28:29.312 INFO [main] org.apache.catalina.core.StandardEngine.startInternal Starting
 Servlet Engine: Apache Tomcat/8.5.13
24-Apr-2017 22:28:29.341 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig
.deployDirectory Deploying web application directory /usr/local/tomcat/webapps/openam
24-Apr-2017 22:28:38.677 INFO [localhost-startStop-1] org.apache.jasper.servlet.TldScanner.scanJars At
 least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a
 complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during
 scanning can improve startup time and JSP compilation time.
Starting up OpenAM at Apr 24, 2017 10:28:42 PM
24-Apr-2017 22:28:43.166 INFO [localhost-startStop-1] com.sun.jersey.server.impl.application
.WebApplicationImpl.initiate Initiating Jersey application, version 'Jersey: 1.1.1-ea 07/14/2009 07:18 PM'
24-Apr-2017 22:28:43.743 INFO [localhost-startStop-1] com.sun.jersey.server.impl.application
.WebApplicationImpl.initiate Initiating Jersey application, version 'Jersey: 1.1.1-ea 07/14/2009 07:18 PM'
24-Apr-2017 22:28:58.674 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig
.deployDirectory Deployment of web application directory /usr/local/tomcat/webapps/openam has finished in
 29,331 ms
24-Apr-2017 22:28:58.687 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler
 ["http-nio-8080"]
24-Apr-2017 22:28:58.697 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 29438 ms
```

6.3.6. Accessing Files in Kubernetes Pods

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

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

```
$ kubectl exec openam-306579823-3brgj -it /bin/bash
root@openam-306579823-3brgj:/usr/local/tomcat# cd /root/openam/openam/log
root@openam-306579823-3brgj:~/openam/openam/log# ls
access.audit.json    authentication.audit.json
activity.audit.json  config.audit.json
root@openam-306579823-3brgj:~/openam/openam/log# cat authentication.audit.json
{"realm":"/","transactionId":"71214ca4-282f-42d3-ad01-ca1c08ff991c-0","component":"Authentication"
,"eventName":"AM-LOGIN-MODULE-COMPLETED","result":"SUCCESSFUL","entries":[{"moduleId":"Application"
,"info":{"authIndex":"module_instance","authControlFlag":"REQUIRED","moduleClass":"Application"
,"ipAddress":"172.17.0.11","authLevel":"0"}}],"userId":"","principal":["cn=dsameuser,ou=DSAME Users
,dc=openam,dc=forgerock,dc=org"],"timestamp":"2017-04-24T22:28:52.405Z","trackingIds":["71214ca4-282f-42d3
-ad01-ca1c08ff991c-35"],"_id":"71214ca4-282f-42d3-ad01-ca1c08ff991c-37"}
{"realm":"/","transactionId":"71214ca4-282f-42d3-ad01-ca1c08ff991c-0","component":"Authentication"
,"eventName":"AM-LOGIN-COMPLETED","result":"SUCCESSFUL","entries":[{"moduleId":"Application","info":
{"authIndex":"module_instance","ipAddress":"172.17.0.11","authLevel":"0"}}],"userId":"cn=dsameuser
,ou=DSAME Users,dc=openam,dc=forgerock,dc=org","principal":["dsameuser"],"timestamp":"2017-04-24T22:28:52
.658Z","trackingIds":["71214ca4-282f-42d3-ad01-ca1c08ff991c-35"],"_id":"71214ca4-282f-42d3-ad01-
ca1c08ff991c-43"}
root@openam-306579823-3brgj:~/openam/openam/log# exit
exit
```

In addition to logging into a pod's shell to access files, you can also copy files from a a Kubernetes pod to your local system using the **kubectl cp** command. For more information, see the **kubectl** command reference .

6.3.7. Performing a Dry Run of Helm Chart Installation

The DevOps Examples use Kubernetes Helm to simplify deployment to Kubernetes by providing variable substitution in Kubernetes manifests for predefined, partial, and custom variables.

When Helm chart installation does not proceed as expected, it can sometimes be helpful to review how Helm expanded charts when creating Kubernetes manifests. Helm dry run installation lets you see Helm chart expansion without deploying.

The initial section of Helm dry run installation output shows user-supplied and computed values. The following example shows output from a dry run installation of the `openam-runtime` chart:

```
$ cd /path/to/fretes-repo
$ helm install --dry-run --debug -f custom.yaml /path/to/fretes-repo/helm/openam-runtime/
Created tunnel using local port: '62356'
SERVER: "localhost:62356"
CHART PATH: /tmp/repos/fretes/helm/openam-runtime
NAME:   torpid-otter
REVISION: 1
RELEASED: Mon Apr 24 12:23:10 2017
CHART: openam-runtime-0.1.0
USER-SUPPLIED VALUES:
cookieDomain: .example.com
git:
  branch: master
  repo: https://stash.forgerock.org/scm/cloud/forgeops-init.git

COMPUTED VALUES:
configLdapHost: configstore-0.configstore
```

```
configLdapPort: 389
cookieDomain: .example.com
git:
  branch: master
  repo: https://stash.forgerock.org/scm/cloud/forgeops-init.git
heapSize: 1g
image:
  amster: amster
  openam: openam
  pullPolicy: IfNotPresent
  repository: forgerock
  tag: 14.0.0
logDriver: none
openamInstance: http://openam:80/openam
openamReplicaCount: 1
rootSuffix: dc=openam,dc=forgerock,dc=org

HOOKS:
. . .
```

After the user-supplied and computed values, the generated Kubernetes manifests appear in the dry run output:

```
MANIFEST:

---
# Source: openam-runtime/templates/secrets
.yaml
. .
.
---
# Source: openam-runtime/templates/log-map
.yaml
. .
.
---
# Source: openam-runtime/templates/config-map
.yaml
. . .
```

6.3.8. Accessing the Kubelet Log

If you suspect a low-level problem with Kubernetes cluster operation, access the cluster's shell and run the **journalctl -u localkube.service** command. For example, on Minikube:

```
$ minikube ssh
$ journalctl -u localkube.service
-- Logs begin at Tue 2017-04-25 20:28:27 UTC, end at Tue 2017-04-25 22:47:56 UTC. --
Apr 25 20:28:27 minikube localkube[3785]: E0425 20:28:27.661974    3785 kuberuntime_container
.go:385] ContainerStatus for 3ff0906a1f97df194b12939734c9282aaf439afcc311297fb3b7ce2b98dcc6a1
 error: rpc error: code = 2 desc = unable to inspect docker image
 "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e" while inspecting docker
 container "3ff0906a1f97df194b12939734c9282aaf439afcc311297fb3b7ce2b98dcc6a1": no such image:
 "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e"
Apr 25 20:28:27 minikube localkube[3785]: E0425 20:28:27.661981    3785 kuberuntime_manager
.go:858] getPodContainerStatuses for pod "amster-109299095-9prn5_default(7a7830d5-1f02
-11e7-8f9d-080027298fc3)" failed: rpc error: code = 2 desc = unable to inspect docker image
 "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e" while inspecting docker
 container "3ff0906a1f97df194b12939734c9282aaf439afcc311297fb3b7ce2b98dcc6a1": no such image:
 "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e"
Apr 25 20:28:27 minikube localkube[3785]: E0425 20:28:27.661993    3785 generic.go:269] PLEG: pod
 amster-109299095-9prn5/default failed reinspection: rpc error: code = 2 desc = unable to inspect
 docker image "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e" while inspecting
 docker container "3ff0906a1f97df194b12939734c9282aaf439afcc311297fb3b7ce2b98dcc6a1": no such image:
 "sha256:a5b0509feb84450f8d133ea7d919ebfc8d2443e24164046654100624c160c38e"
. . .
```

Chapter 7
Reference

This reference section covers information needed for multiple DevOps Examples.

The following topics are covered:

- Git Repositories Used by the DevOps Examples
- Naming Docker Images
- Using the build.sh Script to Create Docker Images
- Specifying Deployment Options in the custom.yaml File

7.1. Git Repositories Used by the DevOps Examples

The ForgeRock DevOps Examples use the `release/5.0.0` branch of the following Git repositories:

- The `docker` repository, which contains Dockerfiles and other artifacts for building Docker images.
- The `fretes` repository, which contains scripts, Helm charts, and Kubernetes manifests for orchestrating the DevOps Examples.
- The `forgeops-init` repository, which contains JSON files and Amster scripts for configuring OpenAM, OpenIDM, and OpenIG.

You must clone these Git repositories before using the DevOps Examples, and you should pull updates from the repositories before running the examples to ensure you have the latest bug fixes.

Obtaining the `docker` and `fretes` repositories requires a ForgeRock BackStage account. The `forgeops-init` repository is public and can be obtained without having any accounts or permissions.

Perform the following steps to obtain the `docker` and `fretes` repositories:

Procedure 7.1. To Obtain the docker and fretes Repositories

1. If you do not already have a ForgeRock BackStage account, get one from https://backstage.forgerock.com/.

   A ForgeRock BackStage account provides access to the ForgeRock Bitbucket Server.

2. Clone the ForgeRock Git repositories that comprise the DevOps Examples:

a. Clone the `docker` Git repository, which contains Dockerfiles and other support files for building Docker images:

```
$ git clone https://myBackStageID@stash.forgerock.org/scm/docker/docker.git
```

Enter your BackStage password when prompted to do so.

b. Clone the `fretes` Git repository, which contains Kubernetes deployment examples:

```
$ git clone https://myBackStageID@stash.forgerock.org/scm/docker/fretes.git
```

Enter your BackStage password when prompted to do so.

3. Check out the `release/5.0.0` branch of both repositories:

```
$ cd docker
$ git checkout release/5.0.0
$ cd ../fretes
$ git checkout release/5.0.0
```

If you ever need to access the `docker` and `fretes` Git repositories using the **git** command from a script without being prompted to enter passwords, take the following actions:

- Add your public SSH key to your ForgeRock Bitbucket Server account. For details, see SSH user keys for personal use .
- Add commands to your script to access the Git repository over ssh. For example:

```
$ git clone ssh://git@stash.forgerock.org:7999/docker/docker.git
$ git clone ssh://git@stash.forgerock.org:7999/docker/fretes.git
```

7.2. Naming Docker Images

Docker image names consist of the following components:

- **Docker registry**. Specifies the Docker registry to which an image is pushed. For example, `gcr.io`.
- **Repository**. Specifies the Docker repository—a collection of images with different tags. For example, `engineering-devops/openam`.
- **Tag**. Specifies the alphanumeric identifier attached to images within a repository. For example, `14.0.0`.

The DevOps Examples use the following naming conventions for Docker images.

Table 7.1. Docker Image Naming Conventions

Image Name Component	Naming Conventions and Examples
Registry	**Minikube**. For images pushed to local cache, do not specify a registry component.

Image Name Component	Naming Conventions and Examples
	GKE. For images pushed to your GKE environment, specify `gcr.io`. **Minikube and GKE**. For images pushed to a remote registry, specify the registry's FQHN. For example, `registry.mycompany.io`.
Repository	**Minikube**. Specify `forgerock/` followed by a component name. For example, `forgerock/openam`. **GKE**. Specify the name of your GKE project, followed by a slash (/), followed by a component name. For example, `engineering-devops/openam`.
Tag	Specify the component version.

The following are examples of Docker image names that follow the naming conventions described in the preceding table:

`forgerock/openam:14.0.0`

An image deployed to local cache in a Minikube environment

`gcr.io/engineering-devops/openam:14.0.0`

An image deployed to a GKE environment within the GKE `engineering-devops` project

`registry.mycompany.io/forgerock/openam:14.0.0`

An image pushed to the remote Docker registry, `registry.mycompany.io`

7.3. Using the build.sh Script to Create Docker Images

Create Docker images for the DevOps Examples with the **build.sh** script.

Before running the script, familiarize yourself with the DevOps Examples Docker image naming conventions described in Section 7.2, "Naming Docker Images".

The script takes the following input arguments:

`-R` *`repository`*

Specifies the first component of the repository name. For example, for a repository named `forgerock/openam`, specify `-R forgerock`. Note that for Docker images deployed on GKE, the first part of the repository component of the image name *must* be your GKE project name.

`-t` *`tag`*

Specifies the image tag. For example, `14.0.0`.

`-r` *`registry`*

Specifies a Docker registry to push the image to. For example, `-r gcr.io` or `-r registry.mycompany.io`. Do not specify the `-r` argument when deploying the image to the local Docker cache.

`-g`

Indicates that you intend to deploy the image in a GKE environment.

The following are example **build.sh** commands to create Docker images that follow the naming conventions in Section 7.2, "Naming Docker Images":

- Build an image and deploy it in local cache:

```
./build.sh -R forgerock -t 14.0.0 openam
```

- Build an image and deploy it to a GKE environment:

```
./build.sh -R engineering-devops -t 14.0.0 -r gcr.io -g openam
```

- Build an image and deploy it to a remote registry:

```
./build.sh -R forgerock -t 14.0.0 -r registry.mycompany.io openam
```

7.4. Specifying Deployment Options in the custom.yaml File

Kubernetes options specified in the `/path/to/fretes/helm/custom.yaml` file override default options specified in Helm charts in the DevOps Examples reference deployments.

The following are deployment options commonly overridden in the `custom.yaml` file.

Table 7.2. Kubernetes Deployment Options for the OpenAM and OpenDJ Example

Option	Explanation and Example Value
`cookieDomain`	For OpenAM deployments, specifies the deployed OpenAM server's cookie domain. For OpenIDM and OpenIG deployments, specifies the domain portion of the FQDN to which the Kubernetes ingress routes requests. Example: `cookieDomain: .example.com`
`registry`	Specifies a Docker registry from which to pull images. You must specify a slash (/) at the end of the string. When orchestrating the example on Minikube and using Docker images from the local Docker cache, omit this option. Example: `registry: registry.mycompany.io/`
`repo`	**GKE only**. Specifies the GKE project containing your Kubernetes cluster. Example:

Option	Explanation and Example Value
	`repo: engineering-devops`

`stackConfigSource`	Specifies either a directory or a Git repository that contains the initial configuration for the OpenAM instance in the example deployment.
	
	With `hostPath`, specifies a directory that has a subdirectory named `forgeops-init`. During initialization of the OpenAM pod, the **amster** command installs OpenAM and then imports the OpenAM configuration from the `forgeops-init/openam` subdirectory.
	
	Example:
	
	`stackConfigSource:`
	`  hostPath:`
	`  path: /path/to/config`
	
	With `gitRepo`, specifies a Git repository and branch that has a top-level directory named `forgeops-init`. During initialization of the OpenAM pod, the **amster** command installs OpenAM, clones the Git repository, and then imports the OpenAM configuration from the `forgeops-init/openam` subdirectory of the cloned repository.
	
	Example:
	
	`stackConfigSource:`
	`  gitRepo:`
	`  repository: https://stash.forgerock.org/scm/cloud/forgeops-init.git`
	`  revision: HEAD`

`amster`	**Minikube only**. Specifies that the **amster** command should import configuration during initialization of the pod.
	
	Example:
	
	`amster:`
	`  skipImport: true`
	
	When you specify the `amster: skipImport` option with the value `true`:
	
	• The `stackConfigSource` option is still required. Specify the value `emptyDir: {}` to indicate that no OpenAM configuration should be applied after installation.
	
	• External CTS and user stores are ready and available for use, but are *not* used unless you explicitly configure OpenAM to use them.

Appendix A. Getting Support

For more information or resources about the ForgeRock DevOps Examples and ForgeRock Support, see the following sections:

A.1. 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.
- ForgeRock core 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.

Core documentation therefore follows a three-phase review process designed to eliminate errors:

- Product managers and software architects review project documentation design with respect to the readers' software lifecycle needs.
- Subject matter experts review proposed documentation changes for technical accuracy and completeness with respect to the corresponding software.
- Quality experts validate implemented documentation changes for technical accuracy, completeness in scope, and usability for the readership.

The review process helps to ensure that documentation published for a ForgeRock release is technically accurate and complete.

Fully reviewed, published core documentation is available at http://backstage.forgerock.com/. Use this documentation when working with a ForgeRock Identity Platform release.

A.2. Joining the ForgeRock Community

Visit the Community resource center where you can find information about each project, download trial builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and find the source code for open source software.

A.3. How to Report Problems or Provide Feedback

If you have questions regarding the DevOps Examples that are not answered by the documentation, you can ask questions on the DevOps forum at https://forum.forgerock.com/forum/devops.

If you have a valid subscription with ForgeRock, report issues or reproducible bugs at https://backstage.forgerock.com.

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
- Description of the environment, including the following information:
  - Environment type (Minikube or Google Container Engine (GKE))
  - Software versions of supporting components:
    - Oracle VirtualBox (Minikube environments only)
    - Docker client (all environments)
    - Minikube (Minikube environments only)
    - **kubectl** command (all environments)
    - Kubernetes Helm (all environments)
    - Google Cloud SDK (GKE environments only)
  - DevOps Examples release version
  - Any patches or other software that might be affecting the problem
- Steps to reproduce the problem
- Any relevant access and error logs, stack traces, or core dumps

A.4. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes 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, visit https://www.forgerock.com, or send an email to ForgeRock at info@forgerock.com.

Appendix B. Change Log

This appendix covers bug fixes and documentation changes made since ForgeRock DevOps Examples 5.0.0 was originally released.

B.1. Deprecated Features

Functionality listed in this section has been deprecated and will be removed in a future release of the ForgeRock DevOps Examples.

B.1.1. Deprecated in ForgeRock DevOps Examples 5.0.0

- `stackConfigSource: hostPath` option in the `custom.yaml` file

  The `stackConfigSource: hostPath` option specifies a path that has a subdirectory named `forgeops-init`. The DevOps Examples use JSON files in this path to initialize OpenAM, OpenIDM, and OpenIG.

  Note that you can use configuration files stored in a Git repository specified by the `stackConfigSource: gitRepo` option. Using Git for initializing configuration will be enhanced and will continue to be supported in later versions of the DevOps Examples.

- `amster: skipImport` option in the `custom.yaml` file

  The `amster: skipImport` option specifies that the **amster** command should not import configuration during initialization of an OpenAM pod.

  An alternative mechanism for starting with a clean OpenAM configuration will be provided in later versions of the DevOps Examples.

B.2. Bug Fixes

The following bugs have been fixed since the original release of DevOps Examples 5.0.0.

Table B.1. Bug Fixes

Date	Fix
2017-05-01	The LDIF to initialize external user stores was corrected.
2017-04-21	The ingresses in all the deployment examples have been reconfigured to provide sticky load balancing. Note that in order to take advantage of sticky load balancing, you must use an ingress that supports the feature: • The built-in ingress in Minikube version 0.18.0 and later • Version 0.9.0-beta.5 and later of the nginx controller See https://github.com/kubernetes/ingress/blob/master/controllers/nginx/Changelog.md for more information.
2017-04-13	For GKE deployments, the `pd-ssd` storage class is no longer configured as the default storage class.
2017-04-05	The part of the `remove-all.sh` script that deletes Helm charts was corrected.
2017-04-04	In the `helm/amster/templates/config-map.yaml` file, the **amster install-openam** command's `--serverUrl` argument was changed from a hardcoded value to a variable.

B.3. Documentation Updates

The following changes have been made to the DevOps Examples 5.0.0 documentation.

Table B.2. Documentation Updates

Date	Change
2017-05-23	• Steps have been added to Procedure 2.4, "To Delete a GKE Cluster" to remove all components (including persistent volume claims), and then, optionally to delete persistent volumes, before deleting a cluster. • Wording in the second list of bullet points in Section 1.3, "Limitations" regarding limitations of the DevOps Examples has been corrected.
2017-05-01	• Chapter 6, "*Troubleshooting DevOps Deployments*" has been added to the documentation. • The software versions in Table 2.2, "Software Versions for Minikube Deployment Environments" and Table 2.4, "Software Versions for GKE Deployment

Date	Change

	Environments" have been updated, and supported versions of Kubernetes clusters have been added.
	• Section 1.3, "Limitations" has been modified as follows:
	  • The section originally stated that configuration as an artifact is not supported for OpenAM. This limitation has been removed.
	  • The section now mentions that example commands in this guide have been tested on the macOS operating system only, and that modifications might be required when running the example commands on other operating systems.
	• The path to specify for the `hostPath` option has been corrected in Table 4.1, "Example Workflow, OpenIDM DevOps Deployment" and Table 5.1, "Example Workflow, OpenIG DevOps Deployment".
	• Originally, guidance in the "Deploy the OpenAM and OpenDJ example in Minikube" step in Section 3.2, "Working With the OpenAM and OpenDJ Example" was to set the `skipImport: true` option in the `custom.yaml` file.
	This guidance has been modified: users should *not* set this option. The reason for this change is that with the `skipImport: true` option enabled, the external CTS and user stores deployed in the OpenAM and OpenDJ example are not used unless additional configuration is performed.
	References to the `skipImport: true` option in Section 3.5.1.1, "custom.yaml File Examples" and Section 7.4, "Specifying Deployment Options in the custom.yaml File" now mention that when you use this option, the external CTS and user stores are not used without performing additional configuration.
	• Originally, guidance to clone the `docker` and `fretes` Git repositories, considered a one-time-only activity, appeared in Section 2.1, "Kubernetes Running on a Minikube Virtual Machine" and Section 2.2, "Kubernetes Running on Google Container Engine".
	This guidance has been modified: users should not only clone the repositories, but also pull them to obtain up-to-date versions with. bug fixes. Getting an up-to-date version of the repository is *not* a one-time activity because of the necessity to pull the repositories.
	The guidance to clone and pull the repositories in order to obtain up-to-date versions has been added to each chapter that describes a DevOps example, and to Section 7.1, "Git Repositories Used by the DevOps Examples".
	• The following incorrect links have been fixed:
	  • The link to the support forum in Section A.3, "How to Report Problems or Provide Feedback"
	  • Links to the documentation for the OpenIDM bidirectional data synchronization sample in Chapter 4, "*Deploying the OpenIDM Example*"

Date	Change
	• The link to the *ForgeRock Service Broker Guide* in Section 1.2.2, "Introducing Container Orchestration"