DevOps Developer's Guide

/ ForgeRock Identity Platform 6.5

Latest update: 6.5.2

Gina Cariaga David Goldsmith Shankar Raman

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

Abstract

Guide to ForgeRock Identity Platform™ deployment for developers using DevOps techniques.

Table of Contents

Preface

The *DevOps Developer's Guide* explains basic concepts and strategies for deploying ForgeRock software using DevOps tools and methods.

Before You Begin

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

About ForgeRock Identity Platform Software

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

The platform includes the following components:

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

Chapter 1

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 "Software Deployment Approaches".
- Containers. See "Containerization".
- Orchestration. See "Container Orchestration".

1.1. Software Deployment Approaches

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 by name, for example, `ldap.mycompany.com`. If the server fails, it might need to be "nursed" to be brought

[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.

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 6.5. You can use these examples to deploy the ForgeRock Identity Platform using DevOps practices.

1.2. Deployment Automation 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 DevOps Examples.

1.2.1. 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 AM image is based on a Tomcat image that is itself based on an OpenJDK JRE image. In this example, the AM container has AM 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 DevOps Examples include:

- Evaluation-only Docker images for the ForgeRock Identity Platform that can be used for test deployments. These images are available from ForgeRock's public Docker registry.
- Scripts and descriptor files, such as Dockerfiles, that you can use to build Docker images for the ForgeRock Identity Platform for production deployments. These files are available from the public `forgeops` Git repository.

For more information about Docker images for the ForgeRock Identity Platform, see "*Building and Pushing Docker Images*".

1.2.2. 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.

1.2.2.1. Orchestration Frameworks

Orchestration frameworks are frameworks that enable automated, repeatable, managed deployments 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 run on the Kubernetes orchestration framework.

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.

1.2.2.2. Supported Kubernetes Implementations

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 have been tested on the following implementations:

- 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.

- Cloud-based Kubernetes orchestration frameworks, which are suitable for both development and production deployment of the ForgeRock Identity Platform:
  - Google Kubernetes Engine (GKE)
  - Amazon Elastic Container Service for Kubernetes (Amazon EKS)
  - Azure Kubernetes Service (AKS)

1.2.2.3. Kubernetes Manifests

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 Helm charts you'll need to deploy the ForgeRock Identity Platform.

These files are publicly available in a Git repository at https://github.com/ForgeRock/forgeops.git.

You can either use the reference Helm charts available in the `forgeops` 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 the ones listed in "Supported Kubernetes Implementations" is possible, although significant customization might be required.

1.2.2.4. Configuration as an Artifact

The DevOps Examples support managing *configuration as an artifact* for the AM, IDM, and IG components of the ForgeRock Identity Platform.

A cloud-based Git *configuration repository* holds AM, IDM, and IG *configuration versions*—named sets of configuration updates.

Managing configuration as an artifact involves the following activities:

- Initializing and managing one or more configuration repositories. For more information, see "Creating Your Configuration Repository".
- Updating ForgeRock component configuration:
  - For AM and IDM deployments, use the administration consoles, command-line tools, and REST APIs to update configuration. Push the configuration changes to the configuration repository as desired.
  - For IG deployments, manually update the IG configuration maintained in the configuration repository.
- Identifying sets of changes that comprise configuration versions. This activity varies depending on your deployment. For example, to identify configuration version `6.5.3` of an AM deployment, you

might merge the `autosave-my-namespace` branch with the configuration repository's `master` branch, and then create the `6.5.3` branch from the `master` branch.

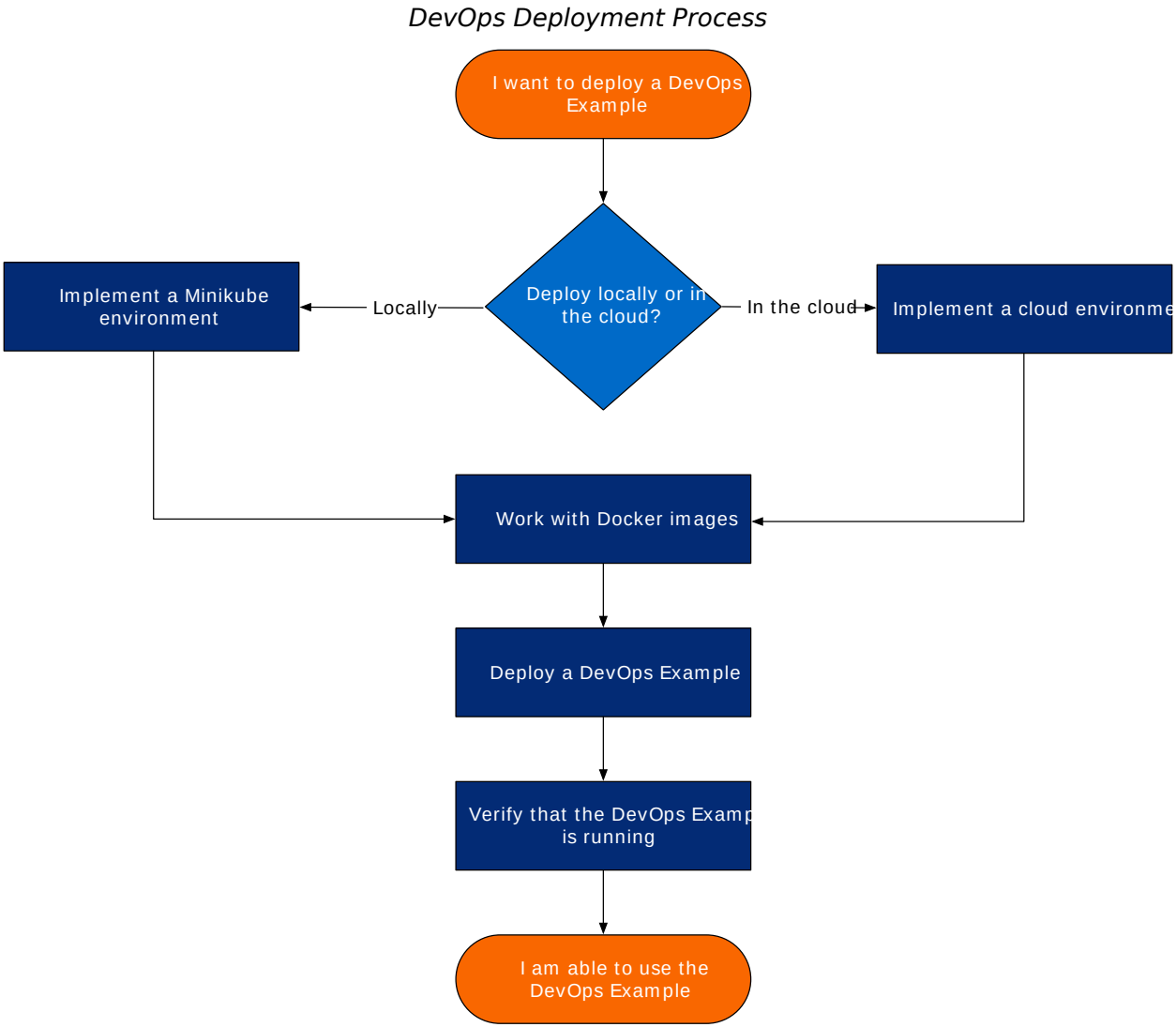
- Redeploying AM, IDM, and IG based on any given configuration version.

1.3. Deployment Process Overview

To get the ForgeRock Identity Platform up and running as quickly as possible, see the DevOps Quick Start Guide, which provides instructions for the simplest possible DevOps ForgeRock Identity Platform deployment.

Use this guide for more complex DevOps deployments.

The following diagram illustrates a high-level workflow you'll use to set up a DevOps environment and deploy ForgeRock Identity Platform components:



Finer-grained workflows in this guide provide more detailed task breakouts:

Task	Reference
Implement a Minikube or cloud environment.	"DevOps Environment Implementation"
Work with Docker images.	"Building and Pushing Docker Images"

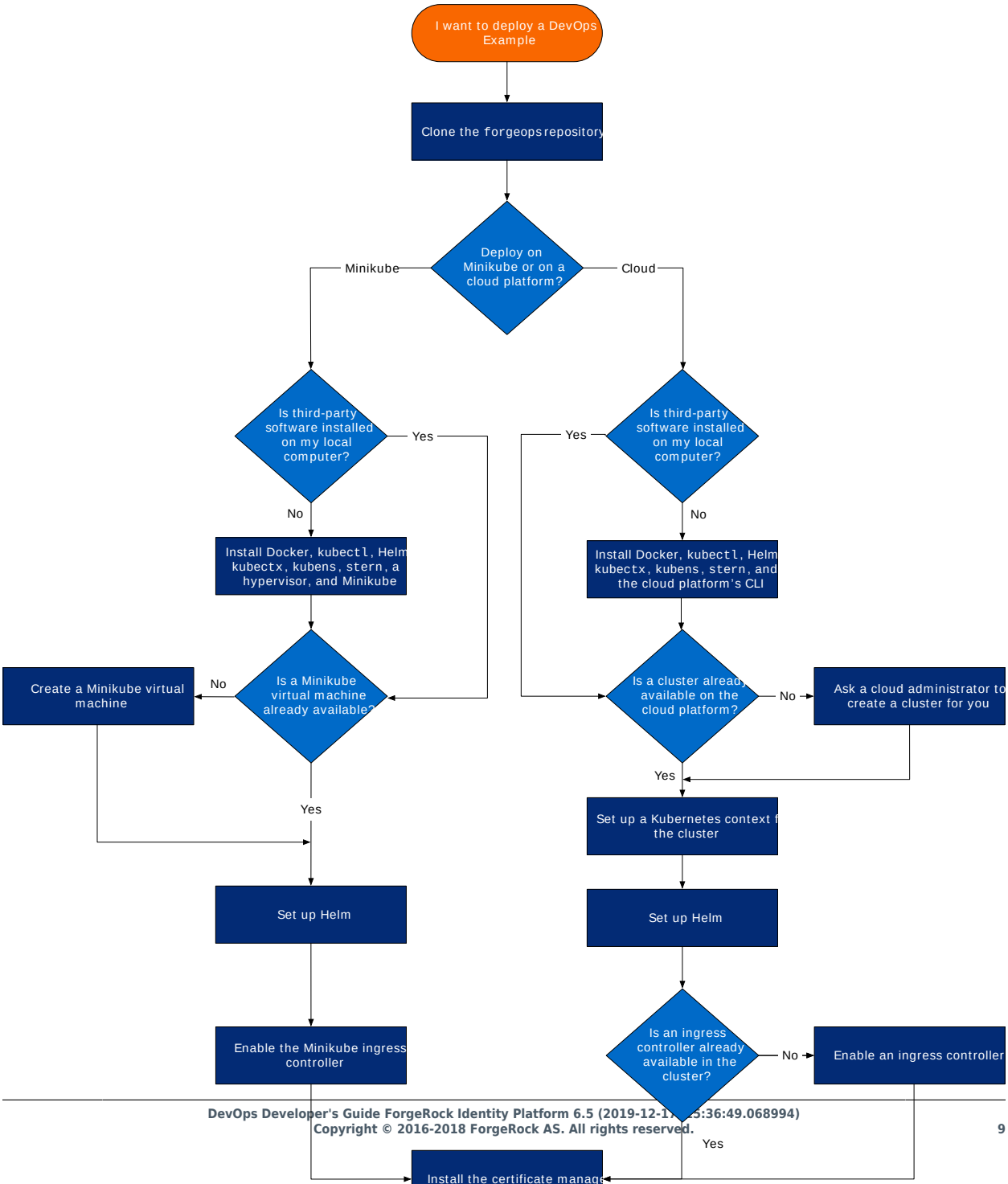
Task	Reference
Deploy a DevOps example and verify that the example is running.	"AM and DS Example Deployment Process", "IDM Example Deployment Process", and "IG Example Deployment Process"

Chapter 2

Implementing DevOps Environments

The following diagram illustrates a high-level workflow you'll use to implement a DevOps environment:

DevOps Environment Implementation



To implement a DevOps environment, perform the tasks listed in one of the following tables:

- Minikube: "Setting up a Minikube Environment"
- Cloud: "Setting up a Cloud Environment"

Setting up a Minikube Environment

Task	Steps
Clone the `forgeops` repository.	Follow the instructions in "Cloning the forgeops Repository".
Install third-party software (if necessary).	Follow the instructions in "Installing Required Third-Party Software".
Create a Minikube virtual machine (if necessary).	Follow the instructions in "Configuring Your Kubernetes Cluster".
Set up Helm.	Follow the instructions in "Setting up Helm".
Enable the Minikube ingress controller.	Perform "Deploying an Ingress Controller".
Install the certificate manager.	Perform "To Install the Certificate Manager".
Create a Kubernetes namespace.	Perform "To Create a Namespace to Run the DevOps Examples".
If you're using a private Docker registry for ForgeRock images, create a secret to enable registry access. Then add the secret to your namespace's default service account.	Perform "To Create a Kubernetes Secret for Accessing a Private Docker Registry".
Create a configuration repository.	Follow the instructions in "Creating Your Configuration Repository".
Configure and install the `frconfig` Helm chart.	Follow the instructions in "Installing the frconfig Helm Chart".

Setting up a Cloud Environment

Task	Steps
Clone the `forgeops` repository.	Follow the instructions in "Cloning the forgeops Repository".
Install third-party software (if necessary).	Follow the instructions in "Installing Required Third-Party Software".
Ask an administrator to create a cluster for you.	Refer to your cloud provider's documentation for more information.
Set up a Kubernetes context for your cluster.	Follow the instructions in "Setting up a Kubernetes Context".
Set up Helm.	Follow the instructions in "Setting up Helm".
Enable an ingress controller.	Perform the relevant procedure in "Deploying an Ingress Controller".

Task	Steps
Install the certificate manager.	Perform "To Install the Certificate Manager".
Create a Kubernetes namespace.	Perform "To Create a Namespace to Run the DevOps Examples".
If you're using a private Docker registry for ForgeRock images, create a secret to enable registry access. Then add the secret to your namespace's default service account.	Perform "To Create a Kubernetes Secret for Accessing a Private Docker Registry".
Create a configuration repository.	Follow the instructions in "Creating Your Configuration Repository".
Configure and install the `frconfig` Helm chart.	Follow the instructions in "Installing the frconfig Helm Chart".

2.1. About Environments for Running the DevOps Examples

You can run the DevOps Examples on a local computer or in a cloud-based environment. This section provides an overview of the requirements for each environment type.

2.1.1. Local Environment Overview

Before you can deploy the DevOps Examples on a local computer, such as a laptop or desktop system, you must install the following software on the local computer:

Software	Purpose
VirtualBox	Runs a Minikube virtual machine that contains a Kubernetes cluster.
Minikube	Provides a virtual machine and sets it up with a Kubernetes cluster.
Docker	Builds Docker images and pushes them to a Docker registry.
kubectl client	Performs various operations on the Kubernetes cluster.
Helm	Installs the ForgeRock Identity Platform in the Kubernetes cluster.

The following Kubernetes objects must be present in your cluster:

Kubernetes Object	Purpose
Ingress controller	Provides IP routing and load balancing services to the cluster.
Namespace	Provides logical isolation for deployments.

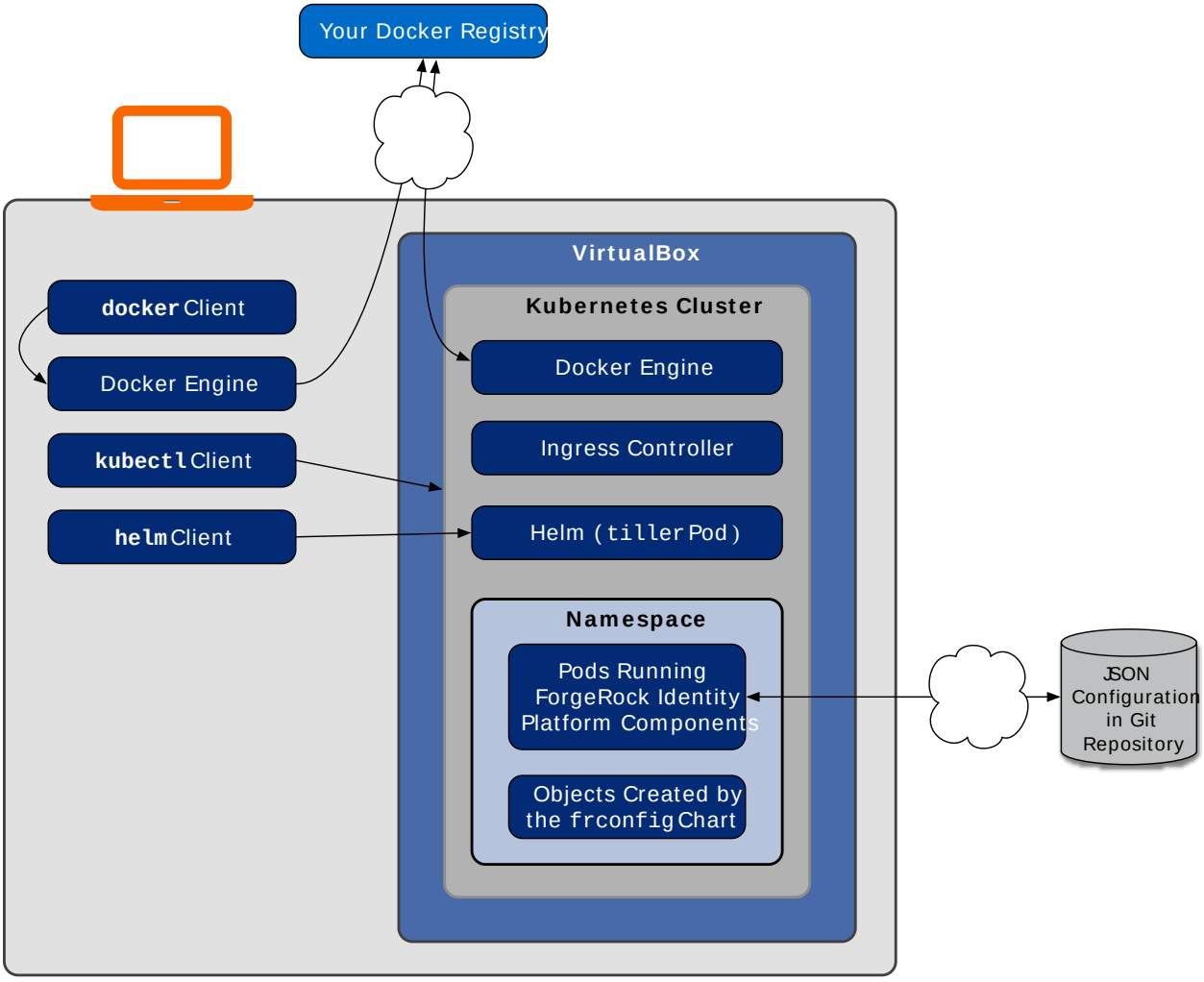
You must have accounts that allow you to use the following types of cloud-based services:

Cloud Service	Purpose
Docker registry	Stores Docker images.
Git repository	Stores JSON configuration for ForgeRock Identity Platform components.

The following diagram illustrates a local environment configured to support the DevOps Examples. The environment includes:

- Clients running on a local computer
- A Kubernetes cluster running in a virtual machine on the local computer
- A Docker registry and a Git repository running in the cloud

Local DevOps Environment



2.1.2. Cloud-Based Environment Overview

Before you can deploy the DevOps Examples in a cloud-based environment, you must install the following software on your local computer:

Software	Purpose
Docker	Builds Docker images and pushes them to a Docker registry.
kubectl client	Performs various operations on the Kubernetes cluster.
Helm	Installs the ForgeRock Identity Platform in the Kubernetes cluster.

You must have accounts that allow you to use the following types of cloud-based services:

Cloud Service	Purpose
Kubernetes hosting provider	Hosts Kubernetes clusters.
Docker registry	Stores Docker images.
Git repository	Stores JSON configuration for ForgeRock Identity Platform components.

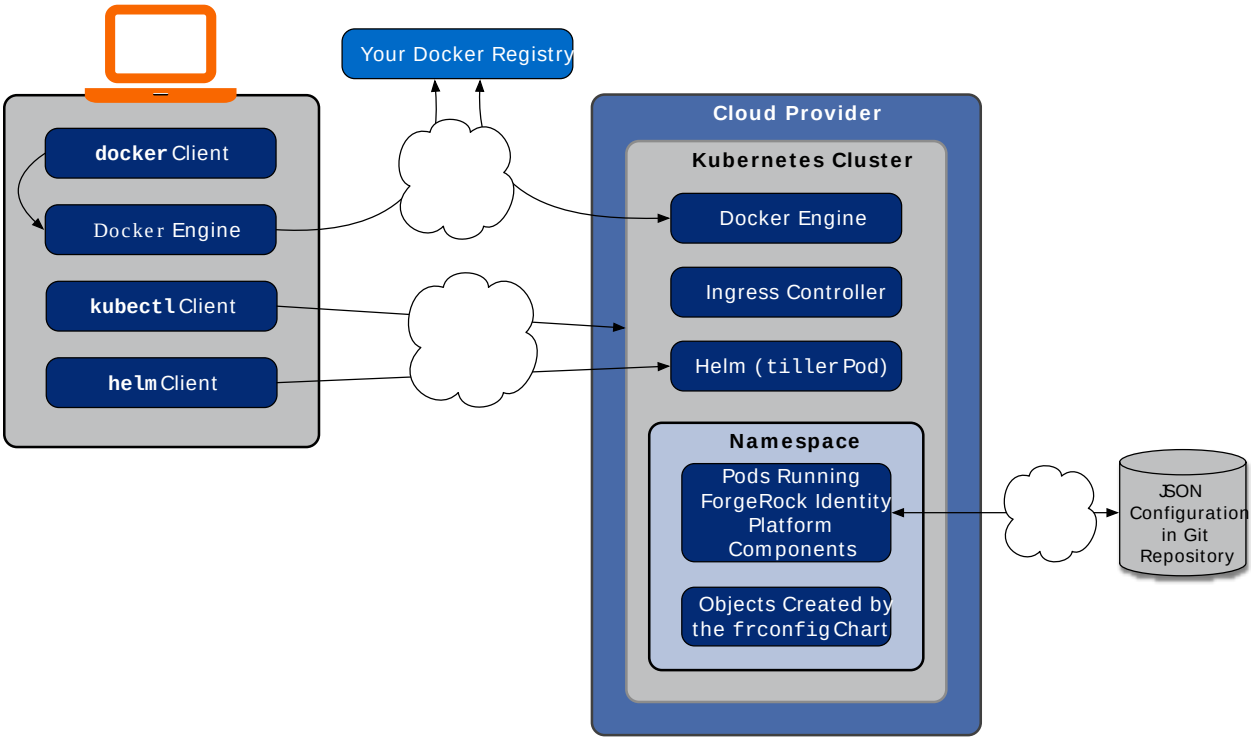
The following Kubernetes objects must be present in your cluster:

Kubernetes Object	Purpose
Ingress controller	Provides IP routing and load balancing services to the cluster.
Namespace	Provides logical isolation for deployments.

The following diagram illustrates a cloud-based environment configured to support the DevOps Examples. The environment includes:

- Clients running on a local computer
- A Kubernetes cluster running on a cloud-based Kubernetes hosting platform
- A Docker registry and a Git repository running in the cloud

Cloud-Based DevOps Environment



2.2. Cloning the forgeops Repository

ForgeRock's `forgeops` repository contains Dockerfiles, Helm charts, and other files required when deploying the DevOps Examples and the CDM.

See "forgeops Repository" in the *DevOps Release Notes* for more information about this repository and for instructions for cloning it to your local computer.

2.3. Installing Required Third-Party Software

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

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

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

2.4. Configuring Your Kubernetes Cluster

The DevOps Examples have been validated with Kubernetes clusters configured as follows:

Category	Requirement
Kubernetes version	See "Choosing the Kubernetes Version" in the *DevOps Release Notes*.
Memory	8 GB or more
Disk space	30 GB or more

The examples *might* also work with older or newer versions of Kubernetes.

On Minikube, use the following example command if you need to create a Kubernetes cluster that meets the minimum requirements for running the DevOps Examples:

```
$ minikube start --memory=8192 --disk-size=30g \
 --vm-driver=virtualbox --bootstrapper kubeadm --kubernetes-version=v1.15.0
##  minikube v1.2.0 on darwin (amd64)
##  Creating virtualbox VM (CPUs=2, Memory=8192MB, Disk=30000MB) ...
##  Configuring environment for Kubernetes v1.15.0 on Docker 18.09.6
##  Downloading kubeadm v1.15.0
##  Downloading kubelet v1.15.0
##  Pulling images ...
##  Launching Kubernetes ...
#  Verifying: apiserver proxy etcd scheduler controller dns
##  Done! kubectl is now configured to use "minikube"
```

In cloud-based environments, ask an administrator to create a Kubernetes cluster for you.

2.5. Setting up a Kubernetes Context

The **kubectl** command uses Kubernetes *contexts* to access Kubernetes clusters. Before you can access a Kubernetes cluster, a context for that cluster must be present on your local computer.

When you create a Kubernetes cluster, the command you use to create the cluster also creates a context for that cluster.

If you are using a Kubernetes cluster on a local environment, then you can assume you created the required Kubernetes context when you created the cluster. No further action is necessary. Note that on Minikube, the Kubernetes context is always named `minikube`; do not attempt to use a context with a different name.

If you are working with a Kubernetes cluster on a cloud-based system, a context referencing the cluster might not exist on your local computer. You must determine whether a context already exists. If no such context exists, then you must create one. Perform one of the following procedures:

- "To Set up Your Kubernetes Context on GKE"
- "To Set up Your Kubernetes Context on Amazon EKS"
- "To Set up Your Kubernetes Context on AKS"

To Set up Your Kubernetes Context on GKE

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted.
2. Take one of the following actions:
   a. If the current context references the cluster in which you want to deploy the DevOps Examples, there is nothing further to do.
   b. If the context of the cluster in which you want to deploy the DevOps Examples is present in the **kubectx** command output, set the context as follows:

      ```
      $ kubectx my-context
      Switched to context "my-context".
      ```

   c. If the context of the cluster in which you want to deploy the DevOps Examples is not present in the **kubectx** command output, set the context as follows:
      i. Configure the Google Cloud SDK standard component to use your Google account. Run the following command:

         ```
         $ gcloud auth login
         ```

      ii. A browser window prompts you to log in to Google. Log in using your Google account.

         A second screen requests several permissions. Click Allow.

         A third screen should appear with the heading, "You are now authenticated with the Google Cloud SDK!"

      iii. Contact the administrator who created the cluster and obtain the Google project name, cluster name, and the zone in which the cluster was created. You will need this information to create a context for the cluster.

      iv. Return to the terminal window and run the following command:

         ```
         $ gcloud container clusters \
          get-credentials cluster-name --zone google-zone --project google-project
         Fetching cluster endpoint and auth data.
         kubeconfig entry generated for cluster-name.
         ```

v. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

To Set up Your Kubernetes Context on Amazon EKS

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted.
2. Take one of the following actions:
   a. If the current context references the cluster in which you want to deploy the DevOps Examples, there is nothing further to do.
   b. If the context of the cluster in which you want to deploy the DevOps Examples is present in the **kubectx** command output, set the context as follows:

```
$ kubectx my-context
Switched to context "my-context".
```

   c. If the context of the cluster in which you want to deploy the DevOps Examples is not present in the **kubectx** command output, set the context as follows:
      i. Contact your AWS administrator and obtain the following information:
         - Your AWS access key ID
         - Your AWS secret access key
         - The AWS region where the EKS cluster resides
         - The cluster's name
      ii. Run the **aws configure** command. This command logs you in to AWS and sets the AWS region:

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

      For AWS Access Key ID, AWS Secret Access Key, and Default region name, enter the information you got from your administrator. You do not need to specify a value for the default output format.

      iii. Run the following command:

```
$ aws eks update-kubeconfig --name my-cluster
Added new context arn:aws:eks:us-east-1:813759318741:cluster/my-cluster
to /Users/my-user-name/.kube/config
```

iv. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

In Amazon EKS environments, the cluster owner must grant access to a user before the user can access cluster resources. For details about how the cluster owner can grant you access to the cluster, refer the cluster owner to "Granting Access to Multiple Users" in the *Site Reliability Guide for Amazon EKS*.

To Set up Your Kubernetes Context on AKS

1. Run the **kubectx** command and review the output. The current Kubernetes context is highlighted.
2. Take one of the following actions:
   a. If the current context references the cluster in which you want to deploy the DevOps Examples, there is nothing further to do.
   b. If the context of the cluster in which you want to deploy the DevOps Examples is present in the **kubectx** command output, set the context as follows:

```
$ kubectx my-context
Switched to context "my-context".
```

   c. If the context of the cluster in which you want to deploy the DevOps Examples is not present in the **kubectx** command output, set the context as follows:
      i. Configure the Azure CLI to use your Microsoft Azure. Run the following command:

```
$ az login
```

      ii. A browser window prompts you to log in to Azure. Log in using your Microsoft account.

      A second screen should appear with the message, "You have logged into Microsoft Azure!"

      iii. Contact the administrator who created the cluster and obtain:

      - The ID of the Azure subscription in which the cluster was created. Be sure to obtain the hexadecimal subscription ID, not the subscription name.
      - The name of the resource group in which the cluster was created.
      - The name of the cluster.

      You will need this information to create a context for the cluster.

      iv. Return to the terminal window and run the following command:

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

v. Run the **kubectx** command again and verify that the context for your Kubernetes cluster is now the current context.

2.6. Setting up Helm

Helm setup varies by environment. Follow the procedure for your environment:

- "To Set up Helm on Minikube"
- "To Set up Helm in a Cloud Environment"

To Set up Helm on Minikube

1. Run the following command to determine whether Helm is already running:

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

2. If the **kubectl** command returned no output, initialize Helm in your Minikube cluster:

```
$ helm init --upgrade --service-account default
$HELM_HOME has been configured at $HOME/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your
-helm-installation
```

To Set up Helm in a Cloud Environment

1. Run the following command to determine whether Helm is already running:

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

2. If the **kubectl** command returned no output, then no Helm `tiller` pod is running. Ask an administrator to start one, and to set up role-based access control (RBAC) to let users access resources. For more information about setting up RBAC for Helm, see the Helm documentation .

   If there is no active `tiller` pod in your cluster, you will not be able to deploy ForgeRock Identity Platform into the cluster.

3. Initialize the Helm client on your local computer:

```
$ helm init --client-only --upgrade
$HELM_HOME has been configured at $HOME/.helm.
Not installing Tiller due to 'client-only' flag having been set
Happy Helming!
```

2.7. Deploying an Ingress Controller

Ingress controller deployment varies by environment.

On Minikube, run the following command to enable the built-in ingress controller plugin:

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

In cloud-based environments, ask your cluster administrator to start an ingress controller if needed. The following links have example commands to start ingress controllers in cloud environments:

- "Deploying an Ingress Controller" in the *Cloud Deployment Model Cookbook for GKE*
- "Deploying an Ingress Controller" in the *Cloud Deployment Model Cookbook for Amazon EKS*
- "Deploying an Ingress Controller" in the *Cloud Deployment Model Cookbook for AKS (Evaluation Edition)*

2.8. Installing the Certificate Manager

The certificate manager generates or dynamically obtains a TLS certificate for securing communication from users to ForgeRock Identity Platform servers in the DevOps Examples or the CDM.

For more information about certificate generation options, see "About Securing Communication With ForgeRock Services".

Perform the following procedure to install the certificate manager:

To Install the Certificate Manager

- Install the `cert-manager` Helm chart:

```
$ helm install stable/cert-manager --namespace kube-system --version v0.5.0
NAME:   listless-magpie
LAST DEPLOYED: Mon Jul 29 17:31:08 2019
NAMESPACE: kube-system
STATUS: DEPLOYED
```

```
RESOURCES:
==> v1/ServiceAccount
NAME                          AGE
listless-magpie-cert-manager  0s

==> v1beta1/ClusterRole
listless-magpie-cert-manager  0s

==> v1beta1/ClusterRoleBinding
listless-magpie-cert-manager  0s

==> v1beta1/Deployment
listless-magpie-cert-manager  0s

==> v1/Pod(related)

NAME                                          READY  STATUS             RESTARTS  AGE
listless-magpie-cert-manager-8499846974-nrv8d  0/1    ContainerCreating  0         0s


NOTES:
cert-manager has been deployed successfully!

In order to begin issuing certificates, you will need to set up a ClusterIssuer
or Issuer resource (for example, by creating a 'letsencrypt-staging' issuer).

More information on the different types of issuers and how to configure them
can be found in our documentation:

https://cert-manager.readthedocs.io/en/latest/reference/issuers.html

For information on how to configure cert-manager to automatically provision
Certificates for Ingress resources, take a look at the `ingress-shim`
documentation:

https://cert-manager.readthedocs.io/en/latest/reference/ingress-shim.html
```

2.9. Creating a Namespace

Perform the following procedure to create a namespace:

To Create a Namespace to Run the DevOps Examples

1. Create a namespace in your Kubernetes cluster:

```
$ kubectl create namespace my-namespace
namespace/my-namespace created
```

2. Make the new namespace your current namespace:

```
$ kubens my-namespace
Context "my-context" modified.
Active namespace is "my-namespace".
```

2.10. Enabling Access to a Private Docker Registry

If your Docker images for the ForgeRock Identity Platform are stored in a private Docker registry, you must set up your namespace to access the registry.

For more information about setting up namespaces to pull images from private Docker registries, see Pull an Image from a Private Registry and Add ImagePullSecrets to a service account in the Kubernetes documentation.

You do *not* need to perform this procedure if the Docker images for the ForgeRock Identity Platform are stored in a public registry.

Note that the ForgeRock evaluation-only Docker images are available from ForgeRock's public registry, so if you are deploying the DevOps Examples using evaluation-only images, do not perform this procedure. For more information about the evaluation-only images, see "Using the Evaluation-Only Docker Images".

To Create a Kubernetes Secret for Accessing a Private Docker Registry

Perform the following steps to enable your namespace to access Docker images from a private registry:

1. Review the example **registry.sh** script. This script creates a Kubernetes image pull secret and associates it with a service account. The path to the script is `/path/to/forgeops/bin/registry.sh`.

   ForgeRock provides this script as an unsupported example. For more information about ForgeRock support for the DevOps Examples, the CDM and the `forgeops` repository, see "*Getting Support*".

2. If necessary, adjust code in the **registry.sh** script.

3. Set the following environment variables in your shell:

Variable	Description
`REGISTRY`	The fully-qualified domain name of your private Docker registry
`REGISTRY_ID`	Your Docker registry username
`REGISTRY_PASSWORD`	Your Docker registry password
`REGISTRY_EMAIL`	Your e-mail address

   For example:

   ```
   $ export REGISTRY=example-docker-registry.io
   $ export REGISTRY_ID=my-user-id
   $ export REGISTRY_PASSWORD=my-password
   $ export REGISTRY_EMAIL=my-email@example.com
   ```

4. Run the `/path/to/forgeops/bin/registry.sh` script to create a secret and configure the default service account in your namespace to use the secret's name as its `imagePullSecrets` value:

```
$ ./registry.sh
secret "frregistrykey" created
serviceaccount "default" replaced
Done
```

2.11. Creating Your Configuration Repository

When you deploy the ForgeRock Identity Platform, the AM, IDM, and IG configurations are retrieved from JSON files. The JSON files are stored in a cloud-based Git *configuration repository*.

Duplicate, fork, or use ForgeRock's sample configuration repository for your configuration repository.

Choose one of the following options when determining how to create your configuration repository:

Private read and private write access to your customized AM, IDM, and IG configurations

Choose this option:

- For developer and production deployments.
- When you do not want your custom configurations to be publicly visible.

To implement, duplicate ForgeRock's sample configuration repository and then configure the duplicate repository as a private repository. For more information about duplicating a GitHub repository, see the GitHub documentation.

Public read and private write access to your customized AM, IDM, and IG configurations

Choose this option:

- For developer and production deployments.
- When you do not care if your custom configurations are publicly visible.

To implement, fork ForgeRock's sample configuration repository.

Read-only access to ForgeRock's sample AM, IDM, and IG configurations

Choose this option for the simplest demonstration deployments only.

To implement, simply use ForgeRock's sample configuration repository

2.12. Installing the frconfig Helm Chart

Several configuration items must be present in your namespace for use by other pods:

- Your configuration repository's URL

- The branch in your configuration repository that contains the AM, IDM, and IG configurations
- The private key needed to access your configuration repository
- A signing certificate that the certificate manager uses to create an SSL certificate to secure communication with ForgeRock services

Installing the `frconfig` Helm chart deploys these items. After installing this chart, the items can be accessed by other pods in your namespace.

This section describes these configuration items. It also includes procedures for configuring and installing the `frconfig` chart.

2.12.1. About the Configuration Repository URL and Branch

The configuration repository URL and branch identifies the location from which AM, IDM, and IG obtain their configurations.

See "To Configure the Configuration Repository URL and Branch" for steps to configure the configuration repository URL and branch.

For more information about configuration repositories, see "Creating Your Configuration Repository".

2.12.2. About the Configuration Repository's Private Key

The pods that run ForgeRock Identity Platform servers need to clone the configuration repository to obtain the AM, IDM, and IG configurations. If either of the following conditions apply to your deployment, you'll need to place your repository's private key in the `frconfig` Helm chart so that pods can clone the repository:

- Your configuration repository is a private repository.
- You plan to export the ForgeRock Identity Platform configuration from the DevOps Examples to the configuration repository.

The private key must be stored in a file named `id_rsa`. It must also be created without a passphrase.

See "To Configure the Configuration Repository Private Key" for steps to configure the configuration repository private key.

2.12.3. About Securing Communication With ForgeRock Services

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

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

Over HTTPS using a self-signed certificate

The certificate manager generates a self-signed certificate using the default signing certificate in the `frconfig` Helm chart.

Communication is encrypted, but users will receive warnings about insecure communication from some browsers.

This is the default configuration for the DevOps Examples.

Over HTTPS using a certificate with a trust chain that starts at a trusted root certificate

The certificate manager generates a certificate with a trust chain that starts at a trusted root certificate using an intermediate signing certificate that you must add to the `frconfig` Helm chart.

Communication is encrypted, and users will not receive warnings from their browsers.

See "To Secure Communication Using a Trust Chain" for steps to configure the `frconfig` chart for this option.

Over HTTPS using a dynamically obtained certificate from Let's Encrypt

The certificate manager calls Let's Encrypt to obtain a certificate.

Communication is encrypted and users will not receive warnings from their browsers.

Configuring the DevOps Examples to dynamically obtain a certificate from Let's Encrypt is complex and requires a unique, public DNS domain. Using Let's Encrypt for certificate management is suitable for production deployments. It is how certificate management is implemented in the CDM. Using a self-signed certificate or a certificate with a trusted root is much easier to configure than obtaining a certificate from Let's Encrypt, and is therefore preferable for developer environments.

For more information about using Let's Encrypt for certificate management, see:

- "Automating Certificate Management" in the *Site Reliability Guide for GKE*
- "Automating Certificate Management" in the *Site Reliability Guide for Amazon EKS*

2.12.4. Configuring and Installing the frconfig Helm Chart

Perform the following procedures:

1. "To Configure the Configuration Repository URL and Branch"
2. "To Configure the Configuration Repository Private Key"
3. "To Secure Communication Using a Trust Chain"
4. "To Install the frconfig Helm Chart"

To Configure the Configuration Repository URL and Branch

- Create the `frconfig.yaml` file. Set values with your configuration repository's location. For example:

```
domain: .my-domain.com
git:
  repo: git@github.com:myAccount/forgeops-init.git
  branch: my-branch
```

  For information about `frconfig.yaml` file options, see "frconfig.yaml" in the YAML File Reference.

To Configure the Configuration Repository Private Key

Perform this procedure if your configuration repository is either a private repository or a read/write public repository.

Do not perform this procedure if your configuration repository is a public read-only repository.

1. Review the flowchart at the start of "*Implementing DevOps Environments*" and verify that you need to copy your configuration repository private key to the `frconfig` Helm chart.
2. Verify that your configuration repository's private key is stored in a file named `id_rsa`.
3. Verify that your configuration repository's private key does not require a passphrase.
4. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
5. In your `forgeops` repository clone, change to the `/path/to/forgeops/helm/frconfig/secrets` directory.
6. Copy your configuration repository's private key into the `/path/to/forgeops/helm/frconfig/secrets` directory:

```
$ cp /path/to/config-repo-key/id_rsa id_rsa
```

   It is safe to overwrite the `id_rsa` file that's already in the directory.

   You'll remove the private key file when you perform Step 4 of "To Install the frconfig Helm Chart".

   Important

   Do not use the **git add** and **git commit** commands while the private key file is temporarily present in your `forgeops` repository clone. The file is needed only during `frconfig` Helm chart installation, and should not be committed to your repository.

To Secure Communication Using a Trust Chain

Perform this procedure if you want to implement secure communication using a trust chain.

Do not perform this procedure if you want to implement secure communication using a either self-signed certificate or Let's Encrypt.

1. Review the flowchart at the start of "*Implementing DevOps Environments*" and verify that you need to add a signing certificate to the `frconfig` Helm chart.

2. Obtain an intermediate signing certificate from your certificate authority. The certificate should have two files named `ca.crt` and `ca.key`.

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

4. In your `forgeops` repository clone, change to the `/path/to/forgeops/helm/frconfig/secrets` directory.

5. Rename the files that contain the default certificate in this directory:

```
$ mv ca.crt ca.crt.self-signed
$ mv ca.key ca.key.self-signed
```

6. Temporarily copy the signing certificate files from your certificate authority into the directory:

```
$ cp /path/to/trusted-root-cert/ca.crt ca.crt
$ cp /path/to/trusted-root-cert/ca.key ca.key
```

   You'll remove the signing certificate files when you perform Step 5 of "To Install the frconfig Helm Chart".

   Important

   Do not use the **git add** and **git commit** commands while the signing certificate files are temporarily present in your `forgeops` repository clone. These files are needed only during `frconfig` Helm chart installation, and should not be committed to your repository.

To Install the frconfig Helm Chart

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

2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install the `frconfig` Helm chart:

```
$ helm install frconfig --values /path/to/frconfig.yaml
NAME:   pouring-vulture
LAST DEPLOYED: Mon Jul 29 17:35:27 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

RESOURCES:
==> v1alpha1/Certificate
NAME                              AGE
wildcard.my-namespace.example.com  0s

==> v1alpha1/Issuer
ca-issuer  0s

==> v1/Secret
NAME                    TYPE                DATA  AGE
certmanager-ca-secret  kubernetes.io/tls  2     0s
frconfig                Opaque              1     0s
frconfig-platform      Opaque              0     0s

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

4. If you copied a private key into the `/path/to/forgeops/helm/frconfig/secrets` directory, remove it:

```
$ cd /path/to/forgeops/helm/frconfig/secrets
$ rm id_rsa
```

5. If you copied a signing certificate into the `/path/to/forgeops/helm/frconfig/secrets` directory, remove it:

```
$ cd /path/to/forgeops/helm/frconfig/secrets
$ rm ca.crt
$ rm ca.key
```

2.13. Deleting ForgeRock Identity Platform From Your Namespace

To delete ForgeRock Identity Platform deployments from a Kubernetes cluster, run the example cleanup script, **remove-all.sh**.

ForgeRock provides this script as an unsupported example. For more information about ForgeRock support for the DevOps Examples, the CDM and the `forgeops` repository, see "*Getting Support*".

By default, the **remove-all.sh** script deletes most ForgeRock-related objects from your namespace. Use the `-N` option to delete all objects from your namespace and then delete the namespace as well:

Object	Deleted when running the remove-all.sh script without the `-N` option?	Deleted when running the remove-all.sh script with the `-N` option?
Pods running ForgeRock Identity Platform components	Yes	Yes
ForgeRock-related stateful sets and persistent volume claims	Yes	Yes
Secrets created by Helm charts (not including the `frconfig` secret)	Yes	Yes
Objects created by installing non-ForgeRock Helm charts	Yes	Yes
All Kubernetes objects created by installing the `frconfig` Helm chart	Yes	Yes
The image pull secret used to access Docker images in a private Docker registry	No	Yes
Objects created by means other than installing Helm charts	No	Yes
Your Kubernetes namespace	No	Yes

To run the cleanup script, perform the following procedure:

To Remove ForgeRock-Related Objects From a Namespace

1. Review the example **remove-all.sh** script. This script deletes residual Kubernetes objects from previous ForgeRock deployments. The path to this script is `/path/to/forgeops/bin/remove-all.sh`.

   ForgeRock provides this script as an unsupported example. For more information about ForgeRock support for the DevOps Examples, the CDM and the `forgeops` repository, see "*Getting Support*".

2. If necessary, adjust code in the **remove-all.sh** script. For example, if you need to retain one or more PVCs from a previous deployment, remove the part of the script that deletes PVCs.

3. Run the **remove-all.sh** script without the `-N` option to delete Kubernetes objects remaining from previous ForgeRock deployments from the active namespace. For example:

   ```
   $ cd /path/to/forgeops/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. The message `Error: release: not found` does not indicate an actual error—it simply indicates that the script attempted to delete Kubernetes objects that did not exist in the cluster.

4. Run the **kubectl get pods** command to verify that no pods that run ForgeRock software [1] are active in the namespace into which you intend to deploy the example.

   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 your namespace 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 your namespace 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
   ```

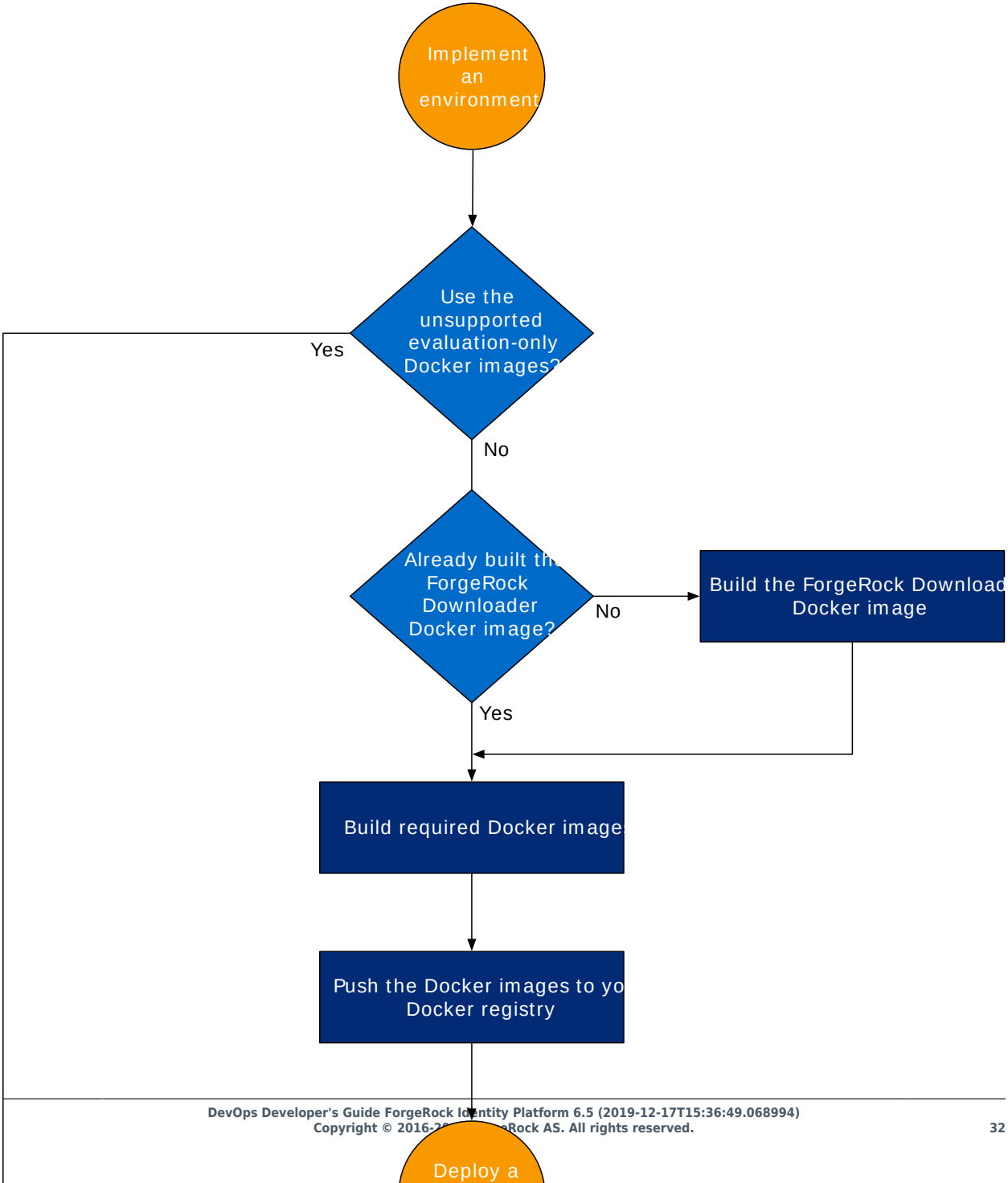
[1] See the deployment diagrams in the introductory section for each DevOps example for the names of pods that run ForgeRock software.

Chapter 3

Building and Pushing Docker Images

The following diagram illustrates a high-level workflow you'll use to build and push Docker images for the ForgeRock Identity Platform:

Building and Pushing Docker Images



To build and push Docker images for the DevOps Examples, perform the following tasks:

Task	Steps
Build the ForgeRock `downloader` Docker image.	"To Build the ForgeRock downloader Docker Image".
Build required Docker images.	"To Build Docker Images for the DevOps Examples".
Push the Docker images to a Docker registry.	Follow the instructions in "Pushing Docker Images".

This chapter does not cover Docker image customization. If you need to customize any of the Dockerfiles provided by ForgeRock, refer to the following resources in the `forgeops` repository:

- `README.md` files
- Comments in the Dockerfiles

3.1. About Docker Images for the Examples

Each DevOps example requires you to build one or more Docker images:

DevOps Example	Required Images	
AM and DS	`openam`, `amster`, `ds`	
IDM	`openidm`, `ds`	
IG	`openig`	

3.1.1. Utility Images Used by the Examples

In addition to the Docker images listed in the preceding table, Dockerfiles and Helm charts in the `forgeops` repository reference the following utility images:

- The `java` image, which includes OpenJDK software. The `ds`, `amster`, and `openidm` Docker images are based on the `java` image.
- The `git` image, which provides functionality to clone the configuration repository. The `openam`, `amster`, `openidm`, and `openig` Helm charts contain init containers that use the `git` image.
- The `util` image, which provides functionality required for AM bootstrapping. The `openam` Helm chart contains an init container that uses the `util` image.

You do not need to build the utility images. They're downloaded from ForgeRock's public regsitry when needed.

3.2. Using the Evaluation-Only Docker Images

You can use the following Docker images, available from ForgeRock's public registry, to evaluate the ForgeRock Identity Platform:

- `gcr.io/forgerock-io/openam:6.5.2`
- `gcr.io/forgerock-io/amster:6.5.2`
- `gcr.io/forgerock-io/ds:6.5.2`
- `gcr.io/forgerock-io/openidm:6.5.2`
- `gcr.io/forgerock-io/openig:6.5.2`

These Docker images are for evaluation purposes only and are *not* supported by ForgeRock.

Production deployments of the ForgeRock Identity Platform must not use these images. When deploying the ForgeRock Identity Platform in production, build Docker images and push them to your Docker registry, following the instructions in "Building the ForgeRock downloader Docker Image", "Building Docker Images", and "Pushing Docker Images".

For more information about support for Docker images for the ForgeRock Identity Platform, see "ForgeRock DevOps Support".

3.3. Building the ForgeRock downloader Docker Image

The `downloader` Docker image obtains binary files for ForgeRock software from the ForgeRock Maven repository. The binary files are needed when building the `openam`, `amster`, `openidm`, `openig`, and `ds` Docker images. You must therefore build a `downloader` Docker image before you can build the Docker images for the ForgeRock Identity Platform.

You need only build the `downloader` Docker image once; you do not need to build it every time you want to build Docker images for the ForgeRock Identity Platform.

To Build the ForgeRock downloader Docker Image

1. Obtain your API key from ForgeRock's Maven repository on Artifactory[1]:
   a. Log in to Artifactory using your BackStage username and password. Be sure to use your username, not your email address, when logging in.
   b. Navigate to your Artifactory user profile by clicking your username.
   c. Enter your password in the Current Password field.
   d. Click Unlock.
   e. If an API key hasn't been created yet, generate one.

[1] If your BackStage account does not support logging in to Artifactory, you can download binary files manually and create Docker images. For more information, see the README.md file for the manual downloader sample.

f. Click the Show API Key icon.

g. Copy the value in the API Key field to the clipboard. You'll use it in the next step.

2. Set the `API_KEY` environment variable:

```
$ export API_KEY=my-artifactory-api-key
```

3. Change to the directory that contains the artifacts for building ForgeRock Docker images:

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

4. Build the `downloader` Docker image:

```
$ docker build --build-arg API_KEY=$API_KEY \
 --tag forgerock/downloader downloader
Sending build context to Docker daemon  9.216kB
Step 1/5 : FROM gcr.io/cloud-builders/gcloud:latest
 ---> 2096d81cfde0
Step 2/5 : ARG API_KEY
 ---> Using cache
 ---> 55f72f23f316
Step 3/5 : RUN echo $API_KEY >/api_key &&     apt-get -q update && apt-get -q install unzip
 ---> Using cache
 ---> 952308776d6f
Step 4/5 : COPY download /usr/bin/
 ---> Using cache
 ---> bd0cac4af919
Step 5/5 : ENTRYPOINT [ "/usr/bin/download" ]
 ---> Using cache
 ---> b9cc7c12038a
Successfully built b9cc7c12038a
Successfully tagged forgerock/downloader:latest
```

3.4. Building Docker Images

Perform the following procedure to build Docker images required to orchestrate the DevOps Examples:

To Build Docker Images for the DevOps Examples

Perform the following steps:

1. Review "About Docker Images for the Examples" and identify the Docker images required for the example you want to orchestrate.

2. Run the **docker images** command and verify that the `downloader` image is available.

   If it is not available, create it. See "To Build the ForgeRock downloader Docker Image".

3. Change to the directory that contains the artifacts for building ForgeRock Docker images:

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

4. Build required Docker images for running the DevOps Examples using the **docker build** command.

   The command's syntax is:

   ```
   $ docker build --tag registry/repository/openam:tag openam
   ```

   Specify values in the **docker build** command as follows:

   - For *registry*, specify the name of the Docker registry to which you will push the image, if required by your registry provider.

     Refer to your Docker registry provider's documentation for details.

   - For *repository*, specify the first qualifier in the name of the Docker repository to which you want to write the image.

     Recommendation: If possible, specify `forgerock`.

   - For *tag*, specify the Docker image's tag.

     Recommendation: If possible, specify `6.5.2`.

   The following example builds the `openam` Docker image:

   ```
   $ docker build --tag my-registry/forgerock/openam:6.5.2 openam
   Sending build context to Docker daemon   16.9kB
   Step 1/18 : FROM forgerock/downloader
    ---> b9cc7c12038a
   Step 2/18 : ARG VERSION="6.5.2"
    ---> Using cache
    ---> d1c246a34f1c
   Step 3/18 : RUN download -v $VERSION openam
    ---> Using cache
    ---> 21f3995262b7
   Step 4/18 : RUN mkdir -p /var/tmp/openam && unzip -q /openam.war -d /var/tmp/openam
    ---> Using cache
    ---> e240f5fbe3d6
   Step 5/18 : FROM tomcat:8.5-alpine
    ---> 4922cef0bc1e
   Step 6/18 : RUN rm -fr "$CATALINA_HOME"/webapps/*
    ---> Running in ce016c3ac7d0
   Removing intermediate container ce016c3ac7d0
    ---> 70dc23207eaa
   Step 7/18 : COPY --from=0 /var/tmp/openam "$CATALINA_HOME"/webapps/ROOT
    ---> eda9168ce4c0
   Step 8/18 : ENV CATALINA_OPTS -server -XX:+UnlockExperimentalVMOptions -XX:
   +UseCGroupMemoryLimitForHeap -Dcom.sun.identity.util.debug.provider=com.sun.identity.shared.debug.impl
   .StdOutDebugProvider -Dcom.sun.identity.shared.debug.file.format=\"%PREFIX% %MSG%\\n%STACKTRACE%\"
    ---> Running in 74f507542b29
   Removing intermediate container 74f507542b29
    ---> 3332924eaaf5
   Step 9/18 : ENV FORGEROCK_HOME /home/forgerock
   ```

```
 ---> Running in 7f9b42f7c9c7
Removing intermediate container 7f9b42f7c9c7
 ---> 00dec5861dfa
Step 10/18 : ENV OPENAM_HOME /home/forgerock/openam
 ---> Running in 924188ccb0e1
Removing intermediate container 924188ccb0e1
 ---> a8534a478ec0
Step 11/18 : RUN apk add --no-cache su-exec unzip curl bash tini && addgroup -g 11111 forgerock   &&
 adduser -s /bin/bash -h "$FORGEROCK_HOME" -u 11111 -D forgerock -G root && mkdir -p "$OPENAM_HOME"
   && mkdir -p "$FORGEROCK_HOME"/.openamcfg   && echo "$OPENAM_HOME" > "$FORGEROCK_HOME"/.openamcfg
/AMConfig_usr_local_tomcat_webapps_ROOT_ && chown -R forgerock:root "$CATALINA_HOME"   && chown -R
 forgerock:root  "$FORGEROCK_HOME"   && chmod -R g+rwx "$CATALINA_HOME"
 ---> Running in a004ab0ddd4f
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
(1/7) Installing nghttp2-libs (1.32.0-r0)
(2/7) Installing libssh2 (1.8.0-r3)
(3/7) Installing libcurl (7.61.1-r1)
(4/7) Installing curl (7.61.1-r1)
(5/7) Installing su-exec (0.2-r0)
(6/7) Installing tini (0.18.0-r0)
(7/7) Installing unzip (6.0-r4)
Executing busybox-1.28.4-r1.trigger
OK: 95 MiB in 69 packages
Removing intermediate container a004ab0ddd4f
 ---> d0a530dd722d
Step 12/18 : USER 11111
 ---> Running in 58e6e3935ec2
Removing intermediate container 58e6e3935ec2
 ---> f6dafa001f36
Step 13/18 : COPY server.xml "$CATALINA_HOME"/conf/server.xml
 ---> e4b4b5e8577a
Step 14/18 : COPY context.xml "$CATALINA_HOME"/conf/context.xml
 ---> 8e1cf371be27
Step 15/18 : ENV CUSTOMIZE_AM /home/forgerock/customize-am.sh
 ---> Running in 3de70f92c456
Removing intermediate container 3de70f92c456
 ---> 3b8c862d8f5d
Step 16/18 : COPY *.sh $FORGEROCK_HOME/
 ---> 61db4473fa7b
Step 17/18 : ENTRYPOINT ["/home/forgerock/docker-entrypoint.sh"]
 ---> Running in a9b96570d3e7
Removing intermediate container a9b96570d3e7
 ---> 9db2fef6cda0
Step 18/18 : CMD ["run"]
 ---> Running in 717684a30fde
Removing intermediate container 717684a30fde
 ---> a12445816915
Successfully built a12445816915
Successfully tagged forgerock/6.5.2
```

5. Run the **docker images** command to verify that the image or images that you built are present.

3.5. Pushing Docker Images

To push the Docker images to your Docker registry, see the registry provider documentation for detailed instructions.

For many Docker registries, you run the **docker login** to log in to the registry, and then run the **docker push** command to push a Docker image to the registry. However, some Docker registries have different requirements. For example, to push Docker images to a registry on Google Container Registry, you use Google Cloud SDK commands rather than the **docker push** command.

Be sure to push all required Docker images for the example you want to orchestrate. Review "About Docker Images for the Examples" and verify that you have pushed all images you need.

3.6. Rebuilding Docker Images

A Docker image's contents are static, so if you need to change the content in the image, you must rebuild it. Rebuild images when:

- You want to upgrade to newer versions of AM, Amster, IDM, IG or DS software.
- You changed files that impact an image's content. Some examples:
  - Changes to security files, such as passwords and keystores.
  - Changes to file locations or other bootstrap configuration in the AM `boot.json` file.
  - Dockerfile changes to install additional software on the images.

If you want to customize the AM web application, you can provision your configuration repository with a script that customizes AM at startup time. You do not need to change what's in the `openam` Docker image. For more information, see "Customizing the AM Web Application".

Chapter 4

Deploying the AM and DS Example

This chapter covers the following topics:

- "About the Example" provides information about the AM and DS example's architecture.
- "About AM Configuration" contains a description of the types of AM configuration, and describes how the deployment uses the Amster pod to manage AM configuration.
- "Working With the AM and DS Example" presents an example workflow that you might use when deploying AM using DevOps techniques.
- "Deploying the Example" provides instructions for deploying the example.
- "Modifying the AM Configuration" describes how to update the AM configuration stored in your configuration repository after you've deployed the example.
- "Customizing the AM Web Application" provides instructions for customizing the AM `.war` file, should you need to do so in the DevOps environment.
- "Redeploying the Example" provides instructions for redeploying the example.

4.1. About the Example

The AM and DS DevOps example has the following architectural characteristics:

- The AM and DS deployment runs in a Kubernetes namespace.
- From outside the deployment, AM is accessed through a Kubernetes ingress controller (load balancer):
  - The ingress controller is configured for session stickiness.
  - A single ingress controller can access every namespace running the example deployment within the cluster.
- The AM configuration is maintained in a Git repository, external to the Kubernetes cluster, called the *configuration repository*.
- The following Kubernetes pods are deployed in each namespace running the example:

`openam` pod(s)

At startup, the `git-init` container clones the configuration repository in order to obtain the AM deployment customization script, if it exists. The container then terminates.

"Customizing the AM Web Application" describes how to use a script that runs immediately before AM starts in order to customize AM.

The `openam` container:

1. Creates the `boot.json` file. The AM server uses this file during startup.
2. Invokes the AM deployment customization script if the script is present in the configuration repository clone.
3. Starts the AM server.

After startup, the `openam` container remains active to run the AM server.

Multiple instances of this pod can be started if required. The Kubernetes ingress controller redirects requests to one of these pods. The pods are elastic and the pod names are created dynamically by Kubernetes at run time.

`amster` pod

At startup, the `git-init` container clones the configuration repository and then terminates.

The `amster` container:

1. Installs AM.
2. Imports AM's configuration from the configuration repository clone.

After startup, the `amster` container remains active to run Amster commands, including commands to export the AM configuration to the configuration repository clone.

When users initiate requests to export the AM configuration, the `git` container pushes changes made to the AM configuration to the configuration repository.

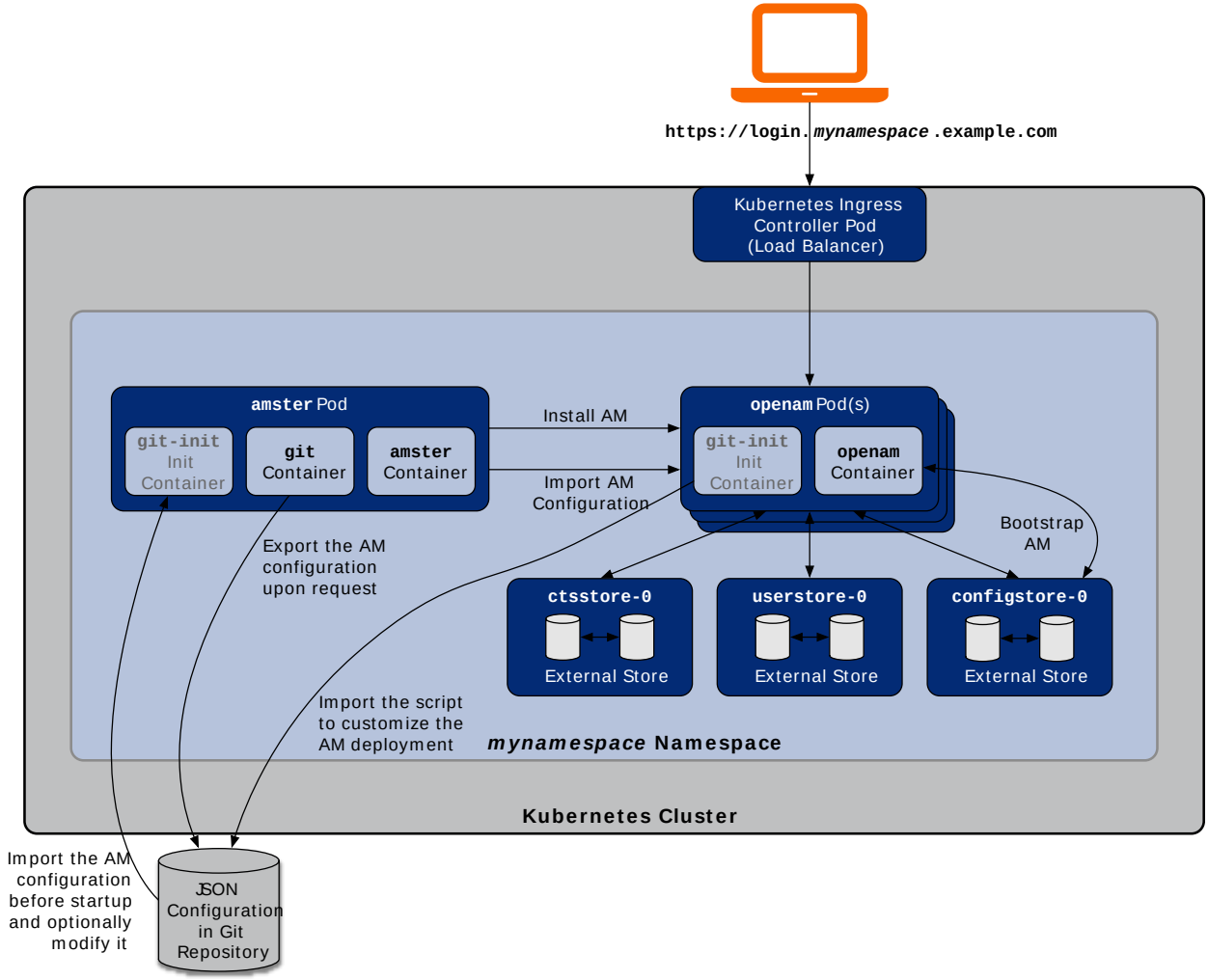
You can start multiple pod instances, but a single instance should be adequate for nearly every deployment scenario. The pods are elastic and the pod names are created dynamically by Kubernetes at run time.

External DS stores for AM configuration, policies, application data, users, and CTS tokens

All of the stores that AM uses are created as external stores that can optionally be replicated. These pods are not elastic. The pod names are fixed. Each pod name starts with the string `configstore-`, `userstore-`, or `ctsstore-` and is followed by a number. Note that the AM configuration, policies, and application data reside in the configuration store.

The following diagram illustrates the example.

AM and DS DevOps Deployment Example



4.2. About AM Configuration

AM uses two types of configuration values:

- *Installation configuration*, a small set of configuration values passed to the **amster install-openam** command, such as AM's server URL and cookie domain.

  In the DevOps Examples, installation configuration resides in Helm charts in the `forgeops` repository.

- *Post-installation configuration*, hundreds or even thousands of configuration values you can set using the AM console and REST API. Examples include realms other than the root realm, service properties, and server properties.

  In the DevOps Examples, post-installation configuration resides in JSON files maintained in a Git repository known as the *configuration repository*. For more information about the configuration repository, see "Creating Your Configuration Repository".

The AM server is started and initialized as follows:

1. The `amster` pod's `git-init` container clones the configuration repository to obtain the post-installation configuration.
2. The `amster` pod runs the **amster install-openam** command, which installs the AM server using installation configuration from Helm charts in the `forgeops` repository.
3. The `amster` pod runs the **amster import-config** command, which imports the post-installation configuration into the AM configuration store.
4. The `openam` pod starts the AM server.

4.3. Working With the AM and DS Example

This section presents an example workflow to set up a development environment in which you configure AM, iteratively modify the AM 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:

Step	Details
Get the latest version of the `forgeops` repository	Make sure you have the latest version of the `release/6.5.2` branch of the `forgeops` Git repository, which contains Docker, Helm, and Kubernetes artifacts needed to build and deploy all of the DevOps Examples. For more information about the `forgeops` Git repository, see "Public Git Repositories".

Step	Details
Implement a development environment	Set up a local or cloud-based environment for developing the AM configuration. See "*Implementing DevOps Environments*".
Obtain Docker images for the example	Make sure you have built Docker images for the example and pushed them them to your Docker registry. For details about creating and pushing Docker images, see "*Building and Pushing Docker Images*". As an alternative, if you are not running a production deployment, you can use unsupported, evaluation-only Docker images from ForgeRock. For more information, see "Using the Evaluation-Only Docker Images".
Deploy the AM and DS example in the development environment	Follow the procedures in "Deploying the Example".
Modify the AM configuration	Repeat the following steps until the configuration meets your needs: 1. Modify the AM configuration using the AM console, the REST API, or the **amster** command. See "To Access the AM Console" for details about how to access the deployed AM server. 2. Push the updates to the configuration repository's `autosave-my-namespace` branch as desired. 3. Merge configuration updates from the `autosave-my-namespace` branch to the branch containing the master version of your AM configuration. For more information about modifying the configuration, see "Modifying the AM Configuration".
Customize the AM web application (optional)	If needed, modify the AM web application to customize AM. For more information about modifying the AM web application, see "Customizing the AM Web Application".
Implement one or more production environments	Set up cloud-based environments for test and production deployments. See "*Implementing DevOps Environments*".
Deploy the AM and DS example in the production environments	Follow the procedures in "Deploying the Example".

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

- Modify the AM configuration on the Minikube deployment and merge changes into the master version of your AM configuration.
- Redeploy the AM and DS example in your cloud environment based on the updated configuration.

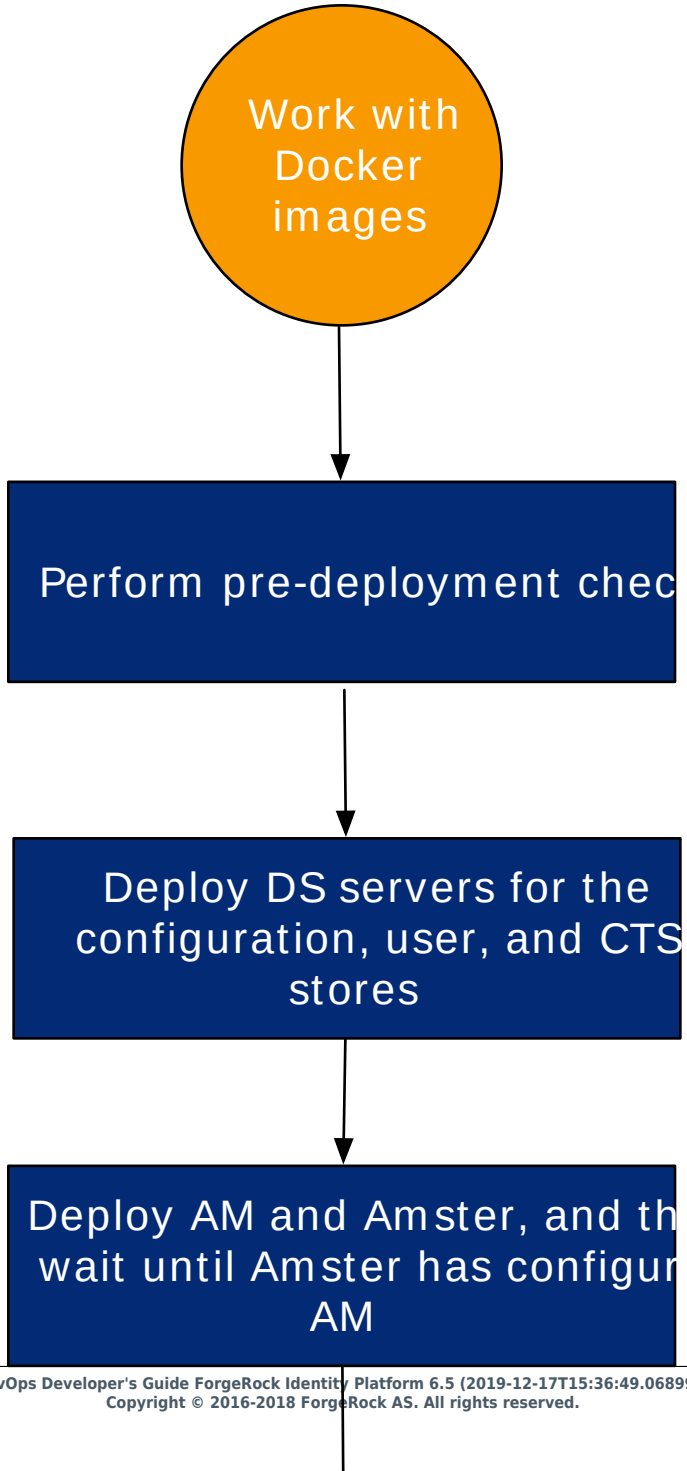
4.4. Deploying the Example

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

- "AM and DS Example Deployment Process"
- "Performing Pre-Deployment Checks"
- "Using Helm to Deploy the AM and DS Example"
- "Scaling AM"

The following diagram illustrates a high-level workflow you'll use to deploy the AM and DS example:

AM and DS Example Deployment Process



To deploy the AM and DS example, perform the following tasks:

Task	Steps
Perform pre-deployment checks.	Follow the instructions in "Performing Pre-Deployment Checks".
Deploy DS servers for the configuration, user, and CTS stores.	Follow the instructions in "Deploying DS Servers Used in the Example".
Deploy AM and Amster, and then wait until Amster has configured AM.	Follow the instructions in "Deploying AM and Amster".
Verify AM is up and has been configured.	Perform "To Verify Successful AM and DS Deployment".
Configure the `hosts` file.	Perform "To Configure the Hosts File".
Access the AM console.	Perform "To Access the AM Console".
Increase or decrease the number of AM replicas.	Follow the instructions in "Scaling AM".

4.4.1. Performing Pre-Deployment Checks

Review the requirements for deployment environments in the following table. If your environment does not meet any of these requirements, perform the necessary procedures before deploying the AM and DS example:

Requirement	Procedure
A `forgeops` repository clone is available on your local computer.	"Cloning the forgeops Repository".
A namespace has been created in your Kubernetes cluster, and it is configured as the active namespace.	Run the **kubens** command and verify that your namespace is the active namespace. If it is not, run the **kubens *my-namespace*** command. If you have not created a namespace, perform "To Create a Namespace to Run the DevOps Examples".
Helm is set up and running in your Kubernetes cluster.	"Setting up Helm".
The Kubernetes context is set up on your local computer.	"Setting up a Kubernetes Context".
An image pull secret is available in your namespace. (Required only if your Docker images are stored in a private Docker registry.)	"To Create a Kubernetes Secret for Accessing a Private Docker Registry".
The `frconfig` Helm chart is installed in your namespace.	"Configuring and Installing the frconfig Helm Chart".
No ForgeRock-related Kubernetes objects from previous deployments are present in your namespace, other than the Kubernetes secrets for configuration	"To Remove ForgeRock-Related Objects From a Namespace".

Requirement	Procedure
repository access, TLS support, and Docker image access.	

4.4.2. Using Helm to Deploy the AM and DS Example

The following sections guide you through deployment of the AM and DS example:

- "Deploying DS Servers Used in the Example"
- "Deploying AM and Amster"

4.4.2.1. Deploying DS Servers Used in the Example

The DS instances for the configuration, user, and CTS stores form the AM persistence tier. In the AM and DS example, the AM configuration, policies, and application data reside in the configuration store.

Perform the following procedures to set up the persistence tier for this example:

1. "To Create .yaml Files for Installing DS Servers Used in the AM and DS Example"
2. "To Install the DS Servers' Helm Charts"

To Create .yaml Files for Installing DS Servers Used in the AM and DS Example

Perform the following steps:

1. Create the `configstore.yaml` file. This file contains information needed to set up the DS server used for the AM configuration store.

   Example `configstore.yaml` file:

   ```
   instance: configstore
   image:
     repository: my-registry/forgerock/ds
   ```

   For information about `configstore.yaml` file options, see "configstore.yaml" in the YAML File Reference.

2. Create the `userstore.yaml` file. This file contains information needed to set up the DS server used for the AM user store.

   Example `userstore.yaml` file:

   ```
   instance: userstore
   image:
     repository: my-registry/forgerock/ds
   ```

For information about `userstore.yaml` file options, see "userstore.yaml" in the YAML File Reference.

3. Create the `ctsstore.yaml` file. This file contains information needed to set up the DS server used for the AM Core Token Service store.

   Example `ctsstore.yaml` file:

   ```
   instance: ctsstore
   image:
     repository: my-registry/forgerock/ds
   ```

   For information about `ctsstore.yaml` file options, see "ctsstore.yaml" in the YAML File Reference.

To Install the DS Servers' Helm Charts

Perform the following steps:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install three `ds` Helm charts to create pods for the configuration, user, and CTS stores:

   a. Install the directory server for the configuration store:

      ```
      $ helm install ds --values /path/to/configstore.yaml
      ```

   b. Install the directory server for the user store:

      ```
      $ helm install ds --values /path/to/userstore.yaml
      ```

   c. Install the directory server for the CTS store:

      ```
      $ helm install ds --values /path/to/ctsstore.yaml
      ```

   Output from each **helm install** command is similar. The following sample output is from the command that installs the `configstore-0` pod for the configuration store:

```
NAME:   hardy-chicken
LAST DEPLOYED: Tue Jul 30 16:42:35 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

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

==> v1/ConfigMap
configstore  0s

==> v1/Service
configstore  0s

==> v1beta1/StatefulSet
configstore  0s

==> v1/Pod(related)

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

4. Review the logs from each pod to verify that all three DS pods started up successfully.

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

```
$ kubectl logs configstore-0 | grep successfully
[30/Jul/2019:00:44:26 +0000] category=CORE severity=NOTICE msgID=135 msg=The Directory Server has
 started successfully
[30/Jul/2019:00:44:26 +0000] category=CORE severity=NOTICE msgID=139 msg=The Directory Server has
 sent an alert notification generated by class org.opends.server.core.DirectoryServer (alert type org
.opends.server.DirectoryServerStarted, alert ID org.opends.messages.core-135): The Directory Server
 has started successfully
```

   The procedure is complete when you have verified that the `configstore-0`, `userstore-0`, and `ctsstore-0` pods all started successfully.

4.4.2.2. Deploying AM and Amster

After you have installed the DS servers for the example, deploy AM and Amster.

Perform the following procedures:

1. "To Create .yaml Files for Deploying AM and Amster"
2. "To Install the AM and Amster Helm Charts"
3. "To Verify Successful AM and DS Deployment"
4. "To Configure the Hosts File"
5. "To Access the AM Console"

To Create .yaml Files for Deploying AM and Amster

Perform the following steps:

1. Create the `openam.yaml` file. This file contains information needed to set up the AM server.

   Example `openam.yaml` file:

   ```
   domain: .my-domain.com
   image:
     repository: my-registry/forgerock/openam
   ```

   For information about `openam.yaml` file options, see "openam.yaml" in the YAML File Reference.

2. Create the `amster.yaml` file. This file contains information needed to run Amster to import and export the AM configuration.

   Example `amster.yaml` file:

   ```
   domain: .my-domain.com
   config:
     importPath: /git/config/path/to/AM/configuration
   image:
     repository: my-registry/forgerock/amster
   ```

   For information about `amster.yaml` file options, see "amster.yaml" in the YAML File Reference.

To Install the AM and Amster Helm Charts

Perform the following steps:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install the `openam` Helm chart:

```
$ helm install openam --values /path/to/openam.yaml
NAME:   lumbering-beetle
LAST DEPLOYED: Tue Jul 30 16:49:17 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

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

==> v1/Pod(related)

NAME                                    READY  STATUS    RESTARTS  AGE
lumbering-beetle-openam-649c7f87b5-pfzcf  0/1    Init:0/3  0         1s

==> v1/Secret

NAME                   AGE
openam-secrets         1s
openam-pstore-secrets  1s

==> v1/ConfigMap
boot-json     1s
am-configmap  1s

==> v1/Service
openam  1s

==> v1beta1/Deployment
lumbering-beetle-openam  1s
```

4. Install the `amster` Helm chart:

```
$ helm install amster --values /path/to/amster.yaml
NAME:   needled-pike
LAST DEPLOYED: Tue Jul 30 16:50:11 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME                AGE
amster-config       0s
amster-needled-pike  0s

==> v1beta1/Deployment
amster  0s

==> v1/Pod(related)

NAME                     READY  STATUS    RESTARTS  AGE
amster-784968d4ff-lwqwq  0/2    Init:0/1  0         0s

==> v1/Secret

NAME           AGE
amster-secrets  0s
```

To Verify Successful AM and DS Deployment

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

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

   ```
   $ kubectl get pods
   NAME                                    READY   STATUS    RESTARTS   AGE
   amster-784968d4ff-lwqwq                 2/2     Running   0          2m
   configstore-0                           1/1     Running   0          9m
   ctsstore-0                              1/1     Running   0          8m
   lumbering-beetle-openam-649c7f87b5-pfzcf   1/1     Running   0          2m
   userstore-0                             1/1     Running   0          8m
   ```

   b. Review the output. Deployment is complete when:

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

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

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

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

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

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

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

```
. . .
Got Response code 200
AM web app is up and ready to be configured
About to begin configuration
Extracting amster version
Amster version is:  6.5.2
Executing Amster to configure AM
Executing Amster script /opt/amster/scripts/00_install.amster
Jul 30, 2019 12:51:52 AM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /opt/amster/scripts/00_install.amster
07/30/2019 12:51:56:339 AM GMT: Checking license acceptance...
07/30/2019 12:51:56:339 AM GMT: License terms accepted.
07/30/2019 12:51:56:349 AM GMT: Checking configuration directory /home/forgerock/openam.
07/30/2019 12:51:56:354 AM GMT: ...Success.
07/30/2019 12:51:56:362 AM GMT: Tag swapping schema files.
07/30/2019 12:51:56:389 AM GMT: ...Success.
07/30/2019 12:51:56:390 AM GMT: Loading Schema odsee_config_schema.ldif
07/30/2019 12:51:56:456 AM GMT: ...Success
.
. . .
```

The following output indicates that deployment is complete:

```
. . .
Configuration complete!
Executing Amster script /opt/amster/scripts/01_import.amster
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------
am> :load /opt/amster/scripts/01_import.amster
Importing directory /git/config/6.5/default/am/empty-import
Import completed successfully
Configuration script finished
+ pause
+ echo 'Args are 0 '
+ echo 'Container will now pause. You can use kubectl exec to run export.sh'
Args are 0
Container will now pause. You can use kubectl exec to run export.sh
+ true
+ wait
+ sleep 1000000
```

To Configure the Hosts File

After you have installed the Helm chart for the example, configure the `/etc/hosts` file on your local computer so that you can access the AM console:

1. Get the ingress controller's IP address:
   - On Minikube, run the **minikube ip** command.

- For cloud environments, ask your cluster administrator which IP address to use to access your cluster's ingress controller.

2. To enable cluster access through the ingress controller, add an entry in the `/etc/hosts` file. For example:

```
192.168.99.100 login.my-namespace.example.com
```

In this example, `login.my-namespace.example.com` is the hostname you use to access the AM console, and `192.168.99.100` is the ingress controller's IP address.

To Access the AM Console

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

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

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

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

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

3. If necessary, start a web browser.

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

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

5. AM prompts you to log in or upgrade depending on whether the version of the AM configuration imported from the Git repository matches the version of AM you just installed:

   - If the version of AM from which the configuration imported from the Git repository matches the version of AM you just installed, then AM prompts you to log in.

   - If the version of AM from which the configuration imported from the Git repository does not match the version of AM you just installed, then AM prompts you to upgrade. Perform the upgrade. Then delete the `openam` pod using the **kubectl delete pod** command, causing the pod to automatically restart, and navigate back to the AM deployment URL. The login page should appear.

     For information about upgrading AM, see the ForgeRock Access Management Upgrade Guide.

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

4.4.3. Scaling AM

By default, the `openam` Helm chart deploys a single AM pod.

You can scale AM by changing the number of pods any time after you have installed the Helm chart. Use the **kubectl scale** command to deploy a specific number of AM pods.

For example, to deploy three AM pods, perform the following procedure:

To Scale AM

1. Get the AM deployment name:

```
$ kubectl get deployments
NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
amster                     1         1         1            1           6m
lumbering-beetle-openam    1         1         1            1           7m
```

2. Scale the number of AM pods to three:

```
$ kubectl scale --replicas=3 deployment lumbering-beetle-openam
deployment.extensions/lumbering-beetle-openam scaled
```

3. Verify that three `openam` pods now appear in the **kubectl get pods** output:

```
$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
amster-784968d4ff-lwqwq                 2/2     Running   0          7m
configstore-0                           1/1     Running   0          15m
ctsstore-0                              1/1     Running   0          14m
lumbering-beetle-openam-649c7f87b5-9vwjx   0/1     Running   0          19s
lumbering-beetle-openam-649c7f87b5-hcl2x   0/1     Running   0          19s
lumbering-beetle-openam-649c7f87b5-pfzcf   1/1     Running   0          8m
userstore-0                             1/1     Running   0          14m
```

The newly-created replicas are made available to AM by the Kubernetes ingress, but they do *not* appear in the AM console under Deployment > Servers.

Note that DS does not support elastic scaling, so do not create multiple DS replicas. For more information, see "DS Limitations" in the *DevOps Release Notes*.

4.5. Modifying the AM Configuration

After you have successfully deployed AM and DS, you can modify the AM configuration. After you save the configuration changes, you can use the revised configuration to initialize a subsequent AM deployment.

When you store the configuration in a version control system like a Git repository, you can take advantage of capabilities such as version control, difference analysis, and repository branches as you manage the AM configuration. The capabilities enable you to migrate from a development

environment to a test environment, and then to a production environment. Deployment migration is one of the primary advantages of DevOps techniques.

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

- The AM console
- The AM REST API
- Amster

Then add, commit, and push AM configuration changes as needed to the `autosave-my-namespace` branch in the configuration repository.

Perform the following steps:

To Save the AM Configuration

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

```
$ kubectl get pods | grep amster
amster-784968d4ff-lwqwq                    2/2     Running   0          8m
```

2. Run the **export.sh** script in the `amster` pod's `amster` container. This script exports the AM configuration to the path defined by the `config.exportPath` property in the `amster.yaml` file:

```
$ kubectl exec amster-6574565b7f-tdjdm -c amster -it /opt/amster/export.sh
+ GIT_ROOT=/git/config
+ NAMESPACE=my-namespace
+ export EXPORT_PATH=default/am/empty-import
+ EXPORT_PATH=default/am/empty-import
+ cd /git/config
+ export AMSTER_EXPORT_PATH=/git/config/default/am/empty-import
+ AMSTER_EXPORT_PATH=/git/config/default/am/empty-import
+ mkdir -p /git/config/default/am/empty-import
+ cat
+ /opt/amster/amster /tmp/do_export.amster
Amster OpenAM Shell (6.5.2 build 314d553429, JVM: 1.8.0_151)
Type ':help' or ':h' for help
.
---------------------------------------------------------------------------------
am> :load /tmp/do_export.amster
Export completed successfully
```

3. Run the **git-sync.sh** script in the `amster` pod's `git` container. This script pushes the configuration exported in the previous step to the `autosave-my-namespace` branch in your configuration repository.

```
$ kubectl exec amster-6574565b7f-tdjdm -c git -it /git-sync.sh
+ DEF_BRANCH=autosave-my-namespace
+ GIT_AUTOSAVE_BRANCH=autosave-my-namespace
+ cd /git/config
+ git config core.filemode false
+ git config user.email auto-sync@forgerock.net
+ git config user.name 'Git Auto-sync user'
+ git checkout -B autosave-my-namespace
Switched to a new branch 'autosave-my-namespace'
++ date
+ t='Tue Nov 27 17:10:31 UTC 2018'
+ git add .
+ git commit -a -m 'autosave at Tue Nov 27 17:10:31 UTC 2018'
[autosave-my-namespace 38fd0b7] autosave at Tue Nov 27 17:10:31 UTC 2018
 191 files changed, 5915 insertions(+)
 create mode 100644 default/am/empty-import/global/ActiveDirectoryModule.json
 create mode 100644 default/am/empty-import/global/AdaptiveRiskModule.json
 . . .
 create mode 100644 default/am/empty-import/realms/root/ZeroPageLoginCollector/c9a2cec4-3f2d-425f-95c7
-d8f4495d8a66.json
+ git push --set-upstream origin autosave-my-namespace -f
Counting objects: 243, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (230/230), done.
Writing objects: 100% (243/243), 94.63 KiB | 2.43 MiB/s, done.
Total 243 (delta 90), reused 0 (delta 0)
remote: Resolving deltas: 100% (90/90), completed with 2 local objects.
To github.com:my-account/forgeops-init.git
 * [new branch]      autosave-my-namespace -> autosave-my-namespace
Branch 'autosave-my-namespace' set up to track remote branch 'autosave-my-namespace' from 'origin'.
```

4. When you are ready to update the master AM configuration, merge the changes in the `autosave-*my-namespace*` branch into the branch containing the master AM configuration.
5. Redeploy the AM and DS example using the updated configuration at any time.

   See the section on configuration changes in "Redeploying the Example" for more information.

4.6. Customizing the AM Web Application

Sometimes, customizing AM requires you to modify the AM web application. For example:

- **Deploying a custom authentication module**. Requires copying the authentication module's `.jar` file into the `WEB-INF/lib` directory.
- **Implementing cross-origin resource sharing (CORS)**. Requires replacing the `WEB-INF/web.xml` file bundled in the `openam.war` file with a customized version.
- **Changing the AM web application configuration**. Requires modifications to the `context.xml` file.

Should you need to customize the AM web application in a DevOps deployment, use one of the following techniques:

- Apply your changes to the `openam.war` file before building the `openam` Docker image.

  Modifying the `openam.war` file is a simple way to customize AM, but it is brittle. You might need different Docker images for different deployments. For example, a deployment for the development environment might need slightly different customization than a deployment for the production environment, requiring you to:

  - Create a `.war` file for each environment
  - Manage all the `.war` files
  - Manage multiple versions of customization code

  For more information about building the `openam` Docker image, see "*Building and Pushing Docker Images*".

- Write a customization script named `customize-am.sh` and add it to your configuration repository. Place the script and any supporting files it needs at the path defined by the `config.importPath` property in the `custom.yaml` file. Be sure to make the file executable.

  The `openam` Dockerfile runs the customization script before it starts the Tomcat web container that runs AM, giving you the ability to modify the expanded AM web application before startup.

  The DevOps Examples support storing multiple configurations in a single configuration repository. When using a single configuration repository for different deployments—for example, development, QA, and production deployments—store customization scripts and supporting files together with the configurations they apply to. Then, when deploying the AM and DS example, identify a configuration's location with the `config.importPath` property.

The following is an example of a simple AM customization script. The script copies a customized version of the `web.xml` file that supports CORS into the AM web application just before AM starts:

```
#!/usr/bin/env bash

# This script and a supporting web.xml file should be placed in the
# configuration repository at the path defined by the config.importPath
# property in the custom.yaml file.

echo "Customizing the AM web application"
echo ""

echo "Available environment variables:"
echo ""
env
echo ""

# Copy the web.xml file that is in the same directory as this script to the
# webapps/openam/WEB-INF directory
cp /git/config/${CONFIG_PATH}/web.xml ${CATALINA_HOME}/webapps/openam/WEB-INF

echo "Finished customizing the AM web application"
```

The script does the following:

1. The **env** command logs all environment variables to standard output. You can review the environment variables that are available for use by customization scripts by reviewing the **env** command's output in the `openam` pod's log using the **kubectl log *openam-pod-name*** command.

   The **env** and **echo** commands in the sample script provide helpful information and are not required in customization scripts.

2. The **cp** command copies a customized version of a `web.xml` file that supports CORS into the `openam` web application.

   The script copies the file from the same path in the configuration repository clone at which the `customize-am.sh` script is located. The destination path is the Tomcat home directory's `webapps/openam/WEB-INF` subdirectory, specified by using the `CATALINA_HOME` environment variable provided at run time.

4.7. Redeploying the Example

After you deploy this example, you might want to change your deployment as follows:

- **Run-time changes**. To make run-time changes, reconfigure your deployment using Kubernetes tools. There is no need to terminate or restart running Kubernetes objects.

  An example of a run-time change is scaling the number of replicas.

  To make a run-time change, use the Kubernetes dashboard or the **kubectl** command.

  Run-time changes take effect while a deployment is running, with no need to terminate or restart any Kubernetes objects.

- **Configuration changes**. To restart a ForgeRock component (AM, IDM, or IG) after a configuration change, restart any pods running the component.

  To restart a pod, execute the **kubectl get pods** command to get the pod's name or names. If you have scaled the pod, more than one will be present. Then run the **kubectl delete pods** command against each pod. Pods in the DevOps Examples are created by Kubernetes `Deployment` objects configured with the default restart policy of `Always`. Therefore, when you delete a pod, Kubernetes automatically restarts a new pod of the same type.

- **Changes requiring full redeployment**. To fully redeploy ForgeRock components, clean up your namespace and optionally rebuild Docker containers. Then, reorchestrate your deployment by following the instructions in previous sections in this chapter.

  Full redeployment is required when making changes such as the following:

  - Deploying a new version of ForgeRock software.
  - Using a new Minikube virtual machine.

- Redeploying one of the DevOps Examples using an updated version of your configuration repository. The updated version might include *any* AM, IDM, or IG configuration changes. For example:
  - New AM realms or changes to service definitions.
  - Updated IDM mappings or authentication configuration.
  - New IG routes.
- Recreating a deployment from scratch.

Chapter 5

Deploying the IDM Example

This chapter covers the following topics:

- "About the Example" provides information about the IDM example's architecture.
- "Working With the IDM Example" presents an example workflow that you might use when deploying IDM using DevOps techniques.
- "Deploying the Example" provides instructions for deploying the example.
- "Modifying the IDM Configuration" describes how to update the IDM configuration stored in your configuration repository after you've deployed the example.
- "Redeploying the Example" provides instructions for redeploying the example.

5.1. About the Example

The IDM DevOps example has the following architectural characteristics:

- The IDM deployment runs in a Kubernetes namespace.
- From outside the deployment, IDM is accessed through a Kubernetes ingress controller (load balancer):
  - The ingress controller is configured for session stickiness.
  - A single ingress controller can access every namespace running the example deployment within the cluster.
- The IDM configuration is maintained in a Git repository, external to the Kubernetes cluster, called the *configuration repository*.
- The following Kubernetes pods are deployed in each namespace running the example:

  `openidm` pod(s)

  At startup, the `git-init` container clones the configuration repository and then terminates.

  The `openidm` container accesses the IDM configuration from the configuration repository clone, starts IDM, and remains active while running the IDM server.

When you run IDM in a container, the IDM configuration is always retrieved from the file system and not the IDM repository. Because of this, always start IDM with the `openidm.config.repo.enabled` property set to `false` in DevOps deployments.

If a user initiates a request to export the IDM configuration, the `git` container pushes changes made to the the IDM configuration to the configuration repository.

Multiple instances of this pod can be started if required. The Kubernetes ingress controller redirects requests to one of these pods. The pods are elastic and the pod names are created dynamically by Kubernetes at run time.

`openidm-postgres` pod(s)

This pod runs the IDM repository as a PostgreSQL database.

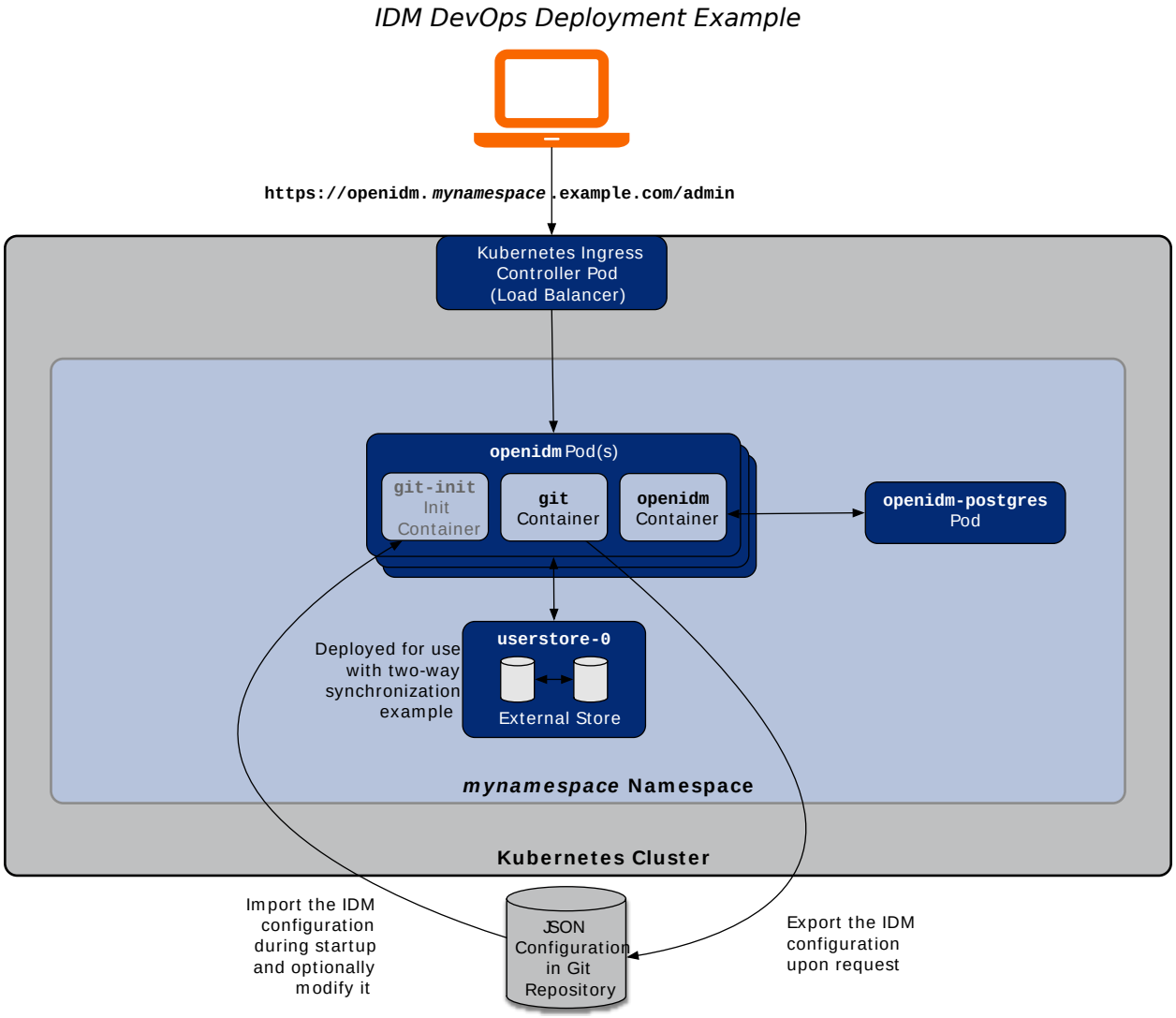
The `openidm-postgres` pod is for development use only. When deploying IDM in production, configure your JDBC repository to support clustered, highly available operations.

DS user store

The IDM example implements bidirectional data synchronization between IDM and LDAP described in Synchronizing Data Between LDAP and IDM in the *Samples Guide*. The DS user store contains the LDAP entries that are synchronized.

This pod, which is not elastic, has the name `userstore-0`. It is preconfigured with a small number of sample users.

The following diagram illustrates the example.



5.2. Working With the IDM Example

This section presents an example workflow to set up a development environment in which you configure IDM, iteratively modify the IDM 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:

Step	Details
Get the latest version of the `forgeops` repository	Make sure you have the latest version of the `release/6.5.2` branch of the `forgeops` Git repository, which contains Docker, Helm, and Kubernetes artifacts needed to build and deploy all of the DevOps Examples. For more information about the `forgeops` Git repository, see "Public Git Repositories".
Implement a development environment	Set up a local or cloud-based environment for developing the IDM configuration. See "*Implementing DevOps Environments*".
Obtain Docker images for the example	Make sure you have built Docker images for the example and pushed them them to your Docker registry. For details about creating and pushing Docker images, see "*Building and Pushing Docker Images*". As an alternative, if you are not running a production deployment, you can use unsupported, evaluation-only Docker images from ForgeRock. For more information, see "Using the Evaluation-Only Docker Images".
Deploy the IDM example in the development environment	Follow the procedures in "Deploying the Example".
Modify the IDM configuration	Iterate through the following steps as many times as you need to: 1. Modify the IDM configuration using the IDM Admin UI or the REST API. See "To Access the IDM Admin UI" for details about how to access the deployed IDM server. 2. Push the updates to the configuration repository's `autosave-my-namespace` branch as desired. 3. Merge configuration updates from the `autosave-my-namespace` branch into the branch containing the master version of your IDM configuration. For more information about modifying the configuration, see "Modifying the IDM Configuration".
Implement one or more production environments	Set up cloud-based environments for test and production deployments. See "*Implementing DevOps Environments*".
Deploy the IDM example in the production environments	Follow the procedures in "Deploying the Example".

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

- Modify the IDM configuration on the Minikube deployment and merge changes into the master version of your IDM configuration.
- Redeploy the IDM example in your cloud environment based on the updated configuration.

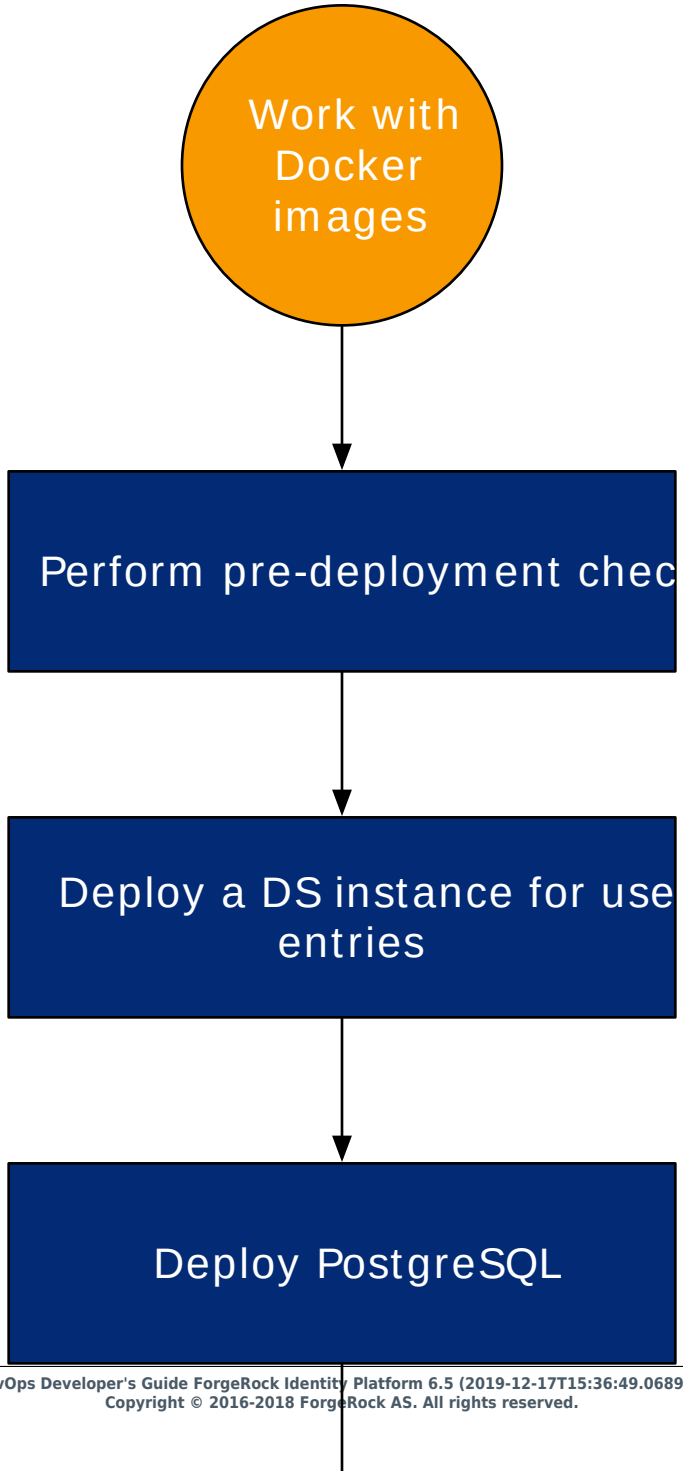
5.3. Deploying the Example

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

- "IDM Example Deployment Process"
- "Performing Pre-Deployment Checks"
- "Using Helm to Deploy the IDM Example"
- "Scaling IDM"

The following diagram illustrates a high-level workflow you'll use to deploy the IDM example:

IDM Example Deployment Process



To deploy the IDM example, perform the following tasks:

Task	Steps
Perform pre-deployment checks.	Follow the instructions in "Performing Pre-Deployment Checks".
Deploy a DS server for user entries.	Follow the instructions in "Deploying the User Directory".
Deploy PostgreSQL.	Follow the instructions in "Deploying the PostgreSQL Server".
Deploy IDM, and then wait until IDM is up.	Follow the instructions in "Deploying IDM".
Verify IDM is up.	Perform "To Verify Successful IDM Deployment".
Configure the `hosts` file.	Perform "To Configure the Hosts File".
Access the IDM Admin UI.	Perform "To Access the IDM Admin UI".
Increase or decrease the number of IDM replicas.	Follow the instructions in "Scaling IDM".

5.3.1. Performing Pre-Deployment Checks

Review the requirements for deployment environments in the following table. If your environment does not meet any of these requirements, perform the necessary procedures before deploying the IDM example:

Requirement	Procedure
A `forgeops` repository clone is available on your local computer.	"Cloning the forgeops Repository".
A namespace has been created in your Kubernetes cluster, and it is configured as the active namespace.	Run the **kubens** command and verify that your namespace is the active namespace. If it is not, run the **kubens *my-namespace*** command. If you have not created a namespace, perform "To Create a Namespace to Run the DevOps Examples".
Helm is set up and running in your Kubernetes cluster.	"Setting up Helm".
The Kubernetes context is set up on your local computer.	"Setting up a Kubernetes Context".
An image pull secret is available in your namespace. (Required only if your Docker images are stored in a private Docker registry.)	"To Create a Kubernetes Secret for Accessing a Private Docker Registry".
The `frconfig` Helm chart is installed in your namespace.	"Configuring and Installing the frconfig Helm Chart".
No ForgeRock-related Kubernetes objects from previous deployments are present in your namespace, other than the Kubernetes secrets for configuration	"To Remove ForgeRock-Related Objects From a Namespace".

Requirement	Procedure
repository access, TLS support, and Docker image access.	

5.3.2. Using Helm to Deploy the IDM Example

The following sections guide you through deployment of the IDM example:

- "Deploying the User Directory"
- "Deploying the PostgreSQL Server"
- "Deploying IDM"

5.3.2.1. Deploying the User Directory

The persistence tier for the IDM example includes a DS server. The DS server contains user entries that are synced with IDM when running LDAP bidirectional synchronization. Perform the following procedures to set up the user directory for this example:

1. "To Create a .yaml File for Installing the DS Server Used in the IDM Example"
2. "To Install the DS Server's Helm Chart"

To Create a .yaml File for Installing the DS Server Used in the IDM Example

Perform the following step:

- Create the `userstore.yaml` file. This file contains information needed to set up the DS server used for user entries.

  Example `userstore.yaml` file:

  ```
  instance: userstore
  image:
    repository: my-registry/forgerock/ds
  ```

  For information about `userstore.yaml` file options, see "userstore.yaml" in the YAML File Reference.

To Install the DS Server's Helm Chart

Perform the following steps:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install the `ds` Helm chart to create the `userstore-0` pod.

   Run the following command, making sure you specify the correct `.yaml` file:

```
$ helm install ds --values /path/to/userstore.yaml
NAME:   zooming-mongoose
LAST DEPLOYED: Tue Jul 30 17:19:19 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/StatefulSet
NAME       AGE
userstore  1s

==> v1/Pod(related)

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

==> v1/Secret

NAME       AGE
userstore  1s

==> v1/ConfigMap
userstore  1s

==> v1/Service
userstore  1s
```

4. Review the logs from each pod to verify that the DS pod started up successfully.

   The following command verifies that the `userstore-0` pod started successfully:

```
$ kubectl logs userstore-0 | grep successfully
[[30/Jul/2019:01:19:37 +0000] category=CORE severity=NOTICE msgID=135 msg=The Directory Server has
 started successfully
[30/Jul/2019:01:19:37 +0000] category=CORE severity=NOTICE msgID=139 msg=The Directory Server has
 sent an alert notification generated by class org.opends.server.core.DirectoryServer (alert type org
.opends.server.DirectoryServerStarted, alert ID org.opends.messages.core-135): The Directory Server
 has started successfully
```

5.3.2.2. Deploying the PostgreSQL Server

The persistence tier for the IDM example also includes a PostgreSQL server that is used for the IDM repository. Perform the following procedure to deploy the PostgreSQL server:

To Install the PostgreSQL Server's Helm Chart

Perform the following steps:

1. Review the commented `values.yaml` file for the `postgres-openidm` Helm chart in the `forgeops` repository and determine whether you want to override any of the default deployment options.

   For simple deployments, you typically do not need to override the default options.

2. If the default values are not suitable for your environment, create a `postgresql.yaml` file that overrides the default values.

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

4. Change to the Helm charts directory in your `forgeops` repository clone:

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

5. Install the `postgres-openidm` Helm chart to create the `postgres-openidm` pod.

   Run one of the following commands:

   - To install the Helm chart using the default options:

     ```
     $ helm install postgres-openidm
     ```

   - To install the Helm chart and override default options:

     ```
     $ helm install postgres-openidm --values /path/to/postgresql.yaml
     ```

   Output similar to the following is displayed:

   ```
   NAME:   vehement-flee
   LAST DEPLOYED: Tue Jul 30 17:20:54 2019
   NAMESPACE: my-namespace
   STATUS: DEPLOYED

   RESOURCES:
   ==> v1/Secret
   NAME               AGE
   postgres-openidm  1s

   ==> v1/ConfigMap
   openidm-sql  1s

   ==> v1/PersistentVolumeClaim
   postgres-openidm  1s

   ==> v1/Service
   postgresql  1s

   ==> v1beta1/Deployment
   postgres-openidm  1s

   ==> v1/Pod(related)

   NAME                               READY  STATUS    RESTARTS  AGE
   postgres-openidm-79496f548b-xs92d  0/1    Init:0/1  0         1s
   ```

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

To get your user password run:

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

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

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

6. Run the **kubectl get pods** command to verify that the `postgres-openidm` pod is in `Running` status:

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgres-openidm-79496f548b-xs92d   1/1     Running   0          54s
userstore-0                         1/1     Running   0          2m
```

5.3.2.3. Deploying IDM

After you have installed the DS user store and the PostgreSQL server for the example, deploy IDM.

Perform the following procedures:

1. "To Create a .yaml File for Deploying IDM"
2. "To Install the IDM Helm Chart"
3. "To Verify Successful IDM Deployment"
4. "To Configure the Hosts File"
5. "To Access the IDM Admin UI"

To Create a .yaml File for Deploying IDM

Perform the following step:

- Create the `openidm.yaml` file. This file contains information needed to set up the AM server.

  Example `openidm.yaml` file:

```
domain: .my-domain.com
config:
  path: /git/config/path/to/IDM/configuration
  immutable: false
image:
  repository: my-registry/forgerock/openidm
```

To deploy IDM with a read/write configuration, set the `config.immutable` key to `false`. By default, the IDM configuration is immutable (read-only). No changes to IDM configuration are saved.

For information about `openidm.yaml` file options, see "openidm.yaml" in the YAML File Reference.

To Install the IDM Helm Chart

Perform the following steps:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install the `openidm` Helm chart:

```
$ helm install openidm --values /path/to/openidm.yaml
NAME:   quaffing-porcupine
LAST DEPLOYED: Tue Jul 30 17:28:21 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

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

==> v1beta1/StatefulSet
quaffing-porcupine-openidm  0s

==> v1beta1/Ingress
openidm  0s

==> v1/Pod(related)

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

==> v1/Secret

NAME                AGE
openidm-secrets-env  0s
openidm-secrets      0s

==> v1/ConfigMap
quaffing-porcupine-openidm  0s
idm-boot-properties         0s
idm-logging-properties      0s


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

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

To Verify Successful IDM Deployment

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

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

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgres-openidm-79496f548b-dw2dt   1/1     Running   0          2m
quaffing-porcupine-openidm-0        2/2     Running   0          2m
userstore-0                         1/1     Running   0          2m
```

2. Review the output. Deployment is complete when:
   - The `READY` column indicates all containers are available. The entry in the `READY` column represents [total number of containers/number of available containers].
   - All entries in the `STATUS` column indicate `Running`.
3. If necessary, continue to query your deployment's status until all the pods are ready.

To Configure the Hosts File

After you have installed the Helm chart for the example, configure the `/etc/hosts` file on your local computer so that you can access the IDM Admin UI:

1. Get the ingress controller's IP address:
   - On Minikube, run the **minikube ip** command.
   - For cloud environments, ask your cluster administrator which IP address to use to access your cluster's ingress controller.
2. To enable cluster access through the ingress controller, add an entry in the `/etc/hosts` file. For example:

```
192.168.99.100 openidm.my-namespace.example.com
```

   In this example, `openidm.my-namespace.example.com` is the hostname you use to access the IDM Admin UI, and `192.168.99.100` is the ingress controller's IP address.

To Access the IDM Admin UI

1. If necessary, start a web browser.
2. Navigate to the IDM Admin UI's deployment URL, for example, https://openidm.*my-namespace*.example.com/admin.

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

3. Log in to IDM as the `openidm-admin` user with password `openidm-admin`.

5.3.3. Scaling IDM

By default, the `openidm` Helm chart deploys a single IDM pod.

You can scale IDM by changing the number of pods any time after you have installed the Helm chart. Use the **kubectl scale** command to deploy a specific number of IDM pods.

For example, to deploy three IDM pods, perform the following procedure:

To Scale IDM

1. Get the IDM stateful set name:

```
$ kubectl get statefulsets
NAME                         DESIRED   CURRENT   AGE
quaffing-porcupine-openidm   1         1         3m
userstore                    1         1         3m
```

2. Scale the number of IDM pods to three:

```
$ kubectl scale --replicas=3 statefulset quaffing-porcupine-openidm
statefulset.apps/quaffing-porcupine-openidm scaled
```

3. Verify that three `openidm` pods now appear in the **kubectl get pods** output:

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgres-openidm-79496f548b-dw2dt   1/1     Running   0          4m
quaffing-porcupine-openidm-0        2/2     Running   0          4m
quaffing-porcupine-openidm-1        2/2     Running   0          26s
quaffing-porcupine-openidm-2        2/2     Running   0          22s
userstore-0                         1/1     Running   0          4m
```

Note that DS does not support elastic scaling, so do not create multiple DS replicas. For more information, see "DS Limitations" in the *DevOps Release Notes*.

5.4. Modifying the IDM Configuration

After you have successfully deployed IDM with a read/write (mutable) configuration, you can modify the IDM configuration. After you save the configuration changes, you can use the revised configuration to initialize a subsequent IDM deployment.

When you store the configuration in a version control system like a Git repository, you can take advantage of capabilities such as version control, difference analysis, and repository branches as you manage the IDM configuration. The capabilities enable you to migrate from a development environment to a test environment, and then to a production environment. Deployment migration is one of the primary advantages of DevOps techniques.

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

- The IDM Admin UI
- The IDM REST API

Then add, commit, and push IDM configuration changes as needed to the `autosave-my-namespace` branch in the configuration repository.

Perform the following steps:

To Save the IDM Configuration

1. Verify that you deployed IDM with a read/write configuration by checking the value of the `config.immutable` key in your `openidm.yaml` file.

   If the `config.immutable` key has a value of `false`, you deployed IDM with a read/write configuration; you are able to save the configuration. Proceed to the next step.

   If the `config.immutable` key has a value of `true`, you deployed IDM with a read-only configuration; any changes you might have made to the IDM configuration using the IDM console or the REST APIs have not been saved. Redeploy the IDM example, being sure to set the `config.immutable` key set to `false`. For information on redeploying the IDM example, see "Redeploying the Example".

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

   ```
   $ kubectl get pods | grep openidm
   postgres-openidm-79496f548b-dw2dt   1/1     Running       0          5m
   quaffing-porcupine-openidm-0        2/2     Running       1          5m
   ```

3. Run the **git-sync.sh** script in the `openidm` pod's `git` container. This script pushes the IDM configuration to the `autosave-my-namespace` branch in your configuration repository.

```
$ kubectl exec quaffing-porcupine-openidm-0 -c git -it /git-sync.sh
Switched to branch 'autosave-my-namespace'
[autosave-my-namespace 99d4b0af] autosave at Tue Apr 10 20:11:19 UTC 2018'
 14 files changed, 3497 insertions(+), 14 deletions(-)
 create mode 100644 custom/idm/am-protects-idm/conf/authentication.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/identityProviders.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/info-login.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/info-ping.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/info-version.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/managed.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/policy.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/selfservice-registration.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/selfservice.kba.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/ui-dashboard.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/ui.context-admin.json.patch
 create mode 100644 custom/idm/am-protects-idm/conf/ui.context-selfservice.json.patch
Counting objects: 19, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (19/19), 14.12 KiB | 7.06 MiB/s, done.
Total 19 (delta 9), reused 7 (delta 4)
remote: Resolving deltas: 100% (9/9), completed with 5 local objects.
To github.com:my-account/forgeops-init.git
 + cebff7ed...99d4b0af autosave-my-namespace -> autosave-my-namespace (forced update)
Branch 'autosave-my-namespace' set up to track remote branch 'autosave-my-namespace' from 'origin'.
```

4. When you are ready to update the master IDM configuration, merge the changes in the `autosave-my-namespace` branch into the branch containing the master IDM configuration.
5. Redeploy the IDM example using the updated configuration at any time.

   See the section on configuration changes in "Redeploying the Example" for more information.

5.5. Redeploying the Example

After you deploy this example, you might want to change your deployment as follows:

- **Run-time changes**. To make run-time changes, reconfigure your deployment using Kubernetes tools. There is no need to terminate or restart running Kubernetes objects.

  An example of a run-time change is scaling the number of replicas.

  To make a run-time change, use the Kubernetes dashboard or the **kubectl** command.

  Run-time changes take effect while a deployment is running, with no need to terminate or restart any Kubernetes objects.

- **Configuration changes**. To restart a ForgeRock component (AM, IDM, or IG) after a configuration change, restart any pods running the component.

  To restart a pod, execute the **kubectl get pods** command to get the pod's name or names. If you have scaled the pod, more than one will be present. Then run the **kubectl delete pods** command

against each pod. Pods in the DevOps Examples are created by Kubernetes `Deployment` objects configured with the default restart policy of `Always`. Therefore, when you delete a pod, Kubernetes automatically restarts a new pod of the same type.

- **Changes requiring full redeployment**. To fully redeploy ForgeRock components, clean up your namespace and optionally rebuild Docker containers. Then, reorchestrate your deployment by following the instructions in previous sections in this chapter.

  Full redeployment is required when making changes such as the following:

  - Deploying a new version of ForgeRock software.
  - Using a new Minikube virtual machine.
  - Redeploying one of the DevOps Examples using an updated version of your configuration repository. The updated version might include *any* AM, IDM, or IG configuration changes. For example:
    - New AM realms or changes to service definitions.
    - Updated IDM mappings or authentication configuration.
    - New IG routes.
  - Recreating a deployment from scratch.

Chapter 6

Deploying the IG Example

This chapter covers the following topics:

- "About the Example" provides information about the IG example's architecture.
- "Working With the IG Example" presents an example workflow that you might use when deploying IG using DevOps techniques.
- "Deploying the Example" provides instructions for deploying the example.
- "Modifying the IG Configuration" describes how to update the IG configuration stored in your configuration repository after you've deployed the example.
- "Redeploying the Example" provides instructions for redeploying the example.

6.1. About the Example

The IG DevOps example has the following architectural characteristics:

- The IG deployment runs in a Kubernetes namespace.
- From outside the deployment, IG is accessed through a Kubernetes ingress controller (load balancer):
  - The ingress controller is configured for session stickiness.
  - A single ingress controller can access every namespace running the example deployment within the cluster.
- The IG configuration is maintained in a Git repository, external to the Kubernetes cluster, called the *configuration repository*.
- After installation, the deployment starts IG, referencing the JSON files stored in the configuration repository.
- The following Kubernetes pod is deployed in each namespace running the example:

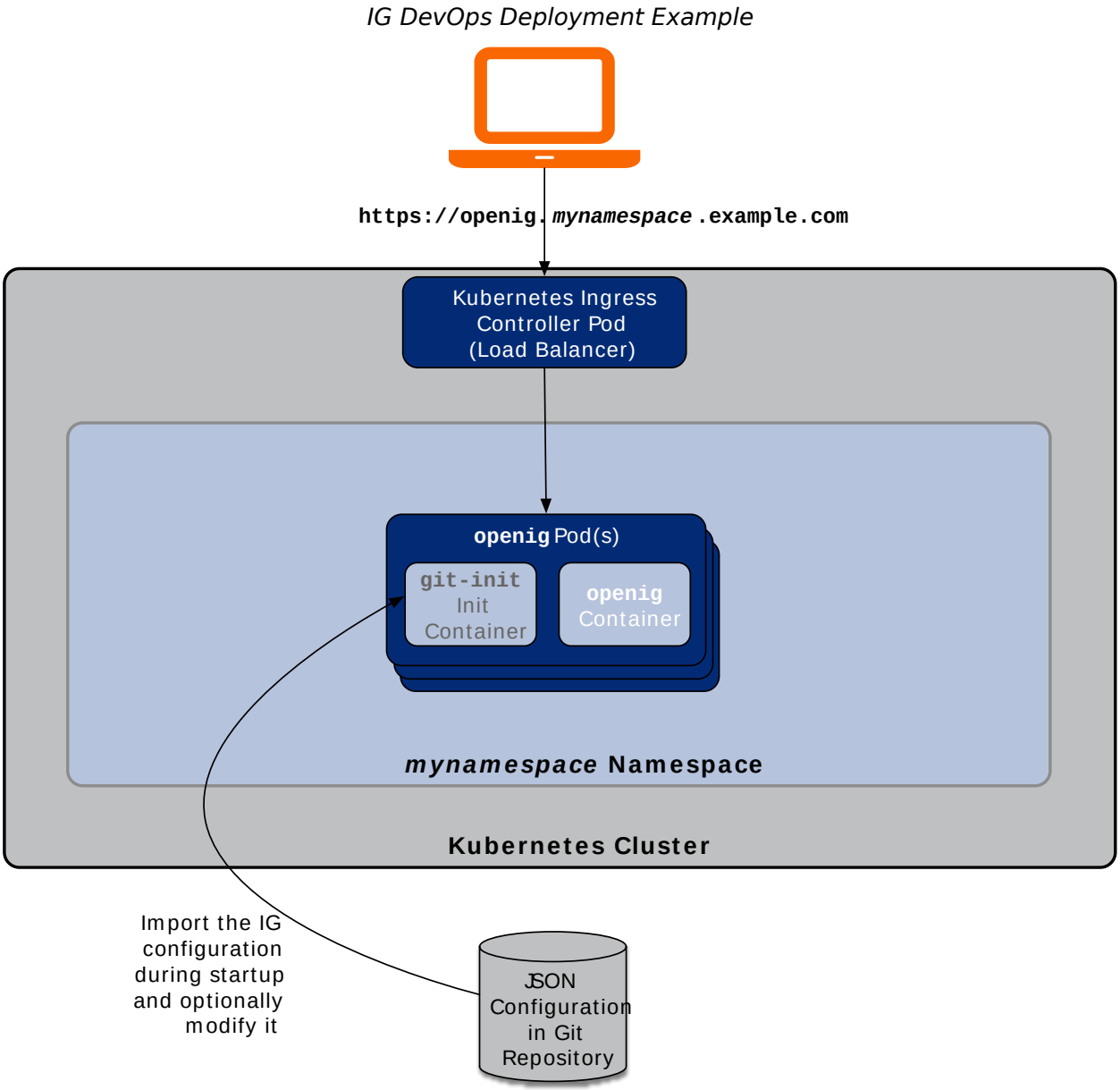
  `openig` pod(s)

  At startup, the `git-init` container clones the configuration repository and then terminates.

The `openig` container accesses the IG configuration from the configuration repository clone, starts IG, and remains active while running the IG server.

Multiple instances of this pod can be started if required. The Kubernetes ingress controller redirects requests to one of these pods. The pods are elastic and the pod names are created dynamically by Kubernetes at run time.

The following diagram illustrates the example.



6.2. Working With the IG Example

This section presents an example workflow to set up a development environment in which you configure IG, iteratively modify the IG 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:

Step	Details
Get the latest version of the `forgeops` repository	Make sure you have the latest version of the `release/6.5.2` branch of the `forgeops` Git repository, which contains Docker, Helm, and Kubernetes artifacts needed to build and deploy all of the DevOps Examples. For more information about the `forgeops` Git repository, see "Public Git Repositories".
Implement a development environment	Set up a local or cloud-based environment for developing the IG configuration. See "*Implementing DevOps Environments*".
Obtain Docker images for the example	Make sure you have built Docker images for the example and pushed them them to your Docker registry. For details about creating and pushing Docker images, see "*Building and Pushing Docker Images*". As an alternative, if you are not running a production deployment, you can use unsupported, evaluation-only Docker images from ForgeRock. For more information, see "Using the Evaluation-Only Docker Images".
Deploy the IG example in the development environment	Follow the procedures in "Deploying the Example".
Modify the IG configuration	Iteratively modify the IG configuration by manually editing JSON files. See the following sections for more information: • "To Verify IG is Operational", for details about about how to verify IG is operational • "Modifying the IG Configuration", for information about how to save IG configuration changes to the configuration repository
Implement one or more production environments	Set up cloud-based environments for test and production deployments. See "*Implementing DevOps Environments*".
Deploy the IG example in the production environments	Follow the procedures in "Deploying the Example".

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

- Modify the IG configuration on the Minikube deployment and merge changes into the master version of your IG configuration.
- Redeploy the IG example in your cloud environment based on the updated configuration.

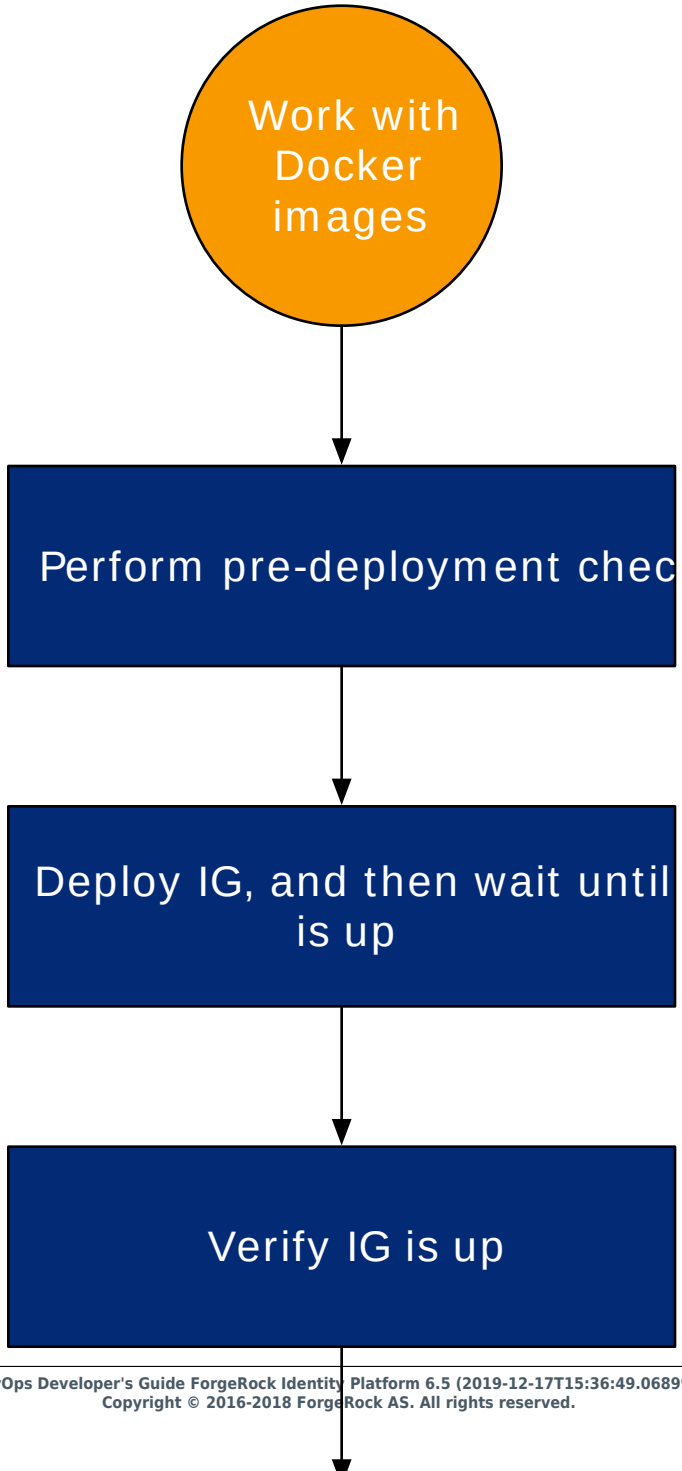
6.3. Deploying the Example

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

- "IG Example Deployment Process"
- "Performing Pre-Deployment Checks"
- "Using Helm to Deploy the IG Example"
- "Scaling IG"

The following diagram illustrates a high-level workflow you'll use to deploy the IG example:

IG Example Deployment Process



To deploy the IG example, perform the following tasks:

Task	Steps
Perform pre-deployment checks.	Follow the instructions in "Performing Pre-Deployment Checks".
Deploy IG, and then wait until IG is up.	Follow the instructions in "Using Helm to Deploy the IG Example".
Verify IG is up.	Perform "To Verify Successful IG Deployment".
Configure the `hosts` file.	Perform "To Configure the Hosts File".
Verify IG is operational.	Perform "To Verify IG is Operational".
Increase or decrease the number of IG replicas.	Follow the instructions in "Scaling IG".

6.3.1. Performing Pre-Deployment Checks

Review the requirements for deployment environments in the following table. If your environment does not meet any of these requirements, perform the necessary procedures before deploying the IG example:

Requirement	Procedure
A `forgeops` repository clone is available on your local computer.	"Cloning the forgeops Repository".
A namespace has been created in your Kubernetes cluster, and it is configured as the active namespace.	Run the **kubens** command and verify that your namespace is the active namespace. If it is not, run the **kubens *my-namespace*** command. If you have not created a namespace, perform "To Create a Namespace to Run the DevOps Examples".
Helm is set up and running in your Kubernetes cluster.	"Setting up Helm".
The Kubernetes context is set up on your local computer.	"Setting up a Kubernetes Context".
An image pull secret is available in your namespace. (Required only if your Docker images are stored in a private Docker registry.)	"To Create a Kubernetes Secret for Accessing a Private Docker Registry".
The `frconfig` Helm chart is installed in your namespace.	"Configuring and Installing the frconfig Helm Chart".
No ForgeRock-related Kubernetes objects from previous deployments are present in your namespace, other than the Kubernetes secrets for configuration repository access, TLS support, and Docker image access.	"To Remove ForgeRock-Related Objects From a Namespace".

6.3.2. Using Helm to Deploy the IG Example

Perform the following procedures to deploy the IG example:

1. "To Create .yaml Files for Deploying IG"
2. "To Install the IG Helm Chart"
3. "To Verify Successful IG Deployment"
4. "To Configure the Hosts File"
5. "To Verify IG is Operational"

To Create .yaml Files for Deploying IG

Perform the following step:

- Create the `openig.yaml` file. This file contains information needed to set up the IG server.

  Example `openig.yaml` file:

  ```
  domain: .my-domain.com
  config:
    path: /git/config/path/to/IG/configuration
  image:
    repository: my-registry/forgerock/openig
  ```

  For information about `openig.yaml` file options, see "openig.yaml" in the YAML File Reference.

To Install the IG Helm Chart

Perform the following steps:

1. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
2. Change to the Helm charts directory in your `forgeops` repository clone:

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

3. Install the `openig` Helm chart:

   ```
   $ helm install openig --values /path/to/openig.yaml
   NAME:   wiggly-aardvark
   LAST DEPLOYED: Tue Jul 30 18:01:30 2019
   NAMESPACE: my-namespace
   STATUS: DEPLOYED

   RESOURCES:
   ==> v1/ConfigMap
   NAME                    AGE
   openig-wiggly-aardvark  0s
   ```

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

==> v1beta1/Deployment
wiggly-aardvark-openig  0s

==> v1beta1/Ingress
openig  0s

==> v1/Pod(related)

NAME                                   READY  STATUS    RESTARTS  AGE
wiggly-aardvark-openig-95cfc9bd9-c2g95  0/1    Init:0/1  0         0s

==> v1/Secret

NAME                AGE
openig-secrets-env  0s


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


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

To Verify Successful IG Deployment

Check the status of the IG pod until it is ready:

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

```
$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
wiggly-aardvark-openig-95cfc9bd9-c2g95   1/1     Running   0          1m
```

2. Review the output. Deployment is complete when:

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

3. If necessary, continue to query the IG pod's status until it is ready.

To Configure the Hosts File

After you have installed the Helm chart for the example, configure the `/etc/hosts` file on your local computer so that you can access IG:

1. Get the ingress controller's IP address:
   - On Minikube, run the **minikube ip** command.
   - For cloud environments, ask your cluster administrator which IP address to use to access your cluster's ingress controller.
2. To enable cluster access through the ingress controller, add an entry in the `/etc/hosts` file. For example:

   ```
   192.168.99.100 openig.my-namespace.example.com
   ```

   In this example, `openig.my-namespace.example.com` is the hostname you use to access IG, and `192.168.99.100` is the ingress controller's IP address.

To Verify IG is Operational

By default, the IG example runs in production mode. Therefore, IG Studio is not available unless you reconfigure IG to run in development mode.

1. If necessary, start a web browser.
2. Navigate to https://openig.*my-namespace*.example.com.

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

   You should see a message similar to the following:

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

6.3.3. Scaling IG

By default, the `openig` Helm chart deploys a single IG pod.

You can scale IG by changing the number of pods any time after you have installed the Helm chart. Use the **kubectl scale** command to deploy a specific number of IG pods.

For example, to deploy three IG pods, perform the following procedure:

To Scale IG

1. Get the IG deployment name:

   ```
   $ kubectl get deployments
   NAME                    DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   wiggly-aardvark-openig   1         1         1            1           2m
   ```

2. Scale the number of IG pods to three:

   ```
   $ kubectl scale --replicas=3 deployment wiggly-aardvark-openig
   deployment.extensions/wiggly-aardvark-openig scaled
   ```

3. Verify that three `openig` pods now appear in the **kubectl get pods** output:

```
$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
wiggly-aardvark-openig-95cfc9bd9-2wtrt   1/1     Running   0          27s
wiggly-aardvark-openig-95cfc9bd9-c2g95   1/1     Running   0          3m
wiggly-aardvark-openig-95cfc9bd9-h6xm5   1/1     Running   0          27s
```

6.4. Modifying the IG Configuration

After you have successfully deployed IG, you can modify the IG configuration. After you save the configuration changes, you can use the revised configuration to initialize a subsequent IG deployment.

When you store the configuration in a version control system like a Git repository, you can take advantage of capabilities such as version control, difference analysis, and repository branches as you manage the IG configuration. The capabilities enable you to migrate from a development environment to a test environment, and then to a production environment. Deployment migration is one of the primary advantages of DevOps techniques.

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

To modify the IG configuration, manually edit the configuration in a local clone of your cloud-based configuration repository. Then add, commit, and push the changes from the clone to a branch in the remote configuration repository.

When you are ready to update the master IG configuration, merge the branch containing the changes into the branch containing the master IG configuration. Then delete all active IG pods. If you have more than one IG pod as a result of scaling the deployment, be sure to delete all the pods. Immediately after you delete an IG pod, Kubernetes automatically starts a new IG pod in its place. When the new pods start up, they obtain the updated IG configuration.

6.5. Redeploying the Example

After you deploy this example, you might want to change your deployment as follows:

- **Run-time changes**. To make run-time changes, reconfigure your deployment using Kubernetes tools. There is no need to terminate or restart running Kubernetes objects.

  An example of a run-time change is scaling the number of replicas.

  To make a run-time change, use the Kubernetes dashboard or the **kubectl** command.

  Run-time changes take effect while a deployment is running, with no need to terminate or restart any Kubernetes objects.

- **Configuration changes**. To restart a ForgeRock component (AM, IDM, or IG) after a configuration change, restart any pods running the component.

To restart a pod, execute the **kubectl get pods** command to get the pod's name or names. If you have scaled the pod, more than one will be present. Then run the **kubectl delete pods** command against each pod. Pods in the DevOps Examples are created by Kubernetes `Deployment` objects configured with the default restart policy of `Always`. Therefore, when you delete a pod, Kubernetes automatically restarts a new pod of the same type.

- **Changes requiring full redeployment**. To fully redeploy ForgeRock components, clean up your namespace and optionally rebuild Docker containers. Then, reorchestrate your deployment by following the instructions in previous sections in this chapter.

  Full redeployment is required when making changes such as the following:

  - Deploying a new version of ForgeRock software.
  - Using a new Minikube virtual machine.
  - Redeploying one of the DevOps Examples using an updated version of your configuration repository. The updated version might include *any* AM, IDM, or IG configuration changes. For example:
    - New AM realms or changes to service definitions.
    - Updated IDM mappings or authentication configuration.
    - New IG routes.
  - Recreating a deployment from scratch.

For example, new routes that you have added to the IG configuration do not take effect until after you have redeployed the example regardless of whether you run IG in development or production mode.

Chapter 7

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 Kubernetes environment.

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

Step	More Information
Verify that you installed supported software versions in your environment.	"Verifying Versions of Required Software"
If you are using Minikube, verify that the Minikube VM is configured adequately.	"Verifying the Minikube VM's Configuration (Minikube Only)"
If you are using Minikube, verify that the Minikube VM has sufficient disk space.	"Checking for Sufficient Disk Space (Minikube Only)"
Review the names of your Docker images.	"Reviewing Docker Image Names"
Enable bash completion for the **kubectl** command to make running the command easier.	"Enabling kubectl bash Tab Completion"
Review pod descriptions and container logs in your Kuberntes cluster.	"Reviewing Pod Descriptions and Container Logs"
View ForgeRock-specific files, such as audit, debug, and application logs, and other files.	"Accessing Files in Kubernetes Containers"
Perform a dry run of Helm chart creation and examine the YAML that Helm sends to Kubernetes.	"Performing a Dry Run of Helm Chart Installation"
Review logs of system components such as Docker and Kubernetes.	"Accessing the Kubelet Log"

7.1. Troubleshooting the Environment

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

7.1.1. Verifying Versions of Required Software

Environments in which you run the DevOps Examples must be based on supported versions of software, documented in "Installing Required Third-Party Software".

Use the following commands to determine software versions:

Software	Command
Docker	**docker version**
kubectl (Kubernetes client)	**kubectl version**
Kubernetes cluster	**kubectl version**
Kubernetes Helm	**helm version**
Kubernetes logging display utility	**stern --version**
Oracle VirtualBox	**VBoxManage --version**
Minikube	**minikube version**
Google Cloud SDK	**gcloud version**
Amazon AWS CLI	**aws --version**

Note that as of this writing, the Kubernetes context switching utilities (**kubectx** and **kubens**), and the AWS authentication utility (**aws-iam-authenticator**) do not provide an option to determine the software version.

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

The **minikube start** command example in "Configuring Your Kubernetes Cluster" 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.

7.1.3. 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  383M  3.6G  10% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
tmpfs           3.9G   64K  3.9G   1% /
tmp
/dev/sda1        25G  7.7G   16G  33% /mnt/
sda1
/Users          465G  219G  247G  48% /Users
$ exit
logout
```

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

7.2. Troubleshooting Containerization

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

7.2.1. Reviewing Docker Image Names

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. In either case, Docker images must have the exact names expected by the Helm charts. If they do not have the same names, 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 "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, perform the following procedure:

To Review Docker Image Names

1. Navigate to your Docker registry's website and locate the repositories that contains ForgeRock Identity Platform Docker images.
2. Compare the image names in your Docker registry to the image names expected by Helm. The following image names are the default names hardcoded in the DevOps Examples Helm charts:

Repository	Tag
gcr.io/forgerock-io/openam	6.5.2

Repository	Tag
gcr.io/forgerock-io/amster	6.5.2
gcr.io/forgerock-io/ds	6.5.2
gcr.io/forgerock-io/openidm	6.5.2
gcr.io/forgerock-io/openig	6.5.2

For information about overriding the default Docker image names expected by the Helm charts, see "YAML File Reference".

7.3. Troubleshooting Orchestration

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

7.3.1. 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.

7.3.2. Reviewing Pod Descriptions and Container Logs

Look at pod descriptions and container log files for irregularities that indicate problems.

Pod descriptions contain information about active Kubernetes pods, including configuration information, pod status, a list of the containers in the pod (including containers that have finished running), volume mounts, and a log of pod-related events.

Container logs contain startup and run-time messages that might indicate problem areas. Each Kubernetes container has its own log that contains output written to `stdout` by the application running in the container. `openam` container logs are especially important for troubleshooting AM issues in DevOps deployments: the `openam` Helm chart configures AM to write its debug logs to `stdout`. Therefore, the `openam` container logs include all the AM debug logs.

Here's an example of how you can use pod descriptions and container logs to troubleshoot. Events in the pod description indicate that Kubernetes was unsuccessful in pulling a Docker image required

to run a container. You can review your Docker registry's configuration to determine whether a misconfiguration caused the problem.

7.3.2.1. Running the debug-logs.sh Script

The **debug-logs.sh** script generates the following HTML-formatted output, which you can view in a browser:

- Descriptions of all the Kubernetes pods running the ForgeRock Identity Platform in your namespace
- Logs for all of the containers running in these pods

Perform the following procedure to run the **debug-logs.sh** script and then view the output in a browser:

To Run the debug-logs.sh Script

1. Run the **kubens** command and verify that your namespace is the active namespace. If it is not, run the **kubens *my-namespace*** command to set your Kubernetes context to reference your namespace.
2. Make sure you've checked out the release/6.5.2 branch of the `forgeops` repository.
3. Change to the `/path/to/forgeops/bin` directory in your `forgeops` repository clone.
4. Run the **debug-logs.sh** script:

```
$ ./debug-logs.sh
Generating debug log for namespace my-namespace
rm: /tmp/forgeops/*: No such file or directory
Generating amster-75c77f6974-rd2r2 logs
Generating configstore-0 logs
Generating ctsstore-0 logs
Generating snug-seal-openam-6b84c96b78-xj8vs logs
Generating userstore-0 logs
open file:///tmp/forgeops/log.html in your browser
```

5. In a browser, navigate to the URL shown in the **debug-logs.sh** output. For example, `file:///tmp/forgeops/log.html`. The browser displays a screen with a link for each ForgeRock Identity Platform pod in your namespace:

debug-logs.sh Output

6. (Optional) To navigate to the information for a pod, select its link from the start of the **debug-logs.sh** output.

   Selecting the link takes you to the pod's description. Logs for each of the pod's containers follow the pod's description.

7. (Optional) To modify the output to contain the latest updates to the pod descriptions and container logs, run the **debug-logs.sh** script again, and then refresh your browser.

7.3.2.2. Example: Docker Image Name Mismatch

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.

A 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:

To Diagnose and Correct Docker Name Mismatches

1. Review the default Docker image names expected by the DevOps Examples Helm charts covered in "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 **debug-logs.sh** command, and then open its output in a browser.

4. Navigate to the pod description for the pod that won't start and review the Events section at the bottom of the pod description:

```
. . .
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{ds} Normal  Pulling   pulling image "forgerock/ds:6.5
.2"
  13m  2m  7 kubelet, minikube spec.containers{ds} Warning  Failed   Failed to pull image "forgerock/
ds:6.5.2": rpc error: code = 2 desc = Error: image forgerock/ds not found
  13m  2m  7 kubelet, minikube    Warning  FailedSync  Error syncing pod, skipping: failed to
 "StartContainer" for "ds" with ErrImagePull: "rpc error: code = 2 desc = Error: image forgerock/ds
 not found"

  13m 9s 53 kubelet, minikube spec.containers{ds} Normal BackOff  Back-off pulling image "forgerock/
ds:6.5.2"
  13m 9s 53 kubelet, minikube    Warning FailedSync Error syncing pod, skipping: failed to
 "StartContainer" for "ds" with ImagePullBackOff: "Back-off pulling image \"forgerock/ds:6.5.2\""
```

   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.

5. Navigate to your Docker registry's website and locate the repositories that contain ForgeRock Identity Platform Docker images.

   A Docker image name mismatch occurs when Kubernetes attempts to retrieve a Docker image that is not available in the Docker registry.

   In the preceding example, observe that Kubernetes attempts to access the `forgerock/ds:6.5.2` image. If this image is not available in the Docker registry, a Docker name mismatch error occurs.

6. Terminate the deployment, recreate the Docker image with the correct name, and redeploy.

7.3.2.3. Example: Misconfigured Configuration Repository

The `frconfig` Helm chart installs the following objects in your namespace:

- A configmap containing the URL for accessing the configuration repository
- A secret containing the SSH key for accessing the repository

The `openam`, `openidm`, and `openig` pods run the `git-init` init container when they start up. This init container uses the objects created by the `frconfig` chart to clone the configuration repository. The clone can then be accessed by the pods when needed.

If the `frconfig` Helm chart has been misconfigured, the `git-init` init container is not able to access the configuration repository. When this happens, the `openam`, `openidm`, or `openig` pod remains in pending status. AM, IDM, and IG services are unable to start.

Troubleshoot and fix a misconfigured configuration repository as follows:

To Diagnose and Reconfigure a Misconfigured Configuration Repository

1. Run the **kubectl get pods** command. For example:

```
$ kubectl get pods
NAME                                      READY     STATUS     RESTARTS   AGE
userstore-0                               1/1       Running    0          17s
postgres-openidm-6647594db9-h2hgp         1/1       Running    0          17s
truculent-snail-openidm-5bbfc8b9f9-2vxh4  0/1       Pending    0          17s
```

   In this example, the `openidm` pod is unable to start.

2. Run the **debug-logs.sh** command, and then view the output in a browser.

3. Select the `openidm` pod from the list of pods that appears at he start of the output.

4. Scroll past the `openidm` pod's description until you see the heading, `Logs for init container: git-init`.

   The log should be similar to the following:

```
Command is init
+ GIT_BRANCH=release/6.5.2
+ ls -lR /etc/git-secret
+ [ ! -z git@github.com:myAccount/forgeops-init.git ]
+ mkdir -p /git/config
+ cd /git/config
+ rm -fr
 *
/etc/git-secret:
total 0
lrwxrwxrwx 1 root root 13 Aug 17 20:42 id_rsa -> ..data/id_rsa
+ git clone git@github.com:myAccount/forgeops-init.git /git/config
Cloning into '/git/config'...
ERROR: Repository not found.
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
+ [ 128 -ne 0 ]
+ echo git clone failed. Will sleep for 5 minutes for debugging.
+ sleep 300
git clone failed. Will sleep for 5 minutes for debugging.
```

5. Review the log. In the preceding example, the `Repository not found` error indicates that the configuration repository could not be accessed.
6. Remove the deployment from your cluster.
7. Make corrections to the configuration repository details in the `frconfig` Helm chart.

   For more information, see "Installing the frconfig Helm Chart".
8. Redeploy the DevOps example.
9. Review the `git-init` init container logs.

   The init container should now be able to clone the configuration repository without errors.

7.3.3. Accessing Files in Kubernetes Containers

You can log in to the bash shell of any container in the DevOps Examples with the **kubectl exec** command. From 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 AM authentication audit log as follows:

```
$ kubectl exec openam-960906639-wrjd8 -c openam -it /bin/bash
bash-4.3$ pwd
/usr/local/tomcat
bash-4.3$ cd
bash-4.3$ pwd
/home/forgerock
bash-4.3$ cd openam/openam/log
bash-4.3$ ls
access.audit.json  activity.audit.json authentication.audit.json config.audit.json
bash-4.3$ cat authentication.audit.json
{"realm":"/","transactionId":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-86","component":"Authentication"
,"eventName":"AM-LOGIN-MODULE-COMPLETED","result":"SUCCESSFUL","entries":[{"moduleId":"Amster","info":
{"authIndex":"service","authControlFlag":"REQUIRED","moduleClass":"Amster","ipAddress":"172.17.0.3"
,"authLevel":"0"}}],"principal":["amadmin"],"timestamp":"2017-09-29T18:14:46.200Z","trackingIds":
["29aac0af-4b62-48cd-976c-3bb5abbed8c8-79"],"_id":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-88"}
{"realm":"/","transactionId":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-86","userId":"id=amadmin,ou=user
,dc=openam,dc=forgerock,dc=org","component":"Authentication","eventName":"AM-LOGIN-COMPLETED"
,"result":"SUCCESSFUL","entries":[{"moduleId":"Amster","info":{"authIndex":"service","ipAddress":"172.17
.0.3","authLevel":"0"}}],"timestamp":"2017-09-29T18:14:46.454Z","trackingIds":["29aac0af-4b62-48cd-976c
-3bb5abbed8c8-79"],"_id":"29aac0af-4b62-48cd-976c-3bb5abbed8c8-95"}
bash-4.3$ exit
```

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

7.3.4. 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 the first part of a dry run installation of the `openam` chart:

```
$ cd /path/to/forgeops/helm
$ helm install openam --values /path/to/openam.yaml --dry-run --debug
[debug] Created tunnel using local port: '63967'

[debug] SERVER: "127.0.0.1:63967"

[debug] Original chart version: ""
[debug] CHART PATH: $HOME/Repositories/forgeops/helm/openam

NAME:   undercooked-wasp
REVISION: 1
RELEASED: Thu Aug  1 14:00:37 2019
CHART: openam-6.5.2
USER-SUPPLIED VALUES:
domain: .example.com

COMPUTED VALUES:
amCustomizationScriptPath: customize-am.sh
auditLogs: []
catalinaOpts: |
  -server -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dcom.sun.identity.util.debug
.provider=com.sun.identity.shared.debug.impl.StdOutDebugProvider -Dcom.sun.identity.shared.debug.file
.format='%PREFIX% %MSG%\\n%STACKTRACE%' -Dcom.iplanet.services.stats.state=off
component: openam
config:
  name: frconfig
  strategy: git
configLdapHost: configstore-0.configstore
configLdapPort: 1389
createBootstrap: true
domain: .example.com
image:
  pullPolicy: IfNotPresent
  repository: gcr.io/forgerock-io/openam
  tag: 6.5.2
ingress:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/cors-allow-credentials: "false"
    nginx.ingress.kubernetes.io/cors-allow-headers: authorization,x-requested-with
    nginx.ingress.kubernetes.io/cors-allow-methods: PUT,GET,POST,HEAD,PATCH,DELETE
    nginx.ingress.kubernetes.io/cors-allow-origin: '*'
    nginx.ingress.kubernetes.io/cors-max-age: "600"
    nginx.ingress.kubernetes.io/enable-cors: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/session-cookie-hash: sha1
    nginx.ingress.kubernetes.io/session-cookie-name: route
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  class: nginx
  enabled: true
```

```
livenessPeriod: 60
livenessTimeout: 15
logDriver: none
openamInstance: http://openam:80
openamReplicaCount: 1
resources:
  limits:
    memory: 1300Mi
  requests:
    memory: 1200Mi
rootSuffix: ou=am-config
service:
  externalPort: 80
  internalPort: 8080
  name: openam
  type: ClusterIP
useConfigMapWebxml: false

HOOKS:
MANIFEST:

---
# Source: openam/templates/secrets.yaml
# Copyright (c) 2016-2017 ForgeRock AS.
# Secrets for AM stack deployment. This is mounted on all containers so they can get their
# passwords, etc.
# Note that secret values are base64-encoded.
# The base64-encoded value of 'password' is 'cGFzc3dvcmQ='
# Watch for trailing \n when you encode!
apiVersion: v1
kind: Secret
metadata:
    name: "openam-secrets"
type: Opaque
data:
  .keypass: Y2hhbmdlaXQ=
  .storepass: MDdVK1pEeURxQlNZeTAwQStIdFVtdzhlU0h2SWp3SUU=
  amster_rsa: LS0t...
  amster_rsa.pub: c3No...
  authorized_keys: c3No...
  id_rsa: LS0t...
  keypass: Y2hhbmdlaXQ=
  keystore.jceks: zs7O...
  keystore.jks: /u3+7QA...
  openam_mon_auth: ZGVtbyBBUUlDY1h3RXVsLzJrQVNqc2Nmb1RRcjVUQ2tOVUdHL2EzN0oK
  storepass: MDdVK1pEeURxQlNZeTAwQStIdFVtdzhlU0h2SWp3SUU=
---
# Source: openam/templates/secrets.yaml
apiVersion: v1
kind: Secret
metadata:
    name: "openam-pstore-secrets"
type: Opaque
data:
  .keypass: Y2hhbmdlaXQ=
  .storepass: MDdVK1pEeURxQlNZeTAwQStIdFVtdzhlU0h2SWp3SUU=
  keypass: Y2hhbmdlaXQ=
  storepass: MDdVK1pEeURxQlNZeTAwQStIdFVtdzhlU0h2SWp3SUU=
---
```

```
# Source: openam/templates/config-map.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: am-configmap
data:
  CATALINA_OPTS: "-server -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dcom.sun
.identity.util.debug.provider=com.sun.identity.shared.debug.impl.StdOutDebugProvider -Dcom.sun.identity
.shared.debug.file.format='%PREFIX% %MSG%\\n%STACKTRACE%' -Dcom.iplanet.services.stats.state=off
"
---
# Source: openam/templates/config-map.yaml
# Copyright (c) 2016-2017 ForgeRock AS. Use of this source code is subject to the
# Common Development and Distribution License (CDDL) that can be found in the LICENSE file
# Config map holds the boot.json for this instance.
# This is now *DEPRECATED*. The boot.json file is now created by the init container. This is here for
# sample purposes, and will be removed in the future.
apiVersion: v1
kind: ConfigMap
metadata:
  name: boot-json
data:
  boot.json: |
   {
     "instance" : "http://openam:80",
     "dsameUser" : "cn=dsameuser,ou=DSAME Users,ou=am-config",
     "keystores" : {
       "default" : {
         "keyStorePasswordFile" : "/home/forgerock/openam/.storepass",
         "keyPasswordFile" : "/home/forgerock/openam/.keypass",
         "keyStoreType" : "JCEKS",
         "keyStoreFile" : "/home/forgerock/openam/keystore.jceks"
       }
     },
     "configStoreList" : [ {
       "baseDN" : "ou=am-config",
       "dirManagerDN" : "uid=am-config,ou=admins,ou=am-config",
       "ldapHost" : "configstore-0.configstore",
       "ldapPort" : 1389,
       "ldapProtocol" : "ldap"
     } ]
   }
---
# Source: openam/templates/service.yaml
# Copyright (c) 2016-2017 ForgeRock AS. Use of this source code is subject to the
# Common Development and Distribution License (CDDL) that can be found in the LICENSE file
apiVersion: v1
kind: Service
metadata:
  name: openam
  labels:
    app: undercooked-wasp-openam
    chart: "openam-6.5.2"
    component: openam
    vendor: forgerock
spec:
  type: ClusterIP
  ports:
  - port: 80
```

```
    targetPort: 8080
    protocol: TCP
    name: openam
  selector:
    app: openam
    release: undercooked-
wasp
---
# Source: openam/templates/openam-deployment.yaml
# Copyright (c) 2016-2018 ForgeRock AS. All rights reserved
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: undercooked-wasp-openam
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: openam
        component: openam
        vendor: forgerock
        release: undercooked-wasp
        heritage: Tiller
    spec:
      terminationGracePeriodSeconds: 10
      initContainers:

      - name: git-init
        image: gcr.io/forgerock-io/git:6.5.2
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - name: git
          mountPath: /git
        - name: git-secret
          mountPath: /etc/git-secret
        args: ["init"]
        envFrom:
        - configMapRef:
            name:  frconfig

      # The init containers below should be removed once file based configuration is in place.
      - name: wait-for-configstore
        image: "gcr.io/forgerock-io/util:6.0.0"
        imagePullPolicy: "IfNotPresent"
        args: [ "wait", "configstore-0.configstore", "1389" ]
      - name: bootstrap
        image: gcr.io/forgerock-io/util:6.5.2
        imagePullPolicy:  IfNotPresent
        env:
        - name: BASE_DN
          value: ou=am-config
        volumeMounts:
        - name: openam-root
          mountPath: /home/forgerock/openam
        - name: openam-secrets
          mountPath: /var/run/secrets/openam
        - name: openam-boot
          mountPath: /var/run/openam
```

```
        - name: configstore-secret
          mountPath: /var/run/secrets/configstore
        args:  ["bootstrap"]
      containers:
      - name: openam
        image: gcr.io/forgerock-io/openam:6.5.2
        imagePullPolicy:  IfNotPresent
        ports:
        - containerPort: 8080
          name: http
        volumeMounts:
        - name: openam-root
          mountPath: /home/forgerock/openam
        - name: configstore-secret
          mountPath: /var/run/secrets/configstore
        - name: openam-secrets
          mountPath: /var/run/secrets/openam/keystore
        - name: openam-pstore-secrets
          mountPath: /var/run/secrets/openam/password

        - name: git
          mountPath: /git


        envFrom:
        - configMapRef:
            name: am-configmap

        - configMapRef:
            name:  frconfig

        env:
        - name: NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        resources:
          limits:
            memory: 1300Mi
          requests:
            memory: 1200Mi

        # For slow environments like Minikube you need to give OpenAM time to come up.
        readinessProbe:
          httpGet:
            path: /isAlive.jsp
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 5
          periodSeconds: 20
        livenessProbe:
          httpGet:
            path: /isAlive.jsp
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 15
          periodSeconds: 60
      # audit logging containers
      volumes:
```

```
      - name: openam-root
        emptyDir: {}
      - name: openam-secrets
        secret:
          secretName: openam-secrets
      - name: openam-pstore-secrets
        secret:
          secretName: openam-pstore-secrets
      - name: openam-boot
        configMap:
          name: boot-json
      - name: configstore-secret
        secret:
          secretName: configstore
          #defaultMode: 256

      - name: git
        emptyDir: {}
      - name: git-secret
        secret:
          secretName:  frconfig
---
# Source: openam/templates/ingress.yaml
# Copyright (c) 2016-2017 ForgeRock AS. Use of this source code is subject to the
# Common Development and Distribution License (CDDL) that can be found in the LICENSE file
# Ingress definition to configure external routes.
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: openam
  labels:
    app: undercooked-wasp-openam
    vendor: forgerock
    chart: "openam-6.5.2"
    release: "undercooked-wasp"
    heritage: "Tiller"
  annotations:
      kubernetes.io/ingress.class: "nginx"
      nginx.ingress.kubernetes.io/affinity: "cookie"
      nginx.ingress.kubernetes.io/cors-allow-credentials: "false"
      nginx.ingress.kubernetes.io/cors-allow-headers: "authorization,x-requested-with"
      nginx.ingress.kubernetes.io/cors-allow-methods: "PUT,GET,POST,HEAD,PATCH,DELETE"
      nginx.ingress.kubernetes.io/cors-allow-origin: "*"
      nginx.ingress.kubernetes.io/cors-max-age: "600"
      nginx.ingress.kubernetes.io/enable-cors: "true"
      nginx.ingress.kubernetes.io/rewrite-target: "/"
      nginx.ingress.kubernetes.io/session-cookie-hash: "sha1"
      nginx.ingress.kubernetes.io/session-cookie-name: "route"
      nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - login.my-namespace.example.com
    secretName: wildcard.my-namespace.example.com
  rules:
    - host: login.my-namespace.example.com
      http:
        paths:
```

```
        - path: /
          backend:
            serviceName: openam
            servicePort: 80
        - path: /openam
          backend:
            serviceName: openam
            servicePort: 80
```

7.3.5. 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 Fri 2019-08-01 18:13:17 UTC, end at Fri 2017-09-29 19:10:21 UTC. --
Aug  1 18:13:17 minikube localkube[3530]: W0929 18:13:17.106167    3530 docker_sandbox.go:342] failed to
 read pod IP from plugin/docker: Couldn't find network status for default/openam-3041109167-682s7 through
 plugin: invalid network status for
Aug  1 18:13:17 minikube localkube[3530]: E0929 18:13:17.119039    3530 remote_runtime
.go:277] ContainerStatus "b345c269b7569f50fb081545b4983b7dcfa7fdda074cbb0b2c3dc64ac049914f"
 from runtime service failed: rpc error: code = 2 desc = unable to inspect docker image
 "sha256:8f44e9539ae15880c60bae933ead4f6a9c12a8bdbd09c97493370e4dcc90baf0" while inspecting docker
 container "b345c269b7569f50fb081545b4983b7dcfa7fdda074cbb0b2c3dc64ac049914f": no such image:
 "sha256:8f44e9539ae15880c60bae933ead4f6a9c12a8bdbd09c97493370e4dcc90baf0"
. . .
```

Chapter 8
Reference

This reference section provides information that might be useful when deploying the DevOps Examples.

The following topics are covered:

- "Public Git Repositories"
- "YAML File Reference"
- "Notes for Microsoft Windows Users"

8.1. Public Git Repositories

The ForgeRock DevOps Examples and the CDM use the following two public repositories on GitHub:

- `forgeops` **repository**. Contains Dockerfiles, Helm charts, Kubernetes manifests, and utility scripts for deploying the DevOps Examples and the CDM. For information about obtaining the `forgeops` repository, see "forgeops Repository" in the *DevOps Release Notes*.
- `forgeops-init` **repository**. Provides a starter configuration repository as described in "Creating Your Configuration Repository". For information about accessing the `forgeops-init` repository, see "forgeops-init Repository" in the *DevOps Release Notes*.

8.2. YAML File Reference

This section documents key/value pairs commonly used when creating `.yaml` files to be used as input values for Helm chart installation.

The following files are covered:

- "amster.yaml"
- "configstore.yaml"
- "ctsstore.yaml"
- "frconfig.yaml"

- "openam.yaml"
- "openidm.yaml"
- "openig.yaml"
- "userstore.yaml"

8.2.1. amster.yaml

The `amster.yaml` file provides options for setting up the Amster command using the `amster` Helm chart.

For more information about Amster command setup, see "Deploying AM and Amster".

Example `amster.yaml` file with commonly-specified key/value pairs:

```
domain: .my-domain.com
config:
  importPath: /git/config/path/to/AM/configuration
image:
  repository: my-registry/forgerock/amster
```

Key	Description
`domain`	Specifies the AM cookie domain. The value must start with a period. The default value is `.example.com`. The AM server's FQDN is built by appending the `domain` value to the string `openam.` and your namespace. For example, `openam.my-namespace.my-domain.com`.
`config.importPath`	Specifies the path from which the `amster` pod imports AM's configuration from the cloned configuration repository. The default import path is `/git/config/6.5/default/am/empty-import`. In the starter `forgeops-init` configuration repository, this path contains configuration for an empty AM installation. Specify the full path relative to the root directory. Note that the configuration repository is cloned into the path `/git/config`, so the first part of the `importPath` value is typically `/git/config/`. If you export the AM configuration, it is exported to the import path by default. If you want to export the configuration to a different path, specify the `exportPath` key.
`image.repository`	Specifies the Docker registry and repository of the Docker image to be pulled when starting the `amster` pod. The default value is the location of the evaluation-only `amster` Docker image in ForgeRock's public registry. The format of the `image.repository` value is `my-registry/my-repository`, where:

Key	Description
	• *my-registry* is the registry name you specify when running the **docker login** command. • *my-repository* is the Docker repository that contains the image. The repository name ends with the qualifier amster. For example, suppose you store Docker images in a private registry, my-registry.io. To pull an image from the forgerock/amster repository, you would specify my-registry.io/forgerock/amster as the image.repository value.

For the full set of key/value pairs that you can override in the amster.yaml file, see the commented values.yaml file for the amster Helm chart in the forgeops repository.

8.2.2. configstore.yaml

The configstore.yaml file provides options for setting up an AM configuration store using the ds Helm chart. In the AM and DS example, the AM configuration, policies, and application data reside in the configuration store.

For more information about configuration store setup, see "Deploying DS Servers Used in the Example".

Example configstore.yaml file with commonly-specified key/value pairs:

```
instance: configstore
image:
  repository: my-registry/forgerock/ds
```

Key	Description
instance	Specifies the type of DS instance to set up: • For a configuration store, specify configstore. • For a CTS store, specify ctsstore. • For a user store, specify userstore.
image.repository	Specifies the Docker registry and repository of the Docker image to be pulled when starting the ds pod. The default value is the location of the evaluation-only ds Docker image in ForgeRock's public registry. The format of the image.repository value is *my-registry/my-repository*, where: • *my-registry* is the registry name you specify when running the **docker login** command.

Key	Description
	• *my-repository* is the Docker repository that contains the image. The repository name ends with the qualifier ds. For example, suppose you store Docker images in a private registry, my-registry.io. To pull an image from the forgerock/ds repository, you would specify my-registry.io/forgerock/ds as the image.repository value.

For the full set of key/value pairs that you can override in the configstore.yaml file, see the commented values.yaml file for the ds Helm chart in the forgeops repository.

8.2.3. ctsstore.yaml

The ctsstore.yaml file provides options for setting up an AM CTS store using the ds Helm chart.

For more information about CTS store setup, see "Deploying DS Servers Used in the Example".

Example ctsstore.yaml file with commonly-specified key/value pairs:

```
instance: ctsstore
image:
  repository: my-registry/forgerock/ds
```

Key	Description
instance	Specifies the type of DS instance to set up: • For a configuration store, specify configstore. • For a CTS store, specify ctsstore. • For a user store, specify userstore.
image.repository	Specifies the Docker registry and repository of the Docker image to be pulled when starting the ds pod. The default value is the location of the evaluation-only ds Docker image in ForgeRock's public registry. The format of the image.repository value is *my-registry/my-repository*, where: • *my-registry* is the registry name you specify when running the **docker login** command. • *my-repository* is the Docker repository that contains the image. The repository name ends with the qualifier ds. For example, suppose you store Docker images in a private registry, my-registry.io. To pull an image from the forgerock/ds repository, you would specify my-registry.io/forgerock/ds as the image.repository value.

For the full set of key/value pairs that you can override in the ctsstore.yaml file, see the commented values.yaml file for the ds Helm chart in the forgeops repository.

8.2.4. frconfig.yaml

The `frconfig.yaml` file provides information about your configuration repository to the `frconfig` Helm chart.

For more information about configuration repository setup, see "Installing the frconfig Helm Chart".

Example `frconfig.yaml` file with commonly-specified key/value pairs:

```
domain: .my-domain.com
git:
  repo: git@github.com:myAccount/forgeops-init.git
  branch: my-branch
```

Key	Description
`domain`	Specifies the domain of the wildcard certificate created by the certificate manager. The value must start with a period and must match the value of the `domain` key in the `openam.yaml`, `openidm.yaml`, and `openig.yaml` files. The default value is `.example.com`.
`git.repo`	Specifies the URL of the configuration repository. The default value is `https://github.com/ForgeRock/forgeops-init.git`. You must specify an SSH URL if your configuration repository is private, or if you want to modify the AM or IDM configuration after you have deployed these components. You can specify an HTTPS URL if your configuration repository is public and your configuration repository is read-only.
`git.branch`	Specifies the branch that is checked out after cloning the configuration repository. The default value is `master`. Override the default if you maintain ForgeRock Identity Platform configurations in branches in your configuration repository.

For the full set of key/value pairs that you can override in the `frconfig.yaml` file, see the commented `values.yaml` file for the `frconfig` Helm chart in the `forgeops` repository.

8.2.5. openam.yaml

The `openam.yaml` file provides options for setting up an AM server using the `openam` Helm chart.

For more information about AM server setup, see "Deploying AM and Amster".

Example `openam.yaml` file with commonly-specified key/value pairs:

```
domain: .my-domain.com
image:
  repository: my-registry/forgerock/openam
```

Key	Description
`domain`	Specifies the AM cookie domain. The value must start with a period. The default value is `.example.com`. The AM server's FQDN is built by appending the `domain` value to the string `openam.` and your namespace. For example, `openam.my-namespace.my-domain.com`.
`image.repository`	Specifies the Docker registry and repository of the Docker image to be pulled when starting the `openam` pod. The default value is the location of the evaluation-only `openam` Docker image in ForgeRock's public registry. The format of the `image.repository` value is *`my-registry/my-repository`*, where: • *`my-registry`* is the registry name you specify when running the **docker login** command. • *`my-repository`* is the Docker repository that contains the image. The repository name ends with the qualifier `openam`. For example, suppose you store Docker images in a private registry, `my-registry.io`. To pull an image from the `forgerock/openam` repository, you would specify `my-registry.io/forgerock/openam` as the `image.repository` value.

For the full set of key/value pairs that you can override in the `openam.yaml` file, see the commented `values.yaml` file for the `openam` Helm chart in the `forgeops` repository.

8.2.6. openidm.yaml

The `openidm.yaml` file provides options for setting up an IDM server using the `openidm` Helm chart.

For more information about IDM server setup, see "Deploying IDM".

Example `openidm.yaml` file with commonly-specified key/value pairs:

```
domain: .my-domain.com
config:
  path: /git/config/path/to/IDM/configuration
  immutable: false
image:
  repository: my-registry/forgerock/openidm
```

Key	Description
`domain`	Specifies a portion of the FQDN that the Kubernetes ingress controller uses for routing requests to IDM. The value must start with a period. The default value is `.example.com`.

Key	Description
	The IDM server's FQDN is built by appending the `domain` value to the string `openidm.` and your namespace. For example, `openidm.`*`my-namespace.my-domain.com`*.
`config.path`	Specifies the path from which the `openidm` pod reads its configuration from the cloned configuration repository. The pod also writes changes to its configuration to the same path. The default path is `/git/config/6.5/default/idm/sync-with-ldap-bidirectional`. In the starter `forgeops-init` configuration repository, this path contains configuration that implements bidirectional data synchronization between IDM and LDAP. Specify the full path relative to the root directory. Note that the the configuration repository is cloned into the path `/git/config`, so the first part of the `path` value is typically `/git/config/`.
`config.immutable`	Specifies whether the IDM configuration is read-only or read/write. When set to `true`, the IDM configuration is read-only (immutable). When set to `false`, the IDM configuration is read/write (mutable). The default value is `true`.
`image.repository`	Specifies the Docker registry and repository of the Docker image to be pulled when starting the `openidm` pod. The default value is the location of the evaluation-only `openidm` Docker image in ForgeRock's public registry. The format of the `image.repository` value is *`my-registry/my-repository`*, where: • *`my-registry`* is the registry name you specify when running the **docker login** command. • *`my-repository`* is the Docker repository that contains the image. The repository name ends with the qualifier `openidm`. For example, suppose you store Docker images in a private registry, `my-registry.io`. To pull an image from the `forgerock/openidm` repository, you would specify `my-registry.io/forgerock/openidm` as the `image.repository` value.
`sedFilter` (Deprecated)	Specifies a substitution pattern for text in the IDM configuration. After cloning the configuration repository, the openidm pod executes the **sed** command recursively on all the files in the cloned repository, using the provided `sedFilter` value as the **sed** command's argument. Specify a `sedFilter` value when you need to globally modify a string in the configuration. By default, there is no global substitution pattern.

For the full set of key/value pairs that you can override in the `openidm.yaml` file, see the commented `values.yaml` file for the `openidm` Helm chart in the `forgeops` repository.

8.2.7. openig.yaml

The `openig.yaml` file provides options for setting up an IG server using the `openig` Helm chart.

For more information about IG server setup, see "Using Helm to Deploy the IG Example".

Example `openig.yaml` file with commonly-specified key/value pairs:

```
domain: .my-domain.com
config:
  path: /git/config/path/to/IG/configuration
image:
  repository: my-registry/forgerock/openig
```

Key	Description
`domain`	Specifies a portion of the FQDN that the Kubernetes ingress controller uses for routing requests to IG. The value must start with a period. The default value is `.example.com`. The IG server's FQDN is built by appending the `domain` value to the string `openig.` and your namespace. For example, `openig.my-namespace.my-domain.com`.
`config.path`	Specifies the path from which the `openig` pod reads its configuration from the cloned configuration repository. The default path is `/git/config/6.5/default/ig/basic-sample`. This path, available in the starter `forgeops-init` configuration repository, contains configuration that implements a basic IG server with minimal configuration. Specify the full path relative to the root directory. Note that the configuration repository is cloned into the path `/git/config`, so the first part of the `path` value is typically `/git/config/`.
`image.repository`	Specifies the Docker registry and repository of the Docker image to be pulled when starting the `openig` pod. The default value is the location of the evaluation-only `openig` Docker image in ForgeRock's public registry. The format of the `image.repository` value is *`my-registry/my-repository`*, where: • *`my-registry`* is the registry name you specify when running the **docker login** command. • *`my-repository`* is the Docker repository that contains the image. The repository name ends with the qualifier `openig`.

Key	Description
	For example, suppose you store Docker images in a private registry, `my-registry.io`. To pull an image from the `forgerock/openig` repository, you would specify `my-registry.io/forgerock/openig` as the `image.repository` value.

For the full set of key/value pairs that you can override in the `openig.yaml` file, see the commented `values.yaml` file for the `openig` Helm chart in the `forgeops` repository.

8.2.8. userstore.yaml

The `userstore.yaml` file provides options for setting up an AM user store using the `ds` Helm chart.

For more information about user store setup, see "Deploying DS Servers Used in the Example".

Example `userstore.yaml` file with commonly-specified key/value pairs:

```
instance: userstore
image:
  repository: my-registry/forgerock/ds
```

Key	Description
`instance`	Specifies the type of DS instance to set up: • For a configuration store, specify `configstore`. • For a CTS store, specify `ctsstore`. • For a user store, specify `userstore`.
`image.repository`	Specifies the Docker registry and repository of the Docker image to be pulled when starting the `ds` pod. The default value is the location of the evaluation-only `ds` Docker image in ForgeRock's public registry. The format of the `image.repository` value is *`my-registry/my-repository`*, where: • *`my-registry`* is the registry name you specify when running the **docker login** command. • *`my-repository`* is the Docker repository that contains the image. The repository name ends with the qualifier `ds`. For example, suppose you store Docker images in a private registry, `my-registry.io`. To pull an image from the `forgerock/ds` repository, you would specify `my-registry.io/forgerock/ds` as the `image.repository` value.

For the full set of key/value pairs that you can override in the `userstore.yaml` file, see the commented `values.yaml` file for the `ds` Helm chart in the `forgeops` repository.

8.3. Notes for Microsoft Windows Users

This section provides adaptations to instructions in this guide that Microsoft Windows users must make when deploying the DevOps Examples.

Use the following table to determine which notes apply to your deployment:

Environment	Read These Notes
Local Minikube environment	"Notes for All Microsoft Windows Environments" "Additional Notes for Minikube Environments Only"
Cloud-based environment	"Notes for All Microsoft Windows Environments"

8.3.1. Notes for All Microsoft Windows Environments

When running the DevOps Examples on Microsoft Windows, make the following adaptations to the instructions in this guide:

- After installing the software listed in "Installing Required Third-Party Software", do the following:
  - Rename the **kubectl** binary to `kubectl.exe`.
  - Add the **kubectl** and Helm binaries to your `PATH`.
  - Download and install Git for Windows software, which includes the Git Bash shell. This shell lets you execute Bash commands and scripts.
- Because there aren't Windows versions of the **kubectx** and **kubens** context switching utilities, use the **kubectl config** command equivalents instead. For example:
  - To set a new context:

    ```
    $ kubectl config set-context my-context
    ```

  - To set a namespace in the current context:

    ```
    $ kubectl config create namespace my-namespace
    ```

  - To display the current context and namespace:

    ```
    $ kubectl config view
    ```

- Use the Git Bash shell (from the Git for Windows software) when performing the following procedure:
  - "To Create a Kubernetes Secret for Accessing a Private Docker Registry"

8.3.2. Additional Notes for Minikube Environments Only

When running the DevOps Examples on Microsoft Windows in a Minikube environment, make the following adaptations to the instructions in this guide:

- After installing the software listed in "Installing Required Third-Party Software", do the following:
  - Rename the Minikube binary to `minikube.exe`.
  - Add the Minikube binary to your `PATH`.
- Use Hyper-V as the hypervisor for Minikube instead of VirtualBox.[1]

  Verify that your Windows software includes Hyper-V, and that Hyper-V has been enabled. Note that Windows 10 Home Edition does *not* include Hyper-V.
- Create a virtual switch in Hyper-V for Minikube to use.
- Configure your network adapter to allow other network users to connect through the Hyper-V virtual switch.
- When creating the Minikube cluster, specify `hyperv` as the VM driver and specify the Hyper-V virtual switch. For example:

  ```
  $ minikube start --memory=8192 --disk-size=30g \
   --vm-driver=hyperv --hyperv-virtual-switch=my-hyperv-switch --kubernetes-version=v1.11
  ```

  Note that Minikube on Windows does not support the `kubeadm` bootstrapper.

[1] Running Minikube on VirtualBox requires using Docker Toolbox, which is deprecated, instead of using standard Docker software.

Appendix A. Getting Support

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

A.1. ForgeRock DevOps Support

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

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

A.1.1. Commercial Support

ForgeRock provides commercial support for the following DevOps resources:

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

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

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

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

A.1.2. Support Limitations

ForgeRock provides no commercial support for the following:

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

A.1.3. Third-Party Kubernetes Services

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

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

A.2. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

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

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

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

A.3. How to Report Problems or Provide Feedback

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

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

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

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

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

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

A.4. Getting Support and Contacting ForgeRock

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

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

Glossary

affinity (AM)	AM affinity based load balancing ensures that the CTS token creation load is spread over multiple server instances (the token origin servers). Once a CTS token is created and assigned to a session, all subsequent token operations are sent to the same token origin server from any AM node. This ensures that the load of CTS token management is spread across directory servers. Source: *Best practices for using Core Token Service (CTS) Affinity based load balancing in AM*
Amazon EKS	Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on Amazon Web Services without needing to set up or maintain your own Kubernetes control plane. Source: *What is Amazon EKS* in the Amazon EKS documentation.
ARN (AWS)	An Amazon Resource Name (ARN) uniquely identifies an Amazon Web Service (AWS) resource. AWS requires an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies and API calls. Source: *Amazon Resource Names (ARNs) and AWS Service Namespaces* in the AWS documentation.
AWS IAM Authenticator for Kubernetes	The AWS IAM Authenticator for Kubernetes is an authentication tool that enables you to use *Amazon Web Services (AWS)* credentials for authenticating to a Kubernetes cluster. Source: *AWS IAM Authenticator for Kubernetes* `README` file on `GitHub`.

cloud-controller-manager	The `cloud-controller-manager` daemon runs controllers that interact with the underlying cloud providers. `cloud-controller-manager` is an alpha feature introduced in Kubernetes release 1.6. The `cloud-controller-manager` daemon runs cloud-provider-specific controller loops only. Source: *cloud-controller-manager* section in the Kubernetes Concepts documentation.
Cloud Developer's Kit (CDK)	The developer artifacts in the `forgeops` Git repository, together with the ForgeRock Identity Platform documentation form the Cloud Developer's Kit (CDK). Use the CDK to stand up the platform in your developer environment.
Cloud Deployment Model (CDM)	The Cloud Deployment Model (CDM) is a common use ForgeRock Identity Platform architecture, designed to be easy to deploy and easy to replicate. The ForgeRock Cloud Deployment Team has developed Helm charts, Docker images, and other artifacts expressly to build the CDM.
CloudFormation (AWS)	CloudFormation is a service that helps you model and set up your Amazon Web Services (AWS) resources. You create a template that describes all the AWS resources that you want. AWS CloudFormation takes care of provisioning and configuring those resources for you. Source: *What is AWS CloudFormation?* in the AWS documentation.
CloudFormation template (AWS)	An AWS CloudFormation template describes the resources that you want to provision in your AWS stack. AWS CloudFormation templates are text files formatted in JSON or YAML. Source: *Working with AWS CloudFormation Templates* in the AWS documentation.
cluster	A container cluster is the foundation of Kubernetes Engine. A cluster consists of at least one cluster master and multiple worker machines called nodes. The Kubernetes objects that represent your containerized applications all run on top of a cluster. Source: *Container Cluster Architecture* in the Kubernetes Concepts documentation.
cluster master	A cluster master schedules, runs, scales and upgrades the workloads on all nodes of the cluster. The cluster master also manages network and storage resources for workloads. Source: *Container Cluster Architecture* in the Kubernetes Concepts docuementation.
ConfigMap	A configuration map, called `ConfigMap` in Kubernetes manifests, binds the configuration files, command-line arguments, environment

	variables, port numbers, and other configuration artifacts to the assigned containers and system components at runtime. The configuration maps are useful for storing and sharing non-sensitive, unencrypted configuration information. Source: *ConfigMap* in the Kubernetes Cocenpts documentation.
container	A container is an allocation of resources such as CPU, network I/O, bandwidth, block I/O, and memory that can be "contained" together and made available to specific processes without interference from the rest of the system. Source *Container Cluster Architecture* in the Google Cloud Platform documentation
DaemonSet	A set of daemons, called `DaemonSet` in Kubernetes manifests, manages a group of replicated pods. Usually, the daemon set follows an one-pod-per-node model. As you add nodes to a node pool, the daemon set automatically distributes the pod workload to the new nodes as needed. Source *DaemonSet* in the Google Cloud Platform documentation.
Deployment	A Kubernetes deployment represents a set of multiple, identical pods. A Kubernetes deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. Source: *Deployment* in the Google Cloud Platform documentation.
deployment controller	A deployment controller provides declarative updates for pods and replica sets. You describe a desired state in a deployment object, and the deployment controller changes the actual state to the desired state at a controlled rate. You can define deployments to create new replica sets, or to remove existing deployments and adopt all their resources with new deployments. Source: *Deployments* in the Google Cloud Platform documentation.
Docker Cloud	Docker Cloud provides a hosted registry service with build and testing facilities for Dockerized application images; tools to help you set up and manage host infrastructure; and application lifecycle features to automate deploying (and redeploying) services created from images. Source: About Docker Cloud in the Docker Cloud documentation.
Docker container	A Docker container is a runtime instance of a Docker image. A Docker container is isolated from other containers and its host machine. You can control how isolated your container's network,

	storage, or other underlying subsystems are from other containers or from the host machine. Source: Containers section in the Docker architecture documentation.
Docker daemon	The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A Docker daemon can also communicate with other Docker daemons to manage Docker services. Source: *Docker daemon* section in the Docker Overview documentation.
Docker Engine	The Docker Engine is a client-server application with these components: • A server, which is a type of long-running program called a daemon process (the `dockerd` command) • A REST API, which specifies interfaces that programs can use to talk to the daemon and tell it what to do • A command-line interface (CLI) client (the `docker` command) Source: Docker Engine section in the Docker Overview documentation.
Dockerfile	A Dockerfile is a text file that contains the instructions for building a Docker image. Docker uses the Dockerfile to automate the process of building a Docker image. Source: *Dockerfile* section in the Docker Overview documentation.
Docker Hub	Docker Hub provides a place for you and your team to build and ship Docker images. You can create public repositories that can be accessed by any other Docker Hub user, or you can create private repositories you can control access to. An image is an application you would like to run. A container is a running instance of an image. Source: *Overview of Docker Hub* section in the Docker Overview documentation.
Docker image	A Docker image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.

	A Docker image includes the application code, a runtime engine, libraries, environment variables, and configuration files that are required to run the application. An image is an application you would like to run. A container is a running instance of an image. Source: *Docker objects* section in the Docker Overview documentation. Hello Whales: Images vs. Containers in Dockers.
Docker namespace	Docker namespaces provide a layer of isolation. When you run a container, Docker creates a set of namespaces for that container. Each aspect of a container runs in a separate namespace and its access is limited to that namespace. The `PID` namespace is the mechanism for remapping process IDs inside the container. Other namespaces such as net, mnt, ipc, and uts provide the isolated environments we know as containers. The user namespace is the mechanism for remapping user IDs inside a container. Source: *Namespaces* section in the Docker Overview documentation.
Docker registry	A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can also run your own private registry. Source: *Docker registries* section in the Docker Overview documentation.
Docker repository	A Docker repository is a public, certified repository from vendors and contributors to Docker. It contains Docker images that you can use as the foundation to build your applications and services. Source: *Repositories on Docker Hub* section in the Docker Overview documentation.
Docker service	In a distributed application, different pieces of the application are called “services.” Docker services are really just “containers in production.” A Docker service runs only one image, but it codifies the way that image runs including which ports to use, the number replicas the container should run, and so on. By default, the services are load-balanced across all worker nodes. Source: *About services* in the Docker Get Started documentation.
dynamic volume provisioning	The process of creating storage volumes on demand is called dynamic volume provisioning. Dynamic volume provisioning allows storage

	volumes to be created on-demand. It automatically provisions storage when it is requested by users. Source: *Dynamic Volume Provisioning* in the Kubernetes Concepts documentation.
egress	An egress controls access to destinations outside the network from within a Kubernetes network. For an external destination to be accessed from a Kubernetes environment, the destination should be listed as an allowed destination in the whitelist configuration. Source: *Network Policies* in the Kubernetes Concepts documentation.
firewall rule	A firewall rule lets you allow or deny traffic to and from your virtual machine instances based on a configuration you specify. Each Kubernetes network has a set of firewall rules controlling access to and from instances in its subnets. Each firewall rule is defined to apply to either incoming glossary-ingress(ingress) or outgoing (egress) traffic, not both. Source: *Firewall Rules Overview* in the Google Cloud Platform documentation.
garbage collection	Garbage collection is the process of deleting unused objects. Kubelets perform garbage collection for containers every minute and garbage collection for images every five minutes. You can adjust the high and low threshold flags and garbage collection policy to tune image garbage collection. Source: *Garbage Collection* in the Kubernetes Concepts documentation.
Google Kubernetes Engine (GKE)	The Google Kubernetes Engine (GKE) is an environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machine instances grouped together to form a container cluster. Source: *Kubernetes Engine Overview* in the Google Cloud Platform documentation.
ingress	An ingress is a collection of rules that allow inbound connections to reach the cluster services. Source: *Ingress* in the Kubernetes Concepts documentation.
instance group	An instance group is a collection of instances of virtual machines. The instance groups enable you to easily monitor and control the group of virtual machines together.

	Source: *Instance Groups* in the Google Cloud Platform documentation.
instance template	An instance template is a global API resource that you can use to create VM instances and managed instance groups. Instance templates define the machine type, image, zone, labels, and other instance properties. They are very helpful in replicating the environments. Source: *Instance Templates* in the Google Cloud Platform documentation.
kubectl	The **kubectl** command-line tool supports several different ways to create and manage Kubernetes objects. Source: *Kubernetes Object Management* in the Kubernetes Concepts documentation.
kube-controller-manager	The Kubernetes controller manager is a process that embeds core controllers that are shipped with Kubernetes. Logically each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. Source: *kube-controller-manager* in the Kubernetes Reference documentation.
kubelet	A kubelet is an agent that runs on each node in the cluster. It ensures that containers are running in a pod. Source: *kubelets* in the Kubernetes Concepts documentation.
kube-scheduler	The `kube-scheduler` component is on the master node and watches for newly created pods that do not have a node assigned to them, and selects a node for them to run on. Source: *Kubernetes components* in the Kubernetes Concepts documentation.
Kubernetes	Kubernetes is an open source platform designed to automate deploying, scaling, and operating application containers. Source: *Kubernetes Concepts*
Kubernetes DNS	A Kubernetes DNS pod is a pod used by the kubelets and the individual containers to resolve DNS names in the cluster. Source: *DNS for services and pods* in the Kubernetes Concepts documentation.

Kubernetes namespace	A Kubernetes namespace is a virtual cluster that provides a way to divide cluster resources between multiple users. Kubernetes starts with three initial namespaces: • `default`: The default namespace for user created objects which don't have a namespace • `kube-system`: The namespace for objects created by the Kubernetes system • `kube-public`: The automatically created namespace that is readable by all users Kubernetes supports multiple virtual clusters backed by the same physical cluster. Source: *Namespaces* in the Kubernetes Concepts documentation.
Let's Encrypt	Let's Encrypt is a free, automated, and open certificate authority. Source: Let's Encrypt web site.
network policy	A Kubernetes network policy specifies how groups of pods are allowed to communicate with each other and with other network endpoints. Source: *Network policies* in the Kubernetes Concepts documentation.
node (Kubernetes)	A Kubernetes node is a virtual or physical machine in the cluster. Each node is managed by the master components and includes the services needed to run the pods. Source: *Nodes* in the Kubernetes Concepts documentation.
node controller (Kubernetes)	A Kubernetes node controller is a Kubernetes master component that manages various aspects of the nodes such as: lifecycle operations on the nodes, operational status of the nodes, and maintaining an internal list of nodes. Source: *Node Controller* in the Kubernetes Concepts documentation.
persistent volume	A persistent volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins that have a lifecycle independent of any individual pod that uses the PV. Source: *Persistent Volumes* in the Kubernetes Concepts documentation.
persistent volume claim	A persistent volume claim (PVC) is a request for storage by a user. A PVC specifies size, and access modes such as:

	• Mounted once for read and write access • Mounted many times for read-only access Source: *Persistent Volumes* in the Kubernetes Concepts documentation.
pod anti-affinity (Kubernetes)	Kubernetes pod anti-affinity allows you to constrain which nodes can run your pod, based on labels on the **pods** that are already running on the node rather than based on labels on nodes. Pod anti-affinity enables you to control the spread of workload across nodes and also isolate failures to nodes. Source: *Inter-pod affinity and anti-affinity*
pod (Kubernetes)	A Kubernetes pod is the smallest, most basic deployable object in Kubernetes. A pod represents a single instance of a running process in a cluster. Containers within a pod share an IP address and port space. Source: *Understanding Pods* in the Kubernetes Concepts documentation.
replication controller	A replication controller ensures that a specified number of Kubernetes pod replicas are running at any one time. The `replication controller` ensures that a pod or a homogeneous set of pods is always up and available. Source: *ReplicationController* in the Kubernetes Concepts documentation.
secret (Kubernetes)	A Kubernetes secret is a secure object that stores sensitive data, such as passwords, OAuth 2.0 tokens, and SSH keys in your clusters. Source *Secrets* in the Kubernetes Concepts documentation.
security group (AWS)	A security group acts as a virtual firewall that controls the traffic for one or more compute instances. Source: *Amazon EC2 Security Groups* in the AWS documentation.
service (Kubernetes)	A Kubernetes service is an abstraction which defines a logical set of pods and a policy by which to access them. This is sometimes called a microservice. Source: *Services* in the Kubernetes Concepts documentation.
shard	Sharding is a way of partitioning directory data so that the load can be shared by multiple directory servers. Each data partition, also

	known as a *shard*, exposes the same set of naming contexts, but only a subset of the data. For example, a distribution might have two shards. The first shard contains all users whose name begins with A-M, and the second contains all users whose name begins with N-Z. Both have the same naming context. Source: *Class Partition* in the *OpenDJ Javadoc*.
stack (AWS)	A stack is a collection of AWS resources that you can manage as a single unit. You can create, update, or delete a collection of resources by using stacks. All the resources in a stack are defined by the template. Source: *Working with Stacks* in the AWS documentation.
stack set (AWS)	A stack set is a container for stacks. You can provision stacks across AWS accounts and regions by using a single AWS template. All the resources included in each stack of a stack set are defined by the same template. Source: *StackSets Concepts* in the AWS documentation.
volume (Kubernetes)	A Kubernetes volume is a storage volume that has the same lifetime as the pod that encloses it. Consequently, a volume outlives any containers that run within the pod, and data is preserved across container restarts. When a pod ceases to exist, the Kubernetes volume also ceases to exist. Source: *Volumes* in the Kubernetes Concepts documentation.
VPC (AWS)	A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. Source: *What Is Amazon VPC?* in the AWS documentation.
worker node (AWS)	An Amazon Elastic Container Service for Kubernetes (Amazon EKS) worker node is a standard compute instance provisioned in Amazon EKS. Source: *Worker Nodes* in the AWS documentation.
workload (Kubernetes)	A Kubernetes workload is the collection of applications and batch jobs packaged into a container. Before you deploy a workload on a cluster, you must first package the workload into a container. Source: *Understanding Pods* in the Kubernetes Concepts documentation.