OpenIG PROJECT

Deployment Guide

/ ForgeRock Identity Gateway 5

Latest update: 5.0.0

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

Abstract

Tutorials for deploying ForgeRock® Identity Gateway with Docker, with best practices for containerized deployment in production environments.

Table of Contents

Preface

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

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

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

1. About This Guide

This document describes how to deploy basic and customized configurations of ForgeRock Identity Gateway through Docker. To help you prepare for production deployments, it describes best practices for managing the secret and public configuration parameters that change from one deployment to another.

In progressive stages, you will configure the software to use OpenID Connect to authenticate to your Google login page, and then deploy that configuration through Docker.

This guide introduces information about orchestrating the deployment of multiple containers, with tools such as Kubernetes and Minikube.

This guide is for:

- **ForgeRock Identity Platform developers** who want a first look and easy-to-use example of containerized deployment.
- **ForgeRock Identity Gateway developers** who want to configure a production environment for containerized deployment.

Although full instructions are given in the tutorials, it will help to be familiar with the following subjects:

- **ForgeRock Identity Gateway**, to edit the basic configuration and test the changes.
- **Docker**, to build and run and Docker images. The provided Dockerfiles are extensively commented, and the tutorials contain the Docker instructions and options that you need to use.
- **Git**, to clone a Git repository onto your local workspace so that you can download the example and files.
- **OpenID Connect 1.0** and **Google Developer Console**, to configure the tutorials.

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

3. 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.

4. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

5. 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, visit https://www.forgerock.com, or send an email to ForgeRock at info@forgerock.com.

Chapter 1
Before You Start

This guide describes how to deploy OpenIG with Docker, using a set of sample files. It provides an overview and a link to a sample that demonstrates OpenIG running on Kubernetes.

A typical deployment, as described in Chapter 2, "*Getting Started*" in the *Gateway Guide*, includes the following steps:

- Download and install a web container, such as Tomcat or Jetty.
- Download the OpenIG software.
- Install the software into the root context of the web container.
- Configure it to protect your application.

Using containers and Docker to deploy OpenIG simplifies deployment, especially when you deploy multiple instances.

1.1. About the Sample Files

The sample files and OpenIG configurations used in this guide are available for you to clone or download from a Git repository. This section describes the repository and files, and describes how to get a local copy of the files.

The sample files are stored in https://github.com/ForgeRock/openig-devops-guide. The repository contains a folder for each tutorial:

- `docker/sample1-base` contains the Dockerfile you need to create and run an OpenIG base image. Chapter 2, "*Deploying a Base Image in Docker*" describes how to build and run the image.
- `docker/sample2-config` contains an OpenIG configuration to display a simple "Hello World" page, and a Dockerfile that refers to the base image. Chapter 3, "*Deploying a Basic Configuration in Docker*" describes how to add the OpenIG configuration, and build and run a new image.
- `docker/sample3-oidc` contains an OpenIG configuration to use a social login to access a page and a Dockerfile that refers to the base image. Chapter 4, "*Deploying a Configuration With OpenID Connect 1.0 in Docker*" describes how to create and use client data in the configuration, and build and run a new image.

Chapter 5, "*Preparing For Production Deployment With Docker*" presents considerations for separating the configuration from the code, to make it easier to deploy the same code in different environments.

- `kubernetes/sample4-kubernetes` introduces an OpenIG configuration, a Dockerfile, scripts, and yaml files to demonstrate OpenIG running on Kubernetes. Chapter 6, "*Orchestrating Deployment With Kubernetes*" contains a brief overview of the file and some pointers about how to use them. Detailed instructions are in the `readme` in the Git repository.

The samples are supported on macOS and native Linux hosts, and are tested on Docker 1.12. The versions of software used to test the sample described in Chapter 6, "*Orchestrating Deployment With Kubernetes*" are listed in the `readme` in the Git repository for the sample.

Some operating systems require you create a docker group or to run the commands with **sudo**.

1.2. Getting the Sample Files

Commands used in this guide assume that you store the sample files in `/path/to/openig-devops-guide`. Adapt the commands if you use a different directory.

Clone or download the sample files:

- To clone the files, go to your installation directory and run this command:

  ```
  $ git clone https://github.com/ForgeRock/openig-devops-guide.git
  ```

  The sample directories and files are copied into `/path/to/openig-devops-guide`.

- To download the files, go to the Git repository on https://github.com/ForgeRock/openig-devops-guide and select Clone or Download > Download ZIP.

  Move and unzip the downloaded folder into your installation directory. The tutorial folders are in `/path/to/openig-devops-guide`.

1.3. Getting the OpenIG .war File

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

Copy (or move) and rename the .war file to the sample directories:

```
$ cp IG-5.0.0.war /path/to/openig-devops-guide/docker/sample1-base/openig.war
```

1.4. Configuring the Network

The samples in /path/to/openig-devops-guide/docker run Docker on your local machine, where the IP address is usually 127.0.0.1. Add the following entry to your /etc/hosts file:

```
127.0.0.1  localhost  openig.example.com
```

1.5. Choosing a Port

The samples in this guide run OpenIG on port 8080. Before you run any Docker images, make sure that port 8080 is available. Alternatively, use a different port and adjust the commands accordingly.

To run a Docker image on a port other than 8080, in the **Docker run** command replace my-port with the other port number:

```
$ docker run -p <my-port>:8080 <docker-image>
```

Chapter 2

Deploying a Base Image in Docker

This chapter describes how to deploy OpenIG by building and running a Docker image. At the end of this chapter, you can access the OpenIG welcome page.

2.1. Understanding the Configuration

The Dockerfile for `sample1-base` is in `/path/to/openig-devops-guide/docker/sample1-base`. It contains a series of instructions to specify the installation environment for OpenIG, the location of the OpenIG `.war` file, the web container, installation directories, environment variables, and port numbers to use.

OPENIG_BASE and the OpenIG configuration files are in the following directories:

- OPENIG_BASE in `/var/openig`
- CATALINA_HOME in `/usr/local/tomcat`

The Dockerfile uses the OpenIG `.war` file to build the base image. The Dockerfiles in the other samples build on this base image.

To help you to customize your installation, the Dockerfile is commented with information about the configuration options that you can change. Read the Dockerfile to understand the configuration options.

2.2. Building and Running the OpenIG Base Image

Before you start:

- Read and follow the instructions in Chapter 1, "*Before You Start*". You should have Docker installed, the sample files and the OpenIG .war file downloaded, and the network configured.
- Make sure that port 8080 is not being used, or, alternatively, replace port 8080 as described in Section 1.5, "Choosing a Port".
- Start Docker or make sure it is running.

After running the procedure, remember to stop the OpenIG instance by pressing CTRL-C in the terminal that is running Tomcat.

Procedure 2.1. To Build and Run a Docker Image

1. In a terminal window, go to `/path/to/openig-devops-guide/docker/sample1-base`.

   Take a moment to review the Dockerfile in that directory. The commands and options in the Dockerfile are used to build the OpenIG image.

2. Run the Docker instruction to build a Docker image called `forgerock/openig-base`:

   ```
   $ docker build -t forgerock/openig-base .
   ```

   It takes a few minutes to build a base image. While it is building, the status is displayed. When it is built, a success message is displayed along with the ID of the image.

3. Check that the base image is built:

   ```
   $ docker images
   REPOSITORY               TAG                  IMAGE ID           CREATED            SIZE
   forgerock/openig-base    latest               5a063649fc87       57 seconds ago     428 MB
   ```

   The forgerock/openig-base image should be listed in the repository.

4. Run the Docker image on port 8080:

   ```
   $ docker run -p 8080:8080 forgerock/openig-base
   ```

   Tomcat starts up and the console displays the OpenIG message log. Read the message log to see the progress of the startup. When the startup is complete, a message similar to this is displayed:

   ```
   org.apache.catalina.startup.Catalina.start Server startup in 8385 ms
   ```

5. Go to http://openig.example.com:8080 to view the OpenIG welcome page and confirm that your OpenIG image is running.

6. In the terminal that is running the Docker image, press CTRL-C to stop the image.

2.3. Stopping a Docker Image

If a Docker image is running in the foreground, press CTRL-C in the terminal that is running Tomcat.

If a Docker image is running in the background:

- List the Docker images that are running:

  ```
  $ docker ps
  CONTAINER ID       IMAGE                    COMMAND                  CREATED            STATUS . . .
  fe3768b9c55f       forgerock/openig-base    "/usr/local/tomcat/bi"   11 minutes ago     Up 11 minutes
  . . .
  ```

- If an image has the status `Up`, use the container ID to stop the image:

  ```
  docker stop fe3768b9c55f
  ```

Chapter 3
Deploying a Basic Configuration in Docker

This chapter describes how to make a small change to the configuration of OpenIG and then build and run a new Docker image with the changed configuration. This chapter demonstrates how easy it is to propagate a configuration change into a Docker image.

3.1. Understanding the Configuration

The Dockerfile for `sample2-config` uses the base image created in Chapter 2, "*Deploying a Base Image in Docker*", and specifies the location of the OpenIG configuration files.

The OpenIG configuration file configures a static response handler to return a simple "Hello World" page when the URI of a request finishes with `/hello`.

3.2. Customizing OpenIG and Building a New Docker Image

Before you start:

- Complete the tutorial in Chapter 2, "*Deploying a Base Image in Docker*".
- If an OpenIG image is still running, stop it as described in Section 2.3, "Stopping a Docker Image".

Procedure 3.1. To Customize OpenIG and Build a New Docker Image

1. In a terminal window, go to `/path/to/openig-devops-guide/docker/sample2-config`.

   Take a moment to review the Dockerfile in that directory. The Dockerfile refers to base image you built in Chapter 2, "*Deploying a Base Image in Docker*", and adds the new OpenIG configuration.

2. Review the OpenIG configuration file `custom-config/config/routes/01-hello.json`:

```
{
  "handler": {
    "type": "StaticResponseHandler",
    "config" : {
      "status": 200,
      "reason": "OK",
      "entity": "Hello, world! Your route is ${request.uri.path}"
    }
  },
  "condition": "${matches(request.uri.path, '^/hello')}"
}
```

This configuration contains a static response handler to return a "Hello World" statement when the URI of a request finishes with `/hello`.

3. From `/path/to/openig-devops-guide/docker/sample2-config`, run the Docker instruction to build a new Docker image called `sample2-config`:

   ```
   $ docker build -t sample2-config .
   Step 1 : FROM forgerock/openig-base:latest
    ---> a93fdd6074b8
   Step 2 : ADD custom-config /var/openig
    ---> a720e5472a76
   Removing intermediate container 1507feec0b39
   ```

   When the image is built, a success message is displayed along with the ID of the image.

4. Check that the new image is built:

   ```
   $ docker images
   REPOSITORY              TAG                 IMAGE ID            CREATED              SIZE
   sample2-config          latest              a720e5472a76        About a minute ago   428 MB
   forgerock/openig-base   latest              a93fdd6074b8        3 minutes ago        428 MB
   ```

   The sample2-config image should be in the list of repositories.

5. Run the Docker image on port 8080:

   ```
   $ docker run -p 8080:8080 sample2-config
   ```

   Tomcat starts up and the console displays the OpenIG message log. Read the message log to see the progress of the startup. When the startup is complete, a message similar to this is displayed:

   ```
   org.apache.catalina.startup.Catalina.start Server startup in 8385 ms
   ```

6. Test that OpenIG is running on http://openig.example.com:8080/hello.

   A page displaying the "Hello World" statement is displayed.

7. In the terminal that is running the Docker image, press CTRL-C to stop the image.

Chapter 4

Deploying a Configuration With OpenID Connect 1.0 in Docker

This chapter describes how to set up a more complex configuration of OpenIG that uses OpenID Connect 1.0 and Google credentials to log you in to a web page.

4.1. Understanding the Configuration

The Dockerfile for `sample3-oidc` refers to the base image from Chapter 2, "*Deploying a Base Image in Docker*", and specifies the location of the OpenIG configuration files. The OAuth 2.0 credentials you add to this file are referenced in the OpenIG configuration.

The OpenIG configuration file is described in Section 4.4, "Reviewing the OpenIG Configuration".

4.2. Setting Up OAuth 2.0 Credentials

Procedure 4.1. To Set Up OAuth 2.0 Credentials

1. Browse to https://console.cloud.google.com/apis/credentials.

2. Create credentials for an OAuth 2.0 client ID with the following options:

   - Application type: `Web application`

   - Authorized redirect URI: `http://openig.example.com:8080/openid/callback`

   A client ID and a client secret are created. Make a note of their values or keep the site open for the next step.

4.3. Preparing the Dockerfile

Procedure 4.2. To Prepare the Dockerfile

1. In a terminal window, go to `/path/to/openig-devops-guide/docker/sample3-oidc`.

2. Edit the Dockerfile to replace `<your-client-ID>` and `<your-client-secret>` with the values you created in Section 4.2, "Setting Up OAuth 2.0 Credentials":

```
FROM forgerock/openig-base:latest

#  Replace <your-client-ID> and <your-client-secret> with your own values.
ENV CLIENT_ID <your-client-ID>
ENV CLIENT_SECRET <your-client-secret>
#  These environment variables are referenced in the openid.json config file.
#  To override these values, pass them as environment variables in the docker run command.

ADD custom-config /var/openig
```

4.4. Reviewing the OpenIG Configuration

Procedure 4.3. To Review the OpenIG Configuration

- Review the OpenIG configuration file `/custom-config/config/routes/07-openid.json`:

```
{
  "heap": [
    {
      "name": "accounts.google.com",
      "type": "Issuer",
      "config": {
        "wellKnownEndpoint": "https://accounts.google.com/.well-known/openid-configuration"
      }
    },
    {
      "name": "OidcRelyingParty",
      "type": "ClientRegistration",
      "config": {
        "clientId": "${env['CLIENT_ID']}",
        "clientSecret": "${env['CLIENT_SECRET']}",
        "issuer": "accounts.google.com",
        "scopes": [
          "openid",
          "profile"
        ]
      }
    },
    {
      "name": "capture",
      "type": "CaptureDecorator",
      "config": {
        "captureEntity": true,
        "_captureContext": true
      }
    }
  ],
  "handler": {
    "type": "Chain",
  "capture": "all",
    "config": {
      "filters": [
        {
          "type": "OAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/openid",
            "requireHttps": false,
            "requireLogin": true,
            "registrations": "OidcRelyingParty",
            "target": "${attributes.openid}",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "comment": "Trivial failure handler for debugging only",
                "status": 500,
                "reason": "Error",
                "entity": "${attributes.openid}"
              }
            }
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "entity": "<html><h1>Welcome to the OIDC Test Page</h1><body><p><b>Your are:</b>
 ${attributes.openid.user_info.name}</p><p><b>Working on host:</b> ${env['HOSTNAME']}</p></body></html>"
        }
      }
    }
  },
  "condition": "${matches(request.uri.path, '^/openid')}"
}
```

Notice the following features of the route:

- The `Issuer` and `ClientRegistration` are defined in the heap. The registration refers to the issuer `accounts.google.com` as as the OpenID provider.

  For more information, see Issuer(5) in the *Configuration Reference* and ClientRegistration(5) in the *Configuration Reference*.

- The `OAuth2ClientFilter` enables OpenIG to act as a relying party, referring to the client registration.

  The client endpoint `/openid` causes requests to `/openid` start the delegated authorization process.

  For convenience in this test, `"requireHttps"` is false. In production environments, set it to true. So that you see the delegated authorization process when you make a request, `"requireLogin"` is true.

The target for storing authorization state information is `${attributes.openid}`. This is where subsequent filters and handlers can find access tokens and user information.

If the request fails, the `failureHandler`, in this case a `StaticResponseHandler`, dumps the information in the context into a web page response. While this is helpful for debugging, in production environments consider returning a more user-friendly failure page.

The `StaticResponseHandler` retrieves the user name and host from the context and displays them on a test page.

For more information, see OAuth2ClientFilter(5) in the *Configuration Reference*.

- The route matches requests to `/openid`.

4.5. Building and Running the OpenID Connect 1.0 Sample

Before you start:

- Complete the tutorial in Chapter 2, "*Deploying a Base Image in Docker*".
- If an OpenIG image is still running, stop it as described in Section 2.3, "Stopping a Docker Image".

After running the procedure, remember to stop the OpenIG instance by pressing CTRL-C in the terminal that is running Tomcat.

Procedure 4.4. To Build and Run the OpenID Connect 1.0 Sample

1. In a terminal window, go to `/path/to/openig-devops-guide/docker/sample3-oidc`.

2. Run the Docker instruction to build a new Docker image called `sample3-oidc`:

```
$ docker build -t sample3-oidc .
Sending build context to Docker daemon 14.34 kB
Step 1 : FROM forgerock/openig-base:latest
 ---> a385ade7625e
Step 2 : ENV CLIENT_ID . . .
 ---> Running in 14c20dec7d5c
 ---> b8e4ccefbc83
Removing intermediate container 14c20dec7d5c
Step 3 : ENV CLIENT_SECRET . . .
 ---> Running in 36eae9fb75c9
 ---> f485785a8a0c
Removing intermediate container 36eae9fb75c9
Step 4 : ADD custom-config /var/openig
 ---> fb3e4fdd24fb
Removing intermediate container 8ce983caface
Successfully built fb3e4fdd24fb
```

   The build uses the base image from Chapter 2, "*Deploying a Base Image in Docker*", adds the OAuth 2.0 credentials to the image, and then adds the new OpenIG configuration to the image.

3. Check that the new image is built:

```
$ docker images
REPOSITORY               TAG                   IMAGE ID            CREATED              SIZE
sample3-oidc             latest                fb3e4fdd24fb        About a minute ago   428 MB
sample2-config           latest                a720e5472a76        10 minutes ago       428 MB
forgerock/openig-base    latest                a93fdd6074b8        20 minutes ago       428 MB
```

   The sample3-oidc image should be in the list of repositories.

4. Run the Docker image on port 8080:

```
$ docker run -p 8080:8080 sample3-oidc
```

   Tomcat starts up and the console displays the OpenIG message log. Read the message log to see the progress of the startup. When the startup is complete, a message similar to this is displayed:

```
org.apache.catalina.startup.Catalina.start Server startup in 8385 ms
```

5. Test the setup on http://openig.example.com:8080/openid.

   Depending on whether you logged out of Google, you might need to authenticate.

   The Google login is used to give you access to the OIDC Test Page, and you are asked to allow access. You should see a screen showing something like this:

```
Welcome to the OIDC Test Page

      Howdy George Costanza

      You are working on host a0b1c2345d67
```

   What is happening behind the scenes:

   - After OpenIG gets the browser request, the `OAuth2ClientFilter` redirects you to authenticate with Google. After authentication, Google returns an access token to the filter.

   - The `OAuth2ClientFilter` uses the access token to get the user information, and then injects the authorization state information into `attributes.openid`. The outermost chain then calls the static response handler.

6. In the terminal that is running the Docker image, press CTRL-C to stop the image.

Chapter 5

Preparing For Production Deployment With Docker

This chapter sets out considerations for deploying OpenIG in multiple containers in a production environment.

5.1. Separating Configuration From Code

Many of the configuration parameters in Chapter 4, "*Deploying a Configuration With OpenID Connect 1.0 in Docker*" are hard-coded into the configuration files. The client ID and client secret, for example, are hard-coded in the Dockerfile. The identity issuer is hard-coded in the OpenIG route.

Although this setup works fine for our example, it has limited use in a production environment, where the values for client ID, client secret, and issuer would probably change from one deploy to another.

Separating configuration parameters from the code has many advantages, including flexibility and security. When the configuration parameters are defined outside the code, the same code can be used in multiple deployments with less need for reconfiguration. When confidential information, such as a client ID or secret, are provided separately to the code, they are less likely to be checked in to version control.

5.1.1. Configuration Parameters to Consider Storing as Environment Variables

This section lists some of the OpenIG configuration parameters that you should consider storing in environment variables instead of in the code.

This list includes secret configuration parameters:

- Key pair for JWTSession encryption/decryption
- OIDC client ID and secret
- OpenAM agent password
- CryptoHeaderFilter keys
- KeyStore credentials

This list includes other, non-secret configuration parameters:

- Host names
- Port numbers
- Configuration base
- baseUri values
- URLs
- Redirect URIs
- SQL data source strings
- JDBC URL for JDBC audit event handlers

5.1.2. Removing Configuration Parameters from the Dockerfile

In Chapter 4, "*Deploying a Configuration With OpenID Connect 1.0 in Docker*" the client ID and client secret are hard-coded in the Dockerfile, and the identity issuer, Google, is hard-coded in the OpenIG route.

To override the credentials in the Dockerfile, or to set them in the command instead of in the Dockerfile, include the credentials when you run the docker image:

```
$ docker run --env CLIENT_ID=your-client-ID --env CLIENT_SECRET=your-client-secret -it -p 8080:8080
 sample3-oidc
```

5.2. Making the Deployment Immutable

OpenIG operates in development mode (mutable mode) and production mode (immutable mode):

Development mode

Use development mode to evaluate or demo OpenIG, or to develop configurations on a single instance. This mode is not suitable for production.

In development mode, by default all endpoints are open and accessible. You can create, edit, and deploy routes through OpenIG Studio, and manage routes through Common REST, without authentication or authorization.

To protect specific endpoints in development mode, configure an ApiProtectionFilter in `admin.json` and add it to the OpenIG configuration.

Production mode

After you have developed your configuration, switch to production mode to test the configuration, to run the software in pre-production or production, or to run multiple instances of the software with the same configuration.

In production mode, the `/routes` endpoint is not exposed or accessible. OpenIG Studio is effectively disabled, and you cannot manage, list, or even read routes through Common REST.

By default, other endpoints, such as `/monitoring`, `/share`, and `api/info` are exposed to the loopback address only. To change the default protection for specific endpoints, configure an ApiProtectionFilter in `admin.json` and add it to the OpenIG configuration.

The default mode is development. For information about making the configuration immutable, see Procedure 3.2, "To Switch From Development Mode to Production Mode" in the *Gateway Guide*.

Chapter 6

Orchestrating Deployment With Kubernetes

In the earlier chapters of this guide you used Docker to automate the deployment of OpenIG in a container. This chapter describes some of the options for managing clusters of OpenIG containers as a single system, by using Kubernetes and Minikube.

After you have started using Docker containers to deploy OpenIG, the next challenge is to scale and start the containers across multiple Docker hosts, logically grouping the containers into pools for load balancing and affinity.

To make it easier to scale horizontally and reuse containers, you should almost always run a single process in a single container. In real applications, containers are connected together and orchestrated.

Container orchestration tools simplify container management, provide a framework for the initial container deployment, and manage multiple containers as one entity, facilitating availability, scaling, and networking.

There are many container orchestration tools. This chapter introduces an example of Kubernetes, running locally with Minikube.

6.1. About Kubernetes and Minikube

Kubernetes is an open source project to manage and run a cluster of Docker containers across multiple hosts. Kubernetes offers the following features on top of Docker:

- Container networking
- Service discovery
- Self healing, where containers are monitored and restarted

For an overview of Kubernetes, see *What is Kubernetes?*.

For development and testing, you can use Minikube to start a single-node Kubernetes cluster locally. Minikube packages and configures a VM, Docker, and the Kubernetes components required for local development.

6.2. Understanding the Kubernetes Sample for Deploying OpenIG

The sample in `sample4-kubernetes` contains an OpenIG configuration, a Dockerfile, scripts, and yaml files to demonstrate OpenIG running on Kubernetes. The sample has been tested on Minikube.

The sample deploys OpenIG and a sample application to demonstrate the following features:

- Authentication to Google through OpenID Connect 1.0
- Password replay to access the sample application
- Throttling the rate of requests to the sample application

For detailed information about how to run the sample, see the `readme` in `sample4-kubernetes`. This section describes how to prepare your environment for the sample.

6.3. Preparing to Run the Sample

Do the following before you try to run the Kubernetes sample:

- Read and follow the instructions in Chapter 1, "*Before You Start*". You should have Docker installed, the OpenIG .war file downloaded, and the network configured. The files for this sample should be in `/path/to/openig-devops-guide/kubernetes/sample4-kubernetes`.
- Make sure you have installed the software listed in the readme in the `sample4-kubernetes` directory. Docker, VirtualBox, kubectl, and Minikube should all be installed.
- Configure your network by adding the IP address of Minikube to your `/etc/hosts` file, as described in Section 1.4, "Configuring the Network".
- Build the OpenIG image, as described in Chapter 2, "*Deploying a Base Image in Docker*".