Introduction to Containers and Docker v1.6

Containers are a way of isolating programs or processes from each other. The primary aim of containers is to make programs easy to deploy in a way that doesn't cause them to break.

It's easy to start using containers without being familiar with the technology that makes them work.

In this lab, you learn how to build, run, and distribute an application as a Docker image.

Setup

Step 1

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click Start Lab, shows how long Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access the Google Cloud Platform for the duration of the lab.

What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal GCP account or project, do not use it for this lab.

Step 2

In your project, you have a pre-provisioned VM running Ubuntu Xenial and the necessary tools pre-installed. To connect to it:

Click the **Navigation menu** icon (top-left). 
Select **Compute > Compute Engine > VM Instances**.
Your instance is listed as `k8s-workshop-module-1-lab`.
To the right of the instance, click the **SSH** drop-down arrow and select **Open in browser window**. You may need to hide the **Info Panel** on the right to see the drop-down.
You may need to click SSH twice, if pop-up is blocked (You can click on Pop-up blocked in the Address bar).
This opens another window, copies SSH keys to the host, and logs you in.

Step 3

Make sure the instance is fully provisioned. To do this, run the following command and look for the `kickstart` directory.

```
ls /
```

If the directory is not there, give the instance a few minutes to get fully provisioned before continuing. We've seen it take up to 8 minutes sometimes.

Run and Distribute Containers With Docker

Docker provides a simple means to package applications as containers with a repeatable execution environment.

Explore Docker by creating and running a simple Docker container image that includes a web server written in Python, upload it to a Docker registry, and share it with everyone so it can be run anywhere that supports Docker.

In this lab, you learn how to:

- Build a Docker image.
- Push a Docker image to Google Cloud Registry.
- Run a Docker container.

Run the Web Server Manually

To install and run even a simple web server, you will have dependencies like `apt` and `pypi` (Python) for deployment. Versions of dependencies frequently change, so it's useful to automate the process of getting the latest dependency versions at installation time.
Run the web server manually to see the steps. Later, you will automate the process to run it on other machines.

Step 1

The source code for this lab is available in the `/kickstart` folder. Switch to that directory.

```
cd /kickstart
```

And list the contents.

```
ls -lh
```

You should see a `Dockerfile` and `web-server.py`. web-server.py is a simple Python application that runs a web server which responds to HTTP requests on localhost:8888 and outputs the hostname.

Step 2

Install dependencies.

Install the latest version of Python and PIP.

```
sudo apt-get install -y python3 python3-pip
```

Install Tornado library that is required by the application.

```
pip3 install tornado
```

Step 3

Run the Python application in the background.

```
python3 web-server.py &
```

Step 4

Ensure that the web server is accessible.

```
curl http://localhost:8888
```

The response should look like this:

```
Hostname: k8s-workshop-module-1-lab
```

Step 5

Terminate the web server.

```
kill %1
```

Package Using Docker

Now, see how Docker can help. Docker images are described via Dockerfiles. Docker allows the stacking of images. Your Docker image will be built on top of an existing Docker image library/python that has Python pre-installed.

Step 1

Look at the Dockerfile.

```
cat Dockerfile
```

Step 2

Build a Docker image with the web server.

The image is stored in the local image store.

```
sudo docker build -t py-web-server:v1 .
```

Be sure to include the '.' at the end of the command. This tells Docker to start looking for the `Dockerfile` in the current working directory.

Step 3

Run the web server using Docker.

```
sudo docker run -d -p 8888:8888 --name py-web-server -h my-web-server py-web-server:v1
```

Step 4

Try accessing the web server again, and then stop the container.

```
curl http://localhost:8888
sudo docker rm -f py-web-server
```

The web server and all its dependencies, including the `python` and `tornado` library, have been packaged into a single Docker image that can now be shared with everyone. The `py-web-server:v1` docker image functions the same way on all Docker supported OSes (OS X, Windows, and Linux).

Upload the Image to a Registry

The Docker image needs to be uploaded to a Docker registry to be available for use on other machines. Upload the Docker image to your private image repository in Google Cloud Registry (gcr.io).

Step 1

Add the signed in user to the Docker group so you can run docker commands without `sudo` and push the image to the repository as an authenticated user using the Container Registry credential helper.

```
sudo usermod -aG docker $USER
```

Step 2

Restart the `SSH` session so the group change takes effect and return to the `kickstart` directory.

```
cd /kickstart
```

Step 3

Store your GCP project name in an environment variable.

```
export GCP_PROJECT=`gcloud config list core/project --
format='value(core.project)'`
```

Step 4

Rebuild the Docker image with a tag that includes the registry name `gcr.io` and the project ID as a prefix.

```
docker build -t "gcr.io/${GCP_PROJECT}/py-web-server:v1" .
```

Again, be sure to include the '.' at the end of the command. This tells Docker to store the image in the current working directory.

Click *Check my progress* to verify the objective.

Build the Docker image with a registry name

Check my progress

Make the Image Publicly Accessible

Google Container Registry stores its images on Google Cloud storage.

Step 1

Configure Docker to use gcloud as a Container Registry credential helper (you are only required to do this once).

```
PATH=/usr/lib/google-cloud-sdk/bin:$PATH
gcloud auth configure-docker
```

When prompted, press **ENTER**.

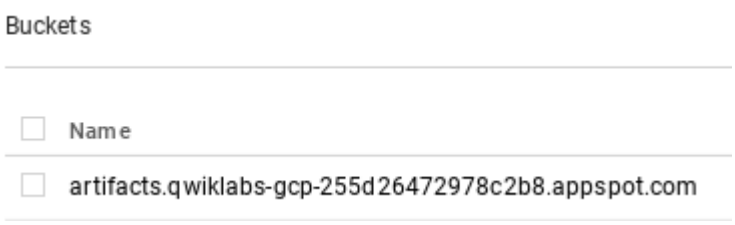
Step 2

Push the image to `gcr.io`.

```
docker push gcr.io/${GCP_PROJECT}/py-web-server:v1
```

Step 3

To see the image stored as a bucket (object) in your Google Cloud Storage repository, click the **Navigation menu** icon and select **Storage**.
You should see an image like the following:



Step 4

Update the permissions on Google Cloud Storage to make your image repository publicly accessible.

```
gsutil defacl ch -u AllUsers:R gs://artifacts.${GCP_PROJECT}.appspot.com
gsutil acl ch -r -u AllUsers:R gs://artifacts.${GCP_PROJECT}.appspot.com
gsutil acl ch -u AllUsers:R gs://artifacts.${GCP_PROJECT}.appspot.com
```

The image is now available to anyone who has access to your GCP project.

Click *Check my progress* to verify the objective.

Make the Image Publicly Accessible

Check my progress

Run the Web Server From Any Machine

The Docker image can now be run from any machine that has Docker installed by running the following command.

```
docker run -d -p 8888:8888 -h my-web-server gcr.io/${GCP_PROJECT}/py-web-server:v1
```

You can test it on your VM instance (re-using the `curl` command from above).
To learn more about Dockerfiles, look at this reference.
To know more about Docker images, look at this reference.
Exit the lab environment and return to the Cloud Shell.

```
exit
```

End your lab