Microservices and Containerization: Create Docker Image for a Web Application Using Dockerfile

Lab 01-1 Practices

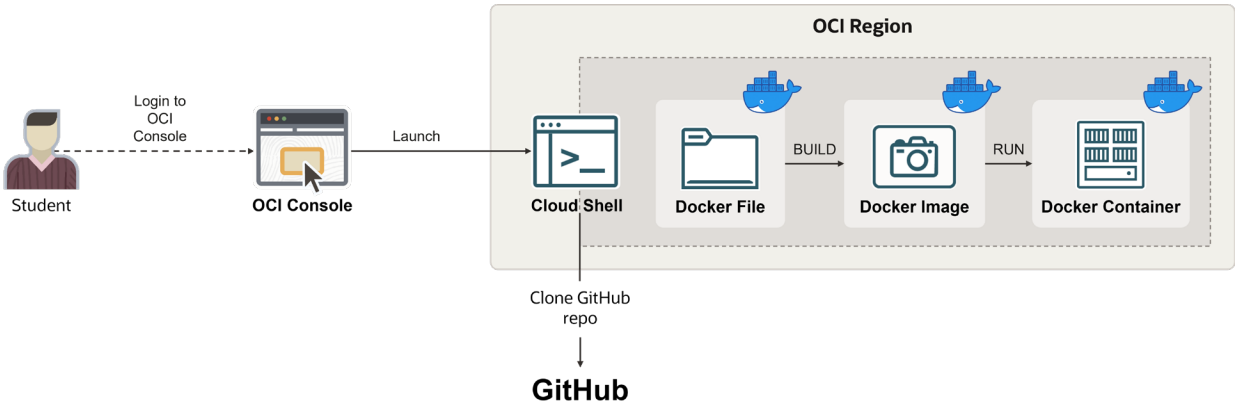
Estimated Time: 30 minutes

Get Started

Overview

There are certain ways for creating, running, and deploying applications in containers using Docker. A Docker image contains application code, libraries, tools, dependencies, and other files needed to make that application run.

In this lab, you will create a Docker image using a Dockerfile, which will further be used to build a container that can run on the Docker platform.



In this lab, you'll:

a. Access the Dockerfile.

b. Build the Docker image.

c. Run your Docker image as a container.

d. Access the web application running within the container.

e. Delete the Docker container.

For more information on Docker, see the [OCI Docker Documentation.]

Assumptions

- You are signed in to your Oracle Cloud Infrastructure (OCI) account using your credentials.
- You have access to the Git repository link that contains the Dockerfile.
- You will replace the `<userID>` placeholder with your Firstname.
- Access the Dockerfile

Access the Dockerfile

Access the Dockerfile needed to generate the Docker image by cloning a Git repository.

Tasks

1. Open **Cloud Shell**.

2. Within Cloud Shell, clone the GitHub repository to access the sample Dockerfile which is a simple Nginx HelloWorld application that you will use to build the Docker image.

   ```
   $ cd ~
   ```

   ```
   $ git clone https://github.com/ou-developers/docker-helloworld-demo
   ```

3. Navigate to the cloned directory.

   ```
   $ cd docker-helloworld-demo/
   ```

4. Open Code Editor. Code Editor allows you to view the files and source codes present in the home directory within the Cloud Shell terminal.

   The tool bar is on the left side of the **Code Editor** window. Click the **Explorer** (top) icon from the left-side menu within the Code Editor window.

   Browse to the cloned Git directory "`docker-helloworld-demo`" to view the various files you have in the directory including application code and Dockerfile for creating the sample Nginx application.

Build the Docker Image

You're using Cloud Shell as your development environment which comes preinstalled with Docker.

Tasks

1. Check the Docker version using the following command in Cloud Shell. It will return a string with the Docker version installed.

   ```
   $ docker -v
   ```

   For example, `Docker version 19.03.11-ol, build 9bb540d`

2. Check for existing Docker images in the Cloud Shell.

   ```
   $ docker images
   ```

   It will return an empty response because there are no docker images at present.

3. Create a docker image for the sample Web Application using the `docker build` command. This command needs Dockerfile as one of its parameters.

   ```
   $ docker build -t oci_sample_webapp_<userID>:<tag> .
   ```

   For example,

   ```
   $ docker build -t oci_sample_webapp_mahendra:1.0 .
   ```

   Where,

   - `-t` is the switch used to specify the image name.
   - Enter an image name using this format: `oci_sample_webapp_<userID>`.

     Replace *<userID>* with your Firstname.

     For example, `oci_sample_webapp_mahendra`.
   - A tag is used to give the image a version. In this lab, you will use `1.0` as tag.
   - You are currently in the cloned directory which contains the Dockerfile. Use "." as the relative path at the end of the command.

4. Upon successful build of a Docker image, verify the image in the local repository using the following command:

   ```
   $ docker images
   ```

   You'll see two entries in the output. One is the base image "`nginx`", and the other is the custom Docker image for the Web Application "`oci_sample_webapp_<userID>`".

Run Your Docker Image as a Container

Your Docker image holds the application that you want Docker to run as a container.

Tasks

1. Use the `docker run` command to spin a container based on the image created.

   ```
   $ docker run -d --name webapp-<userID> -p 80:80/tcp
   oci_sample_webapp_<userID>:<tag>
   ```

   Where,

   - `-d` flag is used to run container in background and print `CONTAINER_ID`.
   - `--name` flag is used to assign a name to the container.
   - `-p` flag is used to publish container port 80 to the host machine port 80.
   - Replace *`<userID>`* with your Firstname.

   For example,

   ```
   $ docker run -d --name webapp-mahendra -p 80:80/tcp
   oci_sample_webapp_mahendra:1.0
   ```

   Note: This command returns the CONTAINER_ID of the container started in the background.

2. Check the container that is currently running using the `docker ps` command.

   ```
   $ docker ps
   ```

   You will see a container running with the name `webapp-`*`<userID>`* and a corresponding `CONTAINER_ID`.

Access the Web Application Running Within the Container

Verify whether you can access the web application that is running in your container. Once you have verified, stop the running container.

Tasks

1. Use the `curl` command to connect to the local host on port 80 to access the web application.

   ```
   $ curl -k http://127.0.0.1:80
   ```

   The output must display the webpage code. This confirms that your web application is up and running.

2. Get the `CONTAINER_ID` and copy it on a notepad to use it in your next step.

   ```
   $ docker ps -a
   ```

3. Stop the running container.

   ```
   $ docker stop <CONTAINER_ID>
   ```

   For example,

   ```
   $ docker stop ffab54628f8f
   ```

4. Use the **curl** command to connect to the localhost on port 80 to access the web application.

   ```
   $ curl -k http://127.0.0.1:80
   ```

   Output: `curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection refused`

   This time output will return the above mentioned error, because the container running the application is no longer active.

Delete the Docker Container

Clean up your resources by removing the container used in this lab.

Tasks

1. Check the status of all the containers in the system.

   ```
   $ docker ps -a
   ```

   The status for the container must show **exited** which means the container is stopped.

2. Delete the existing container using the **rm** flag.

   ```
   $ docker rm webapp-<userID>
   ```

   For example,

   ```
   $ docker rm webapp-mahendra
   ```

   Output: `webapp-mahendra`

   Note: On successful deletion it'll return the container name.

3. Verify if the container is deleted.

   ```
   $ docker ps -a
   ```

   The container entry should be gone.

   Important Note: Do not delete the Docker image created in this lab, because it will be used as an artifact in the upcoming labs.

Congratulations! You have successfully built and containerized a docker image.