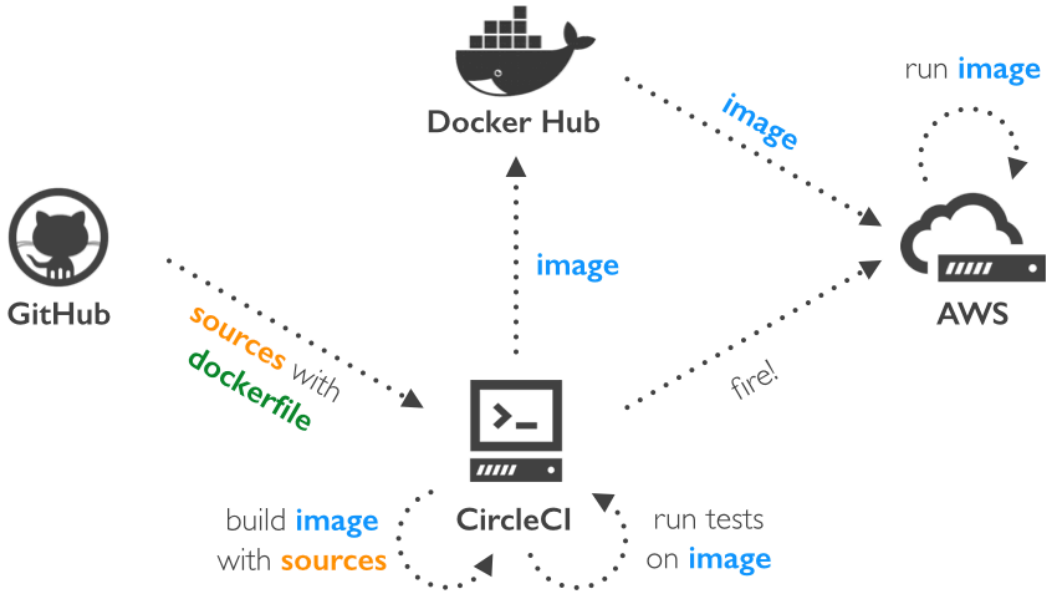
CONTAINER IMAGES USING DOCKERFILE

Overview

In this lab, you will learn how to create container images using a Dockerfile. You will learn to construct a Dockerfile using Keywords for building the container images. This will allow you manage your applications changes easier and build complex solutions.



Pre-requisites

Docker Hub Account

If you do not have a DockerID (a free login used to access Docker Hub), please visit Docker Hub and register for one. You will need this for later steps.

GitHub Repo

Use the following command to clone the lab's repo from GitHub. This will make a copy of the lab's repo in a new sub-directory called linux_tweet_app.

git clone https://github.com/iimran-muhammad/linux_tweet_app

Procedure

Package and run a custom app using Dockerfile

Build a simple website image

In this step you'll learn how to package your own apps as Docker images using a Dockerfile.

- Make sure you're in the linux_tweet_app directory.

  cd ~/linux_tweet_app

- Display the contents of the Dockerfile.

  cat Dockerfile

In your own words explain what these lines in the Dockerfile do.

FROM

COPY

COPY

EXPOSE

CMD

- To make the following commands more copy/paste friendly, export an environment variable containing your DockerID.

  export DOCKERID=<your docker id>

- Echo the value of the variable back to the terminal to ensure it was stored correctly.

  echo $DOCKERID

- Use the docker image build command to create a new Docker image using the instructions in the Dockerfile.

  --tag allows us to give the image a custom name. In this case it's comprised of our DockerID, the application name, and a version. Having the Docker ID attached to the name will allow us to store it on Docker Hub in a later step

  . tells Docker to use the current directory as the build context

  Be sure to include period (.) at the end of the command.

  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .

What is the output of the Docker daemon executing each line in the Dockerfile? You should explain all steps of the layers.

- Use the docker container run command to start a new container from the image you created.

  docker container run \

```
--detach \
--publish 80:80 \
--name linux_tweet_app \
$DOCKERID/linux_tweet_app:1.0
```

- Check your website which should be running.
- Once you've accessed your website, shut it down and remove it.

```
docker container rm --force linux_tweet_app
```

Modify a running website

When you're actively working on an application it is inconvenient to have to stop the container, rebuild the image, and run a new version every time you make a change to your source code.

One way to streamline this process is to mount the source code directory on the local machine into the running container. This will allow any changes made to the files on the host to be immediately reflected in the container.

We do this using something called a bind mount.

When you use a bind mount, a file or directory on the host machine is mounted into a container running on the same host.

Start web app with a bind mount

- Let's start our web app and mount the current directory into the container.

  We'll use the --mount flag to mount the current directory on the host into /usr/share/nginx/html inside the container.

  Be sure to run this command from within the linux_tweet_app directory on your Docker host.

```
docker container run \
--detach \
--publish 80:80 \
--name linux_tweet_app \
--mount type=bind,source="$(pwd)",target=/usr/share/nginx/html \
$DOCKERID/linux_tweet_app:1.0
```

- Check your website which should be running.

Modify the running website

Bind mounts mean that any changes made to the local file system are immediately reflected in the running container.

- Copy a new index.html into the container.

  The Git repo that you pulled earlier contains several different versions of an index.html file. You can manually run an ls command from within the ~/linux_tweet_app directory to see a list of them. In this step we'll replace index.html with index-new.html.

cp index-new.html index.html

- Go to the running website and refresh the page. Notice that the site has changed.

How is this possible that you were able to modify the index.html and you never actually built a new image after modifying the running website? Explain your finding.

- Stop and remove the currently running container.

  docker container rm --force linux_tweet_app

- Re-run the current version without a bind mount.

```
docker container run \
--detach \
--publish 80:80 \
--name linux_tweet_app \
$DOCKERID/linux_tweet_app:1.0
```

- Notice the website is back to the original version.
- Stop and remove the current container

  docker container rm --force linux_tweet_app

Update the web app image

To persist the changes you made to the index.html file into the image, you need to build a new version of the image.

- Build a new image and tag it as 2.0

  Remember that you previously modified the index.html file on the Docker hosts local filesystem. This means that running another docker image build command will build a new image with the updated index.html

  Be sure to include the period (.) at the end of the command.

  docker image build --tag $DOCKERID/linux_tweet_app:2.0 .

Now that you have built the second image with an updated index.html file. What have you noticed about the time it took to build the image? Explain your finding.

- Let's look at the images on the system.

  docker image ls

  You now have both versions of the web app on your host.

Test the new version of web app

- Run a new container from the new version of the image.

```
docker container run \
--detach \
--publish 80:80 \
```

```
--name linux_tweet_app \
$DOCKERID/linux_tweet_app:2.0
```

Check the new version of the website (You may need to refresh your browser to get the new version to load).

The web page will have an orange background.

We can run both versions side by side. The only thing we need to be aware of is that we cannot have two containers using port 80 on the same host.

- Run another new container, this time from the old version of the image.

  Notice that this command maps the new container to port 8080 on the host. This is because two containers cannot map to the same port on a single Docker host.

```
docker container run \
--detach \
--publish 8080:80 \
--name old_linux_tweet_app \
$DOCKERID/linux_tweet_app:1.0
```

- View the old version of the website.

Push your images to Docker Hub

- List the images on your Docker host.

  docker image ls -f reference="$DOCKERID/*"

  You will see that you now have two linux_tweet_app images - one tagged as 1.0 and the other as 2.0.

  These images are only stored in your Docker hosts local repository. We will push the images to a public repository so you can run them from any Linux machine with Docker.

  Distribution is built into the Docker platform. You can build images locally and push them to a public or private registry, making them available to other users. Anyone with access can pull that image and run a container from it. The behavior of the app in the container will be the same for everyone because the image contains the fully-configured app - the only requirements to run it are Linux and Docker.

  Docker Hub is the default public registry for Docker images.

- Before you can push your images, you will need to log into Docker Hub.

  docker login

  You will need to supply your Docker ID credentials when prompted.

- Push version 1.0 of your web app using docker image push.

  docker image push $DOCKERID/linux_tweet_app:1.0

  You'll see the progress as the image is pushed up to Docker Hub.

- Now push version 2.0.

  docker image push $DOCKERID/linux_tweet_app:2.0

Have a close look at the output as you push the image. Explain in detail about your findings.

You can browse to https://hub.docker.com/r/<your docker id>/ and see your newly-pushed Docker images. These are public repositories, so anyone can pull the image.