Doing More With Docker Images

Image creation from a container

docker container run -ti ubuntu bash

This command will grab the image called “ubuntu” then run an interactive shell in a ubuntu container

If we want to share our container to other teammate, we need to get the container ID

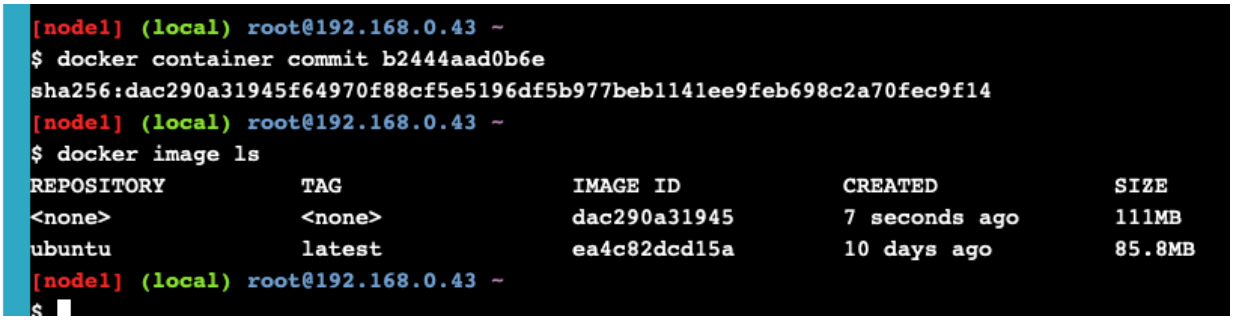
docker container ls -a

To create an image we need to “commit” this container. Commit creates an image locally on the system running the Docker engine.

docker container commit b2444aad0b6e

To see the image just created

docker image ls



To add tag to the image

docker image tag ea4c82dcd15a ourfiglet

```
$ docker image tag ea4c82dcd15a ourfiglet
[node1] (local) root@192.168.0.43 ~
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              dac290a31945        2 minutes ago       111MB
ourfiglet           latest              ea4c82dcd15a        10 days ago         85.8MB
ubuntu              latest              ea4c82dcd15a        10 days ago         85.8MB
[node1] (local) root@192.168.0.43 ~
```

To run a container based on the newly created ourfiglet image:

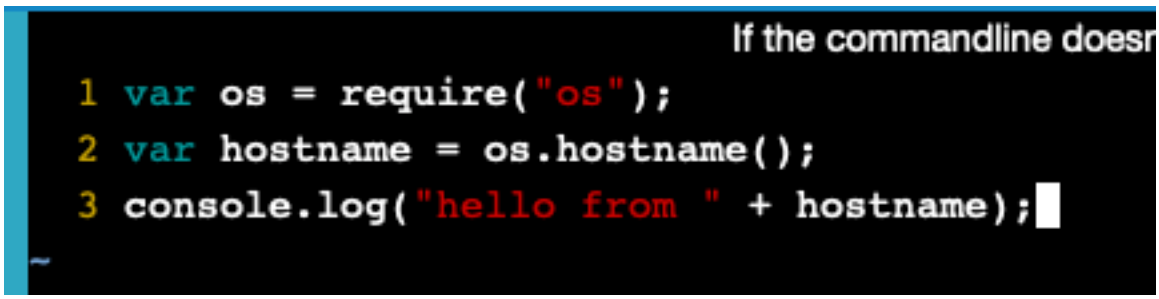
docker container run ourfiglet piglet hello

This example shows that we can create a container, add all the libraries and binaries in it and then commit it in order to create an image. We can then use that image just as we would for images pulled down from the Docker Store.

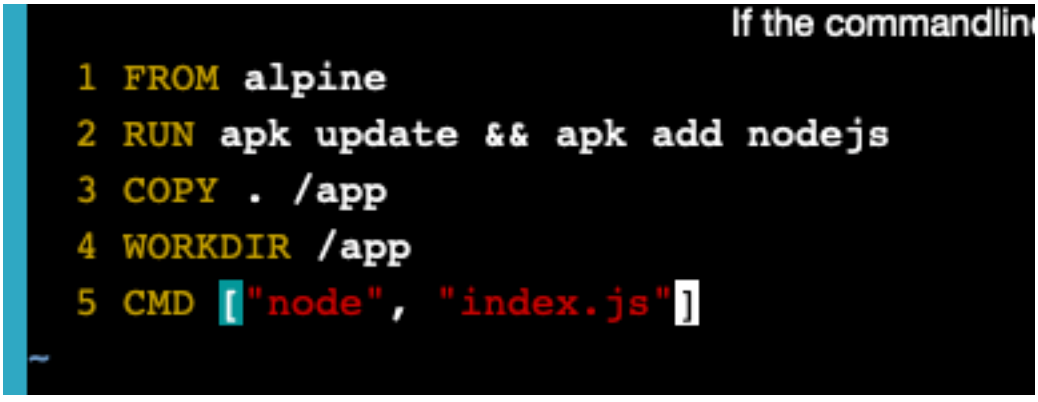
Image creation using a Dockerfile

Dockerfile supply the instructions for building the image, rather than just the raw binary files. This is useful because it becomes much easier to manage changes, especially as your image get bigger and more complex.

Create index.js



Create Dockerfile

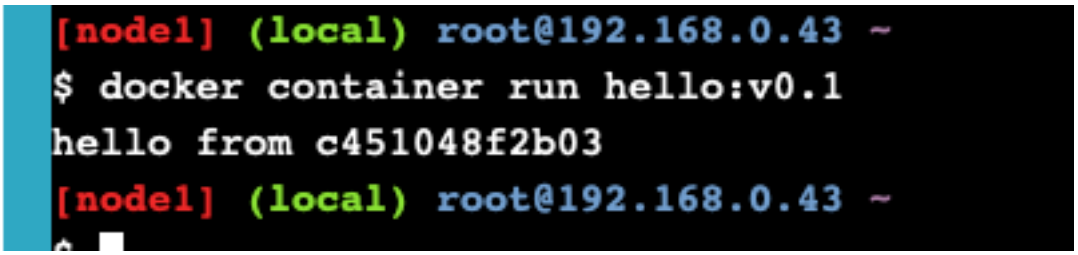


To build first image out of this Dockerfile and name it hello:v0.1

docker image build -t hello:v0.1 .

Start a container to check our application runs correctly:

docker container run hello:v0.1

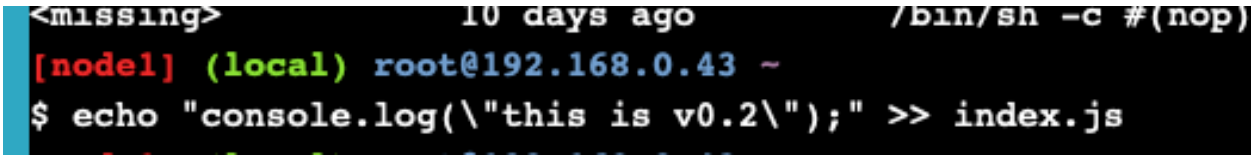


We created two files: out application code (index.js) is a simple bit of javascript code that prints out a message. And the Dockerfile is the instructions for Docker engine to create our custom container. Dockerfile does the following:

1. Specifies a base image to pull FROM - the alpine image we used in earlier labs.
2. Then it RUNs two commands (apk update and apk add) inside that container which installs the Node.js server.
3. Then we told it to COPY files from our working directory in to the container. The only file we have right now is our index.js.
4. Next we specify the WORKDIR - the directory the container should use when it starts up
5. And finally, we gave our container a command (CMD) to run when the container starts.

Image layers

use echo edit the index.js



Rebuild image again

```
[node1] (local) root@192.168.0.43 ~
$ docker image build -t hello:v0.2 .
Sending build context to Docker daemon  1.739MB
Step 1/5 : FROM alpine
 ---> 196d12cf6ab1
Step 2/5 : RUN apk update && apk add nodejs
 ---> Using cache
 ---> 95b6f85af48a
Step 3/5 : COPY . /app
 ---> 09633dba202e
Step 4/5 : WORKDIR /app
 ---> Running in 838280505e00
Removing intermediate container 838280505e00
 ---> 23ede69f15a2
Step 5/5 : CMD ["node", "index.js"]
 ---> Running in 52b37a3aff62
Removing intermediate container 52b37a3aff62
 ---> b3d917c57663
Successfully built b3d917c57663
Successfully tagged hello:v0.2
```

Image Inspection — To know what's inside the container

To inspect the image

docker image inspect alpine

Summary

After this lab, I know the two different methods to create our own images. First from container, and the second from Dockerfile. Using Dockerfile is more convince which support instructions. And I also studied the method to inspect the container.