CCGC 5001 - Virtualization

Module 4B:
Creating and Managing Container Images II

HUMBER

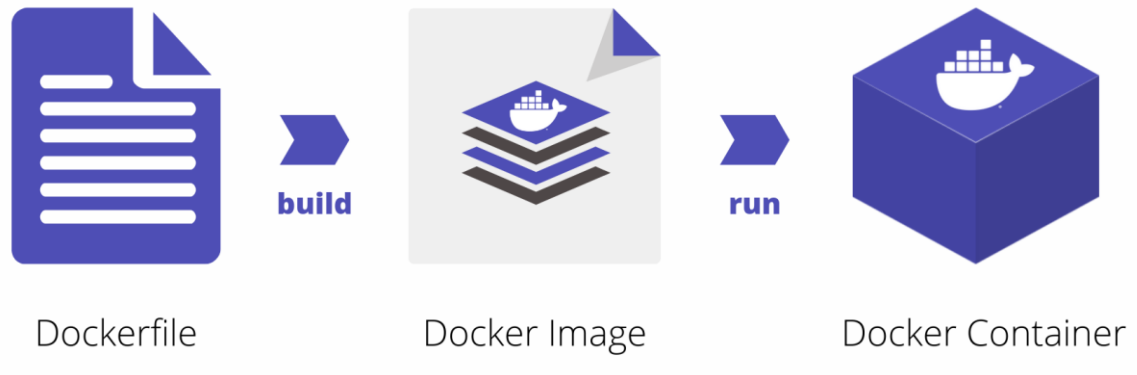
Module objectives

At the end of this module, you should be able to:

- Author a simple Dockerfile to generate a custom image
- Share or ship custom images

Dockerfile

- A text file
- Labelled as Dockerfile
- Contains instructions
- Declarative



Sample Dockerfile

FROM python:2.7

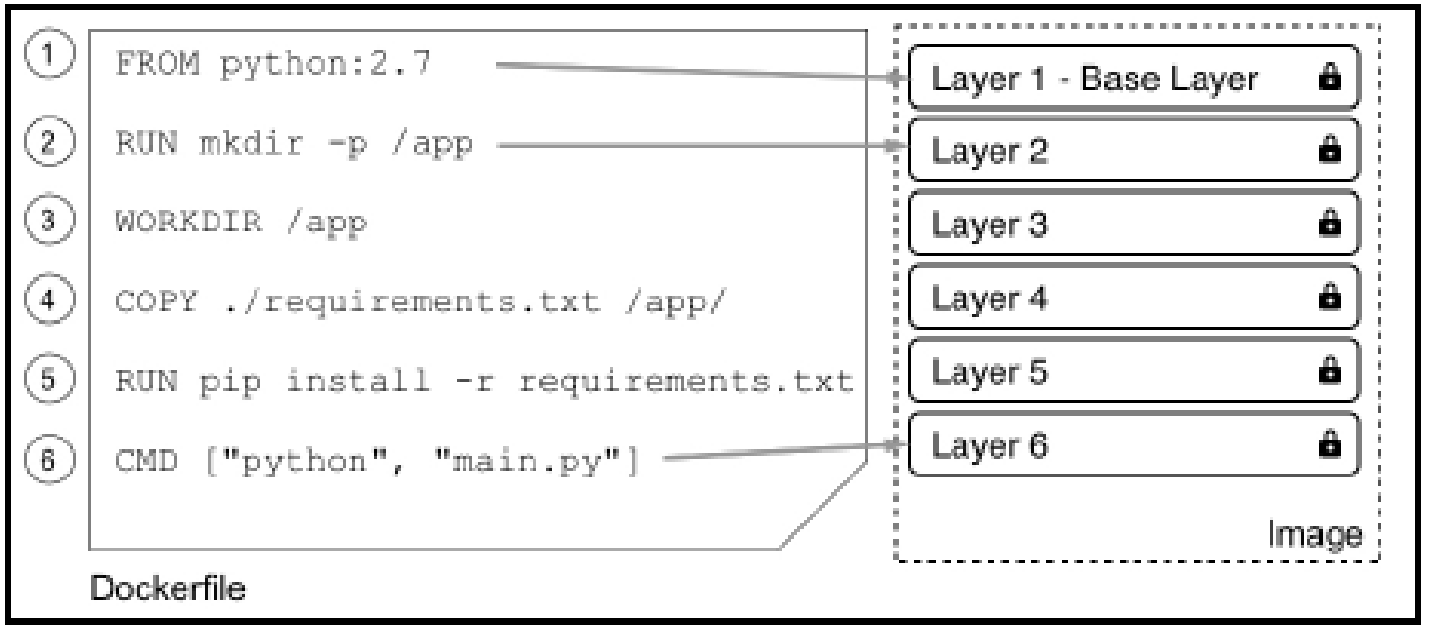
RUN mkdir -p /app

WORKDIR /app

COPY ./requirements.txt /app/

RUN pip install -r requirements.txt

CMD ["python", "main.py"]



FROM keyword

- Every Dockerfile starts with FROM keyword
- Defines the base image to start with
  - Linux distro
  - Development framework

What if we want to start from scratch?

```
FROM scratch
```

- Useful in building minimal images
- Does not generate a layer in the image

RUN keyword

- Argument for RUN keyword is any Linux command

```
RUN yum install –y wget
```

What is the base OS if we use the preceding command?

What happens in the following commands?

```
RUN apt-get update && apt-get install –y wget
```

```
RUN mkdir –p /app && cd /app
```

COPY and ADD keywords

- COPY and ADD keywords are very important and allows us to add some content to the base image to make it custom

Usage of these two keywords, as follows:

```
COPY . /app
COPY ./web /app/web
COPY sample.txt /data/my-sample.txt
ADD sample.tar /app/bin/
ADD http://example.com/sample.txt /data/
COPY ./sample* /mydir/
```

WORKDIR keyword

- WORKDIR keyword defines the working directory or context

  WORKDIR /app/bin

- All activity that happens after the preceding line use this directory as working directory

What happens in the below code?

```
RUN cd /app/bin
RUN touch sample.txt
```

Compare the preceding code with the following code:

```
WORKDIR /app/bin
RUN touch sample.txt
```

CMD and ENTRYPOINT keywords

- Other keywords defined in Dockerfile are executed at the time the image is built
- CMD and ENTRYPOINT defines what will happen when a container is started from image
- They tell Docker what the start process is and how to start that process
- For both ENTRYPOINT and CMD, the values are formatted as a JSON array of strings

```
FROM alpine:3.10
ENTRYPOINT ["ping"]
CMD ["-c","3","8.8.8.8"]
```

CMD and ENTRYPOINT keywords

```
FROM alpine:3.10
CMD wget -O - http://www.google.com
```

What is happening in the preceding code?

A complex Dockerfile

```
FROM node:12.5-stretch
RUN mkdir -p /app
WORKDIR /app
COPY package.json /app/
RUN npm install
COPY . /app
ENTRYPOINT ["npm"]
CMD ["start"]
```

Build an image

Create	Create a file called Dockerfile
Save	Save the file and exit your editor
Build	Build an image using the Dockerfile as a manifest `docker image build -t my-image-name .`

Build an image

What if my Dockerfile has a different name?

```
docker image build -t my-image-name -f Dockerfile .
```

We only ever need the -f parameter if our Dockerfile has a different name or is not located in the current directory.

Exercise

Hello World application:

```
#include <stdio.h>
int main (void)
{
printf ("Hello, world!\n");
return 0;
}
```

Dockerfile:

```
FROM alpine:3.7
RUN apk update && apk add --update alpine-sdk
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN mkdir bin
RUN gcc -Wall hello.c -o bin/hello
CMD /app/bin/hello
```

Let’s build this image.

Shipping or sharing images

Image namespaces

Generic way to define an image is by its FQDN, as follows:

<registry URL>/<User or Org>/<name>:<tag>

https://registry.acme.com/engineering/web-app:1.0

<registry URL>: URL or registry from which we want to pull the image. Default is docker.io.

<User or Org>: This is the private Docker ID of either an individual or an organization.

<name>: This is the name or repository of the image.

<tag>: This is the tag of the image.

Public registries

Google:

- https:/ / cloud. google. com/ container- registry

Amazon AWS Amazon Elastic Container Registry (ECR):

- https:/ / aws. amazon. com/ ecr/

Microsoft Azure:

- https:/ / azure. microsoft. com/ en- us/services/ container- registry/

Red Hat:

- https:/ / access. redhat. com/ containers

Pushing images to a registry

- Use a container registry to ship our images
- Registry can be public or private

Example:

Let’s push latest version of Alpine to my account and give it a tag of 1.0.

```
docker image tag alpine:latest imranathumber/alpine:1.0
```

Now, to be able to push the image, I have to log in to my account, as follows:

```
docker login -u imranathumber -p <my secret password>
```

After a successful login, I can then push the image, like this:

```
docker image push imranathumber/alpine:1.0
```

Exercise

How would you create a Dockerfile that inherits from Ubuntu version 19.04, and that installs ping and runs ping when a container starts? The default address to ping will be 127.0.0.1.

Module summary

In summary, in this module, you learned:

- Anatomy of Dockerfile
- How to author a Dockerfile
- How to ship our custom images

Thank you