Build Your First Docker Image

Your First Dockerfile

We'll cover the following

In the last lesson, we saw how to run an app using a virtual environment. In this lesson, we will automate the steps required to run the app. We will do it with the help of Docker.

> Clone the project using `git clone https://github.com/venky8283/Docker.git`

> Type `git checkout d469419582ee877067f6d449c9691ad16e5f3365` to get to the code used in this lesson.

The Docker engine executes all commands on your behalf, but we need to tell Docker what commands we want to execute. So, Docker has something called a Dockerfile. All the steps mentioned in the Dockerfile are executed step by step as we saw in the Docker fundamentals section. Create a file named '*Dockerfile*' at the same level as of the app.py file. Dockerfile will have no extension.

Writing a Dockerfile

While writing a Dockerfile, we start with a main dependency of the app. Docker runs your app independent of platform. So, can you think about the main dependency of our app? What is the first thing you need to run your app?

This task is like finding the root of all the dependencies. Any luck?

Well, our app needs a Python environment. So, we need a base of Python on which we can build the stack required for our app.

So, the first command to Docker would be to get Python.

```
FROM python:3.9-rc-buster

# Setting up Docker environment
WORKDIR /code
# Export env variables.
ENV FLASK_APP app.py
ENV FLASK_RUN_HOST 0.0.0.0
###


#Copy requirements file from current directory to file in
#containers code directory we have just created.
COPY requirements.txt requirements.txt

#Run and install all required modules in container
RUN pip3 install -r requirements.txt

#Copy current directory files to containers code directory
COPY . .

#RUN app.
CMD ["flask", "run"]
```

Docker commands

Docker isolates the code environment from the host machine using containers and containers are the instances of the image. In a programming language, this Dockerfile will provide us with a blueprint also called an image, from which we can run our app in a container.

> Docker has some predefined keywords to use for providing commands. We can use one keyword in a line to provide one command at a time.

So, let's review some commands.

FROM

The FROM keyword tells Docker which base image to use or what should be the main platform for this image.

The Python:3.9-rc-buster image is a Linux-based image which has Python installed and no other additional programs or software.

WORKDIR

The WORKDIR instruction sets a working directory for other Docker instructions such as RUN and CMD.

If we do not specify a working directory, we have to provide a full path for running our app.py file while using the RUN instruction.

ENV

The ENV instruction sets an environment variable required for the flask app. We can skip these ENV steps, but you have to provide those variables using .flaskenv file.

Here, we are setting host to 0.0.0.0 which means we can access the app using any IP within the container.

> Setting up a host to 0.0.0.0 is required if you are running the app inside the container and want to access it outside. If you change it to anything else, you won't be able to access the app from the host machine.

COPY

The COPY instruction literally copies the file from one location to another. `COPY SOURCE DESTINATION` is the syntax of the command.

Here, we are copying the requirements.txt file first so that we will have all the libraries installed before copying all the files.

RUN

The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

In the previous section, we learnt about layers. The RUN instruction creates a new layer here. It runs the commands provided as an argument in that layer.

One benefit of running requirements before copying all the files is whenever a change occurs in project files, we don't have to build requirements again and Docker will use a cached layer for requirements.

CMD

CMD runs the command inside the container once a container is forked or created from an image. You can only have one CMD instruction in a Dockerfile. If multiple CMD instructions are used, the last one will be executed.

Here, once the container is created, the CMD command will run our project.

We have one more frequently-used instruction.

ENTRYPOINT

The ENTRYPOINT instruction can be used if you want to configure your container as an executable. If you want to override CMD while running a container, use ENTRYPOINT,

for example, ENTRYPOINT ["flask", "run"]

These definitions are enough to start with Dockerfile. However, if you want to know more about these commands, checkout the official documentation.

Click on the run button above and access the app.

Build your app image

Now, it's time to build the image. First, cd into the directory where the Dockerfile is located.

Type `docker build -t flask_app:1.0 .`

If you get any error, try `sudo docker build -t flask_app:1.0 .`

> The last '.' in the command is very important. It denotes the current directory or location of the Dockerfile to use for building the image.

Let’s run our app in the next lesson. This will create an image on your local machine. You can check that using `docker images`.