Using Docker

Short description

Lab focus: Build your own images with Dockerfile and containerize complex applications with docker-compose.
Credits: This is not a graded assignment but finishing it will help you a lot with the next graded assignments.

Intro

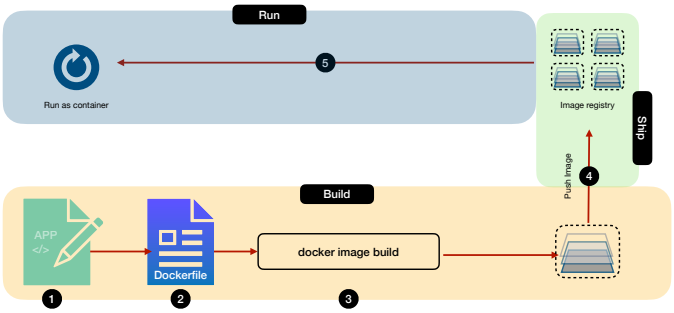

Figure 1: Containarizing an application.

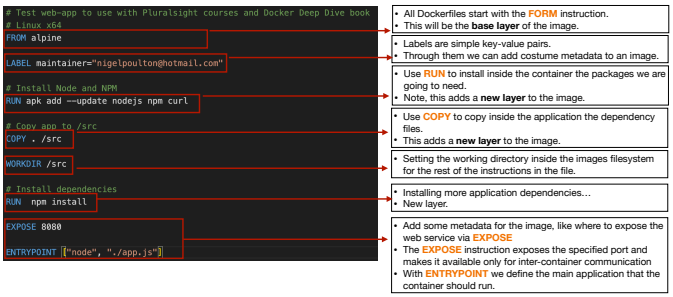

Figure 2: Example of a Dockerfile.

The Figure 1 shows the steps needed to containerize an application. Specifically we follow the following steps:

1. Build you application.
2. Create a **Dockerfile**. This file will contain a list of commands that Docker will go through in order to build the image of your application.
3. Use docker to build your image.

```
docker build -t <name>:<tag> .
```

4. (Optional) You can push your images to a public repository.
5. Run as many as you want instances of your application as an independent container.

```
docker run <name>:<tag> <app-name>
```

What is a Dockerfile: A Dockerfile is the starting point for creating a container image – it describe an application and it tells Docker how to build it into an image. The image you will build using the Dockerfile is built upon a *parent* image. When we execute instructions on this parent image, new layers keep adding up. These layers are created when we run docker build command. The Figure 2 shows an example of a Dockerfile.
Try to build an image from this dockerfile and then run a container.
What happens if you remove the ENTRYPOINT command?.

You can get the whole source code for the example:

```
git clone https://github.com/nigelpoulton/psweb.git
```

Exercise One - Dockerfile

Task 1: Create a Docker file for an image with GIT installed

Task 2 : After the image creation tag your image as `ex1:v1.0`.

Task 3: Create a container based on that image, but use the flag to run the container in the background.

Task 4: Use the docker attach command to enter the container. Verify that git is installed. Discuss on your report the usability of the attach command.

NOTE! Practically, from this exercise we expect the source code and for each step a description. Explain what is created at each step and how you verify that docker engine does what you ask for. Feel free to add as many screenshots as you want.

HINT ;) Use alpine as base image

Exercise Two - `Dockerfile`

Create a Dockerfile to build an image for a simple nginx webpage.

Task 1: Let's build an application. Create a `index.html` file. Inside the file insert the following code:

Task 2: Create a file named `default` and write the following:

Task 3: Create the Dockerfile for your application. Like in Figure 2 make sure to copy in your image the needed files and to install the nginx. Also use as base image: `ubuntu:18.04` .
You need to copy the files in the appropriate directories:

- the default file should be in the /etc/nginx/sites-available/ in the container
- the index.html file should be in the /usr/share/nginx/html/ in the contaier.

```
<html>
  <head>
    <title>Dockerfile</title>
  </head>
  <body>
    <div class="container">
      <h1>My App</h1>
      <h2>This is my first app</h2>
      <p>Hello everyone, This is running via Docker container</p>
    </div>
  </body>
</html>
```

Figure 3: The code of your index.html file.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /usr/share/nginx/html;
    index index.html index.htm;

    server_name _;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Figure 4: The code of default file.

Finally, make sure to start your nginx server using the command CMD in your Dockerfile.

Test 4: Build your image.

Test 5: Run a container with the image you created using the following command:

```
docker run -d -p <xx>:80 -name webserver nginx:1.0
```

The <xx> is the port you prefer to expose the app in your host machine.

Exercise Three - docker-compose

What is docker-compose?

- We can deploy multi-container applications using the Docker Compose.
- Modern cloud-native apps are made of multiple smaller services that interact to form a useful app.
- We call this pattern “microservices”.
- Docker Compose lets you describe an entire app in a single declarative configuration file, and deploy it with a single command.
- Once the app is deployed, you can manage its entire lifecycle with a simple set of commands.

Task I: Take a look at the demo code at getting-started-compose. Use the code for the app. For this exercise you will change the `Dockerfile` and the `docker-compose` file. We want to move the description for the app build from the `Dockerfile` to the `docker-compose` file. Your Dockerfile should contain only the following commands:

```
FROM python:3.9-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
```

Update `docker-compose` via the following steps:

1. create a network that allows the containers to communicate.
2. run the app.
3. Create a volume so that the web service can store the code. Note that we have removed the COPY command from the `Dockerfile`.

HINT ;) Inside the Dockerfile use `python:3.9-alpine` instead of `python:3.7-alpine` (in case the `python:3.7-alpine` does not work on your machine)

HINT ;) Use the commands `docker network/volume ls` to list (i.e., view) the resources you have created.
