**Containerizing a React app using Docker (6/02/2022)**

**What is Docker?**

**Docker** is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure (i.e. a cloud) so you can deliver software quickly.

**Docker Engine** is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with a command line interface (CLI) client **docker**.

**Docker Compose** is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML (YAML Ain’t Markup Language) file to define the required services. Compose enables you to define your application stack in a file, keep it at the root of your project repository and easily enable peers to contribute to your project.

**What is a Docker container?**

According to the Docker website, a container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. Utilizing containers is beneficial because they isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

**What is the difference between a container and a virtual machine (VM)?**

Virtualization enables you to run multiple operating systems on the hardware of a single physical server, while containerization enables you to deploy multiple applications using the same operating system on a server.

**What is a Docker image?**

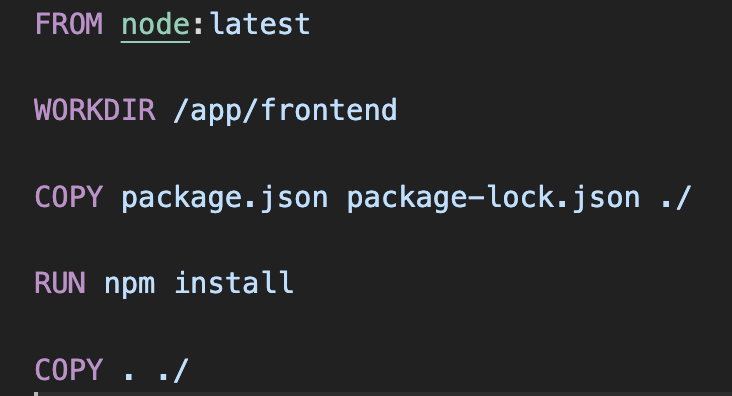
A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

**What is a Dockerfile?**

A Dockerfile is a text document containing all the commands that we want to use to assemble the image. In the Dockerfile, you can define the base image, environment, working directory, dependencies as well the ports to specify (e.g. 8000 aka HTTP).

Below is the basic functioning Dockerfile that we wrote for the React app.

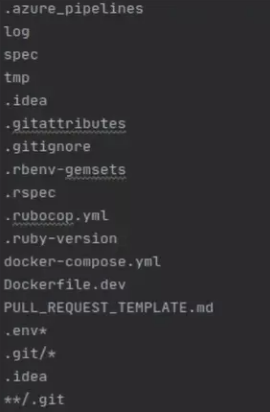
**FROM** sets the base image and **WORKDIR** sets the working directory where the app can be found. The first **COPY** instruction allows us to add the necessary app dependencies in the build context. Something to keep in mind is that to increase the build’s performance, certain files and directories should be excluded by adding them to a created .gitignore file in the directory. We also need to install all the necessary JavaScript packages by using **RUN** to call npm for React. The final **COPY** shown adds the app to the image.



From here, we can build the image and spin up the container.

**.dockerignore**

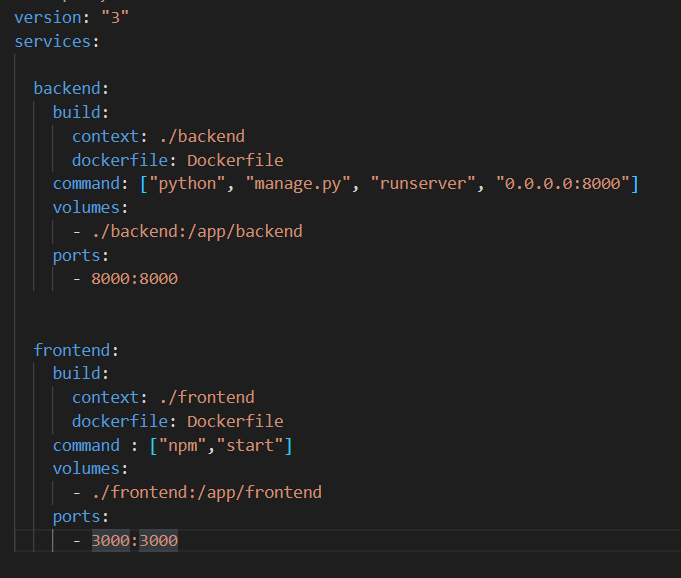
The .dockerignore file is similar to the .gitignore file in that it excludes certain files for a faster and lighter build. Sometimes there are large files or repositories that are not used in the build. The file is located in the root directory. A sample .dockerignore file can be seen below. Though we did not use one in this instance, they can be key to an efficient build time.



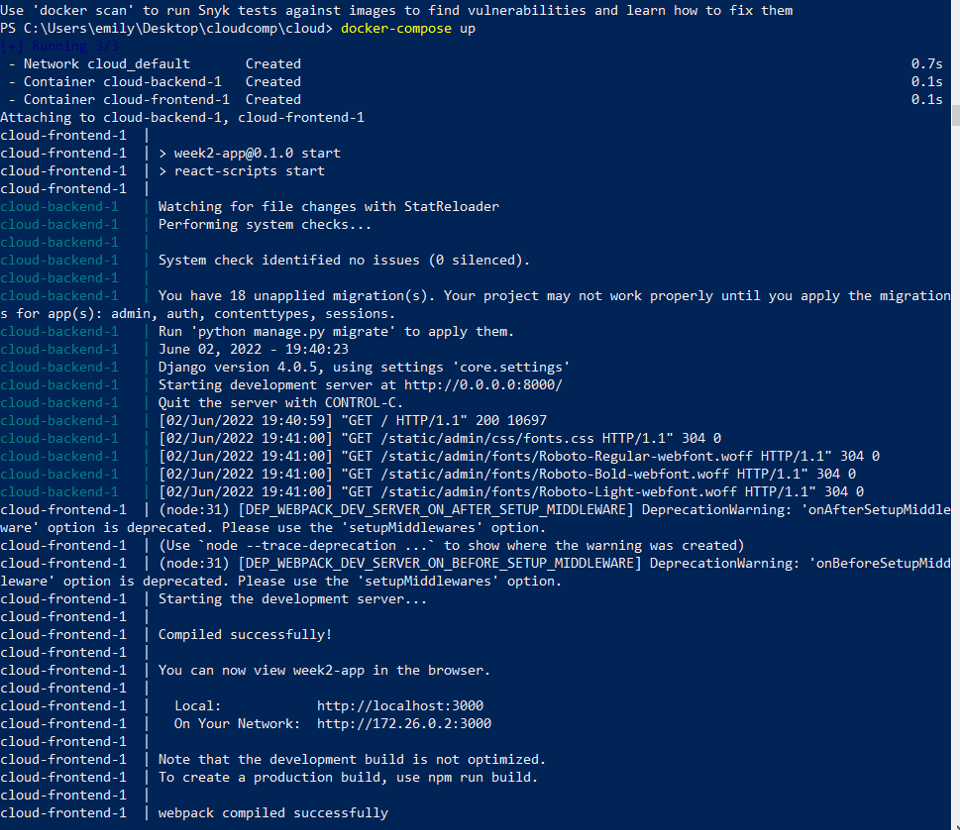
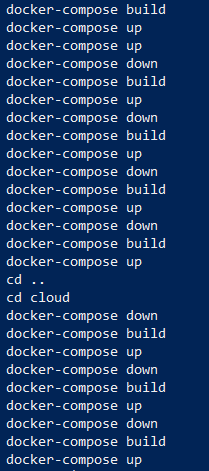
**Docker Compose and How We Used It**

Docker compose allows us to access multi-container applications using a single file. Every part of the application stack is found in our docker-compose.yml file, so all we would need is the app repository in order to run the different parts of our app on separate containers. For this specific app, we have separate Dockerfiles for our Django and React apps (similar to and exactly what is shown above, respectively), but using docker-compose allows us to run both servers without having to build two separate images.

We first wrote a docker-compose file that specifies the directories for each part of the app (Django or React). It also accesses the respective Dockerfiles. We also include the commands to start running the react and django servers, and we also specify the ports in the docker-compose file.

****

To run Docker compose, we first build our container using ***docker-compose build***. Then, we run ***docker-compose up***to start up our containers. If we make any edits, we use ***docker-compose down*** to stop the container and then rebuild and rerun with the first two commands.

****

**Other common docker-compose CLI commands and what they do:**

| **build** | Build or rebuild services |
| --- | --- |
| **down** | Stop and remove containers, networks, images, and volumes |
| **help** | Get help on a command |
| **images** | List containers |
| **kill** | Kill containers |
| **logs** | View output from containers |
| **rm** | Remove stopped containers |
| **run** | Run a one-off command |
| **start** | Start services |
| **stop** | Stop services |
| **up** | Create and start containers |
| **version** | Show the Docker-Compose version information |

**APPENDIX:**

**What is React?**

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. The following is a rudimentary example of React usage in HTML with JSX and JavaScript:



**What is Django?**

Django is a free, open source Python-based web framework that speeds the process of web development. This ability comes from the fact that it doesn’t require building a backend, APIs, javascript and sitemaps from scratch.

**Resources:**

[**https://www.docker.com/resources/what-container/**](https://www.docker.com/resources/what-container/)

[**https://docs.docker.com/compose/reference/**](https://docs.docker.com/compose/reference/)