**Docker Tutorial**

**Why?**

I chose to write a tutorial on Docker, as I liked its ability to use containers, which I can use in other projects, not necessarily in the Web Development Domain. I plan on using it in my thesis and I am glad I found out about it.

**What is Docker?**

It's a platform for containerization. Containers are lightweight, portable units that bundle the application code and its dependencies. Docker provides tools for building, shipping, and running these containers. In our case, we will be using docker to use containerization for our microfrontend and our microservices.

**Installation**

Firstly, it is highly recommended to install Docker Desktop, as, assuming that we are going to be using Visual Studio Code for our project, as it has a Docker functionality, if we install the Docker extensions.

In Visual Studio Code, we will be installing two extensions, Docker and Docker Compose.

And that’s it! We have everything we need to start using Docker and Docker Compose.

**Basic Docker**

For the basic instruction in Docker, we are going to look at an example from our project:

# Use an official Node.js image as the base image

FROM node:latest as build

# Set the working directory inside the container

WORKDIR /app-microf-mf1

# Copy package.json and package-lock.json to the working directory

COPY ../projects/mf1/package\*.json /app-microf-mf1/

# Install dependencies

RUN npm install -g @angular/cli

RUN npx npm install

# Copy the rest of the application code

COPY ../projects/mf1 /app-microf-mf1/

# Arguments

ARG ANGULAR1\_APP\_API\_BASE\_URL

ENV ANGULAR1\_APP\_API\_BASE\_URL=${ANGULAR1\_APP\_API\_BASE\_URL}

# Build the application

RUN npx npm run build

# Use official nginx image as the base image

FROM nginx:latest

# Copy the build output to replace the default nginx contents.

COPY ../nginx/default-1.conf /etc/nginx/conf.d/default-1.conf

COPY --from=build /app-microf-mf1/dist/mf1 /usr/share/nginx/html

# Expose the port the app runs on

EXPOSE 3000 80

CMD ["nginx", "-g", "daemon off;"]

All of the commands used by Docker are simple and self-explanatory. Let’s take a look at a few of them from our code.

The RUN command will execute when the Docker file is built and that is when we would like to perform builds on a project, in our case, we would like to run “npm run”. Docker is useful, because we can have many Docker files that execute npm run, so we don’t have to build it manually for each one, which we will see in the next section.

The CMD command will execute when the image is running. In our case, we want to open nginx, with arguments “-g” and “daemon off;”.

There are many more commands that can be used in Docker, as it isn’t limited to Web Development. For more information, consult the Bibliography.

**Docker Compose**

Now that we have tackled a simple example, let’s look at how Docker uses composition to provide containerization for our entire project. To do this, we will examine our docker-compose.yml file:

version: '3.8'

services:

  movies-microfrontend-shell:

    build:

      context: .

      dockerfile: projects/shell/ShellDockerfile

      args:

        ANGULAR3\_APP\_API\_BASE\_URL: http://127.0.0.1:5000/

    ports:

      - "5000:80"

    links:

      - movies-microfrontend-mf1

      - movies-microfrontend-mf2

    container\_name: movies-microfrontend-shell

    stdin\_open: true

  movies-microfrontend-mf1:

    build:

      context: .

      dockerfile: projects/mf1/Mf1Dockerfile

      args:

        ANGULAR1\_APP\_API\_BASE\_URL: http://127.0.0.1:3000/remoteEntry.js

    ports:

      - "3000:8080"

    container\_name: movies-microfrontend-mf1

    stdin\_open: true

  movies-microfrontend-mf2:

    build:

      context: .

      dockerfile: projects/mf2/Mf2Dockerfile

      args:

        ANGULAR2\_APP\_API\_BASE\_URL: http://127.0.0.1:3001/remoteEntry.js

    ports:

      - "3001:8080"

    links:

      - movies-microfrontend-mf1

    container\_name: movies-microfrontend-mf2

    stdin\_open: true

Firstly, the version is specified, in our case, 3.8.

Then, we are going to write the 3 services we are going to use, in our example, the 3 microfrontends, movies-microfrontend-mf1, movies-microfrontend-mf2 and movies-microfrontend-shell.

For each of them, we will specify the dockerfile, args, ports and container name.

In the case of shell, we are going to specify the links to the other two services. This is because the entry point for our user will be the shell, the other services only provide data to be shown by this shell.

**Conclusion**

Docker makes projects easier to understand and approach. The Divide Et Impera strategy helps in the containerization of otherwise complex projects. For these reasons, I think Docker is a useful and powerful tool.

In conclusion, Docker makes work flow.

**Bibliography**

<https://www.docker.com/products/docker-desktop/>

<https://docs.docker.com/>

<https://docs.docker.com/manuals/>