Creating a Docker file

Open Cmd : touch Dockerfile

The first thing we need to do is define from what image we want to build from. Here we will use the latest LTS (long term support) version 16 of node available:

FROM node:16

Next we create a directory to hold the application code inside the image, this will be the working directory for our application:

# Create app directory

WORKDIR /usr/src/app

This image comes with Node.js and NPM already installed so the next thing we need to do is to install our app dependencies using the npm binary. Please note that if you are using npm version 4 or earlier a package-lock.json file will not be generated.

# Install app dependencies

# A wildcard is used to ensure both package.json AND package-lock.json are copied

# where available (npm@5+)

COPY package\*.json ./

RUN npm install

# If you are building our code for production

# RUN npm ci --only=production

we are only copying the package.json file. This allows us to take advantage of cached Docker layers. Furthermore, the npm ci command, specified in the comments, helps provide faster, reliable, reproducible builds for production environments.

To bundle our app's source code inside the Docker image, use the COPY instruction:

# Bundle app source

COPY . .

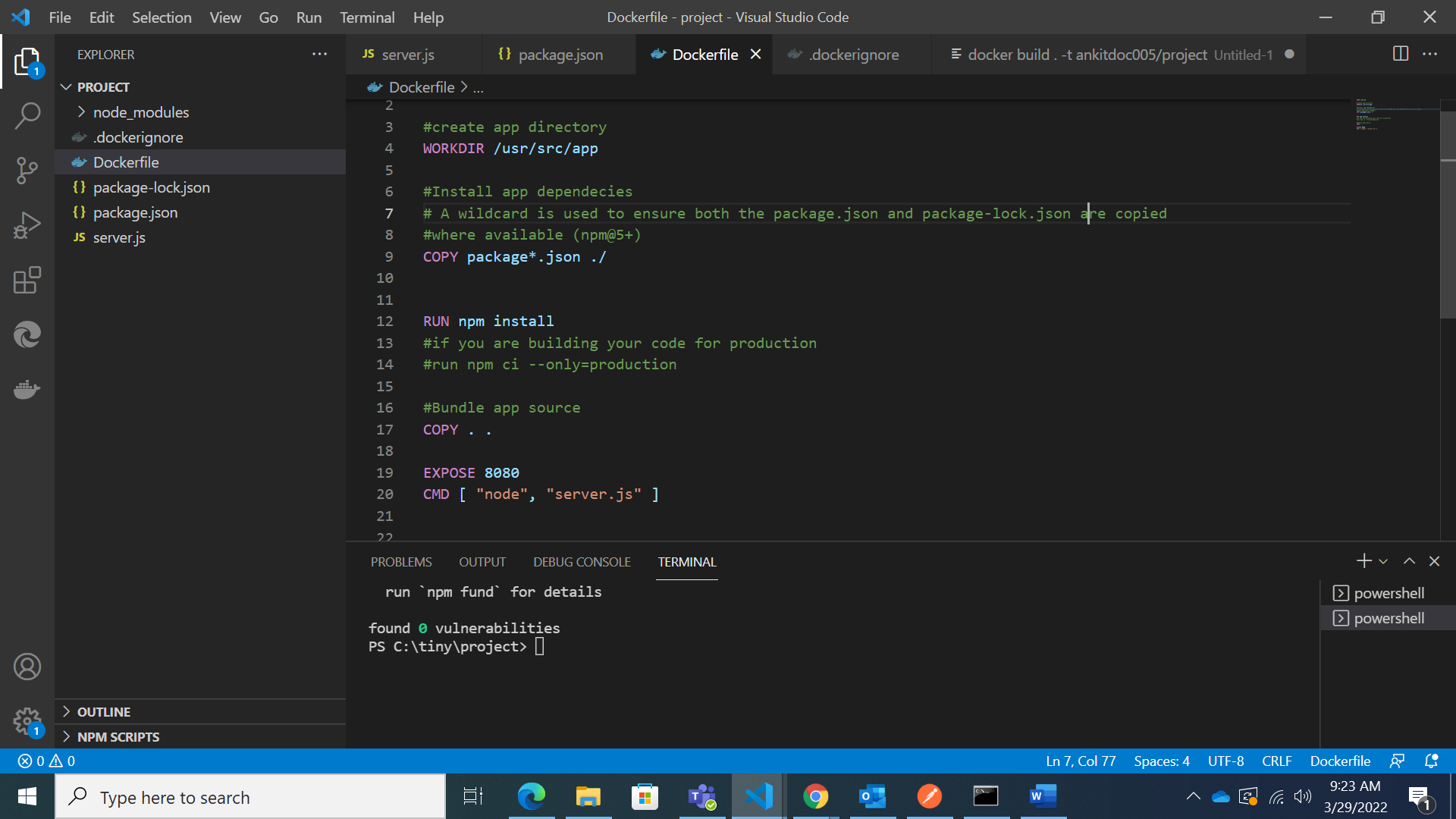
Our Web/app binds to port 8080 so you'll use the EXPOSE instruction to have it mapped by the docker daemon:

EXPOSE 8080

define the command to run our app using CMD which defines our runtime. Here we will use node server.js to start our server:

CMD [ "node", "server.js" ]

Our Dockerfile should now look like this:



## .dockerignore file

Create a .dockerignore file in the same directory as our Dockerfile with following content:

node\_modules

npm-debug.log

This will prevent our local modules and debug logs from being copied onto our Docker image and possibly overwriting modules installed within our image.

## Building our image

Go to the directory that has our Dockerfile and run the following command to build the Docker image. The -t flag lets you tag our image so it's easier to find later using the docker images command:

docker build -t nodeapp .

Our image will now be listed by Docker:

$ docker images

# Example

REPOSITORY TAG ID CREATED

node 16 3b66eb585643 5 days ago

ankitdoc005/project latest d64d3505b0d2 1 minute ago

## Run the image

Running our image with -d runs the container in detached mode, leaving the container running in the background. The -p flag redirects a public port to a private port inside the container. Run the image you previously built:

docker run -p 49160:8080 -d project

Print the output of our app:

# Get container ID

$ docker ps

# Print app output

$ docker logs <container id>

# Example

Running on http://localhost:8080

If you need to go inside the container you can use the exec command:

# Enter the container

$ docker exec -it <container id> /bin/bash

## Test

To test our app, get the port of our app that Docker mapped:

$ docker ps

# Example

ID IMAGE COMMAND ... PORTS

ecce33b30ebf ankitdoc005/project:latest npm start ... 49160->8080

In the example above, Docker mapped the 8080 port inside of the container to the port 49160 on our machine.