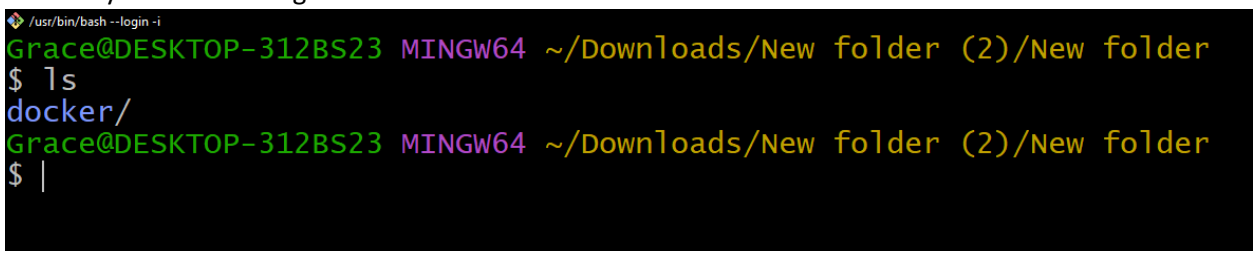
For this journal:

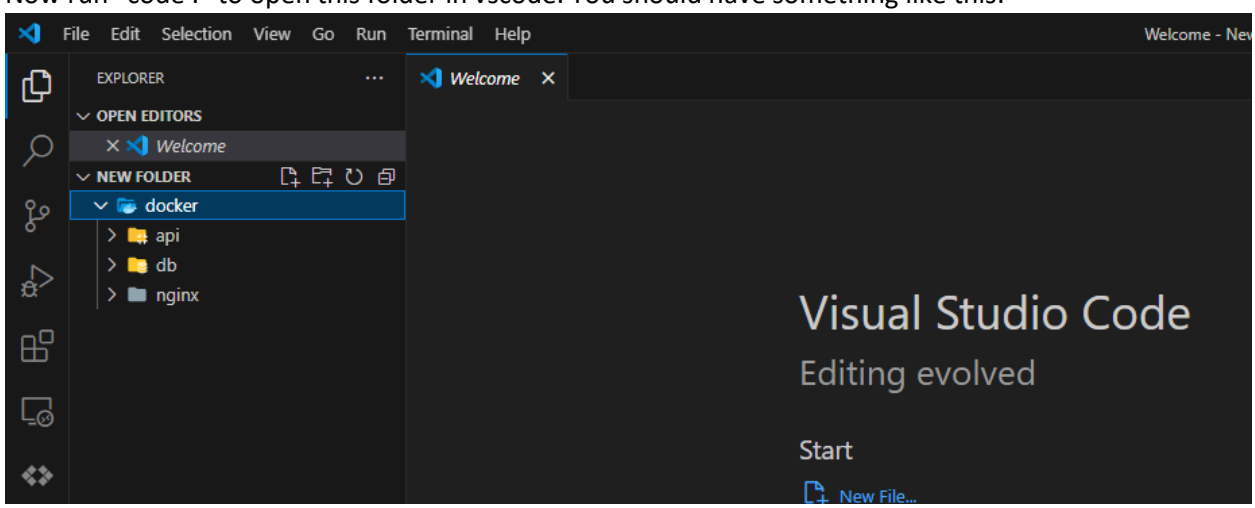
- We will be using the Git Bash terminal. Any other terminal can also be used; e.g., Z Shell (zsh) for Mac users and CMD or Windows PowerShell for Windows users.
- It is assumed you have installed Docker Compose and Docker Desktop on your system. You can follow the guide here (Overview of installing Docker Compose | Docker Docs) to do so.
- If, for instance, we see "run 'a command'", it means type "a command" in your terminal and press enter.

Challenge 3:

1. To start, download and unzip the docker folder into your local storage. Open your git bash terminal and cd into where you saved the docker folder. You can run `ls` in your terminal to confirm you are in the right folder. It should look like this:

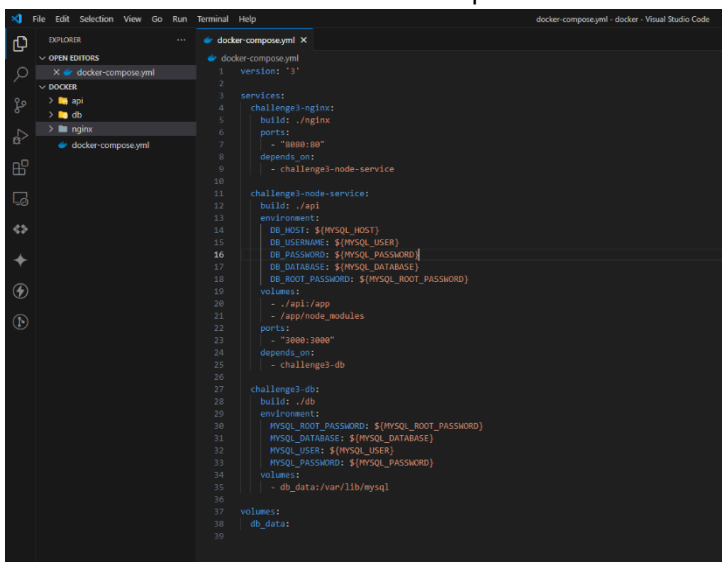


2. Now run "code ." to open this folder in vscode. You should have something like this:



3. Next, we will create a file called "docker-compose.yml". This is where we'll define our web server, application, and database services. This is what your docker-compose file should look

like. Take note of the indentation and spaces.

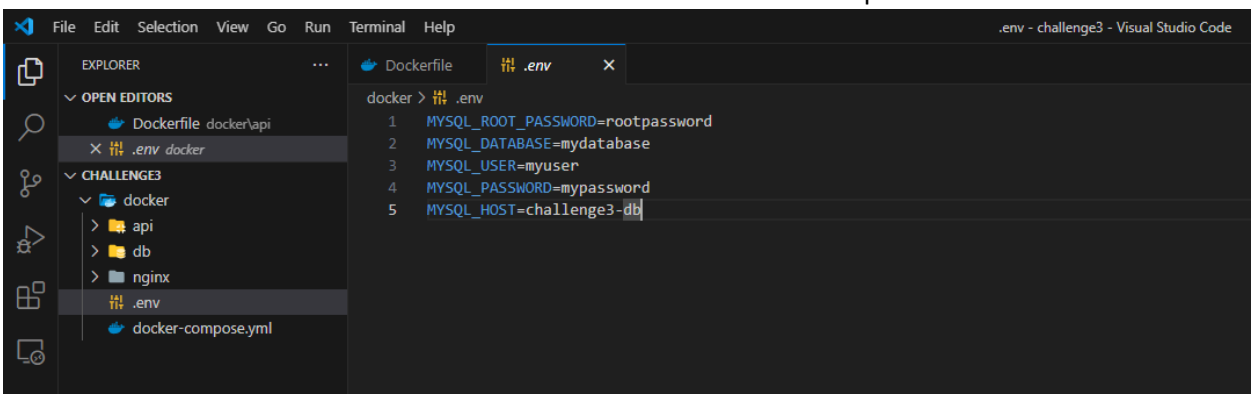


Let us note a few things in this file, specifically **services** and **volumes**:

a. **Services**: This section defines the different services that will make up our entire application. These services are:
   i. **challenge3-nginx**: This is the Nginx server. It is built from the Dockerfile in the nginx folder (or directory). It exposes port 80 inside the container as port 8080 on the host, meaning we can access the server using port 8080 in our browser. It depends on the "challenge3-node-service" to start, meaning "challenge3-node-service" will start before it starts.
   ii. **challenge3-node-service**: This is a Node.js service. It's built from the Dockerfile in the api folder. It uses several environment variables for database configuration. It mounts the api folder on the host to the app folder inside the container, and it excludes the node_modules folder from the volume. This will ensure the node_modules in our container will not be overwritten during the build. It exposes port 3000 inside the container as port 3000 on the host, and it depends on the "challenge3-db" service.
   iii. **challenge3-db**: This is the database service. It's built from the Dockerfile in the db folder. It also uses several environment variables for database configuration. It mounts the db_data volume to /var/lib/mysql inside the container.

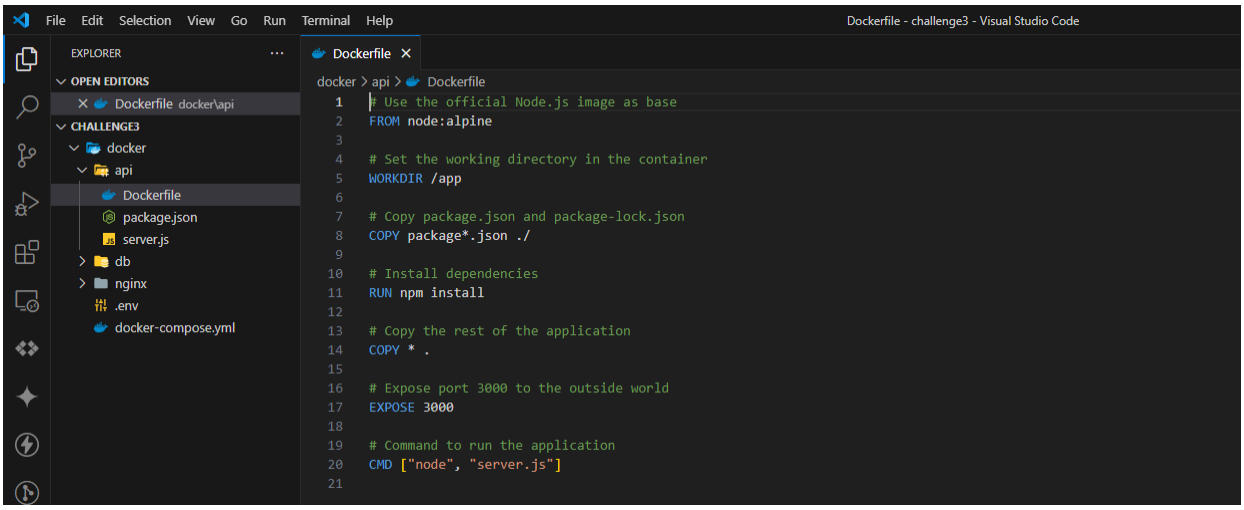
b. **Volumes**: This section defines the volumes that will be used by our services. In this case, there's one volume named "db_data". This volume is used by the "challenge3-db" service to persist database data, a common and recommended practice.

4. Now we will create a ".env" file in the same level where our docker-compose file is like so:

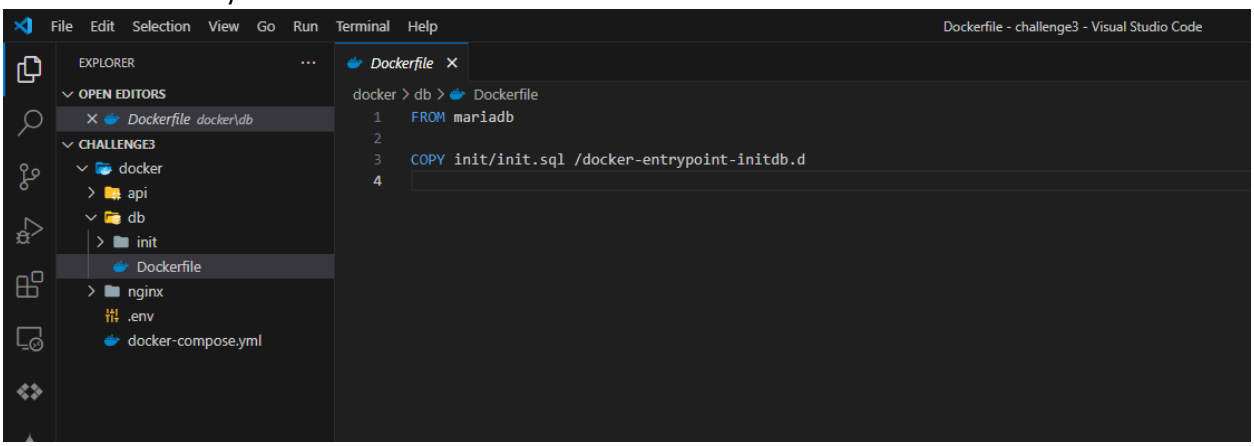


This is where we will store our environment variables. It is worth noting that Docker Compose has built-in support for environment variables stored in a .env file. Now that we created a .env file in the same directory as our docker-compose.yml file, Docker Compose will automatically use it. You can learn more here: [Docker Compose Env File - How to Read in environment variables file (Docker) (youtube.com)]

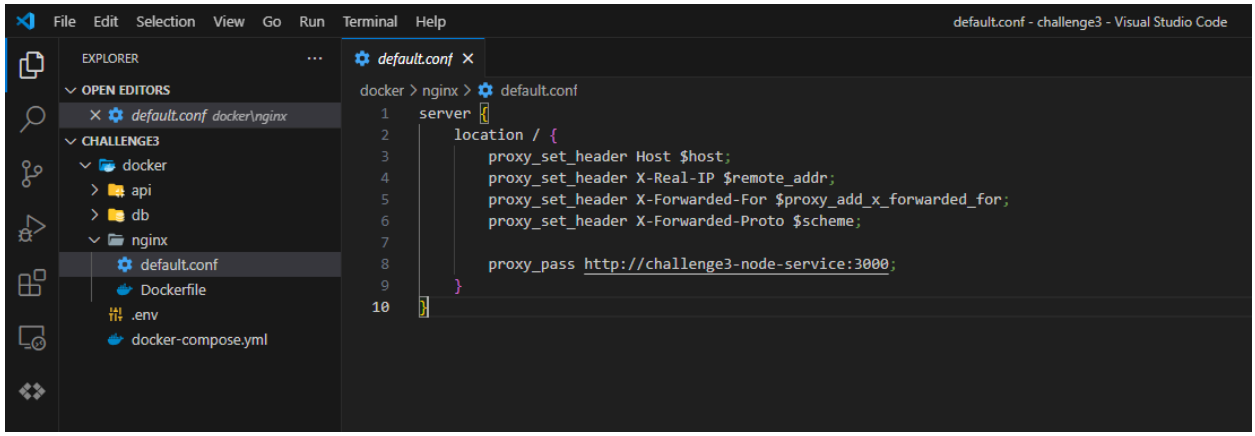
5. Now we will modify some files, starting with the Dockerfile in the api folder. Edit your Dockerfile to look like this.



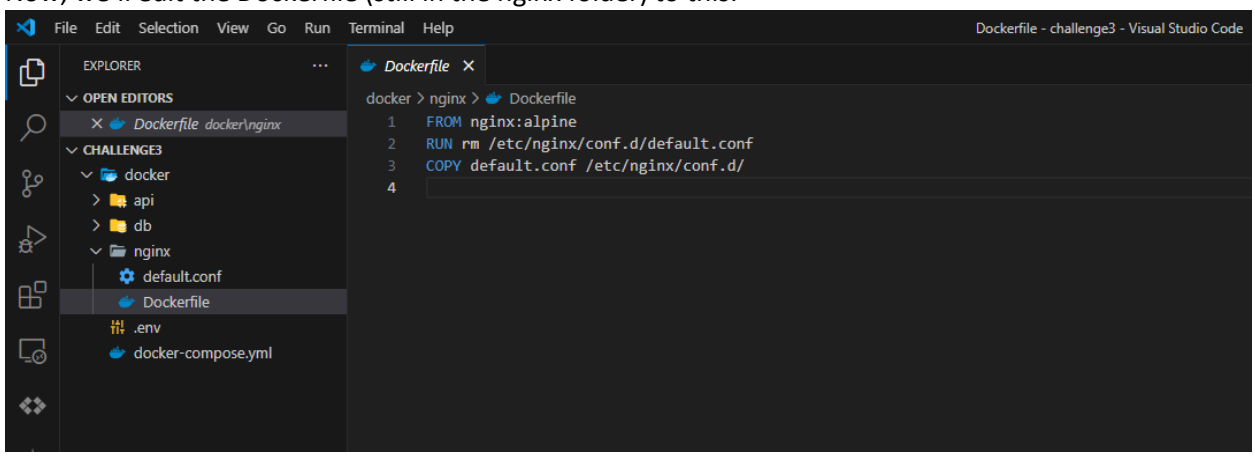
6. Now we will modify the Dockerfile in the db folder to this:



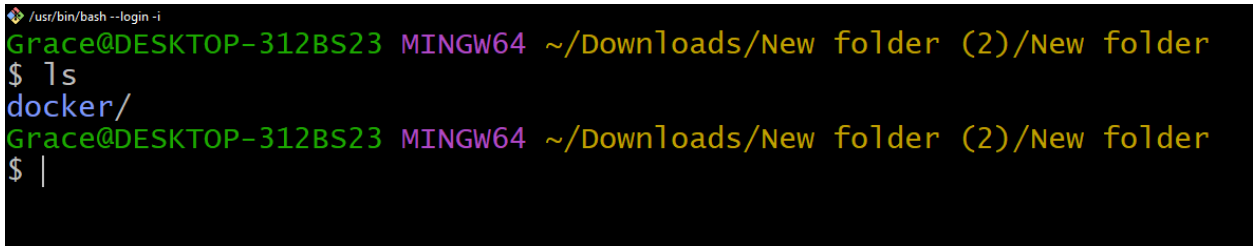
7. Next, inside the nginx folder, we will rename the file “nginx.conf” to “default.conf” and edit the contents of the file like so:



8. Now, we’ll edit the Dockerfile (still in the nginx folder) to this:



9. Now let’s build our services. You should still be in the folder where the docker folder is saved. Your bash terminal should look something like this from the last time we used it. You can still run “ls” to confirm you’re in the right folder.



10. Now we’ll cd into the docker folder and build our services from there because that is where our “docker-compose.yml” file is located.
    a. Run “cd docker”
    b. Then run “ls” to confirm the folder contents
    c. Then run “docker-compose build”. This is the command that will build the services.

After these commands, your terminal should look like this:

```
/usr/bin/bash --login -i
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3
$ cd docker/
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$ ls
api/  db/  docker-compose.yml  nginx/
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$ docker-compose build
#0 building with "default" instance using docker driver

#1 [challenge3-db internal] load build definition from Dockerfile
#1 transferring dockerfile: 98B 0.0s done
#1 DONE 0.1s

#2 [challenge3-db internal] load metadata for docker.io/library/mariadb:latest
#2 DONE 4.3s

#3 [challenge3-db internal] load .dockerignore
#3 transferring context: 2B 0.0s done
#3 DONE 0.1s

#4 [challenge3-db 1/2] FROM docker.io/library/mariadb:latest@sha256:b5e508abc5d889425e90212541e30d29279b7ed34dd74bab5bb715b2f2aeeb7b
#4 DONE 0.0s

#5 [challenge3-db internal] load build context
```

11. If all is successful, your terminal should look like this:

```
#19 DONE 0.0s

#20 [challenge3-nginx internal] load .dockerignore
#20 transferring context: 2B done
#20 DONE 0.0s

#21 [challenge3-nginx 1/3] FROM docker.io/library/nginx:alpine
#21 DONE 0.0s

#22 [challenge3-nginx internal] load build context
#22 transferring context: 338B 0.0s done
#22 DONE 0.1s

#23 [challenge3-nginx 2/3] RUN rm /etc/nginx/conf.d/default.conf
#23 CACHED

#24 [challenge3-nginx 3/3] COPY default.conf /etc/nginx/conf.d/
#24 CACHED

#25 [challenge3-nginx] exporting to image
#25 exporting layers done
#25 writing image sha256:6587277b081e2785rea50c8d287a7777494ee2463d8e9b1b68f92befa3e69d80 0.0s done
#25 naming to docker.io/library/docker-challenge3-nginx 0.0s done
#25 DONE 0.1s
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$
```

This means our services were built successfully.

12. Now let's start our services. To do this, we will run "docker-compose up -d" in the terminal. "-d" here means we want Docker Compose to run the services in detached mode, i.e. run them in the background leaving our terminal free. After running the command, your terminal should look like this:

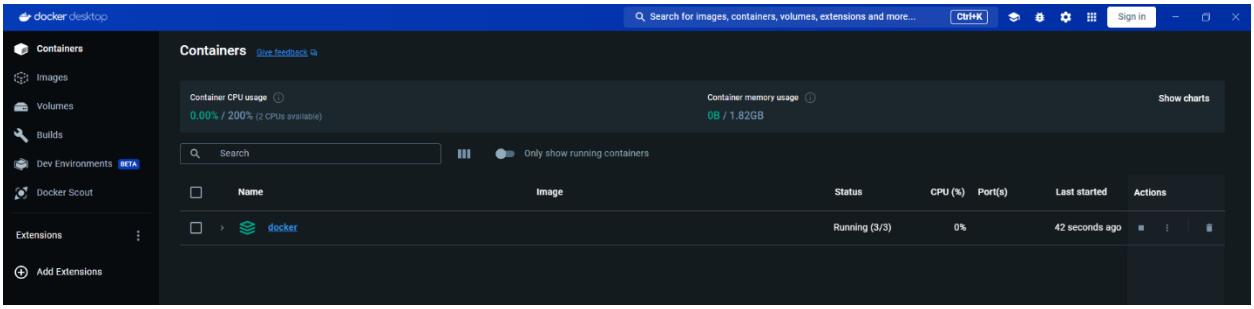
```
/usr/bin/bash --login -i
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$ docker-compose up -d
 Network docker_default  Creating
 Network docker_default  Created
 Container docker-challenge3-db-1  Creating
 Container docker-challenge3-db-1  Created
 Container docker-challenge3-node-service-1  Creating
 Container docker-challenge3-node-service-1  Created
 Container docker-challenge3-nginx-1  Creating
 Container docker-challenge3-nginx-1  Created
 Container docker-challenge3-db-1  Starting
 Container docker-challenge3-db-1  Started
 Container docker-challenge3-node-service-1  Starting
 Container docker-challenge3-node-service-1  Started
 Container docker-challenge3-nginx-1  Starting
 Container docker-challenge3-nginx-1  Started
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$
```

Notice the sections highlighted in red. This means our services were successfully started. We can also run “docker-compose ps” to confirm, like so:

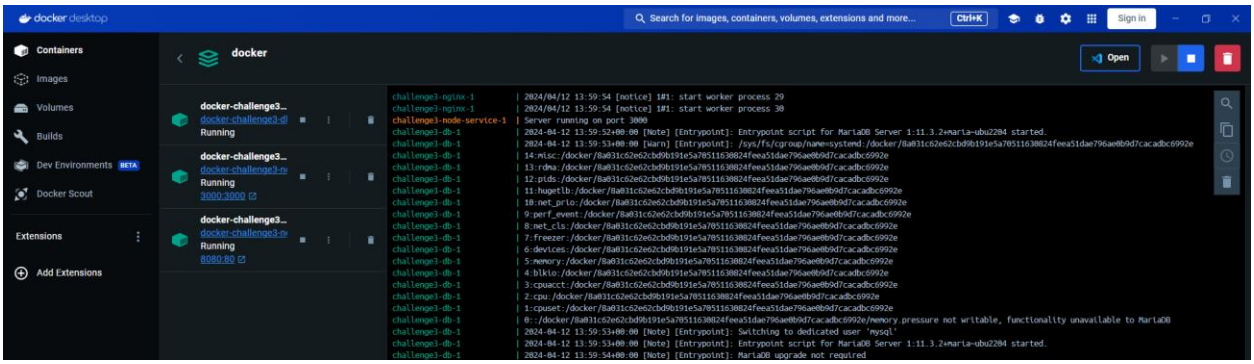
```
 Container docker-challenge3-nginx-1  Started
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$ docker-compose ps
NAME                          IMAGE                         COMMAND                  SERVICE                  CREATED              STATUS
  PORTS
docker-challenge3-db-1        docker-challenge3-db          "docker-entrypoint.s..."  challenge3-db           About a minute ago   Up About a minute
  3306/tcp
docker-challenge3-nginx-1     docker-challenge3-nginx       "/docker-entrypoint...."  challenge3-nginx        About a minute ago   Up About a minute
  0.0.0.0:8080->80/tcp
docker-challenge3-node-service-1  docker-challenge3-node-service  "docker-entrypoint.s..."  challenge3-node-service  About a minute ago  Up About a minute
  0.0.0.0:3000->3000/tcp
Grace@DESKTOP-312BS23 MINGW64 ~/Downloads/New folder (2)/challenge3/challenge3/docker
$
```

Notice the “stats” section for each service. Each service is running successfully.

We can also confirm in Docker Desktop that our services are up and running. When we open up Docker Desktop, we should see this:

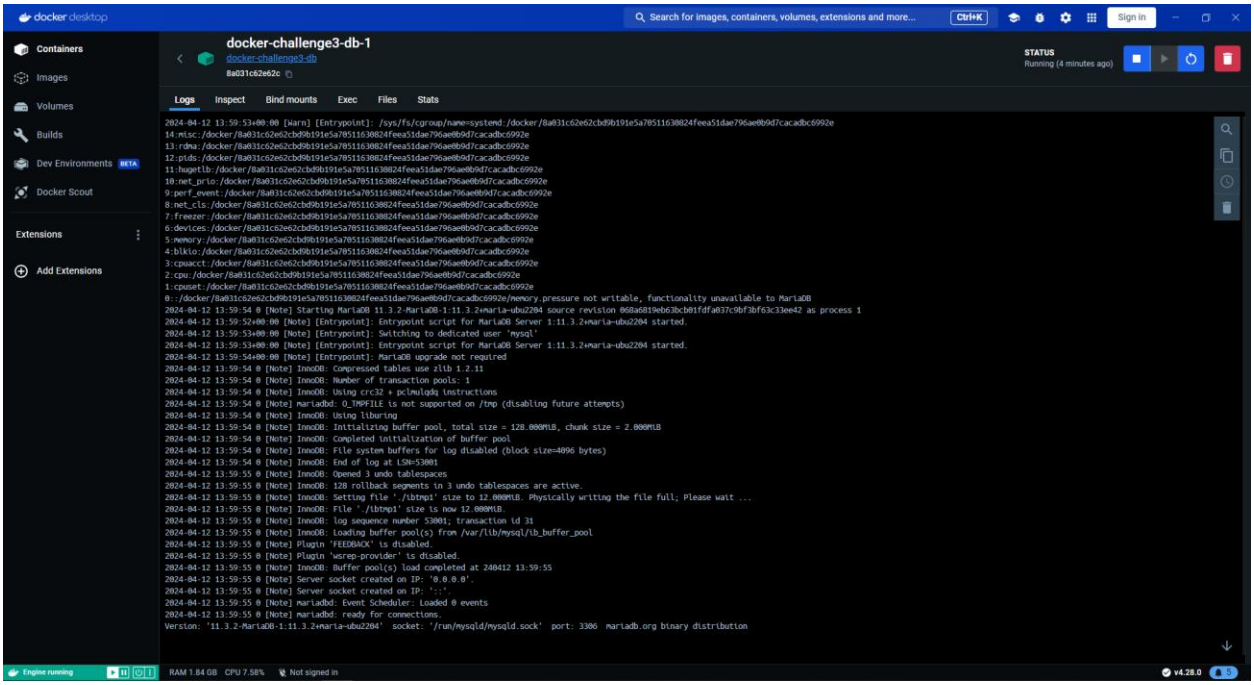


Notice the container named “docker” is green, meaning all services within it are running successfully. We can click on it to see the different services. Like so:



We can also click on each service to see its logs:

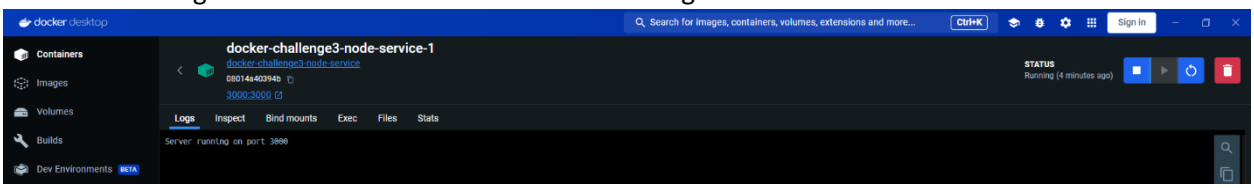
For the first service, which is the “challenge3-db”, we can click on it and select the logs to observe like so:



Observe closely this line near the end of the logs:

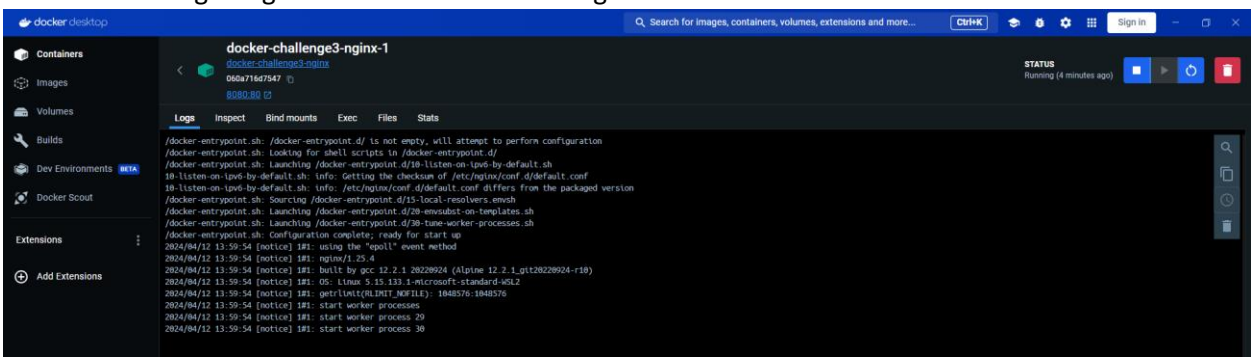
This means the db build was successful and we can connect our node-service to it.

For the “challenge3-node-service” we can also view its logs:



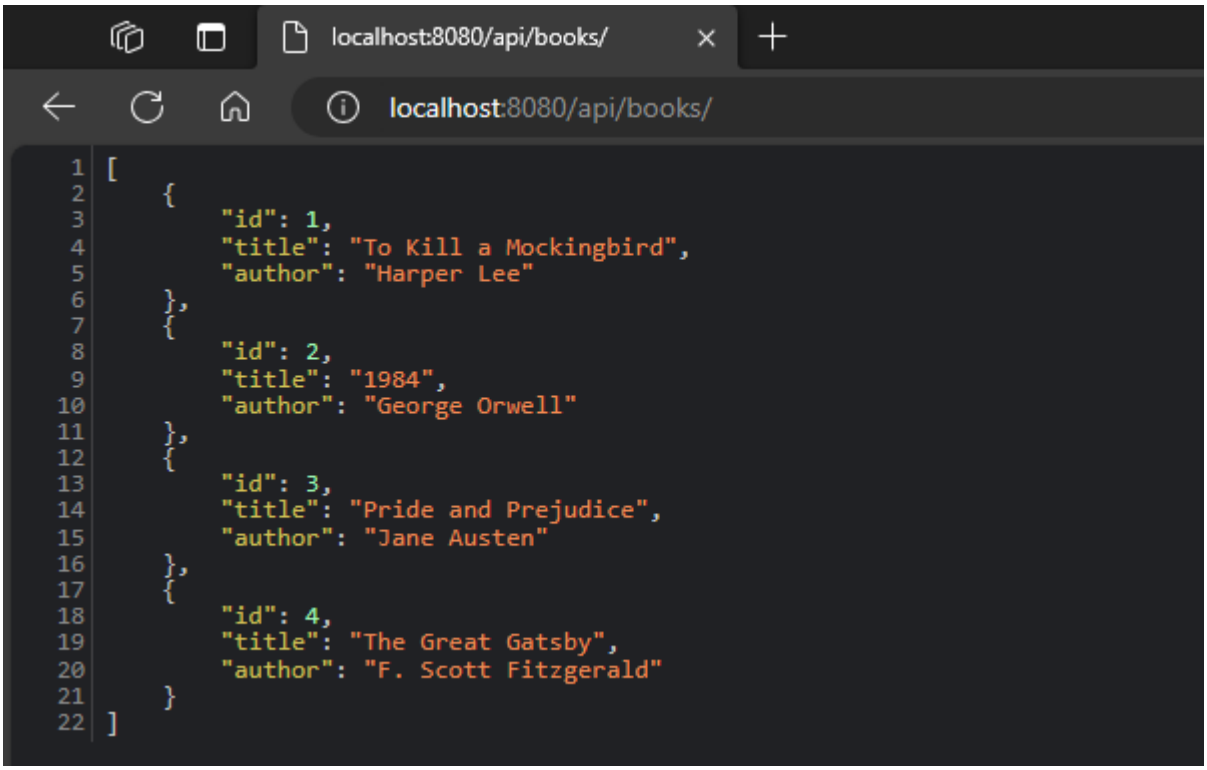
This too is successful.

For our “challenge3-nginx” we can also view its logs:

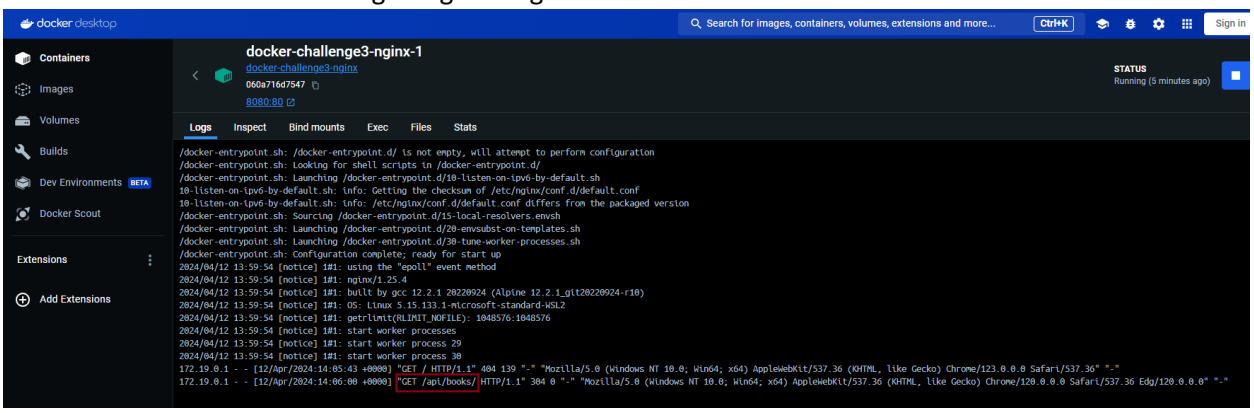


This too also is successful.

We can now access “http://localhost:8080/api/books” successfully in our browser. Remember in our docker-compose file we exposed port 80 inside the container as port 8080. This means we can access our node api with “http://localhost:8080/api/books”. Let’s test it out. Visiting the link should look like this:

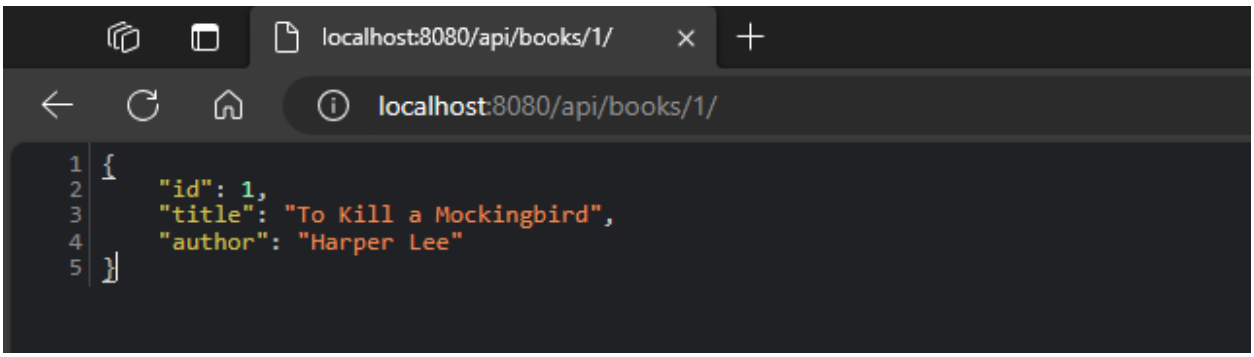


We can also check the “challenge3-nginx” logs to observe like so:

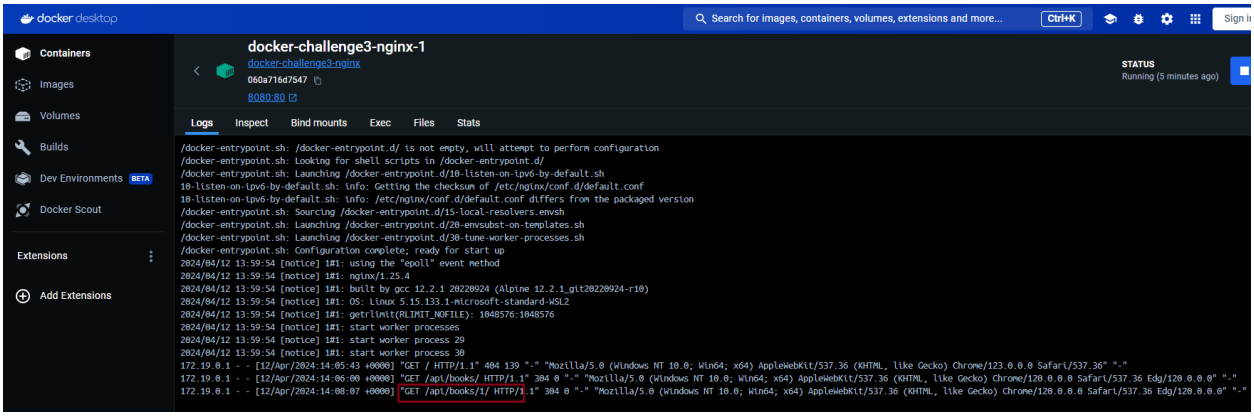


Notice the highlighted section is the same “/api/books” endpoint we visited in our browser.

We can also get one book using “http://localhost:8080/api/books/1”. Visiting the endpoint looks like this:



We can also check the “challenge3-nginx” logs to observe like so:



Highlighted is the same endpoint we visited in our browser.

You can read more about the “docker-compose” command here: [Use Docker Compose | Docker Docs]

Congratulations on reaching this far! You have successfully built a full stack application! Cheers!