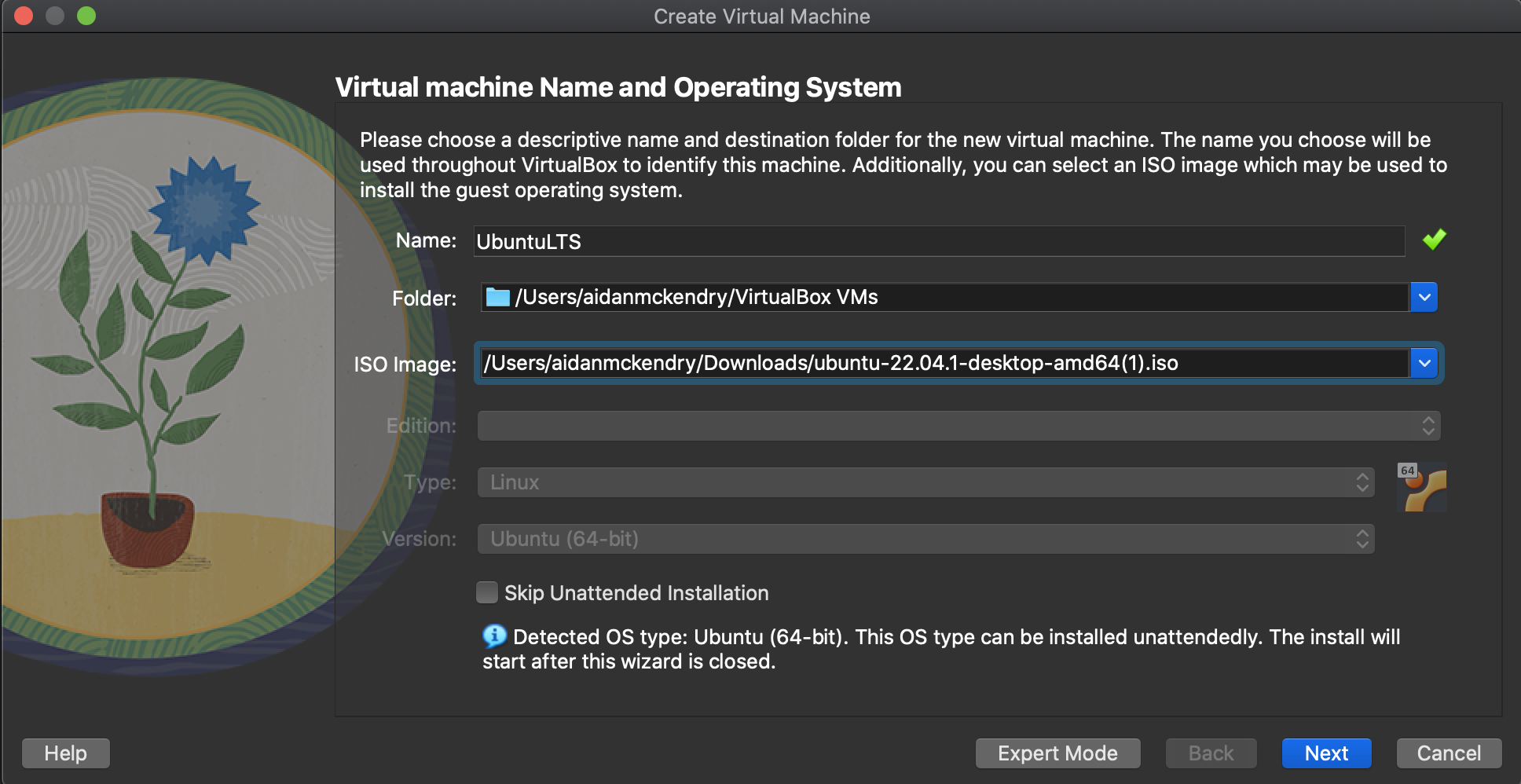
**Project documentation**

**MongoDB silent installation script**

1. Downloaded ubuntu version 22.04.1 (<https://ubuntu.com/download/desktop>)
2. Created a new virtual machine



1. Set the username and password
2. Allocated 4gb RAM and 2 CPUs to the VM.

A screenshot of a computer

Description automatically generated with medium confidence

1. I then set the memory of the virtual hard drive to 25GB.
2. Selected Finish to begin the unattended installation process.
3. Created a new user with a password.
4. I added a new user to the VM with sudo permissions and named it simply newuser with the commands shown below:

Graphical user interface, text, chat or text message

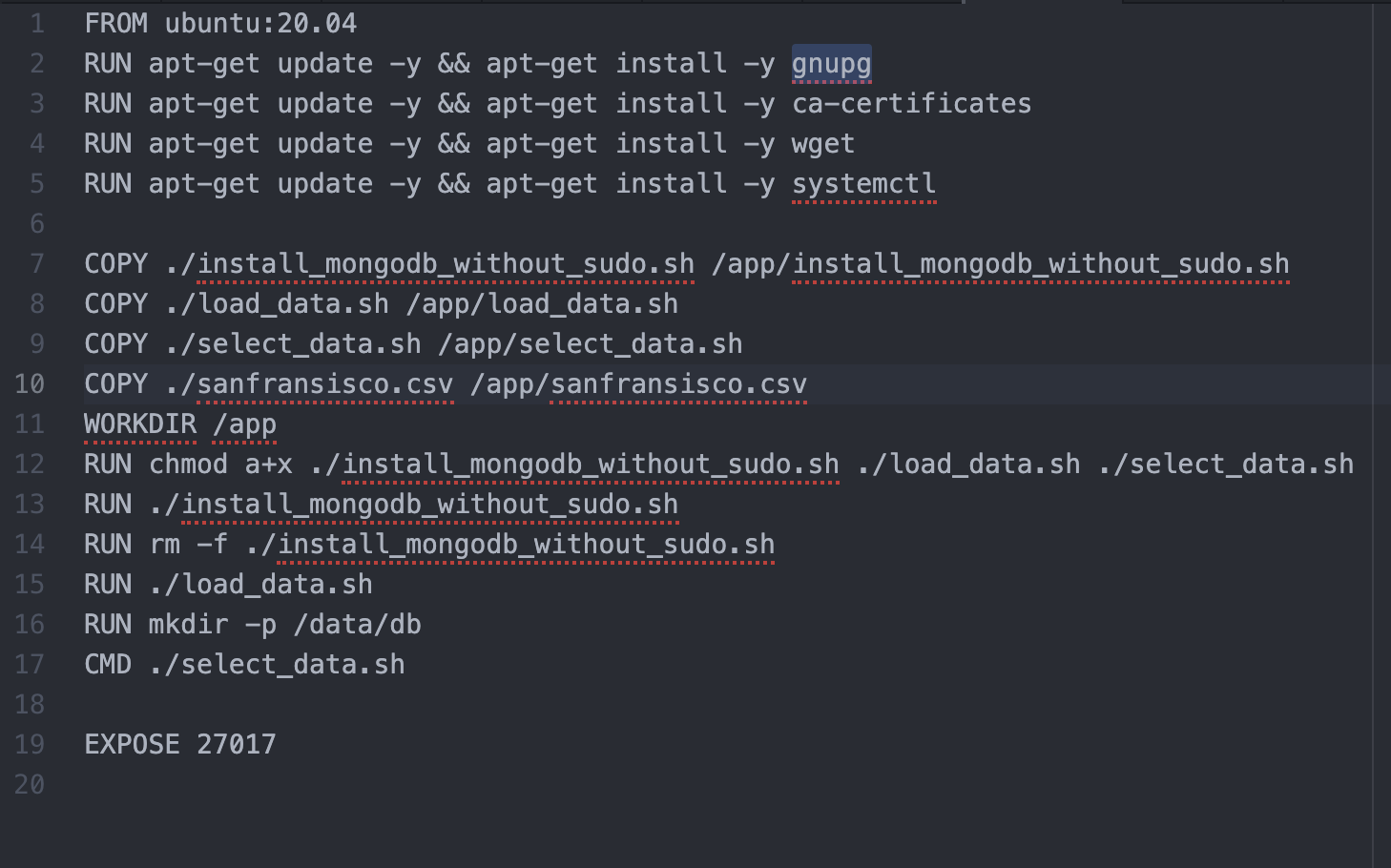
Description automatically generated

1. The newuser now has sudo permissions and will be used for the remainder of the installation process
2. Su newuser
3. Mkdir mongodb – making the mongodb directory for storing scripts for installation and set up.)
4. Nano install\_mongodb\_without\_sudo.sh (creating a shell file to house a list of commands for silent installation of Mongodb)

install\_mongodb\_without\_sudo.sh

1. apt-get update followed by apt-get install to allow installation of the dependencies: ca-certificates (verifying the download for Mongo), gnupg and wget which are needed within the container for installation.
2. Wget the apt key and add it to our container for reference.
3. Then we perform the installation of MongoDB version 4.4.15 since this will be stable.
4. Mongod –version verifies the installation was completed successfully.
5. The service is then stopped in this layer before finishing.
6. I also added the sed command to edit the bindIp value in mongos configuration file to allow connections from all addresses.

**Dockerising the silent installation script**



Dockerfile (Mongo)

1. Above is a screenshot of the dockerfile for creating the MongoDB image.
2. I install all dependencies then run the scripts to perform silent installation.
3. The shell scripts are copied over to an app directory which is then set as the present working directory.
4. The permissions of access to the shell files are changed to allow execution.
5. The Install\_mongodb\_without\_sudo.sh script is ran and MongoDB is successfully installed in the container.
6. Next this silent installation script is removed.
7. Text

   Description automatically generatedThe Load\_data script is then ran (shown below).

load\_data.sh

* 1. The mongod service is started.
  2. The status is checked to verify it is running.
  3. MongoImport method loads the dataset from SanFransiscoMonthlyTmpRecords.csv.
  4. The service is stopped from this layer.

1. Graphical user interface, text, application

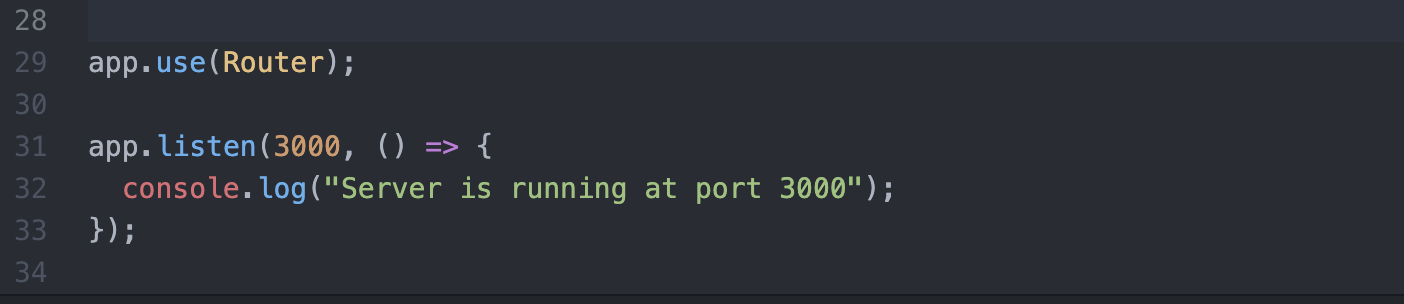
   Description automatically generatedNext in the Dockerfile the entry command for connections to the container executes the select\_data.sh script (shown below).

select\_data.sh

* 1. The service is started once again.
  2. The tail command prints the logs for the mongod service are contiuously printed.

1. Finally, in the dockerfile I exposed port 27017 for the container as this is the port on which mongod will run by default.
2. Build the docker image with “sudo docker build mongodb . “
3. This can then be ran or pushed to dockerhub for easy pulling within a docker-compose.
4. To do this we run the command docker tag <image\_name> <dockerhub\_username>/<imagename>:<version>
5. Docker login -u "<myusername>” -p "<mypassword>" docker.io
6. sudo docker push myusername/my\_mogodb:0.0.1

**Connecting external application to MongoDB container**

**Server.js fileText

Description automatically generated**

1. Mongoose (mongo client), express (framework for building Restful APIs) and the router are inherited into this file from node modules folder which was created when running npm install in the dockerfile.
2. App.use(express.json()) line allows the express to parse incoming JSON payloads since we are using JSON format for storing data in MongoDB.
3. There is then a promise-retry within the connectWithRetry which tries to connect to the mongod service running on “mongodb://mongodb:27017/test”, where mongodb will be the hostname of the database container and test is the name of my collection/ database. If a connection is unsuccessful, this code will print this in the logs and retry to connect in 5 seconds. It will retry 10 times depending on the value specified in the options variable.
4. App.use(Router); allows the express app access to the available routes/ endpoints for this service.
5. The app is set to listen to port 3000 allowing requests to be sent to this port. This is also logged in the console to give visibility in the container.

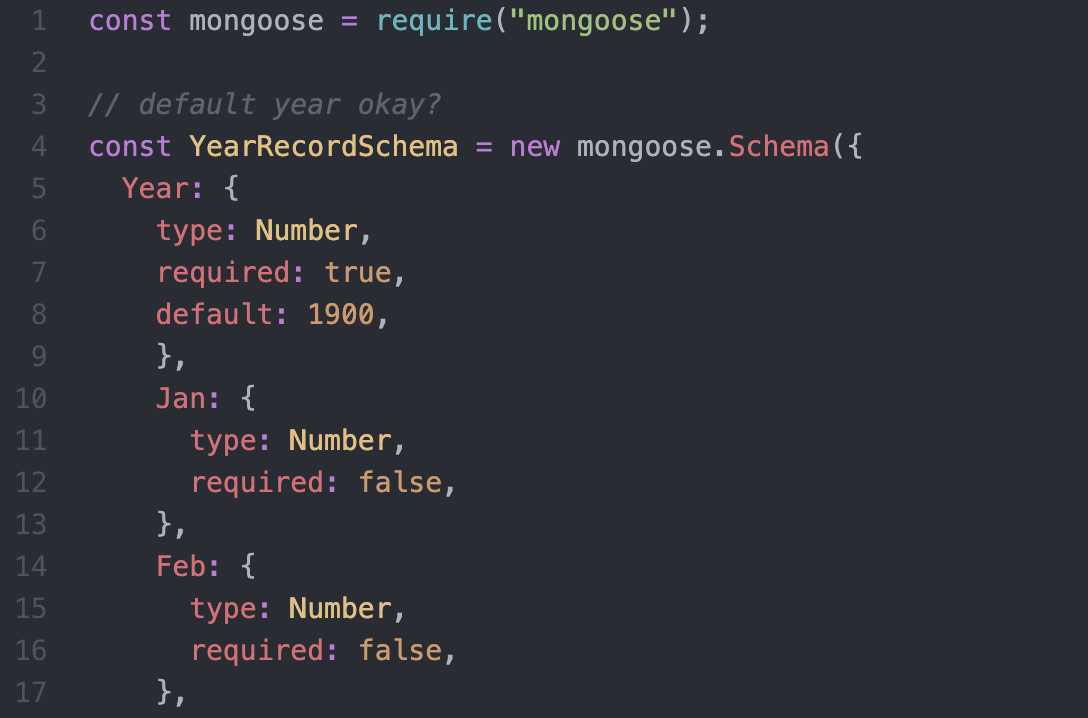
**Routes.js**

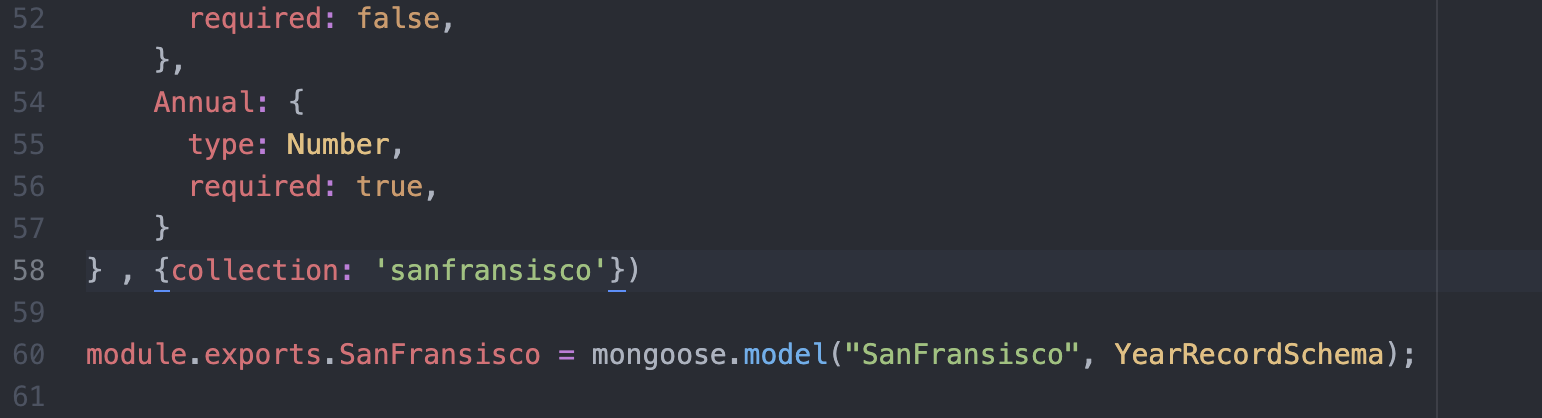
Text

Description automatically generated

1. Express is imported from the node modules.
2. The SanFransisco variable is a JSON schema defined by the models.js file to match the structure of the dataset stored in the Mongo database.
3. Currently there are two routes defined:
   1. ‘/’ currently just returns a status of “ok” – this will be adapted into the base map visualisation route.
   2. ‘/allYears’ returns all data from the dataset which is saved in a collection named “sanfransisco.” This responded JSON object is then displayed with the res.send() method. If an error is encountered retrieving this data, a 500 HTTP status will be returned along with the associated error description.
4. These routes are then exposed to other .js components so they are callable.

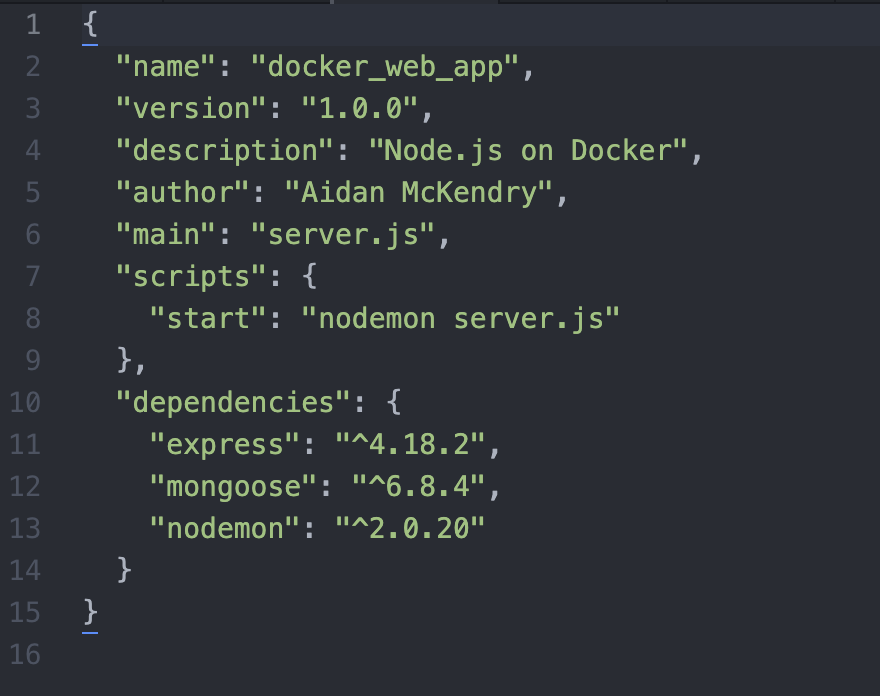
**Model.js**





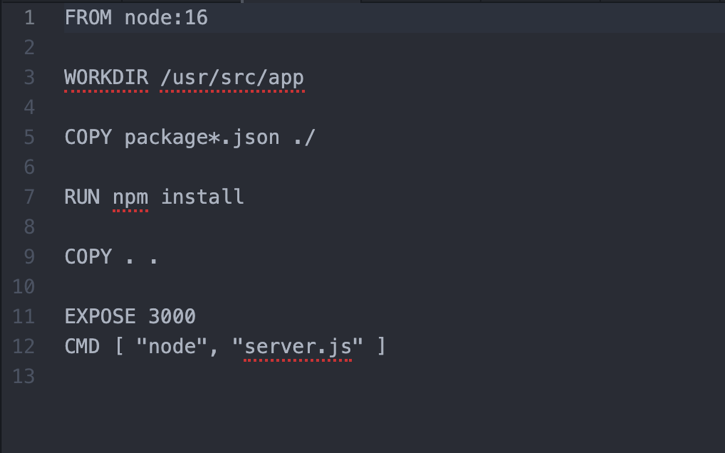
1. As mentioned above this file matches the structure of the JSON objects stored in MongoDB by creating a Mongoose schema and entering all fields that are in the dataset.
2. I have entered the months as optional fields because some monthly records are missing from the datasets, and they should not be vital for backend tasks. The Year and Annual field are set to required: true though.
3. At the end of this schema, an associated collection is specified so Mongodb can search for this collection find method is called within the routes.js file.
4. Finally, this model is exported to be included in other files such as the routes file.

**Package.js**



1. This file holds the dependencies for the Node application in one place along with the versions and some description around the application name, author, version and scripts to start.
2. Additional dependencies will be added here when required for GIS functions and map visualization.

**Dockerfile (Node app)**



1. This file dockerises the application and follows a few simple steps including setting the base image as node:16.
2. Sets a working directory for the container
3. Installs npm for running node.
4. Exposes the container on port 3000.
5. CMD entry command specifies to run the server.js with node on start of the container.

**Docker-compose.yml**

Text

Description automatically generated

1. Sets the version of docker-compose at 3.3.
2. Defines two services: (containers) app & mongodb.
3. App service:
   1. The app service builds the ./mongoose\_tutorial which holds all the files for the node application.
   2. A hostname and dependency to the mongodb service are specified here.
   3. The ports are then mapped as 0.0.0.0:3000 -> 3000 in the container.
   4. I also set the environment to wait on the host mongodb service before builsing the node container.
   5. The volume for this container is set to the working directory of the container that is set in the dockerfile.
4. MongoDB service:
   1. The image is pulled from my dockerhub account which the database container was pushed to.
   2. The container is given a hostname and container\_name for visibility.
   3. Ports are mapped as 0.0.0.0:37017 -> 27017 allowing my machines port 37017 to mao to 27017 in the container where the database service is running.
   4. Volumes for the database data is set to directory data/db to keep the volume and persist its data.