```
EXPOSE 8080

COPY ./ /app

CMD ["python", "main.py"]
```

Let's go through the instructions contained in the preceding Dockerfile one by one:

- The first instruction the Dockerfile executes is to set the base image for our own image using the `FROM` keyword. Other variations of this image can be found at `https://hub.docker.com/_/python`.
- The next line uses the `WORKDIR` keyword to set the working directory to `/app`. A working directory helps organize the structure of the project being built to an image.
- Next, we copy the `requirements.txt` file from the local directory to the working directory on the Docker container using the `COPY` keyword.
- The next instruction is the `RUN` command, which is used to upgrade the pip package and then install the dependencies from the `requirements.txt` file.
- The next command exposes the PORT from which our application can be accessed from the local network.
- The next command copies the rest of the files and folders into the Docker container working directory.
- Lastly, the last command starts the application using the `CMD` command.

Each set of instructions listed in the Dockerfile is built as an individual layer. Docker does an intelligent job of caching each layer during a build to reduce build time and eliminate repetition. If a layer that is essentially an instruction is untouched, the layer is skipped and the previously built one is used. That is, Docker uses the cache system when building images.

Let's create a `.dockerignore` file before proceeding to build our image:

```
(venv)$ touch .dockerignore
```

.dockerignore

```
Venv
.env
.git
```

What Is .dockerignore?

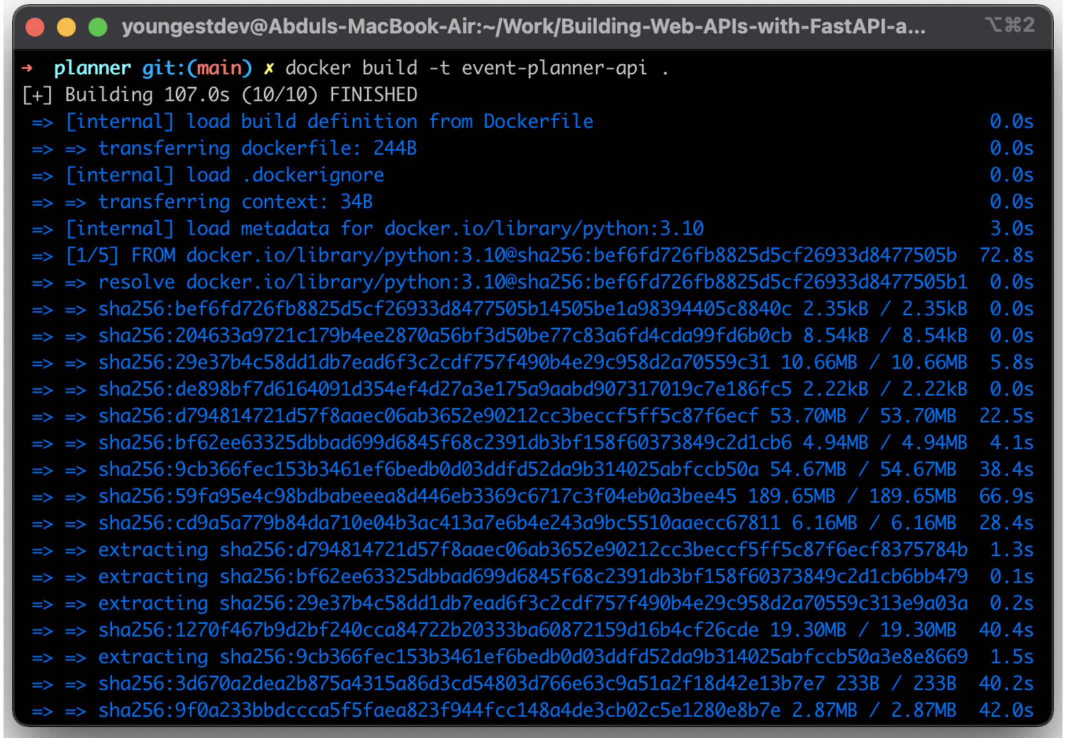
The `.dockerignore` file contains files and folders to be exempted from instructions defined in the Dockerfile.

Building the Docker image

To build the application image, run the following command in the base directory:

```
(venv)$ docker build -t event-planner-api .
```

This command simply tells Docker to build an image with the `event-planner-api` tag from the instructions defined in the current directory, which is represented by the dot at the end of the command. The build process commences once the command is run and the instructions are executed:



```
youngestdev@Abduls-MacBook-Air:~/Work/Building-Web-APIs-with-FastAPI-a...  ⌥⌘2
➜  planner git:(main) ✗ docker build -t event-planner-api .
[+] Building 107.0s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                   0.0s
 => => transferring dockerfile: 244B                                                   0.0s
 => [internal] load .dockerignore                                                      0.0s
 => => transferring context: 34B                                                       0.0s
 => [internal] load metadata for docker.io/library/python:3.10                         3.0s
 => [1/5] FROM docker.io/library/python:3.10@sha256:bef6fd726fb8825d5cf26933d8477505b  72.8s
 => => resolve docker.io/library/python:3.10@sha256:bef6fd726fb8825d5cf26933d8477505b1  0.0s
 => => sha256:bef6fd726fb8825d5cf26933d8477505b14505be1a98394405c8840c 2.35kB / 2.35kB  0.0s
 => => sha256:204633a9721c179b4ee2870a56bf3d50be77c83a6fd4cda99fd6b0cb 8.54kB / 8.54kB  0.0s
 => => sha256:29e37b4c58dd1db7ead6f3c2cdf757f490b4e29c958d2a70559c31 10.66MB / 10.66MB  5.8s
 => => sha256:de898bf7d6164091d354ef4d27a3e175a9aabd907317019c7e186fc5 2.22kB / 2.22kB  0.0s
 => => sha256:d794814721d57f8aaec06ab3652e90212cc3beccf5ff5c87f6ecf 53.70MB / 53.70MB  22.5s
 => => sha256:bf62ee63325dbbad699d6845f68c2391db3bf158f60373849c2d1cb6 4.94MB / 4.94MB  4.1s
 => => sha256:9cb366fec153b3461ef6bedb0d03ddfd52da9b314025abfccb50a 54.67MB / 54.67MB  38.4s
 => => sha256:59fa95e4c98bdbabeeea8d446eb3369c6717c3f04eb0a3bee45 189.65MB / 189.65MB  66.9s
 => => sha256:cd9a5a779b84da710e04b3ac413a7e6b4e243a9bc5510aaecc67811 6.16MB / 6.16MB  28.4s
 => => extracting sha256:d794814721d57f8aaec06ab3652e90212cc3beccf5ff5c87f6ecf8375784b  1.3s
 => => extracting sha256:bf62ee63325dbbad699d6845f68c2391db3bf158f60373849c2d1cb6bb479  0.1s
 => => extracting sha256:29e37b4c58dd1db7ead6f3c2cdf757f490b4e29c958d2a70559c313e9a03a  0.2s
 => => sha256:1270f467b9d2bf240cca84722b20333ba60872159d16b4cf26cde 19.30MB / 19.30MB  40.4s
 => => extracting sha256:9cb366fec153b3461ef6bedb0d03ddfd52da9b314025abfccb50a3e8e8669  1.5s
 => => sha256:3d670a2dea2b875a4315a86d3cd54803d766e63c9a51a2f18d42e13b7e7 233B / 233B  40.2s
 => => sha256:9f0a233bbdccca5f5faea823f944fcc148a4de3cb02c5e1280e8b7e 2.87MB / 2.87MB  42.0s
```

Figure 9.1 – Docker build process

Now that we have successfully built our application's image, let's pull the MongoDB image:

```
(venv)$ docker pull mongo
```

We're pulling a MongoDB image to create a standalone database container accessible from the API container when created. By default, the Docker container has a separate network configuration, and connecting to the machine hosts' localhost address is not allowed.

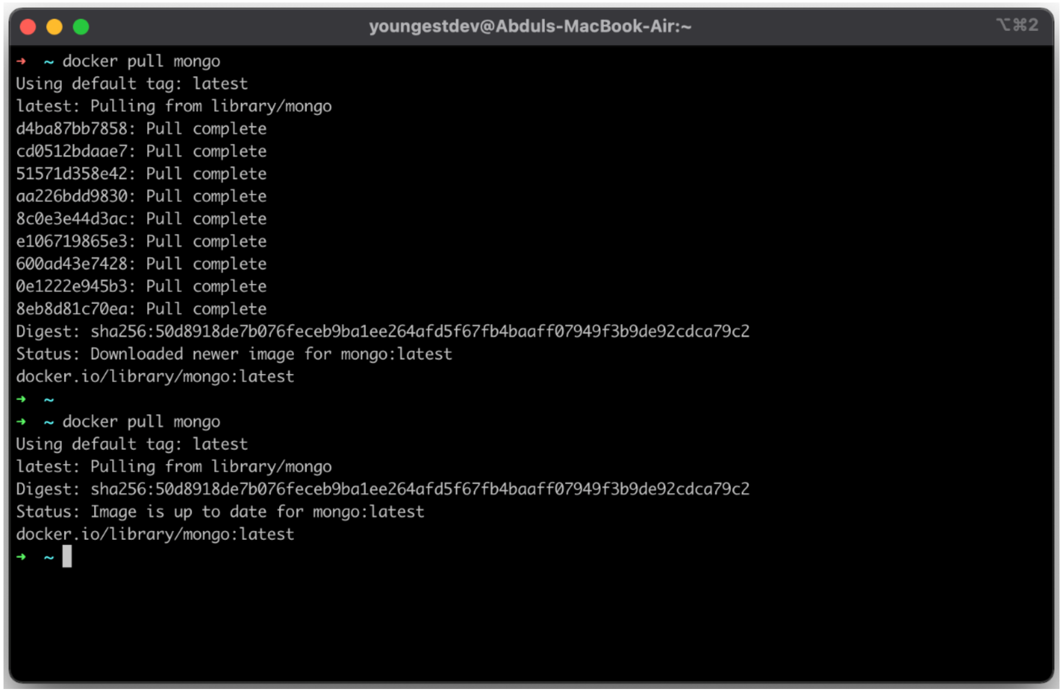

```
youngestdev@Abduls-MacBook-Air:~                                    ⌥⌘2
➜  ~ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
d4ba87bb7858: Pull complete
cd0512bdaae7: Pull complete
51571d358e42: Pull complete
aa226bdd9830: Pull complete
8c0e3e44d3ac: Pull complete
e106719865e3: Pull complete
600ad43e7428: Pull complete
0e1222e945b3: Pull complete
8eb8d81c70ea: Pull complete
Digest: sha256:50d8918de7b076feceb9ba1ee264afd5f67fb4baaff07949f3b9de92cdca79c2
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
➜  ~
➜  ~ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
Digest: sha256:50d8918de7b076feceb9ba1ee264afd5f67fb4baaff07949f3b9de92cdca79c2
Status: Image is up to date for mongo:latest
docker.io/library/mongo:latest
➜  ~
```

Figure 9.2 – Pulling a MongoDB image

What Is docker pull?

The `docker pull` command is responsible for downloading images from a registry. Unless otherwise stated, these images are downloaded from the public Docker Hub registry.

Deploying our application locally

Now that we have created the images for the API and pulled the image for the MongoDB database, let's proceed to write a compose manifest to handle our application deployment. The docker-compose manifest will consist of the API service and the MongoDB database service. In the root directory, create the manifest file:

```
(venv)$ touch docker-compose.yml
```

The contents of the docker-compose manifest file will be as follows:

docker-compose.yml

```
version: "3"

services:
  api:
    build: .
    image: event-planner-api:latest
    ports:
      - "8080:8080"
    env_file:
      - .env.prod

  database:
    image: mongo
    ports:
      - "27017"
    volumes:
      - data:/data/db

volumes:
  data:
```

In the `services` section, we have the `api` service and the `database` service. In the `api` service, the following set of instructions are put in place:

- The `build` field instructs Docker to build the `event-planner-api:latest` image for the `api` service from the Dockerfile situated in the current directory denoted by `.`.
- Port `8080` is exposed from the container to enable us to access the service through HTTP.

- The environment file is set to `.env.prod`. Alternatively, the environment variables can be set in this format:

  ```
  environment:
    - DATABASE_URL=mongodb://database:27017/planner
    - SECRET_KEY=secretkey
  ```

  This format is mostly used when environment variables are to be injected from a deployment service. It is encouraged to use the environment file.

In the database service, the following set of instructions are put in place:

- The `database` service makes use of the `mongo` image we pulled earlier.
- Port `27017` is defined but not exposed externally. The port is only accessible internally by the `api` service.
- A persistent volume is attached to the service to store our data. The folder allocated for this is `/data/db`.
- Lastly, the volume for this deployment is created with the name `data`.

Now that we have understood the content of the compose manifest, let's create the environment file, `.env.prod`:

.env.prod

```
DATABASE_URL=mongodb://database:27017/planner
SECRET_KEY=NOTSTRONGENOUGH!
```

In the environment file, `DATABASE_URL` is set to the name of the MongoDB service created by the compose manifest.

Running our application

We are set to deploy and run the application from the docker-compose manifest. Let's start the services using the compose tool:

```
(venv)$ docker-compose up -d
```