**DOCKER**

* Docker is a container technology: A tool for Creating and managing containers
* Container: A package of code and dependencies to run that code (Node js code + Node JS runtime)
* Example of sample Docker file

FROM node:14

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

CMD [ "node", "app.mjs" ]

COPY . . -> The first dot implies copy all the files in the project ,Second dot specifies the path inside of the image

Npm install -> for install all dependencies for node application

* Command for build this docker file : docker build .
* Command for run image : docker run image-ID
* For interactively run a container : docker run -it image-ID
* Command for add port : docker run -p 3000:3000 image-ID
* To remove all stopped containers: docker container prune
* now we can see our application in localhost address: localhost:3000
* Ways to get an image

1. Using Existing image
2. Using Custom image

* For restart a stopped container: docker start container-ID
* For getting logs : docker logs container-name
* By default, if you run a Container without -d, you run in "attached mode".
* If you started a container in detached mode (i.e. with -d), you can still attach to it afterwards without restarting the Container with the following command: docker attach container-ID
* For stop a container: docker stop container-name
* For remove a container: docker rm container-name
* For remove an image: docker rmi image-id
* docker inspect image-id : For getting all the details of our image

Removing stopped containers automatically:

Docker run -p 3000:80 -d –rm container-id : the --rm command helps to automatically remove the container if container stopped.

Copying files into and from a container:

Docker cp folder\_name/filename container\_name:/path inside container

or Docker cp folder\_name/. container\_name:/path inside container

Naming and tagging containers and images:

Container: docker run -p 3000:3000 -d –rm –name container\_name image\_id

Image: docker build -t goals: latest . or docker build -t arun:1 .

Pushing Images to docker hub:

Docker push image-name

Docker pull image-name

**SECTION 3: Managing Data and working with volume**

* Volumes helps to persist the data
* Volumes are folders on your host machines hard drive which are mounted into containers
* Volume persist data even if container shutdown
* Types of external data storage for docker:

1)Volumes (managed by docker) 2)Bind Mounts (managed by you)

* Volumes can be divided in to two: Anonymous volume and Named Volume
* Docker volume ls : to list all the volume currently manage
* Anonymous volume will remove automatically when the container shutdown
* But Named volumes will be available even if we shutdown our container

Command to create named volume

Docker run -d -p 3000:80 –rm –name feedback-app -v Arun:/app/feedback feedback-node: volume

* Feedback-node: volume is image name
* -v Arun:/app/feedback is the command for create named Volume

Command for delete anonymous volume:

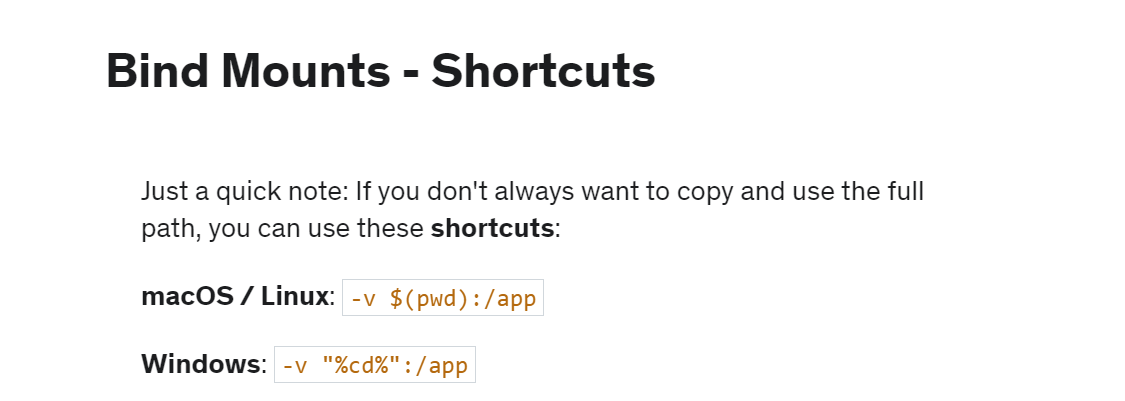
Docker volume rm VOL\_Name or docker volume prune

Bind Mounts

* You define a folder or path In your host machine.
* Command to Bind Mounts:

Docker run -d -p 3000:80 –rm –name feedback-app -v “ABSOLUTE-PATH-OF-FOLDER :/app” feedback-node: volume

* /app is the working director



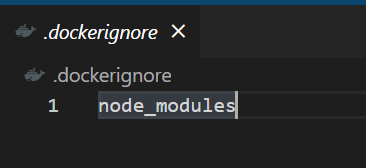
Read only Docker volume

By default, docker volume is read-write, for making it as read only we need to use the following command.

Docker run -d -p 3000:80 –rm –name feedback-app -v Arun:/app/feedback:ro feedback-node: volume

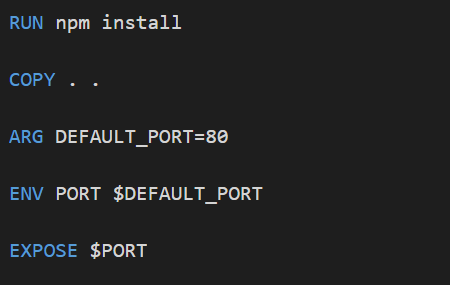
Docker run -d -p 3000:80 –rm –name feedback-app -v “ABSOLUTE-PATH-OF-FOLDER :/app:ro” feedback-node: volume

. dockerignore: File which is used to mention which files and folders need to ignore when using COPY command in our Docker File.

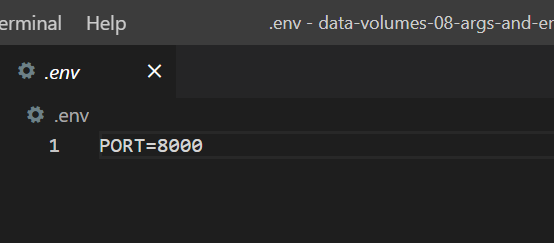


Arguments and Environment variables

* ARG: Available inside docker file, not accessible in CMD or in application code



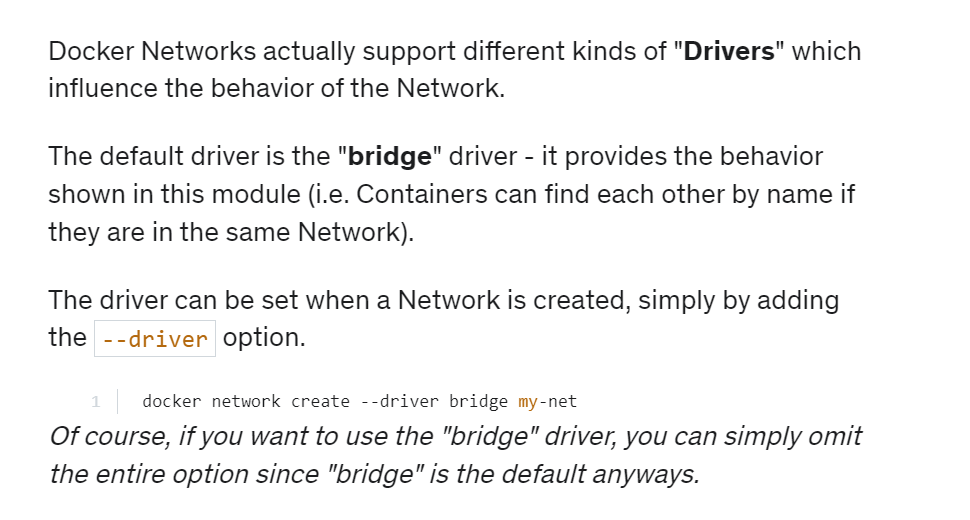
* ENV: Available inside docker file and in application code ,set via ENV in docker file or via –env on docker run.
* Also we can add one file with our code file as .env and we can add all our environment variable inside it , and we can call the file inside our build command as Docker run –d –rm –p 3000:8000 –env-file ./.env



…………………………………………………………………………………………………………………………………………….

**SECTION 4: Networking and cross-container communication**

* Within a docker network, all containers can communicate with each other and Ips are automatically resolved.
* Command to create docker network: docker network create network-name
* Command to create docker network: docker network ls
* Creating a container with adding network : docker run -d –name mongodb –network network-name



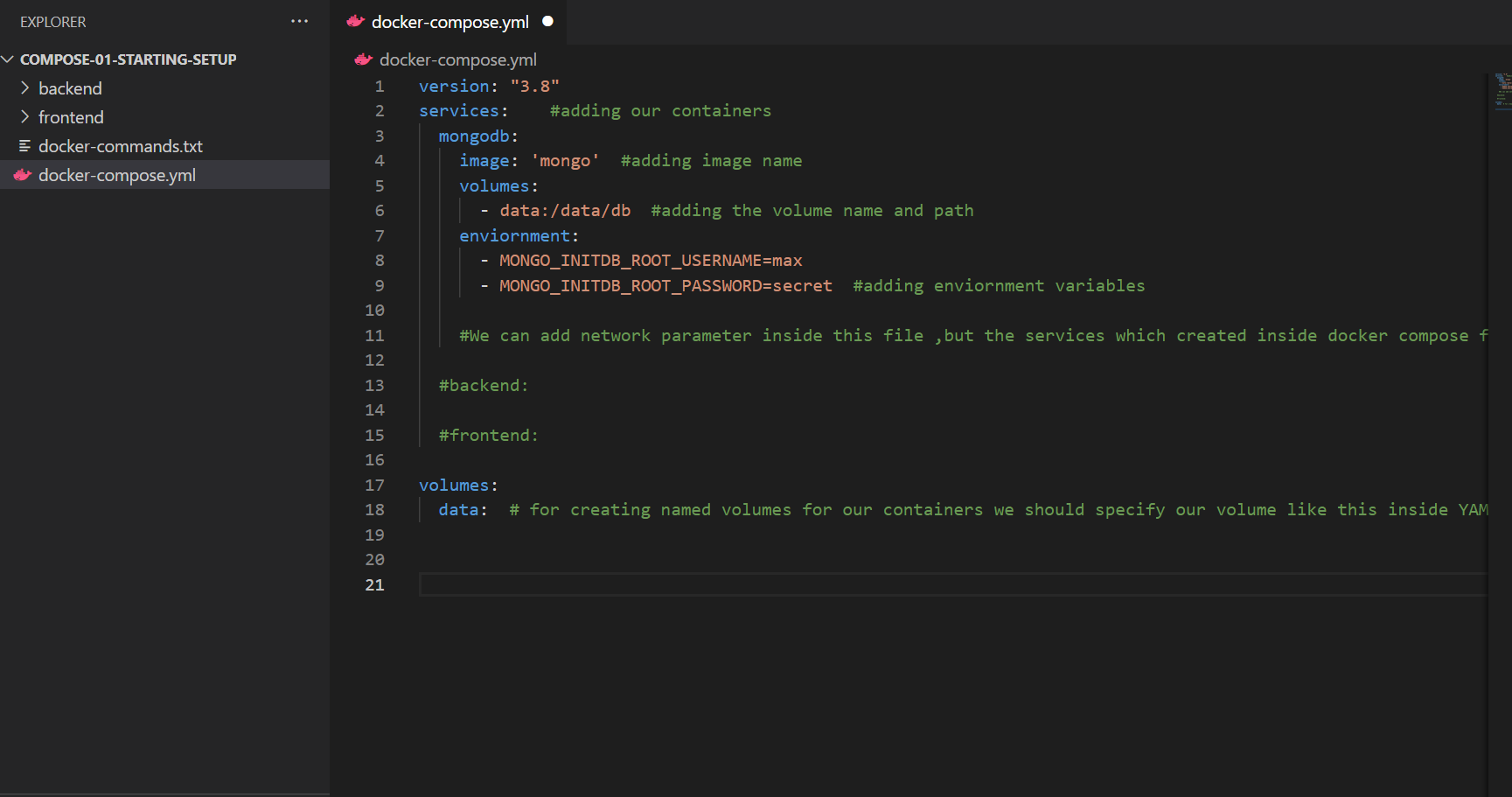
**SECTION 5: Docker Compose : Multi container orchestration**

* Docker compose is a tool that allows you to replace multiple Docker Build and Docker Run commands with one configuration file + set of orchestration commands to build docker images.
* Docker compose does not replace docker files for custom images
* Docker compose does not replace images or containers
* Docker compose is not suited for managing multiple containers on different host(machines)

Creating a compose file

* Create a docker-compose.yml file in the project folder

1)creating compose file for our mongo DB container



version: "3.8"

services:    #adding our containers

  mongodb:

    image: 'mongo'  #adding image name

    volumes:

      - data:/data/db  #adding the volume name and path

    Environment:

      - MONGO\_INITDB\_ROOT\_USERNAME=max

      - MONGO\_INITDB\_ROOT\_PASSWORD=secret  #adding environment variables

    #We can add network parameter inside this file ,but the services which created inside docker compose file automatically comes under one network.

  #backend:

  #frontend:

volumes:

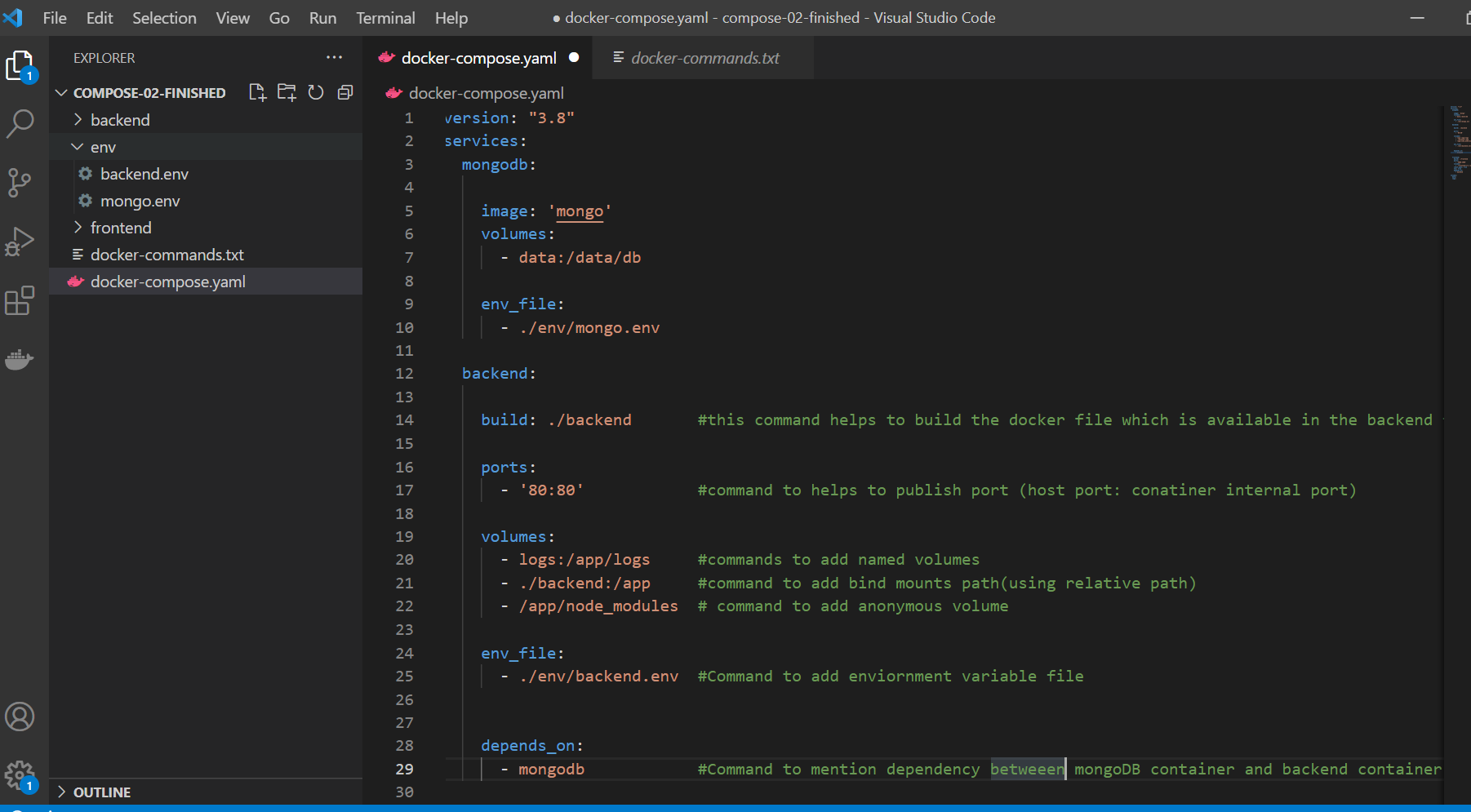
  data:  # for creating named volumes for our containers we should specify our volume like this inside YAML file

Command to run Docker compose file or start services: docker-compose up

Command to stop Docker compose file or stop services: docker-compose down

For deleting volume also: docker-compose down -v

2)Adding second container (Backend container)



version: "3.8"

services:

  mongodb:

    image: 'mongo'

    volumes:

      - data:/data/db

    env\_file:

      - ./env/mongo.env

  backend:

    build: ./backend       #this command helps to build the docker file which is available in the backend folder.

    ports:

      - '80:80'            #command to helps to publish port (host port: container internal port)

    volumes:

      - logs:/app/logs     #commands to add named volumes

      - ./backend:/app     #command to add bind mounts path(using relative path)

      - /app/node\_modules  # command to add anonymous volume

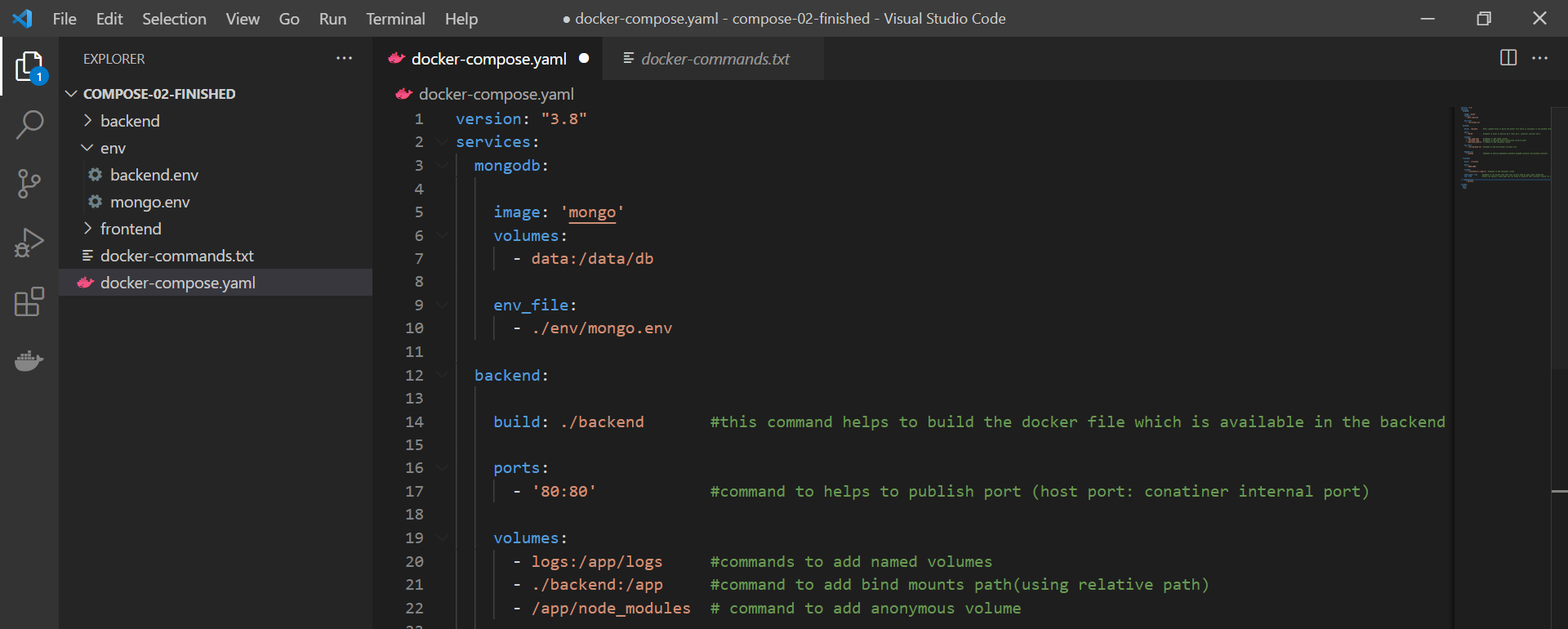
    env\_file:

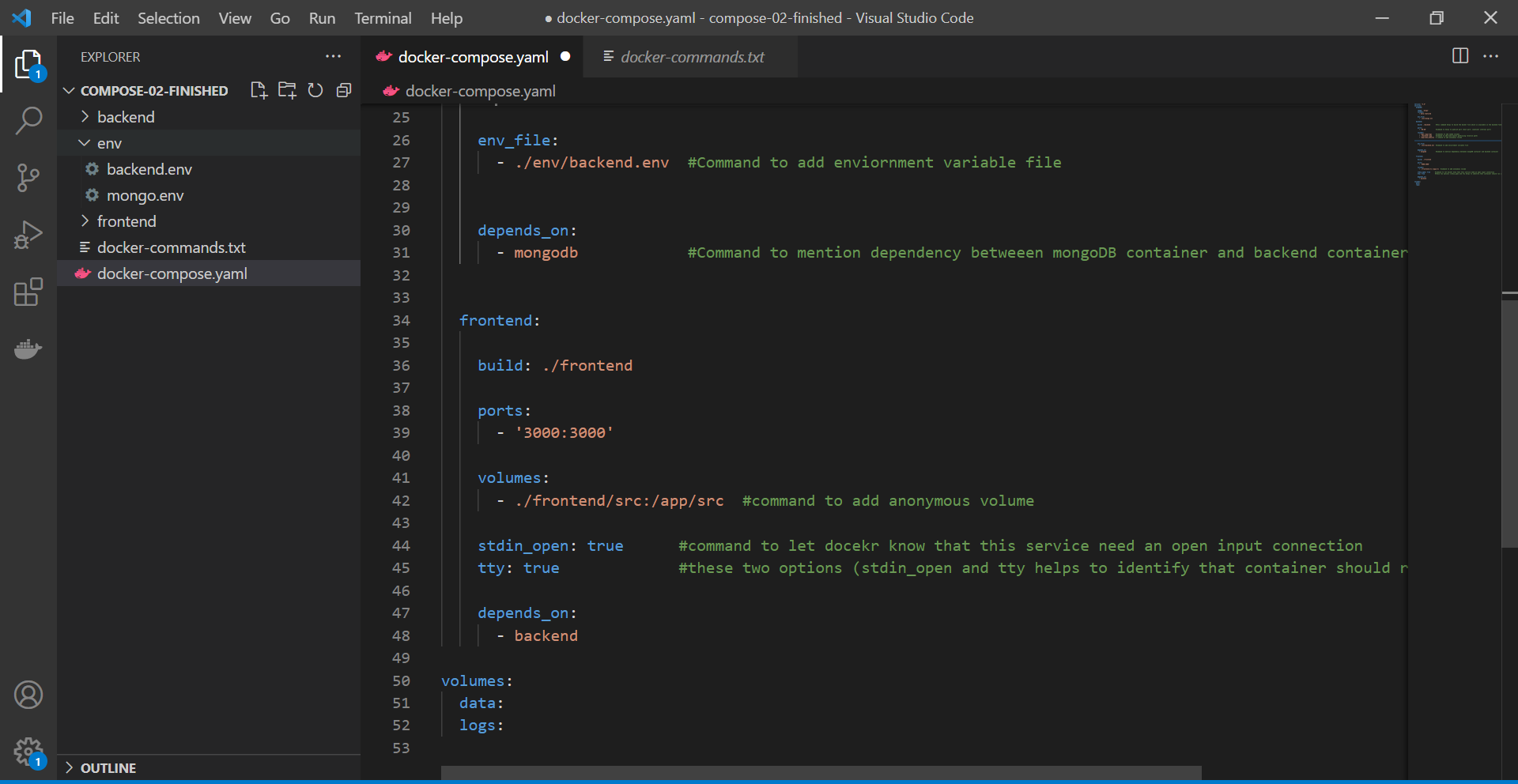
      - ./env/backend.env  #Command to add environment variable file

    depends\_on:

      - mongodb            #Command to mention dependency between mongoDB container and backend container

3)Adding third container (frontend container)





version: "3.8"

services:

  mongodb:

    image: 'mongo'

    volumes:

      - data:/data/db

    env\_file:

      - ./env/mongo.env

  backend:

    build: ./backend       #this command helps to build the docker file which is available in the backend folder.

    ports:

      - '80:80'            #command to helps to publish port (host port: conatiner internal port)

    volumes:

      - logs:/app/logs     #commands to add named volumes

      - ./backend:/app     #command to add bind mounts path(using relative path)

      - /app/node\_modules  # command to add anonymous volume

    env\_file:

      - ./env/backend.env  #Command to add enviornment variable file

    depends\_on:

      - mongodb            #Command to mention dependency betweeen mongoDB container and backend container

  frontend:

    build: ./frontend

    ports:

      - '3000:3000'

    volumes:

      - ./frontend/src:/app/src  #command to add anonymous volume

    stdin\_open: true      #command to let docekr know that this service need an open input connection

    tty: true             #these two options (stdin\_open and tty helps to identify that container should run in interactive mode)

    depends\_on:

      - backend

volumes:

  data:

  logs:

* by default container name to be displayed as project\_foldername-service name , for creating custom container name :

version: "3.8"

services:

  mongodb:

    image: 'mongo'

    volumes:

      - data:/data/db

    env\_file:

      - ./env/mongo.env

    container\_name: arun-container  # command to add custom container name

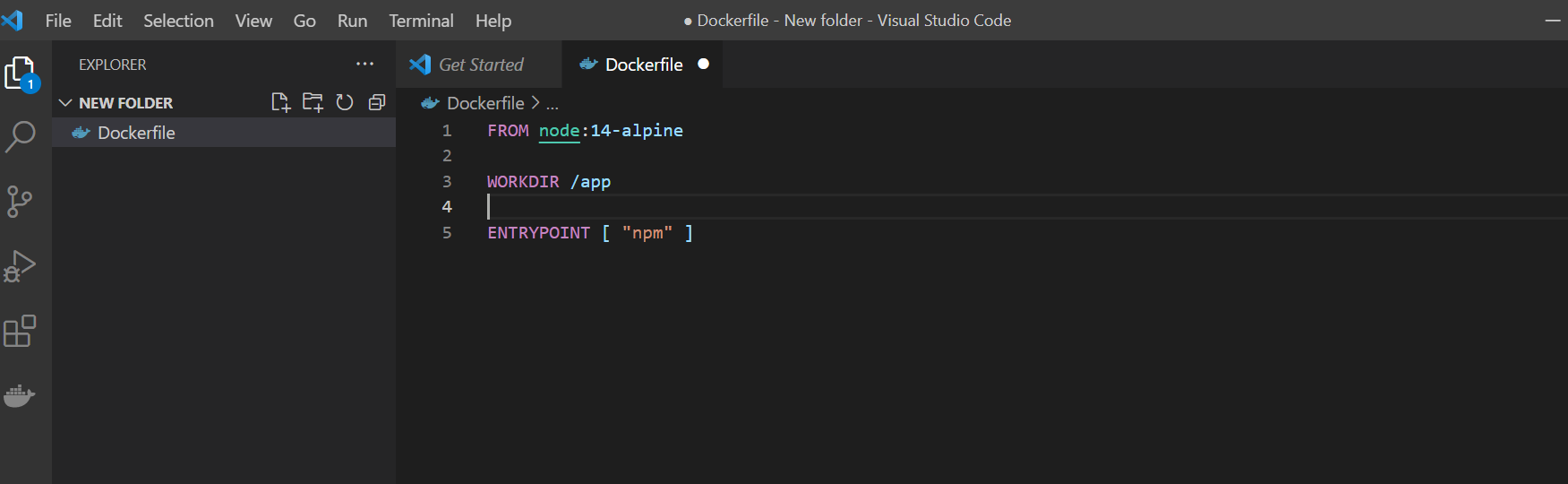
…………………………………………………………………………………………………………………………………

**SECTION 6: Working with Utility Containers:**

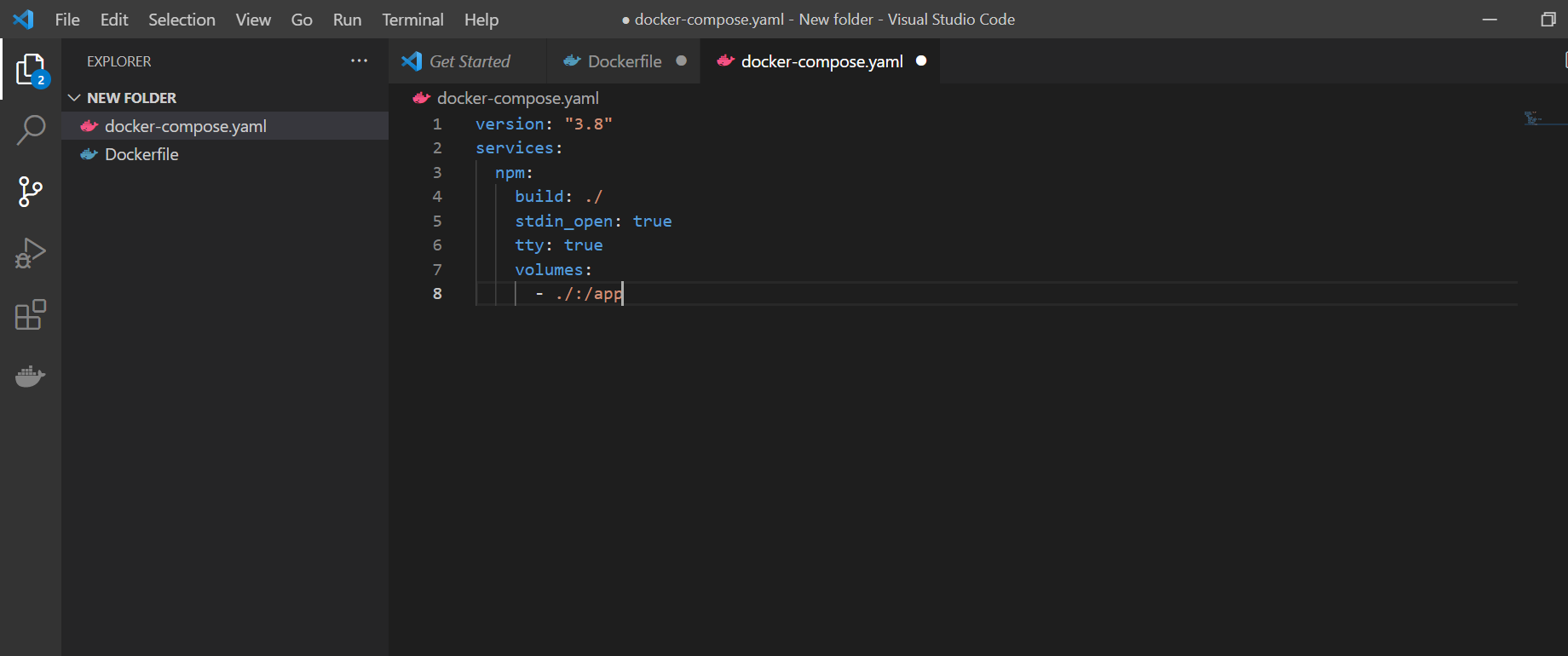
* The utility container provides a space where miscellaneous tools and other software can be installed. Tools and objects can be placed in a utility container if they do not require a dedicated container or if it is impractical to create a new container for a single tool or object.
* This is not a application container, its an environment container.
* Docker exec command helps to run commands inside a running container

Eg: docker exec -it container-name npm init

Creating a utility container



Using Docker compose file for above task



* Commands to run docker compose for utility containers: docker-compose run service-name init

**SECTION 7: Working with Utility Containers: A Larval and PHP Dockerized project:**

1. Adding a Nginx Webserver

version: "3.8"

services:

  server:

    image: 'nginx:stable-alpine'

    ports:

      - '8000:80'

    volumes:

      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro

  # php:

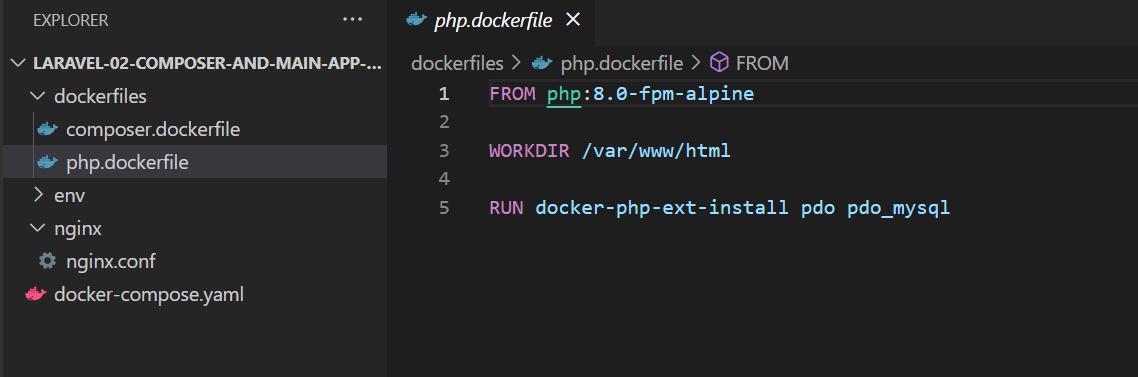
  # mysql:

  # composer:

  # artisan:

  # npm:

2)Adding a PHP Container (from custom image)



FROM php:8.0-fpm-alpine

WORKDIR /var/www/html

RUN docker-php-ext-install pdo pdo\_mysql

Docker Compose file

version: "3.8"

services:

  server:

    image: 'nginx:stable-alpine'

    ports:

      - '8000:80'

    volumes:

      - ./src:/var/www/html

      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf:ro

  php:

    build:

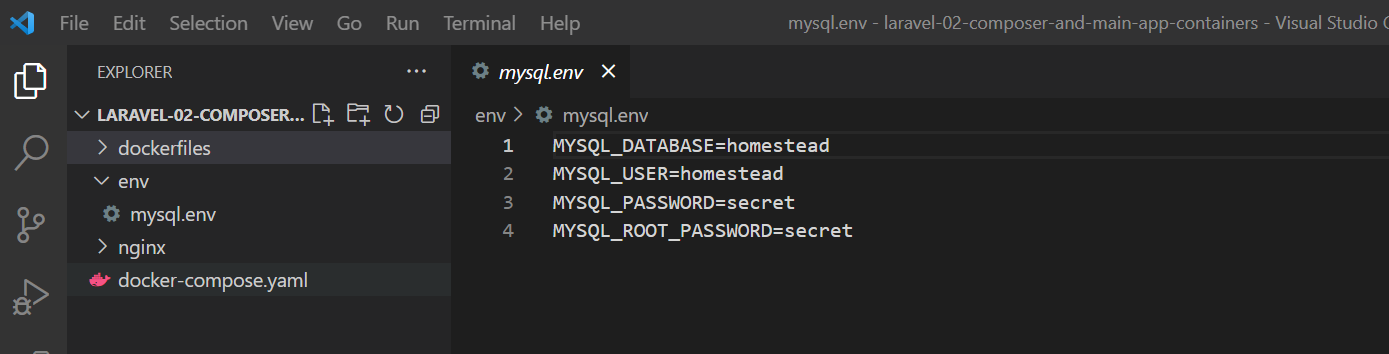
      context: ./dockerfiles

      dockerfile: php.dockerfile

    volumes:

      - ./src:/var/www/html:delegated

3)Creating MYSQL Container



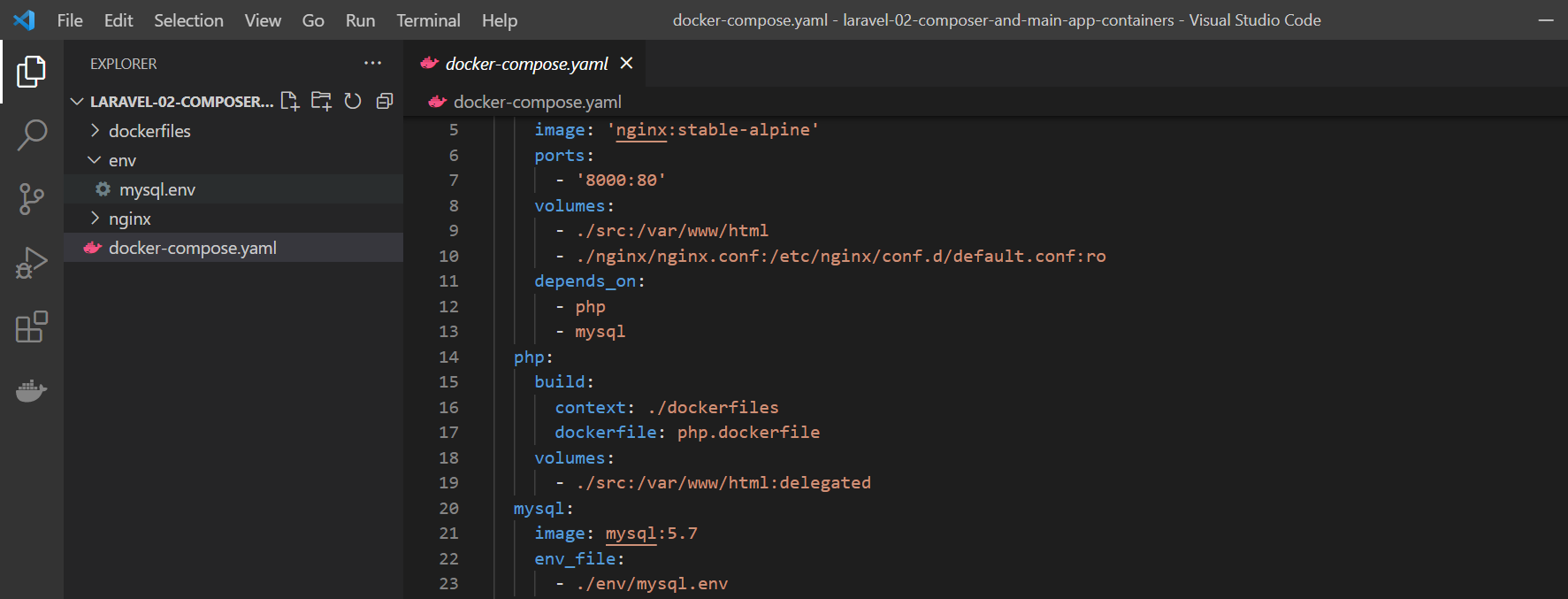
Docker compose file

mysql:

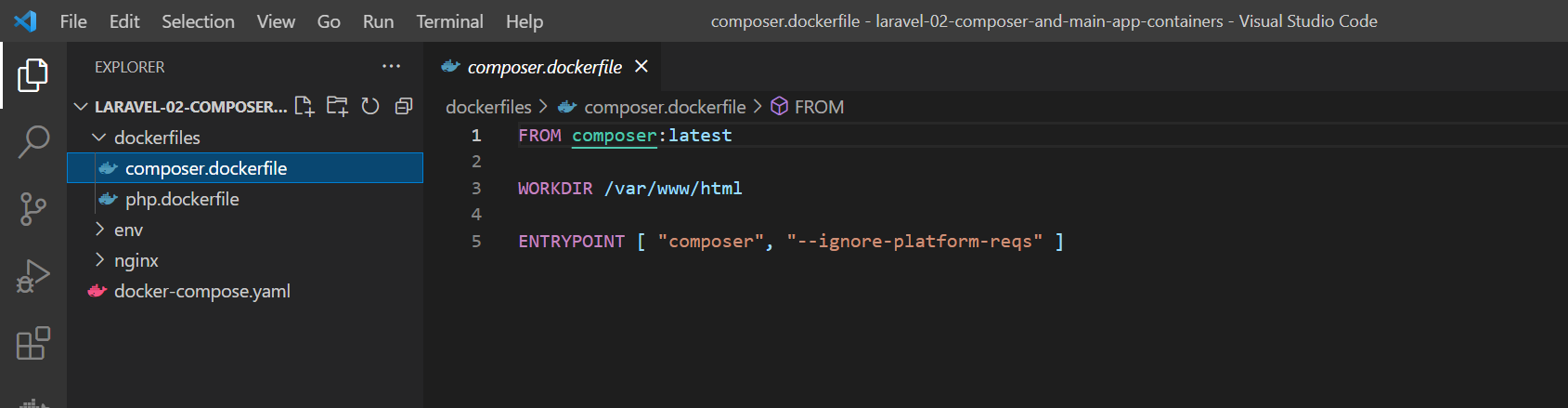
    image: mysql:5.7

    env\_file:

      - ./env/mysql.env



4)Adding a composer utility container (from custom image)

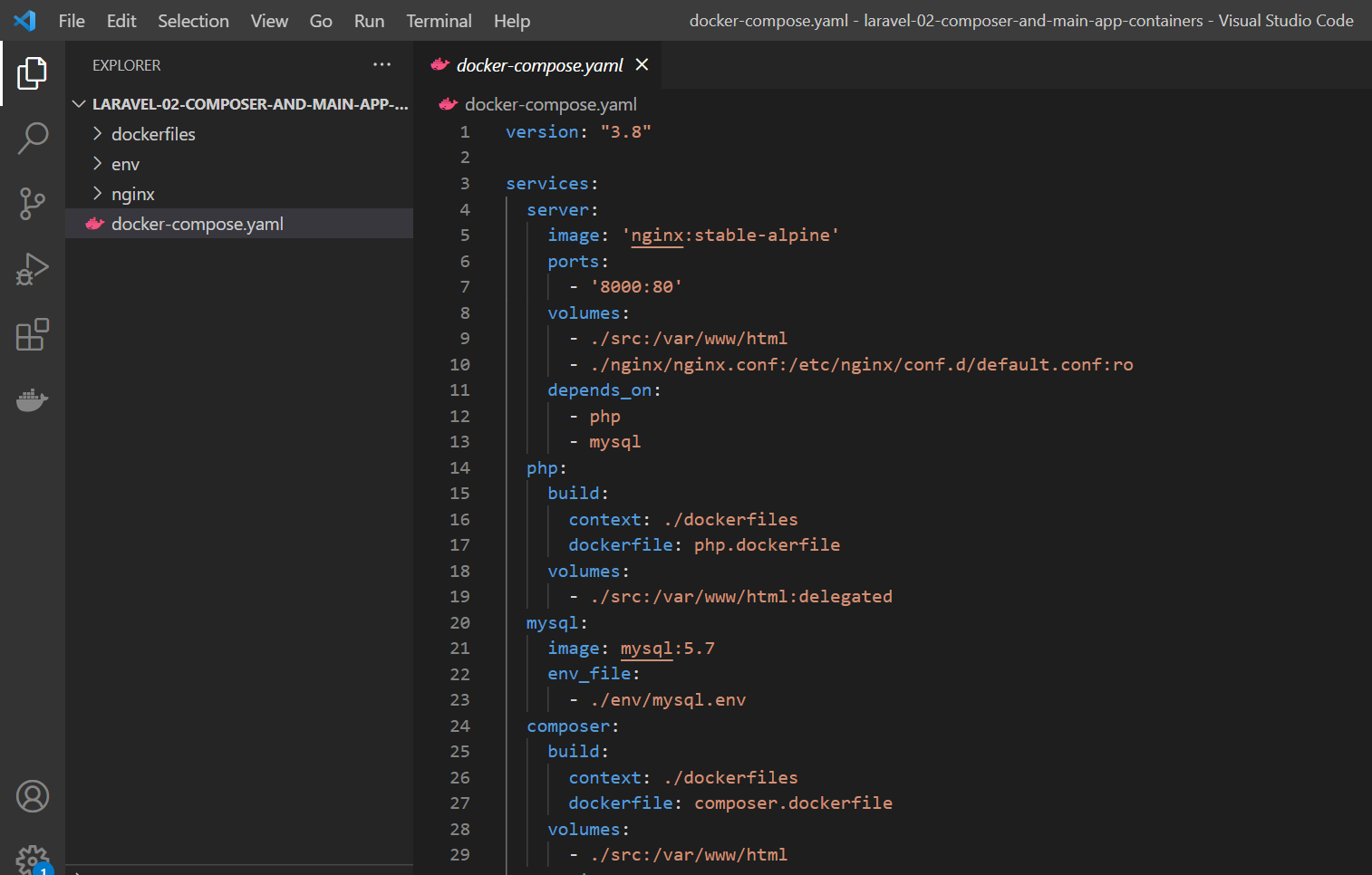


FROM composer:latest

WORKDIR /var/www/html

ENTRYPOINT [ "composer", "--ignore-platform-reqs" ]

Docker Compose file



version: "3.8"

services:

  server:

    image: 'nginx:stable-alpine'

    ports:

      - '8000:80'

    volumes:

      - ./src:/var/www/html

      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf:ro

    depends\_on:

      - php

      - mysql

  php:

    build:

      context: ./dockerfiles

      dockerfile: php.dockerfile

    volumes:

      - ./src:/var/www/html:delegated

  mysql:

    image: mysql:5.7

    env\_file:

      - ./env/mysql.env

  composer:

    build:

      context: ./dockerfiles

      dockerfile: composer.dockerfile

    volumes:

      - ./src:/var/www/html

**SECTION 8: Deploying Docker Containers:**