**Docker**

**By Devi Sri Charan**

[Docker 2](#_Toc1652768975)

[Dockerfile Creation 3](#_Toc726999648)

[Dockerfile for React application (Frontend) 3](#_Toc716955931)

[Dockerfile for NodeJS application (Backend) 3](#_Toc1122514238)

[Build and Run Container 3](#_Toc1266366246)

[Creating Image (Frontend) 3](#_Toc175455907)

[Create Container 4](#_Toc258069694)

[Creating Image (Backend) 5](#_Toc276257247)

[Creating Container (Backend) 6](#_Toc956337166)

[Volume Mount 7](#_Toc1699947804)

[Create Volume (Frontend) 7](#_Toc1901777534)

[Create Volume (Backend) 8](#_Toc362388798)

[Port Expose and Publishing 9](#_Toc299150600)

[Exposing Ports 9](#_Toc2043998638)

[Publishing Ports 9](#_Toc267153810)

[Multi-Container Setup with Docker-Compose 9](#_Toc233493592)

[Explanation 10](#_Toc363829653)

# Docker

## Dockerfile Creation

### Dockerfile for React application (Frontend)

FROM node:20-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 5173

CMD npm run dev

### Dockerfile for NodeJS application (Backend)

FROM node:20-alpine

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

COPY .env .env

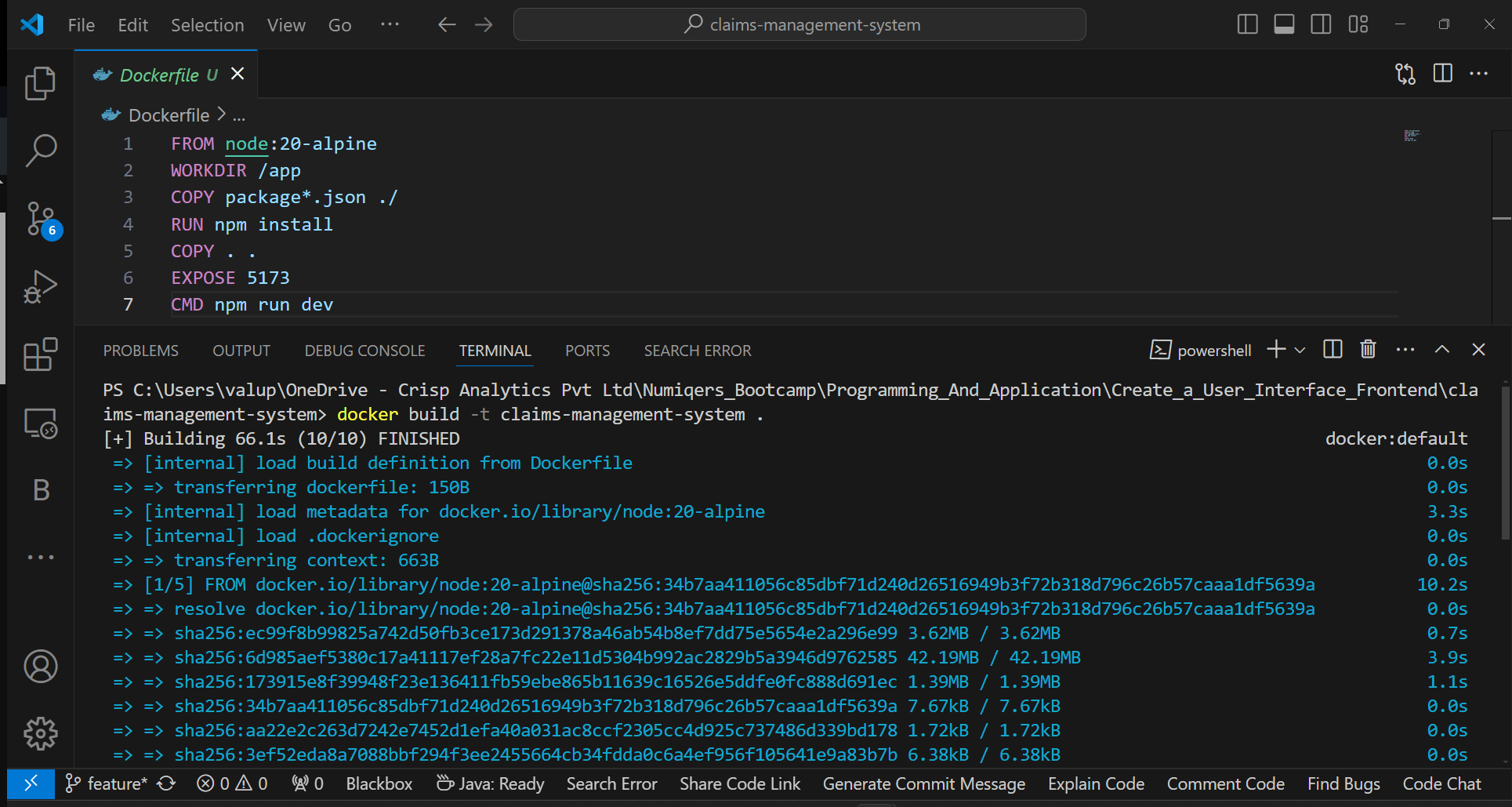
EXPOSE 3000

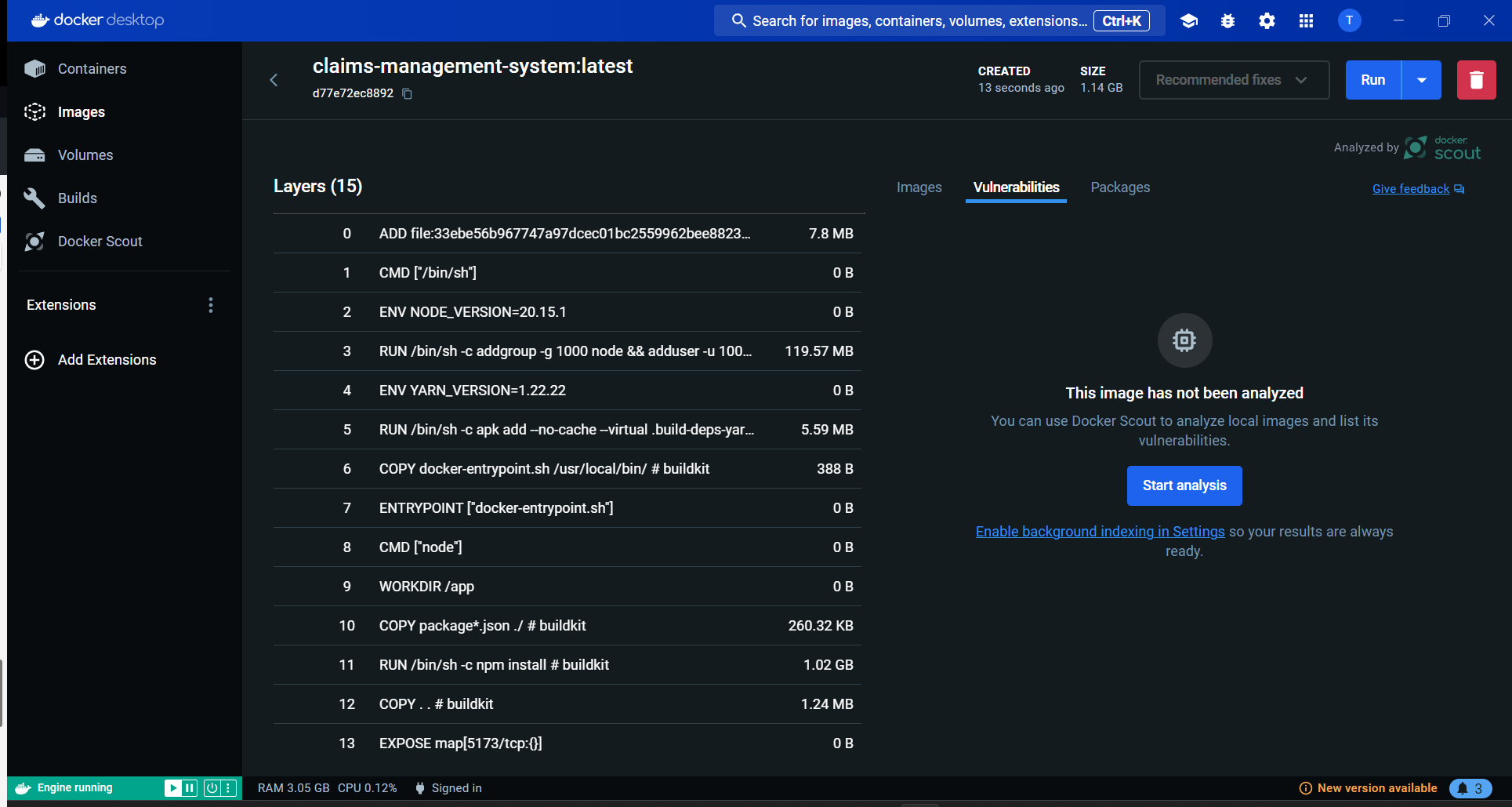
CMD npm start

## Build and Run Container

### Creating Image (Frontend)

To build the image, we must run the following command in the terminal.

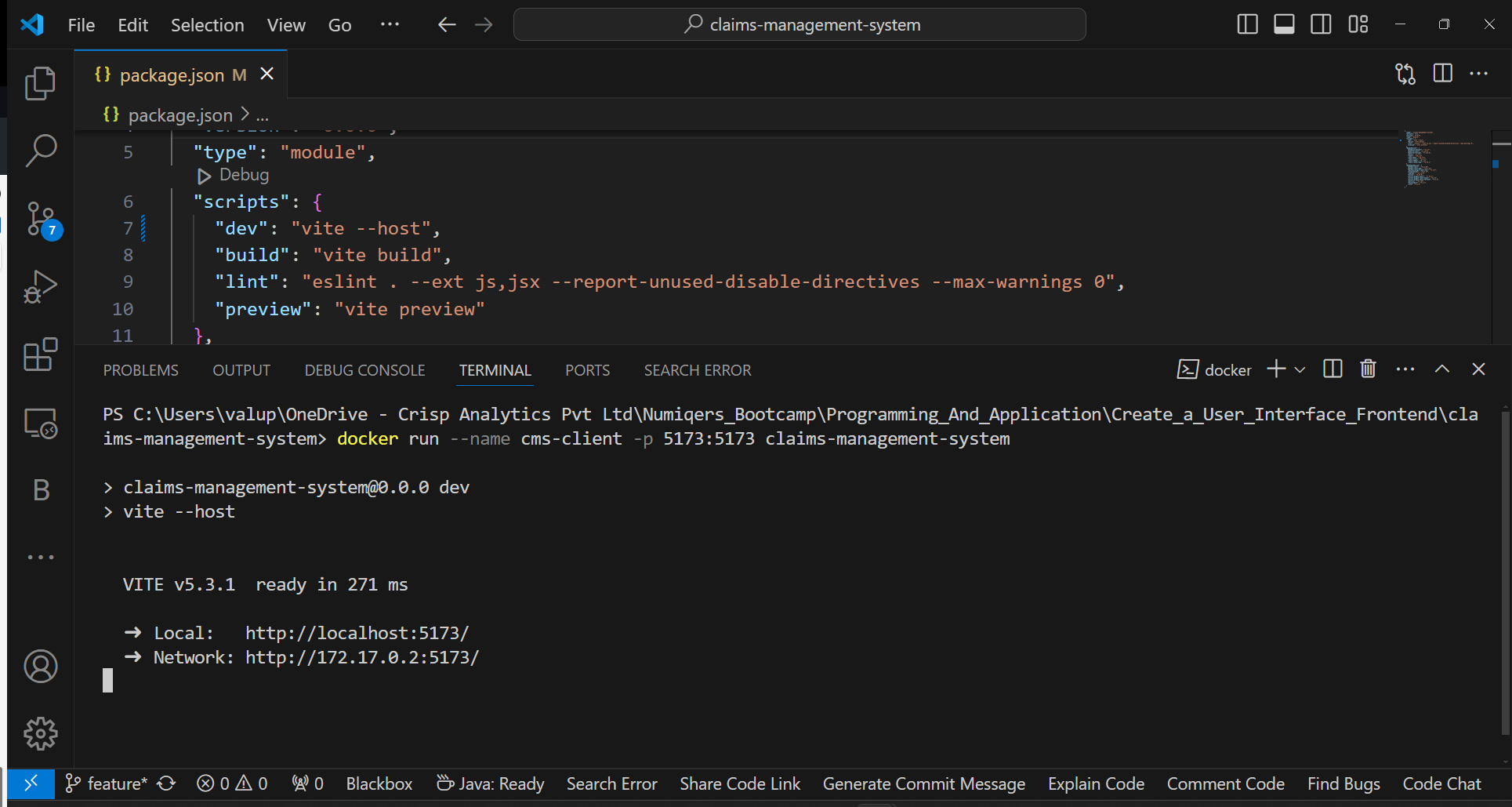
`docker build -t claims-management-system .`

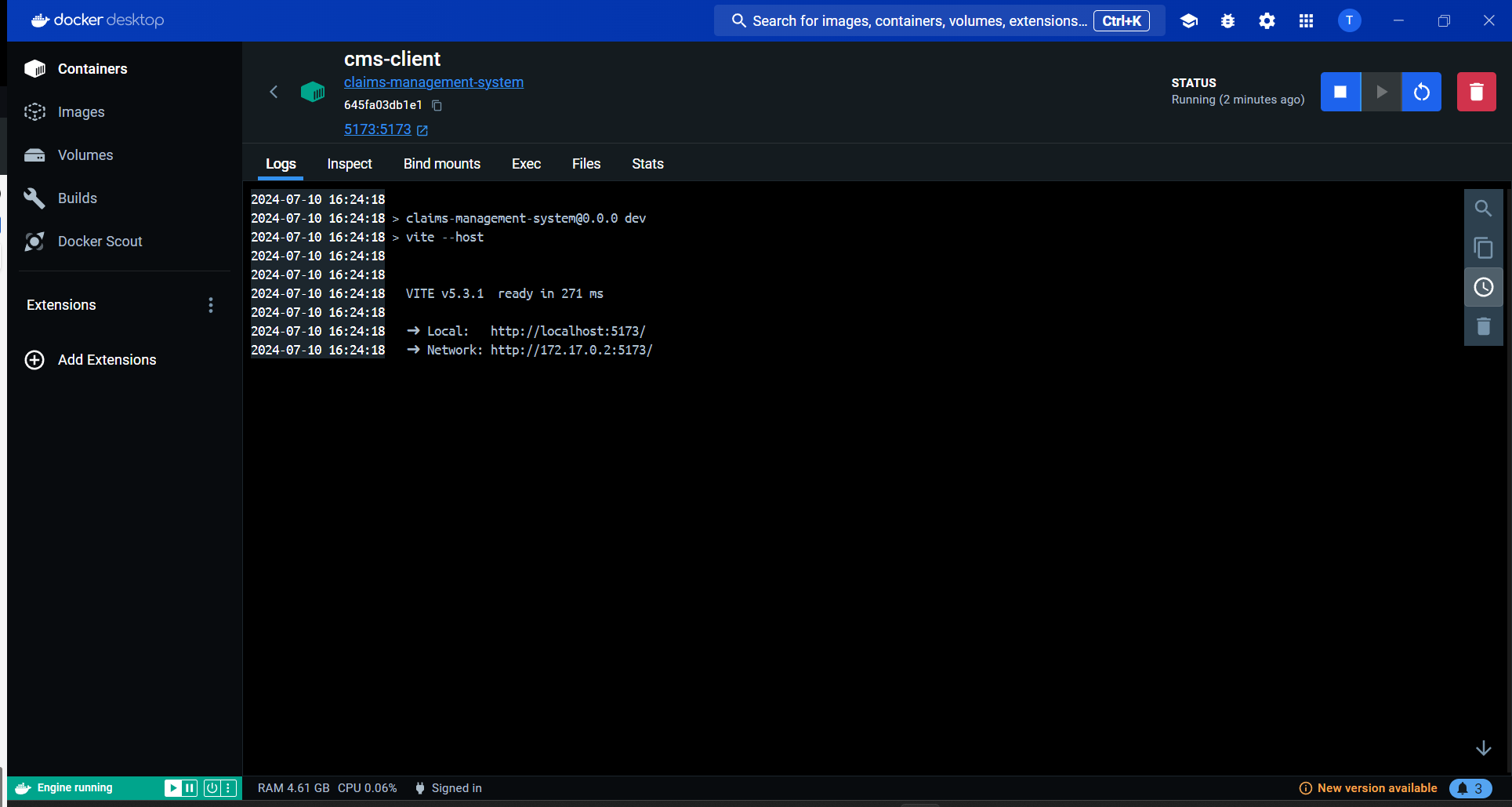


### Create Container

To build the container to run the image, we must run the following command in the terminal.

` docker run --name cms-client -p 5173:5173 claims-management-system`

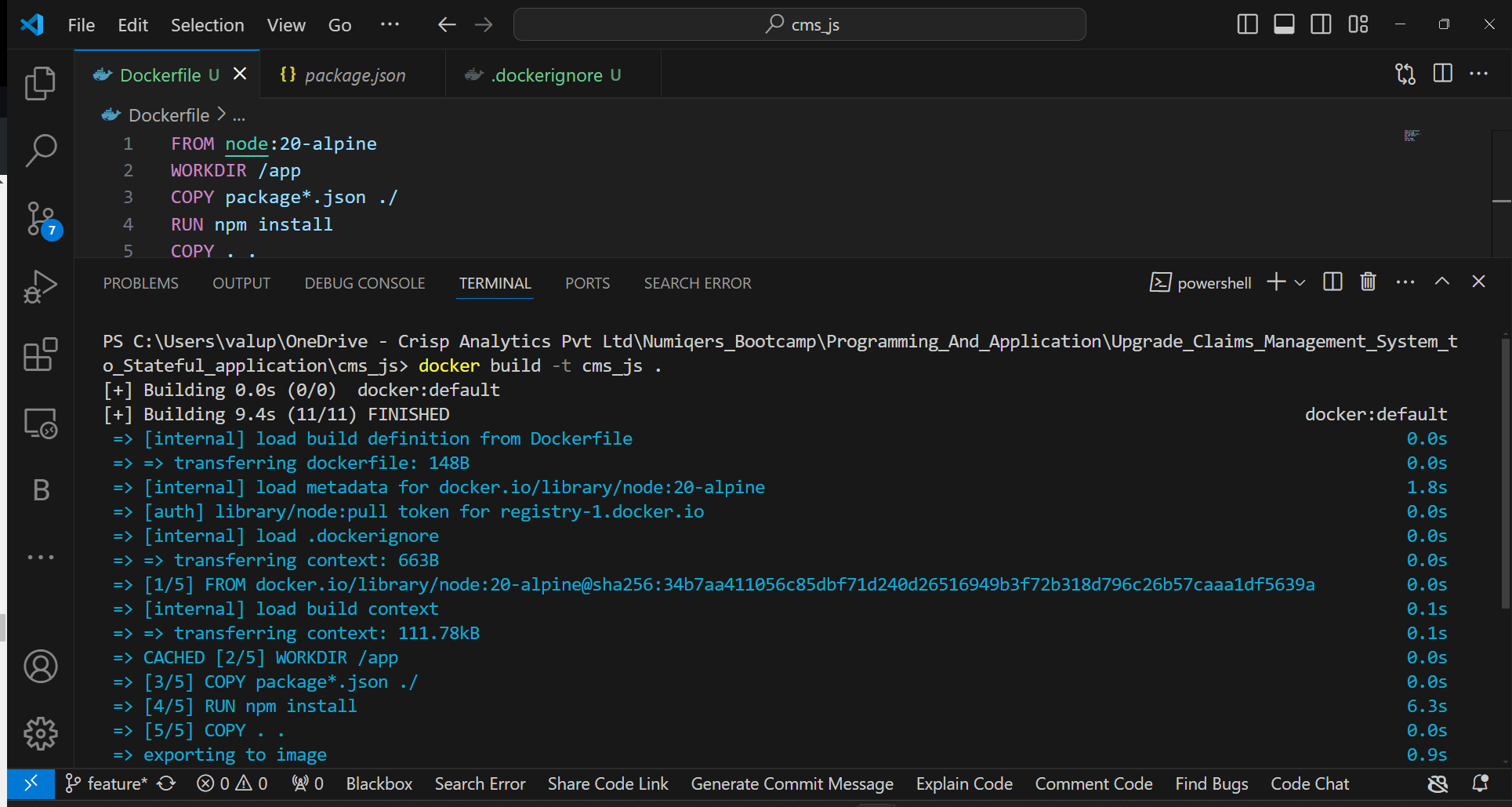


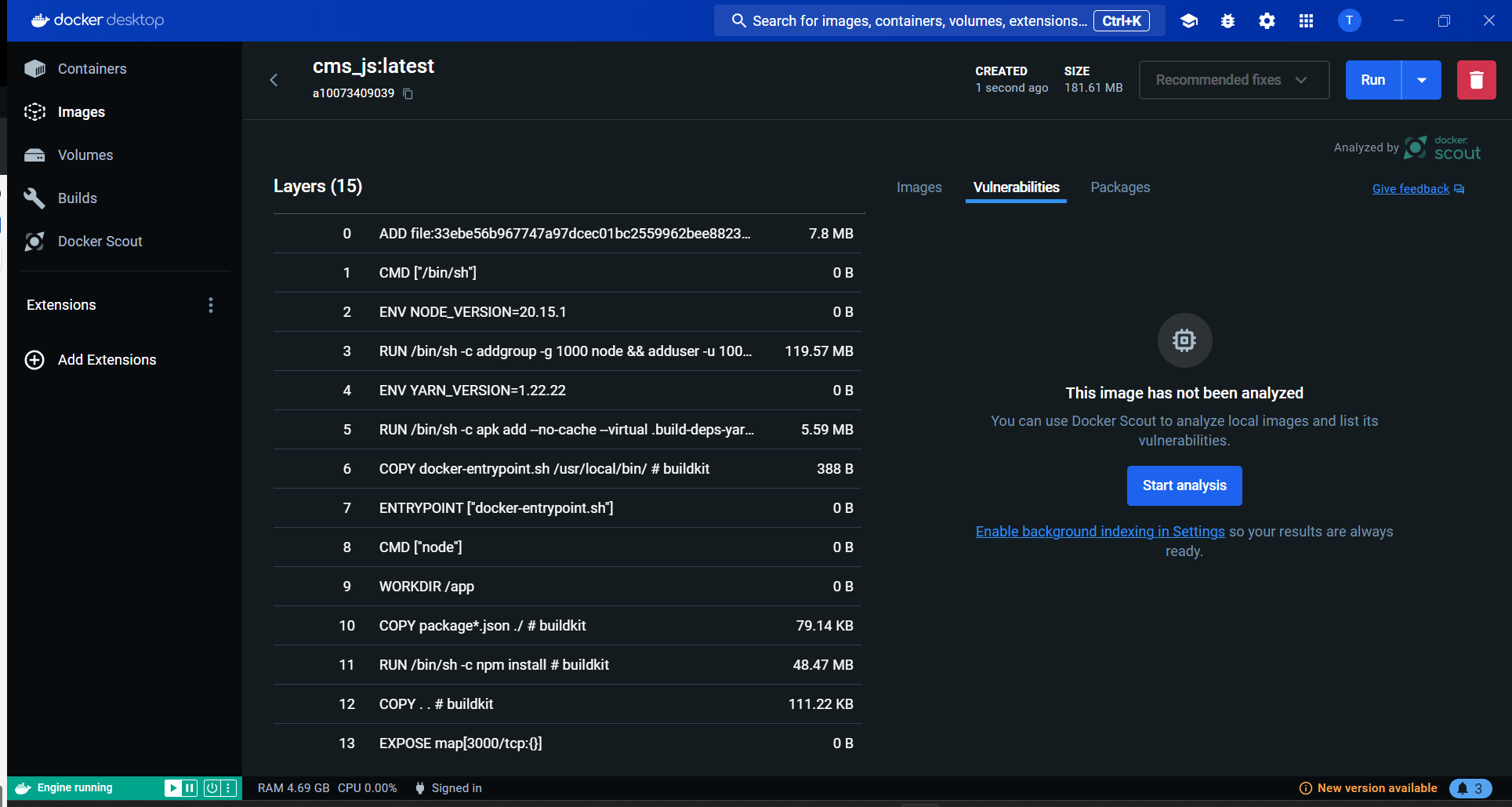


### Creating Image (Backend)

To build the image, we must run the following command in the terminal.

`docker build -t cms\_js .`

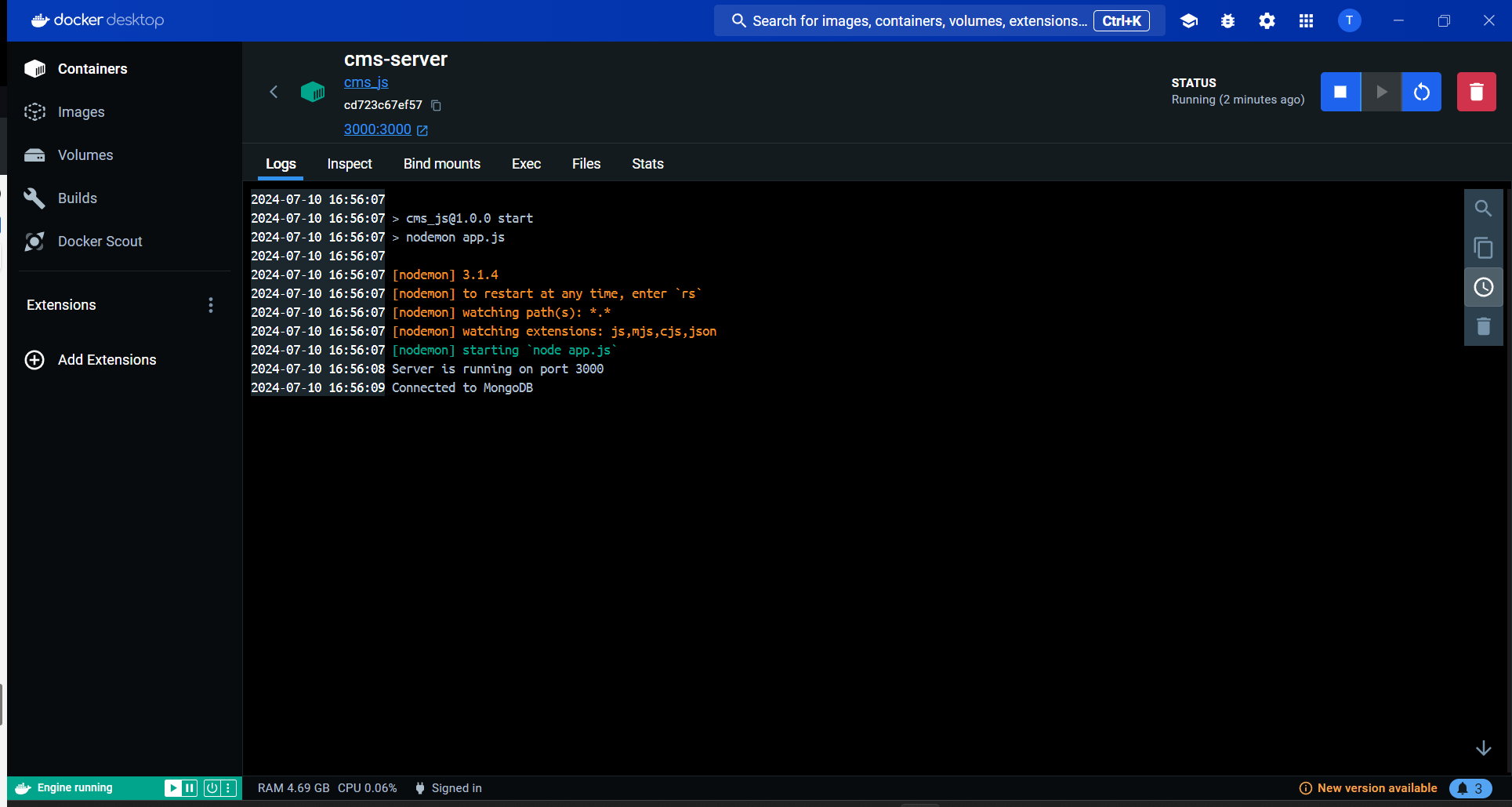
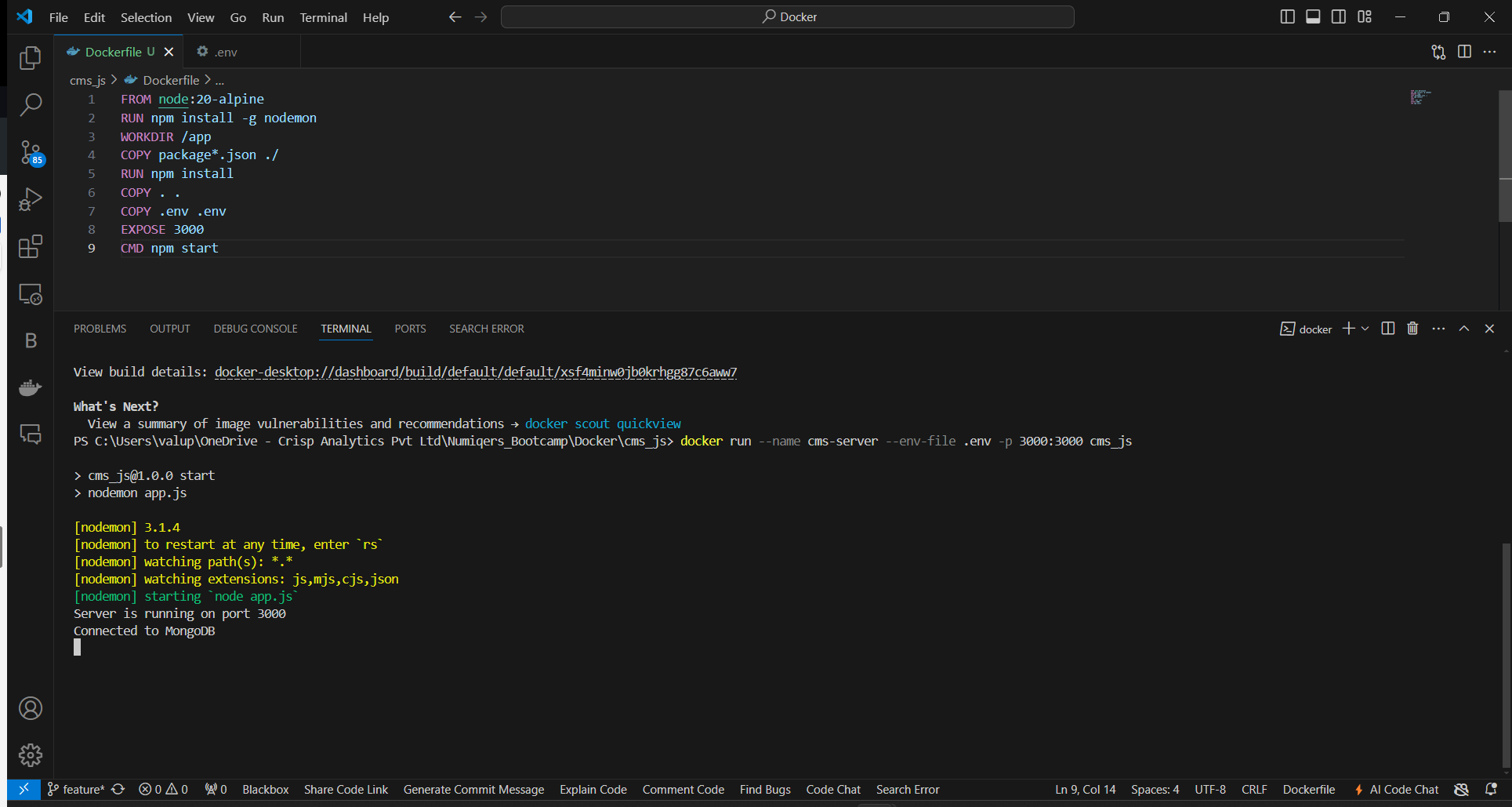




### Creating Container (Backend)

To build the container to run the image, we must run the following command in the terminal.

` docker run --name cms-server --env-file .env -p 3000:3000 cms\_js`



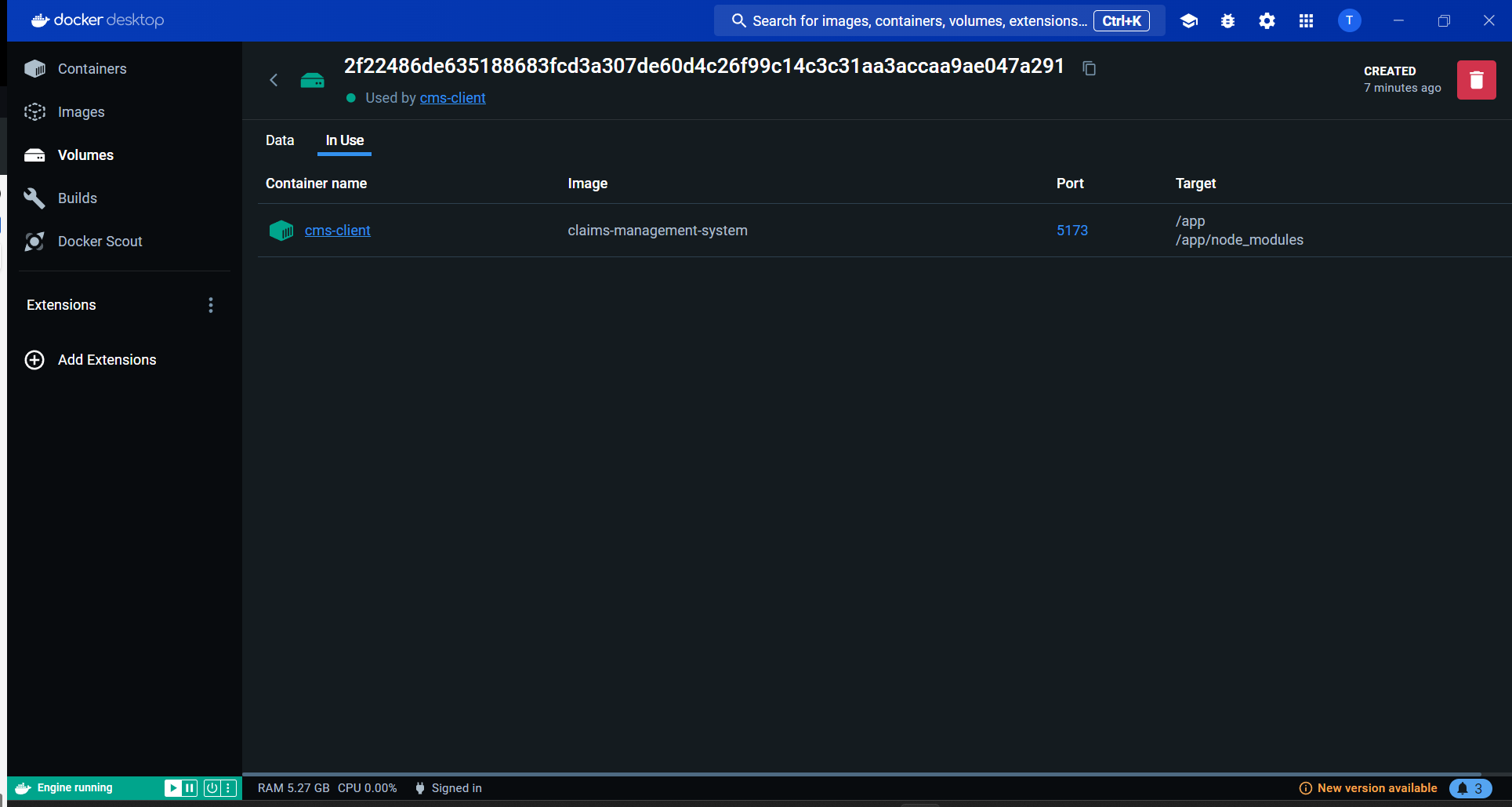
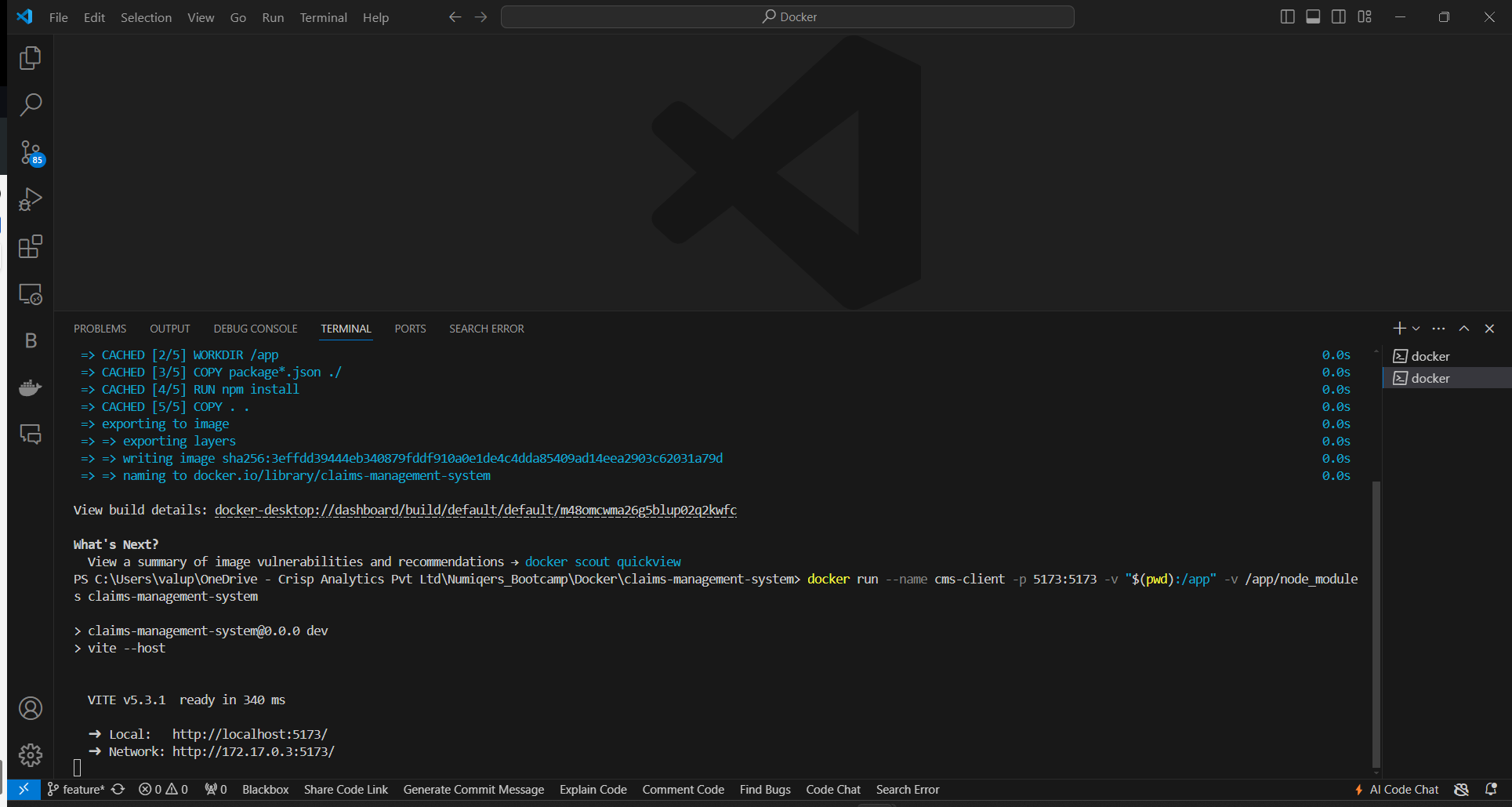
## Volume Mount

We can use Volumes to store data (files, etc.) permanently and map the volumes to containers. As a result, any changes we make will be reflected in the container.

### Create Volume (Frontend)

To create a volume to persist data outside of the container, we must run the following command in the terminal.

`docker run --name cms-client -p 5173:5173 -v "$(pwd):/app" -v /app/node\_modules claims-management-system`



### Create Volume (Backend)

To create a volume to persist data outside of the container, we must run the following command in the terminal.

docker run --name cms-server --env-file .env -p 3000:3000 -v "$(pwd):/app" -v /app/node\_modules cms\_js

## Port Expose and Publishing

### Exposing Ports

In Docker, exposing a port is typically done in the Dockerfile using the EXPOSE instruction:

Dockerfile

EXPOSE 3000

### Publishing Ports

To publish a port, use the -p flag with the docker run command:

docker run -p <host\_port>:<container\_port> <image\_name>

Example:

docker run -p 5173:5173 client

This command maps port 5173 on the host machine to port 5173 inside the container.

## Multi-Container Setup with Docker-Compose

Create a file called compose.yml in the project folder. This is the folder structure:

.   
└── my-project/   
├── client/   
├── server/   
└── compose.yml

services:

client:

build: ./client

container\_name: clientapp

ports:

- 5173:5173

volumes:

- ./client:/app

- /app/node\_modules

depends\_on:

- server

develop:

watch:

- path: ./client/package.json

action: rebuild

- path: ./client/package-lock.json

action: rebuild

- path: ./client

target: /app

action: sync

server:

build: ./server

container\_name: serverapp

ports:

- 3000:3000

volumes:

- ./server:/app

- /app/node\_modules

environment:

- MONGODB\_URL=mongodb+srv://charan522003:BdN6yRGAv1kQdz0h@cluster0.o4fmxl4.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

- JWT\_SECRET=lumiqcorp

- PORT=3000

develop:

watch:

- path: ./server/package.json

action: rebuild

- path: ./server/package-lock.json

action: rebuild

- path: ./server

target: /app

action: sync

### Explanation

1. **Services**: Defines the services (containers) that will be run.
   * **client** and **server**: These are the names of the services.
   * **build**: Specifies the build context and Dockerfile for the service.
   * **container\_name**: Names the container.
   * **ports**: Maps the host machine ports to the container ports (e.g., "5173:5173").
   * **volumes**: Mounts the host directories into the container.
   * **depends\_on**: Specifies the service dependencies.

To run the containers, I used the following command:

docker compose up --build

This command will build the Docker images and start the containers for both the frontend and backend services.

