**Full Completed Docker Note**

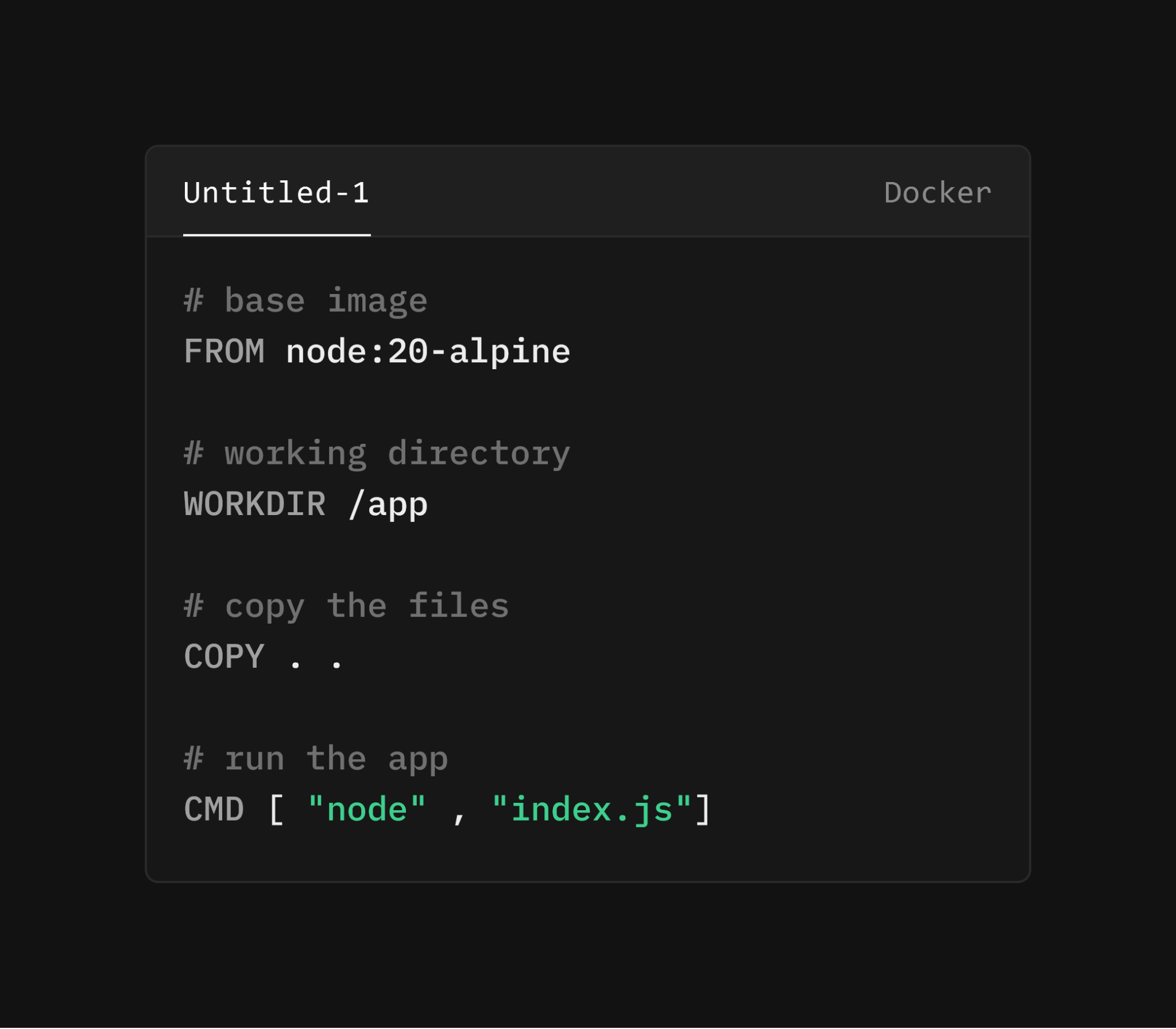
By Adomic Arts [HGS Chandeepa](mailto:saminchandeepa@gmail.com) [adomicarts@gmail.com](mailto:adomicarts@gmail.com)

**🔗Full Tutorial :**[**https://www.youtube.com/watch?v=qtKK7qOGMuE**](https://www.youtube.com/watch?v=qtKK7qOGMuE)

Welcome to the Docker Tutorials and Projects repository! This repository contains code and resources related to a comprehensive Docker tutorial video. The video covers all the essential areas you need to know before using Docker, including hands-on projects to solidify your understanding.

**Dockerize a Simple application**

**Dockerfile**

****

**Commands**

1.Create an docker image using the Dockerfile

*docker build -t image\_name .*

2.To see all the images

*docker images*

3.To create a container from that image

*docker run –name container\_name image\_name*

*docker run –name container\_name -d image\_name (to run in detached mode)*

4.To see all the running services(containers)

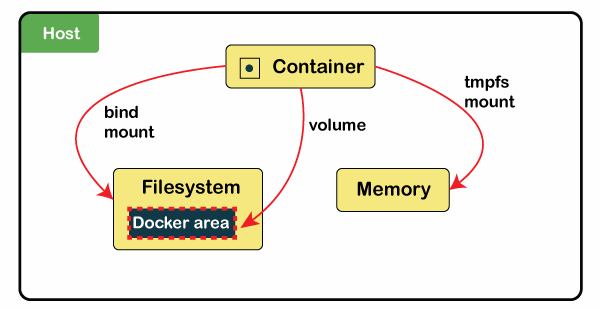
*docker ps*

5.To see all the containers (running ones or not)

*docker ps -a*

**Docker Volumes**

Docker volumes are a mechanism for persisting data generated and used by Docker containers. Unlike bind mounts, which link a directory on the host machine to a directory in the container, volumes are managed by Docker itself and can be stored in a location on the host filesystem that Docker chooses.



### **Types of Docker Volumes**

1. Anonymous Volumes: Created and managed by Docker but not given a specific name. These volumes are typically used for temporary data that doesn't need to be persisted beyond the container's lifecycle.

docker run -d --name my\_container -v /app/data my\_image

1. Named Volumes: Created with a specific name and can be referenced by multiple containers. Named volumes are useful for persisting data that needs to be shared between containers or across container restarts.

docker run -d --name my\_container -v my\_volume:/app/data my\_image

1. Host Volumes (Bind Mounts): Directly map a directory or file on the host to a directory or file in the container. Unlike managed volumes, the host determines where the data is stored. Bind mounts provide more control but less isolation from the host system.

*docker run -d --name my\_container -v /path/on/host:/path/in/container my\_image*

**Commands**

1.Create a docker volume

*docker volume create my\_volume*

2.Run a docker container with a volume

*docker run -d --name my\_container -v my\_volume:/app/data my\_image*

*docker run --name container\_name --rm -v /app/node\_modules -v ${PWD}:/app image\_name*

3.List all docker volumes

*docker volume ls*

4.Inspect docker volumes

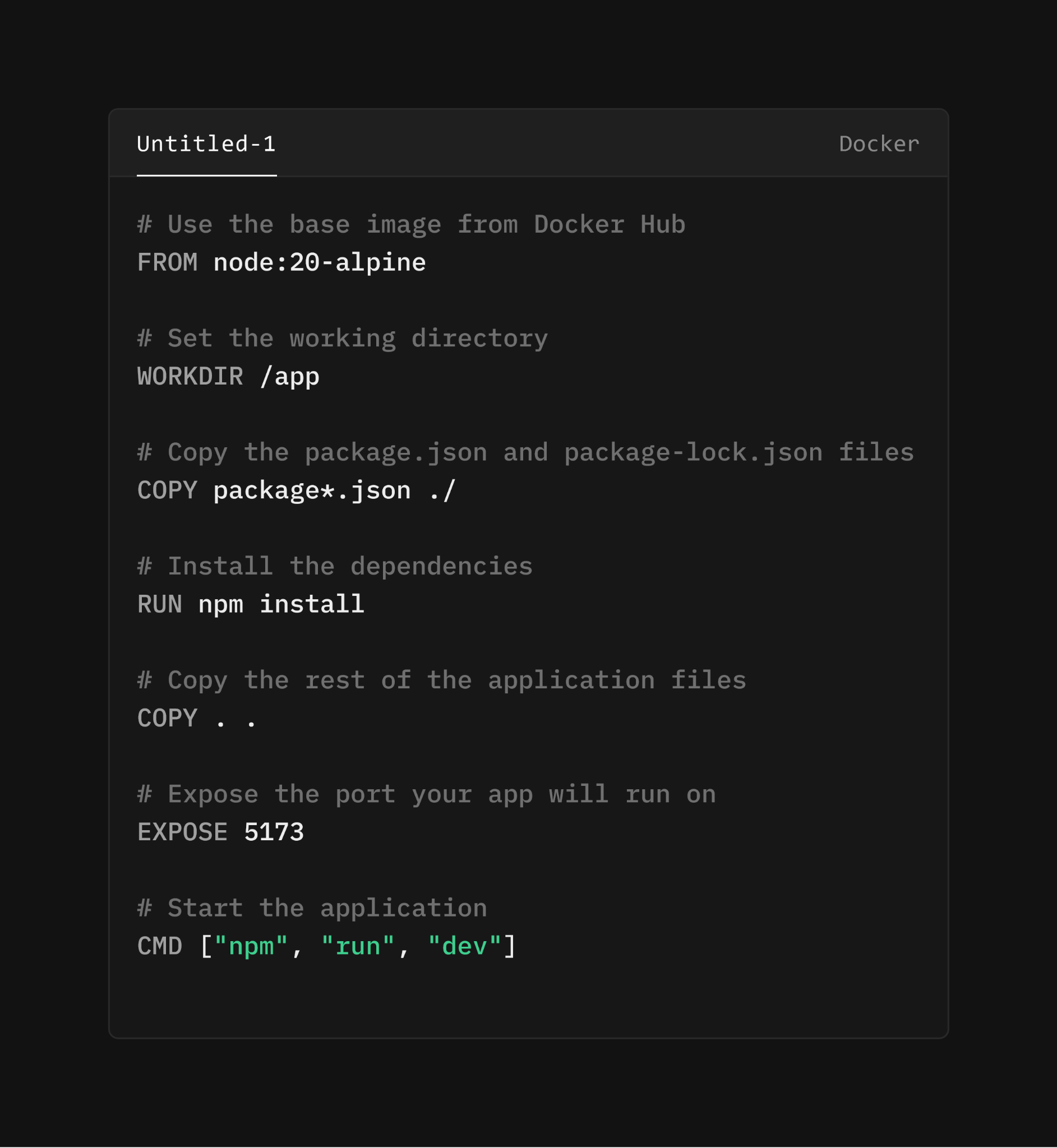
*docker volume inspect my\_volume*

5.Remove

*docker volume rm my\_volume*

**Dockerize a React application**

**Dockerfile**



**Dockerignore**

In Docker, the .dockerignore file is used to specify which files and directories should be excluded from the Docker build context. When you build a Docker image, Docker uses a "build context" which includes all files and directories in the current directory (where your Dockerfile resides) and its subdirectories. The .dockerignore file helps optimize the build process by preventing unnecessary or sensitive files from being sent to the Docker daemon as part of the build context.

**Commands**

1.Create an docker image using the Dockerfile

*docker build -t image\_name .*

2.To see all the images

*docker images*

3.To create a container from that image

*docker run –name container\_name -p 3000:5173 image\_name*

*docker run –name container\_name -p 3000:5173 -d image\_name (to run in detached mode)*

4.To see all the running services(containers)

*docker ps*

5.To see all the containers (running ones or not)

*docker ps -a*

6.To stop a running container

*docker stop container\_name*

*or*

*docker stop container\_id*

7.To restart a docker container

*Docker start container\_name*

*(here we don't need to re configure the port mappings as we did that earlier)*

‘

8.Remove a docker container

*docker ps -a*

*docker container rm CONTAINER\_ID\_OR\_NAME*

*Remove multiple containers*

*docker container rm CONTAINER\_ID\_OR\_NAME CONTAINER\_ID\_OR\_NAME*

9.Remove a docker image

*docker images*

*docker image rm IMAGE\_ID\_OR\_TAG*

10.Remove all containers all images and all volumes

*docker system prune -a*

11.Docker Volumes

This Docker run command sets up a container (vite\_container) based on the vite-app image, with port mappings, volume mounts, environment variable settings, and automatic removal (--rm) when the container stops. It's configured to facilitate local development with live code updates (${PWD}:/app), efficient node\_modules management (/app/node\_modules), and reliable file change detection (CHOKIDAR\_USEPOLLING=true). Adjustments can be made based on specific project requirements or environment configurations.

*docker run --name container\_name -p 3000:5173 --rm -v /app/node\_modules -v ${PWD}:/app -e CHOKIDAR\_USEPOLLING=true image\_name*

The --rm flag in the docker run command stands for "remove". When you use --rm, Docker automatically removes the container and its filesystem when the container exits (stops running).

**Versioning Images**

Here in docker we can manage the version by adding a tag to our images.

Add a tag to the docker image

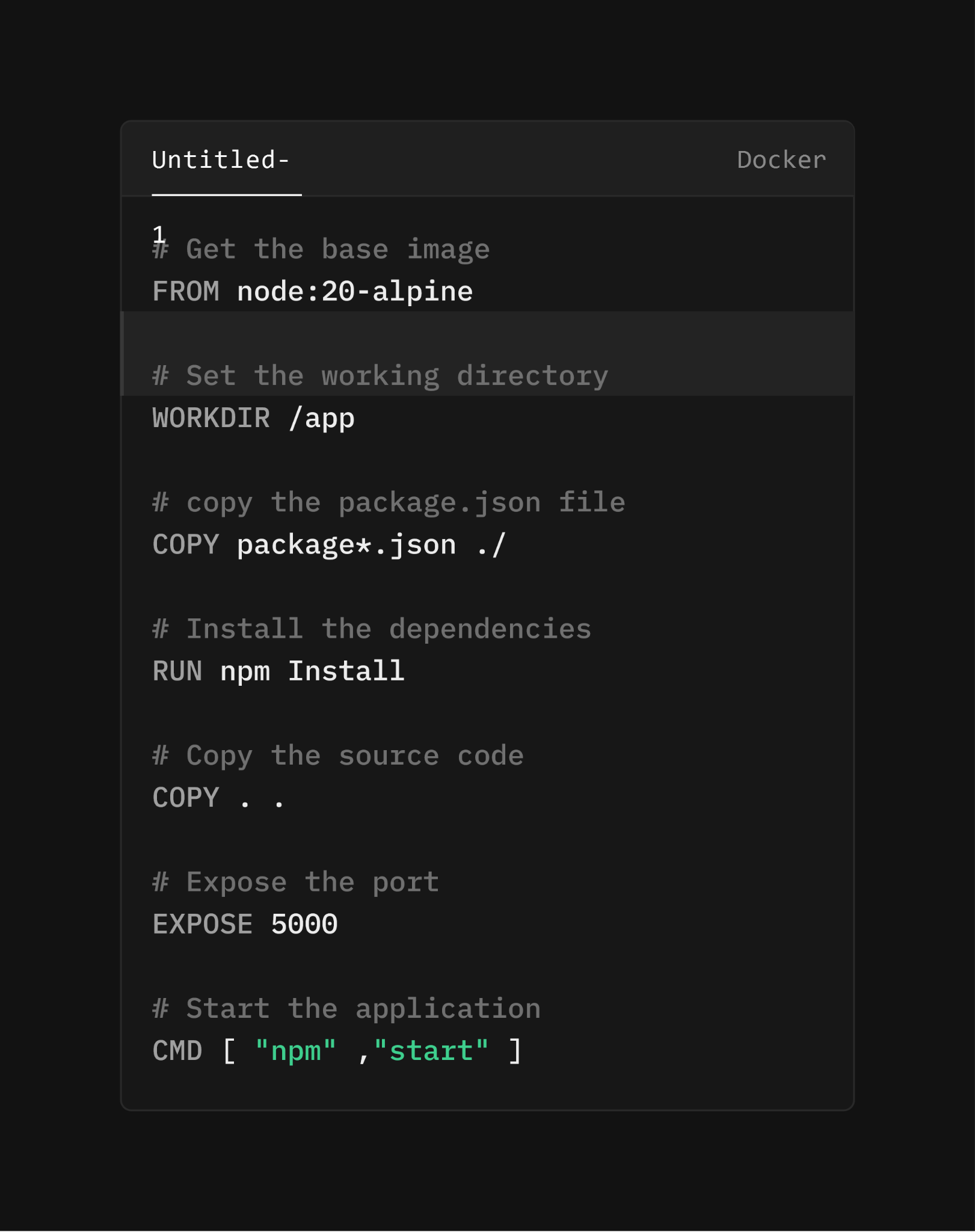
*docker build -t image\_name:tag .*

Create a container with the tag(specified version)

*Docker run –name container\_name -p 3000:4000 image\_name:tag*

**Dockerize a Node application**

**Dockerfile**

****

**Dockerignore**

In Docker, the .dockerignore file is used to specify which files and directories should be excluded from the Docker build context. When you build a Docker image, Docker uses a "build context" which includes all files and directories in the current directory (where your Dockerfile resides) and its subdirectories. The .dockerignore file helps optimize the build process by preventing unnecessary or sensitive files from being sent to the Docker daemon as part of the build context.

**Commands**

1.Create an docker image using the Dockerfile

*docker build -t image\_name .*

2.To see all the images

*docker images*

3.To create a container from that image

*docker run –name container\_name -p 3000:5173 image\_name*

*docker run –name container\_name -p 3000:5173 -d image\_name (to run in detached mode)*

4.To see all the running services(containers)

*docker ps*

5.To see all the containers (running ones or not)

*docker ps -a*

6.To stop a running container

*docker stop container\_name*

*or*

*docker stop container\_id*

7.To restart a docker container

*Docker start container\_name*

*(here we don't need to re configure the port mappings as we did that earlier)*

‘

8.Remove a docker container

*docker ps -a*

*docker container rm CONTAINER\_ID\_OR\_NAME*

*Remove multiple containers*

*docker container rm CONTAINER\_ID\_OR\_NAME CONTAINER\_ID\_OR\_NAME*

9.Remove a docker image

*docker images*

*docker image rm IMAGE\_ID\_OR\_TAG*

10.Remove all containers all images and all volumes

*docker system prune -a*

**11.Docker Volumes**

This Docker run command sets up a container (vite\_container) based on the vite-app image, with port mappings, volume mounts, environment variable settings, and automatic removal (--rm) when the container stops. It's configured to facilitate local development with live code updates (${PWD}:/app), efficient node\_modules management (/app/node\_modules), and reliable file change detection (CHOKIDAR\_USEPOLLING=true). Adjustments can be made based on specific project requirements or environment configurations.

*docker run --name container\_name -p 5000:5000 --rm -v /app/node\_modules -v ${PWD}:/app image\_name*

The --rm flag in the docker run command stands for "remove". When you use --rm, Docker automatically removes the container and its filesystem when the container exits (stops running).

**Versioning Images**

Here in docker we can manage the version by adding a tag to our images.

Add a tag to the docker image

*docker build -t image\_name:tag .*

Create a container with the tag(specified version)

*Docker run –name container\_name -p 5000:5000 image\_name:tag*

**Push Images to Docker Hub**

Docker Hub is a cloud-based registry service provided by Docker, Inc. It allows you to store, manage, and distribute Docker images. Docker Hub serves as a central repository where you can find official Docker images, share your own images, and collaborate with others.

*docker login*

*docker tag image\_name dockerhub\_username/image\_name -app:tag*

*docker push dockerhub\_username/image\_name -app:tag*

**Dockerize a Full Stack Application(MERN)**

**Server Dockerfile**



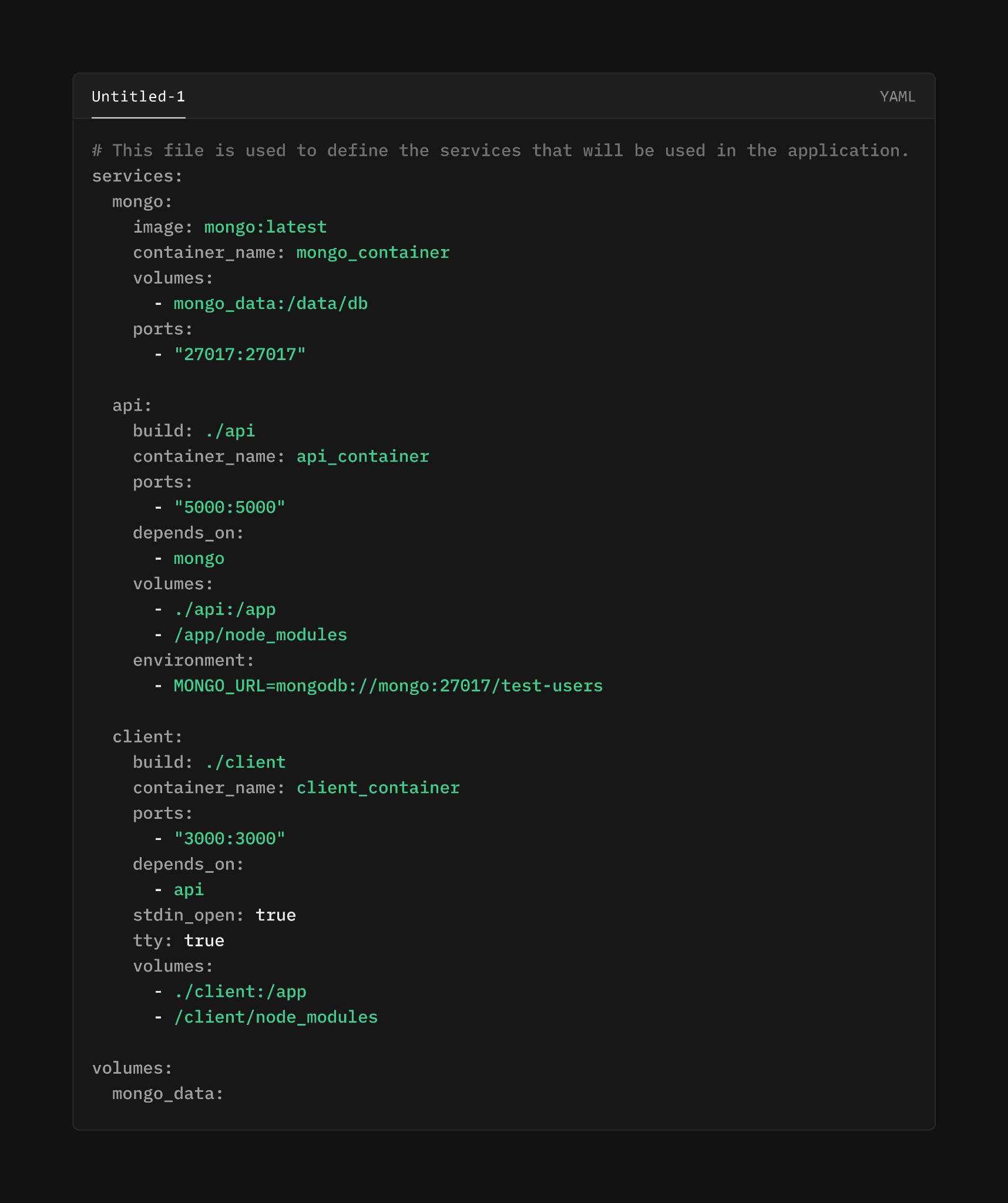
**Client Dockerfile**

****

**Docker Compose file**

•Docker Compose simplifies the process of defining and running multi-container Docker applications.

•By using a docker-compose.yml file, you can easily manage complex applications with multiple interconnected services, volumes, and networks.

****

The stdin\_open: true and tty: true options in Docker Compose are used to keep the container's standard input (stdin) open and to allocate a pseudo-TTY (a terminal) to the container. These options are particularly useful for containers where you need to interact with the shell or command line.

* stdin\_open: true: This keeps the standard input (stdin) open, even if not attached. It's useful for containers where you might want to keep an interactive session open.
* tty: true: This allocates a pseudo-TTY, which is a terminal interface. It enables features such as colored output and allows interactive commands to work as if they are run in a regular terminal.

-END-