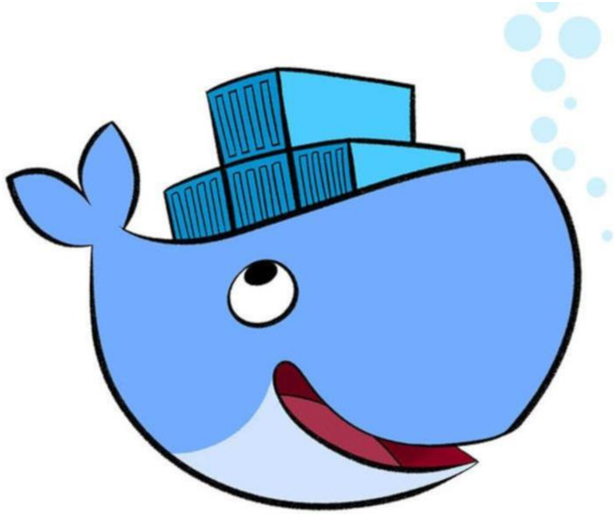
DOCKER

Alessandro Bocci
name.surname@unipi.it

Advanced Software Engineering (Lab)
17/10/2024

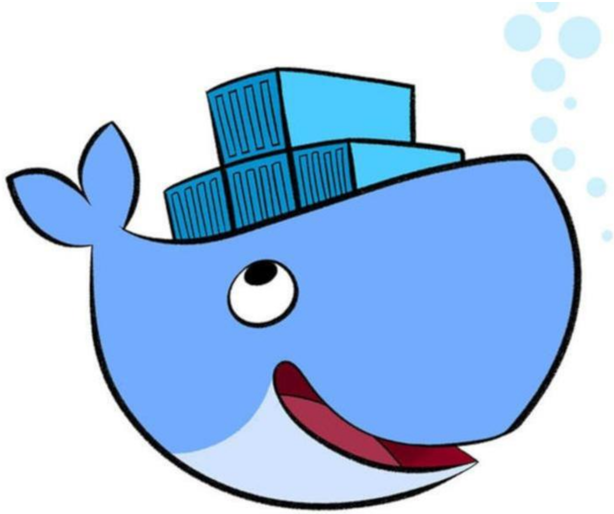
What will you do?

- Write `Dockerfile`(s) to create images to deploy your services.
- Learn the basic commands to manage containers.



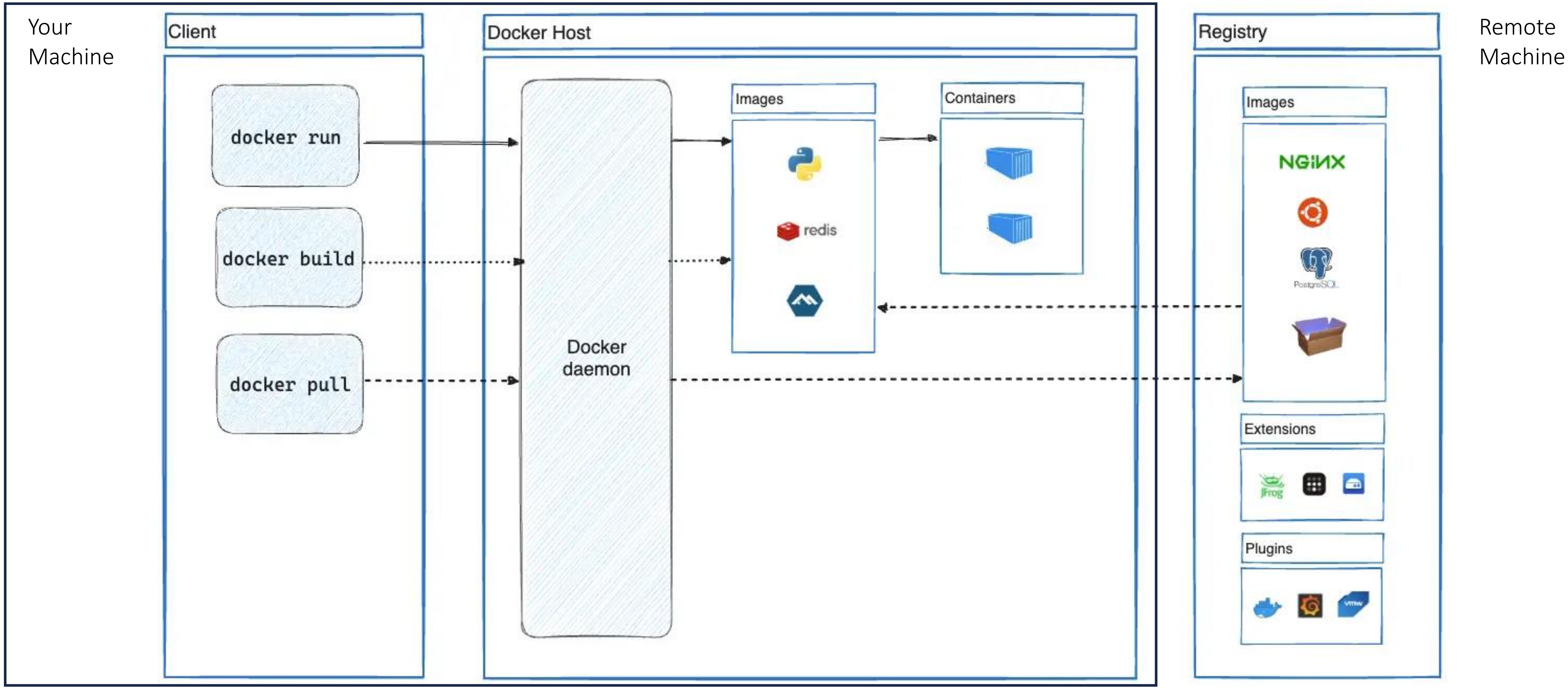
Software Prerequisites

- Docker Engine
- Docker image `python:3.12-slim`



What is Docker?

Docker is an open platform for developing, shipping, and running applications.



Your Machine

Client

docker run

docker build

docker pull

Docker Host

Docker daemon

Images

Containers

Registry

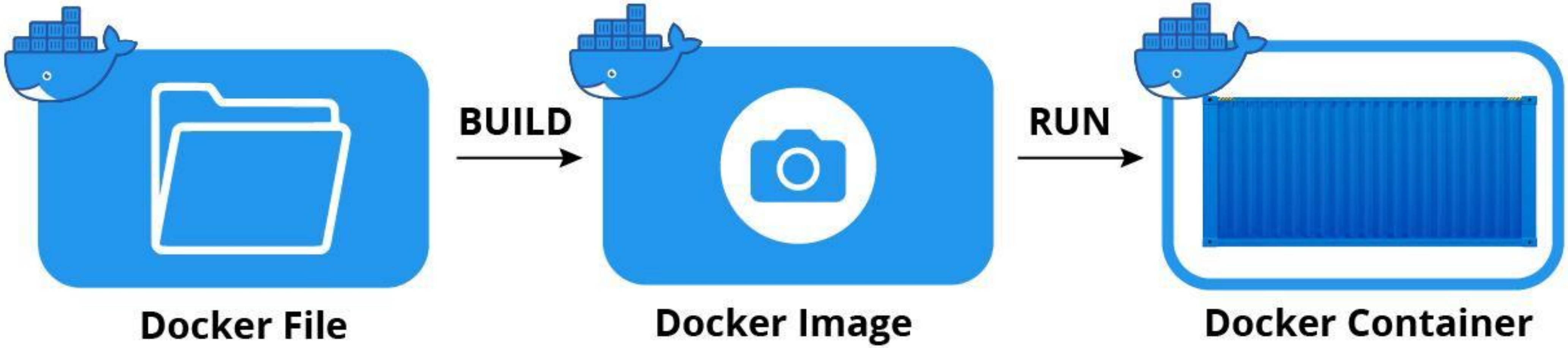
Images

Extensions

Plugins

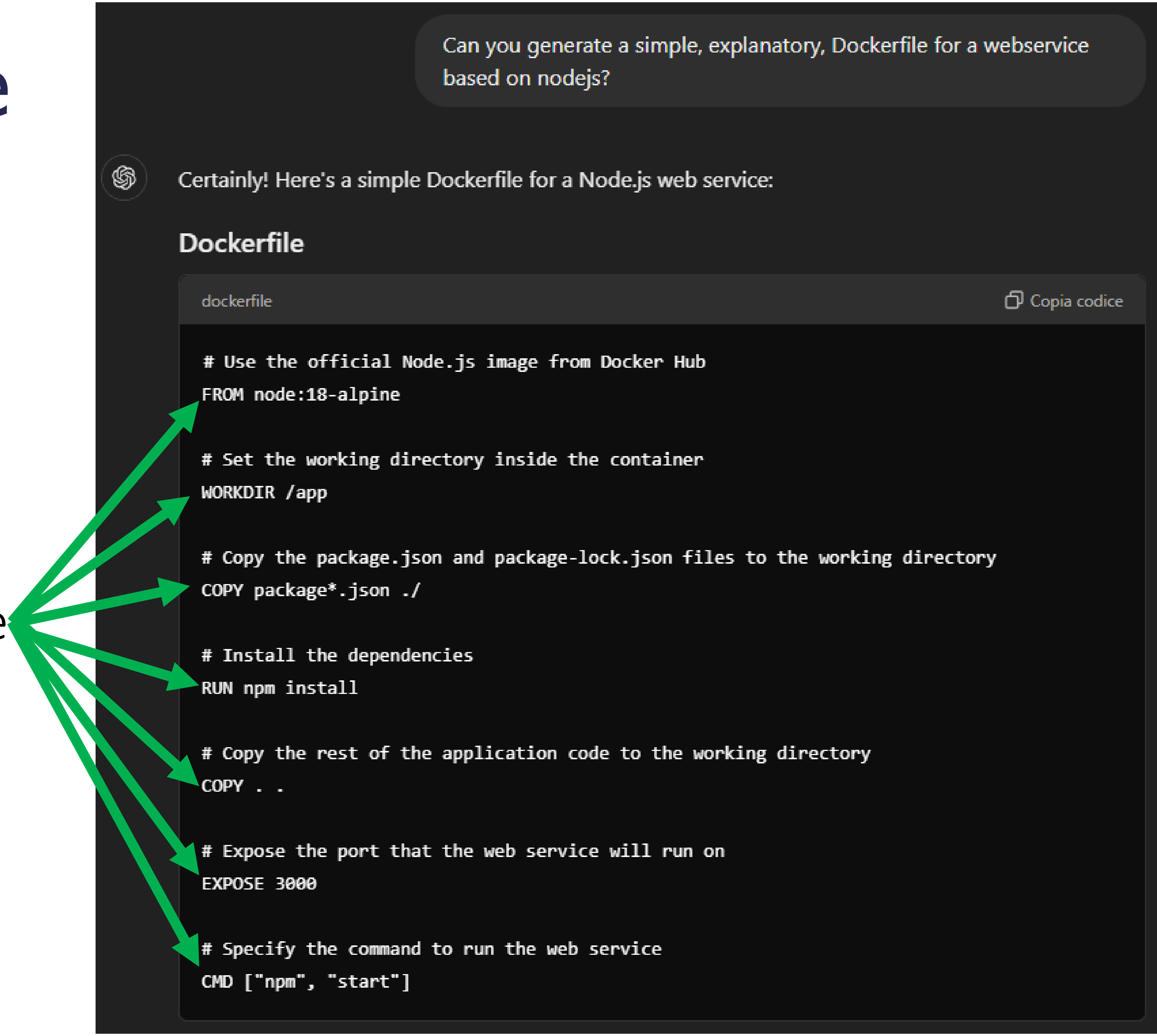
Remote Machine

Docker basic workflow



Dockerfile example

Commands executed in sequence when building the image



Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

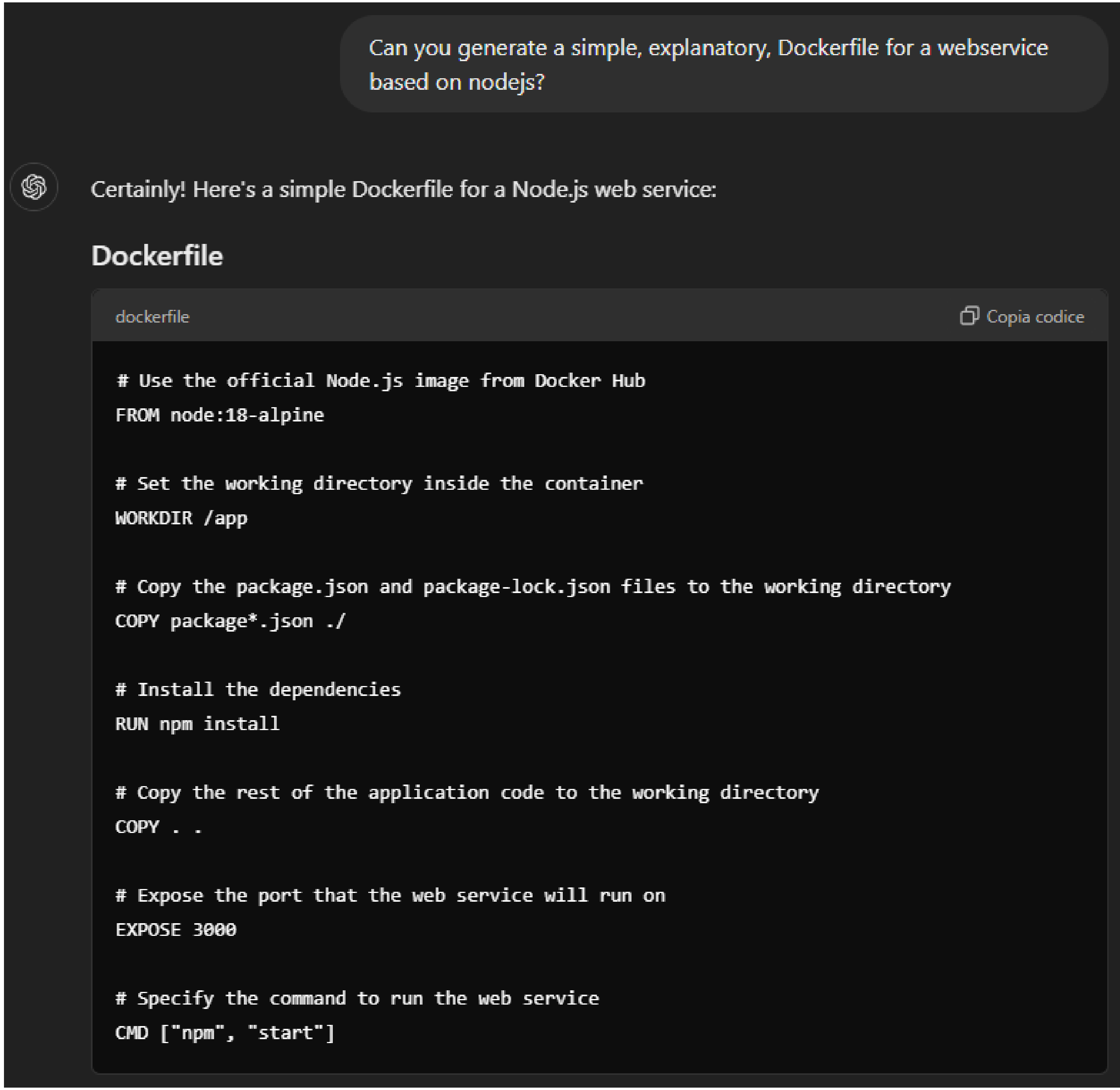
```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example

Local machine

Dockerfile
package.json
package-lock.json
server.js

Image



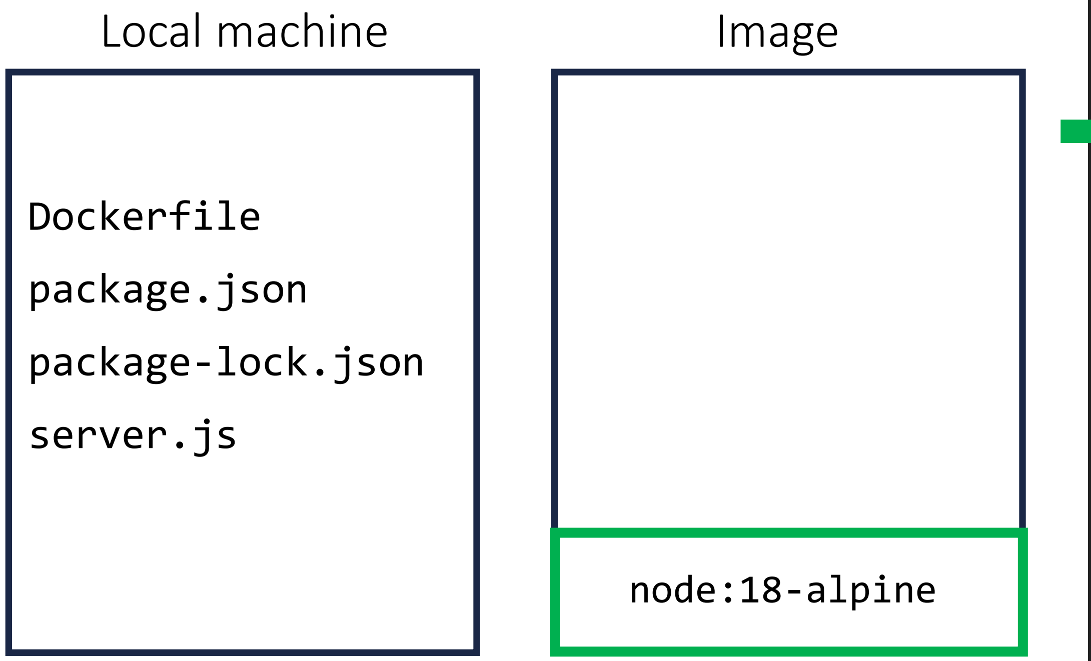
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example

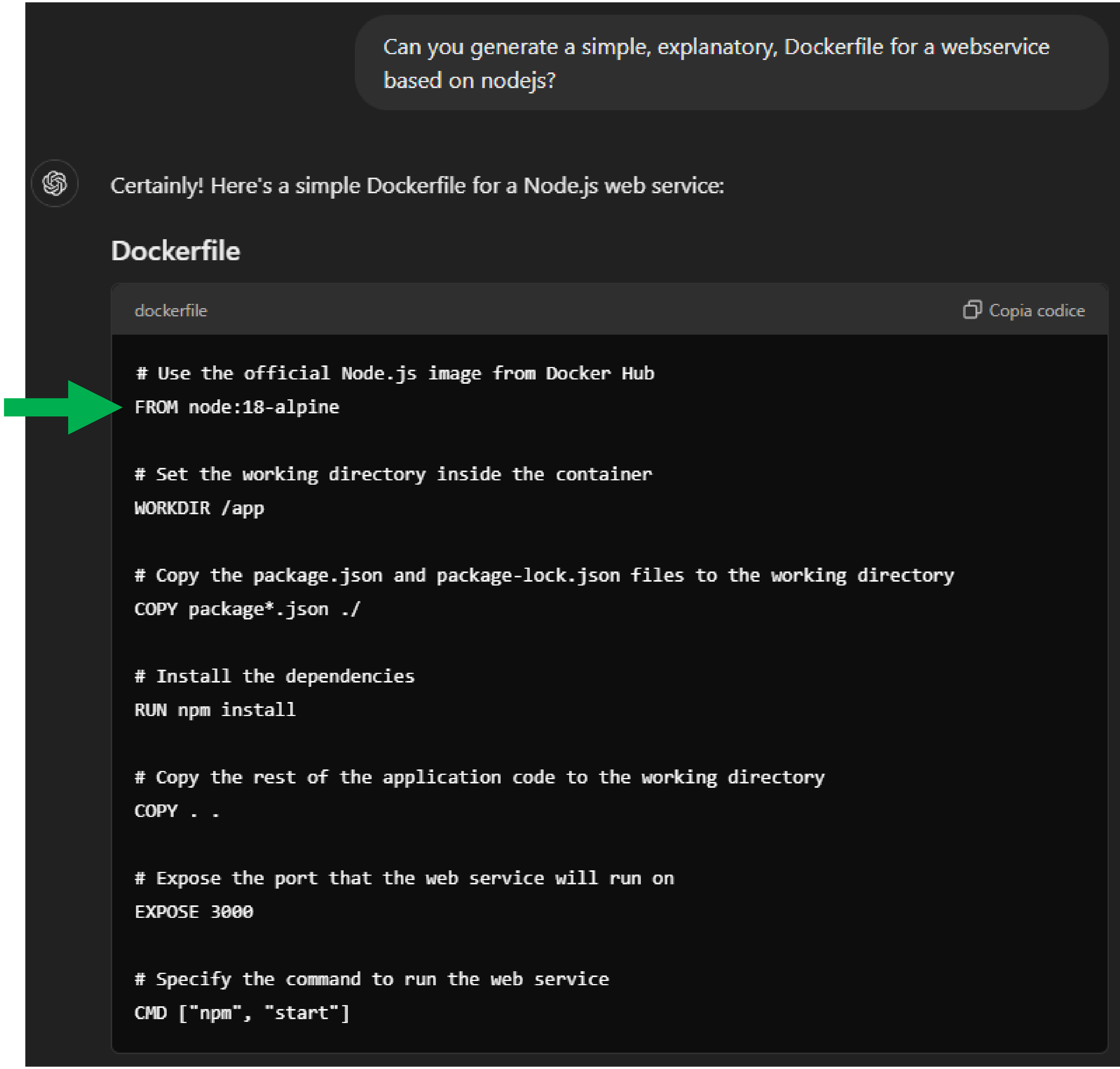


Local machine

Dockerfile
package.json
package-lock.json
server.js

Image

node:18-alpine



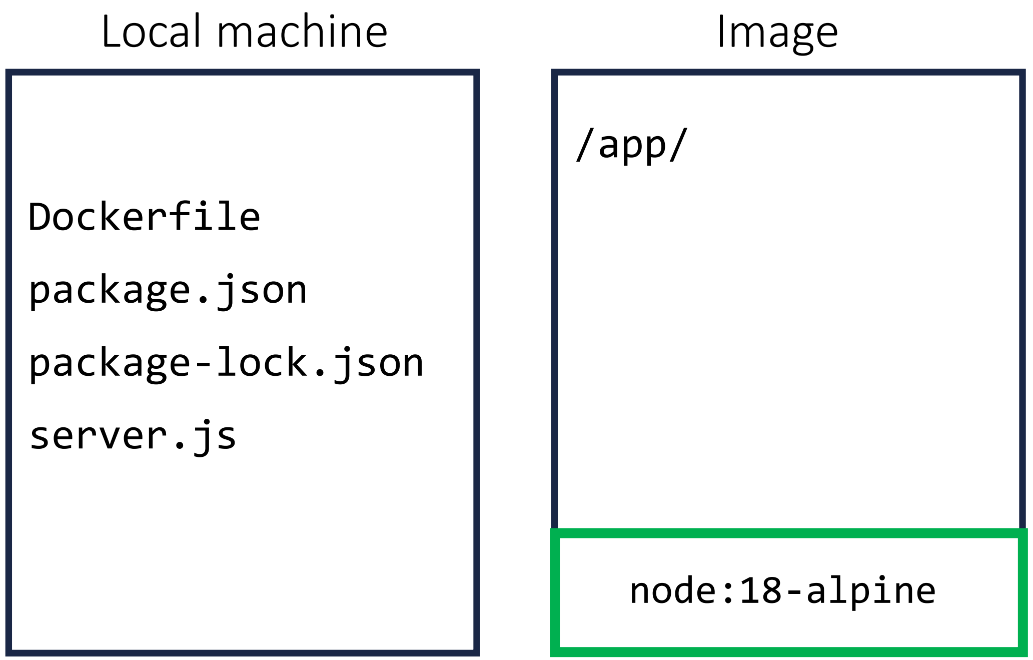
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



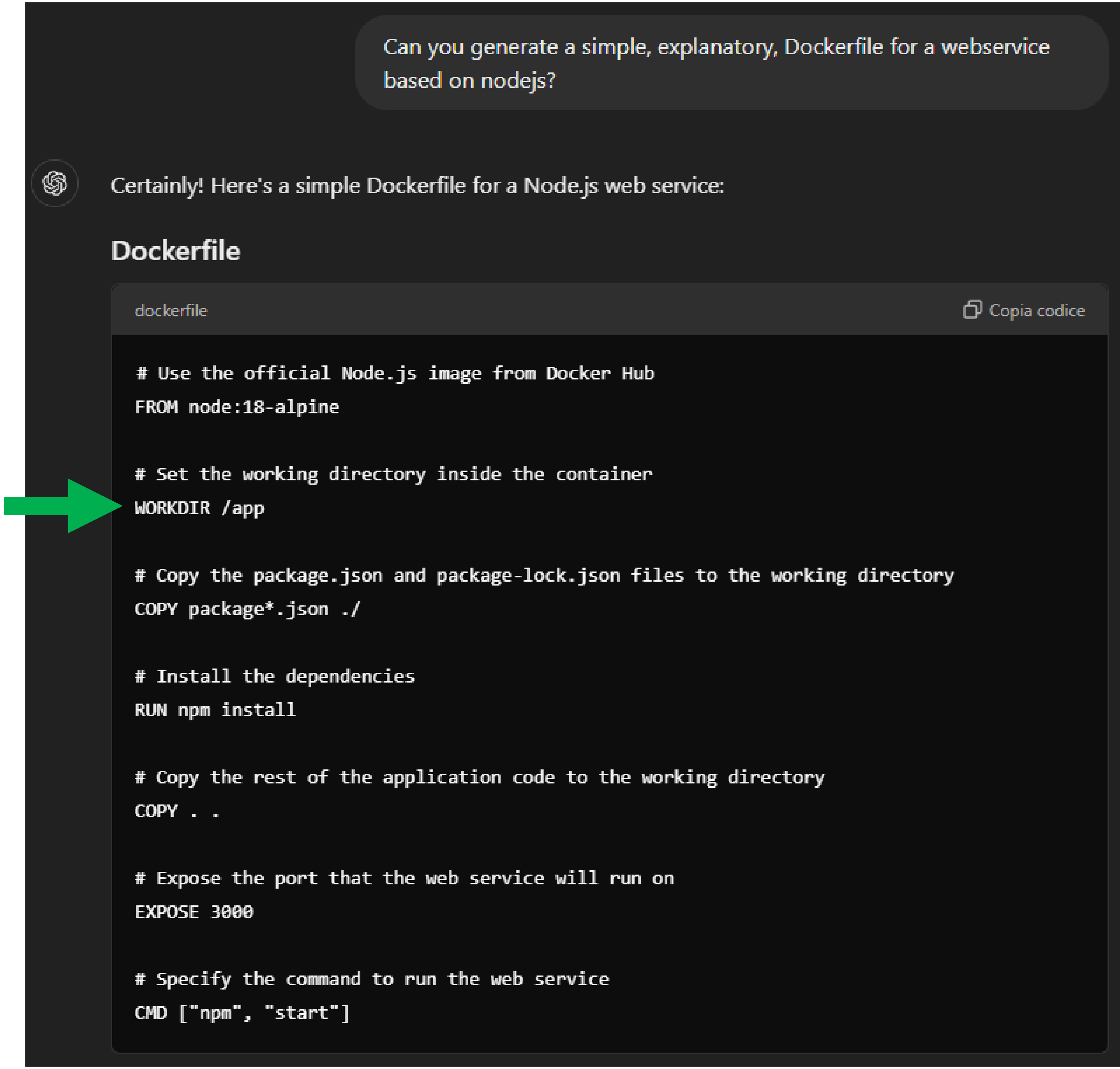
Local machine

Dockerfile
package.json
package-lock.json
server.js

Image

/app/

node:18-alpine



Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

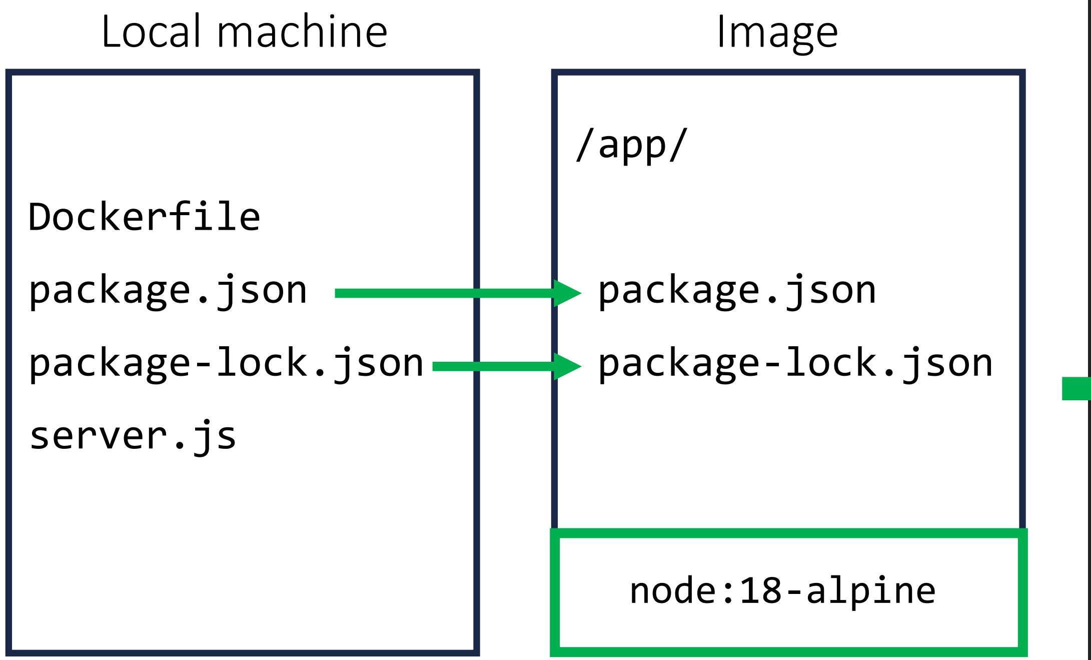
Certainly! Here's a simple Dockerfile for a Node.js web service:

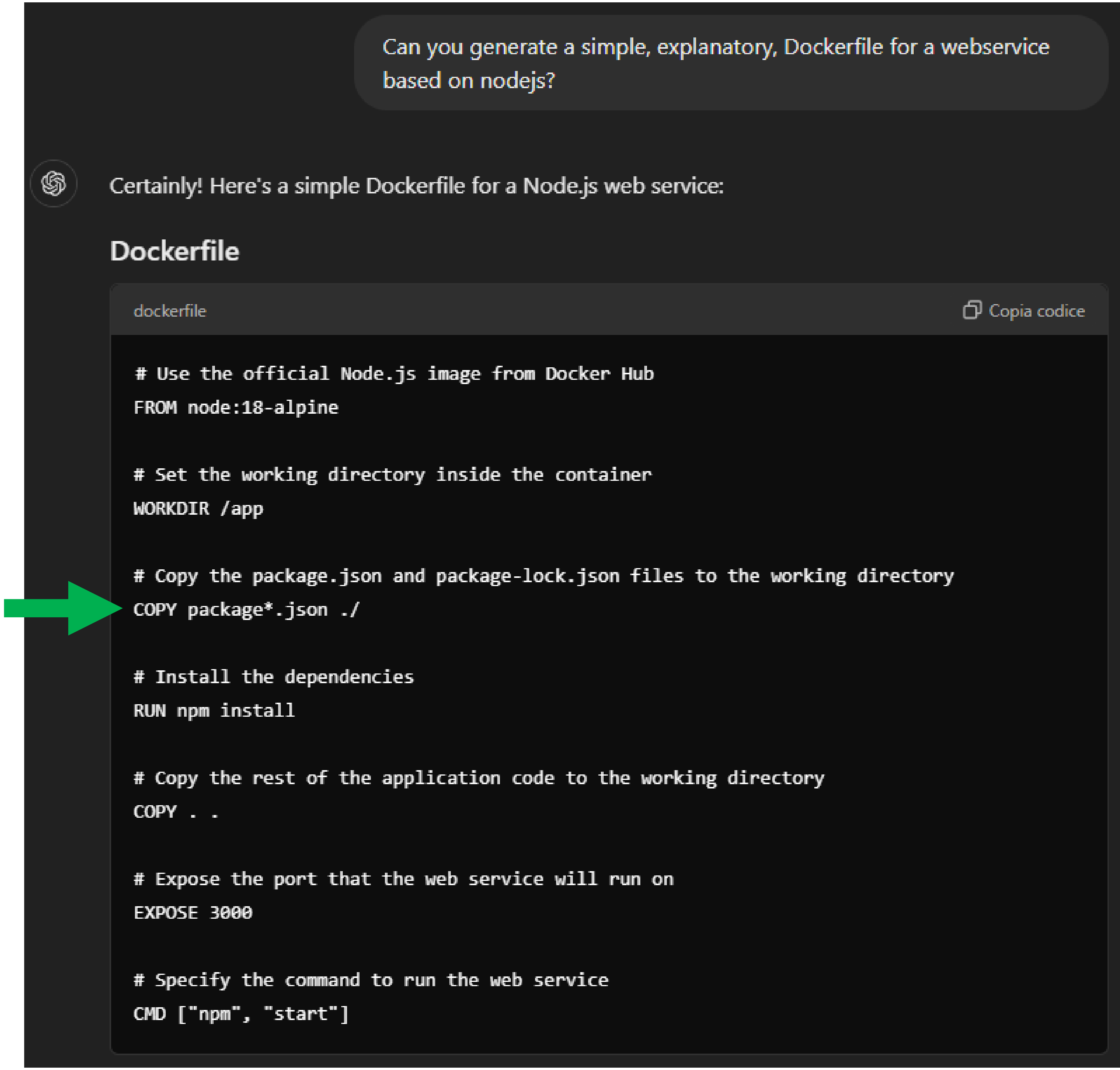
Dockerfile

dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example





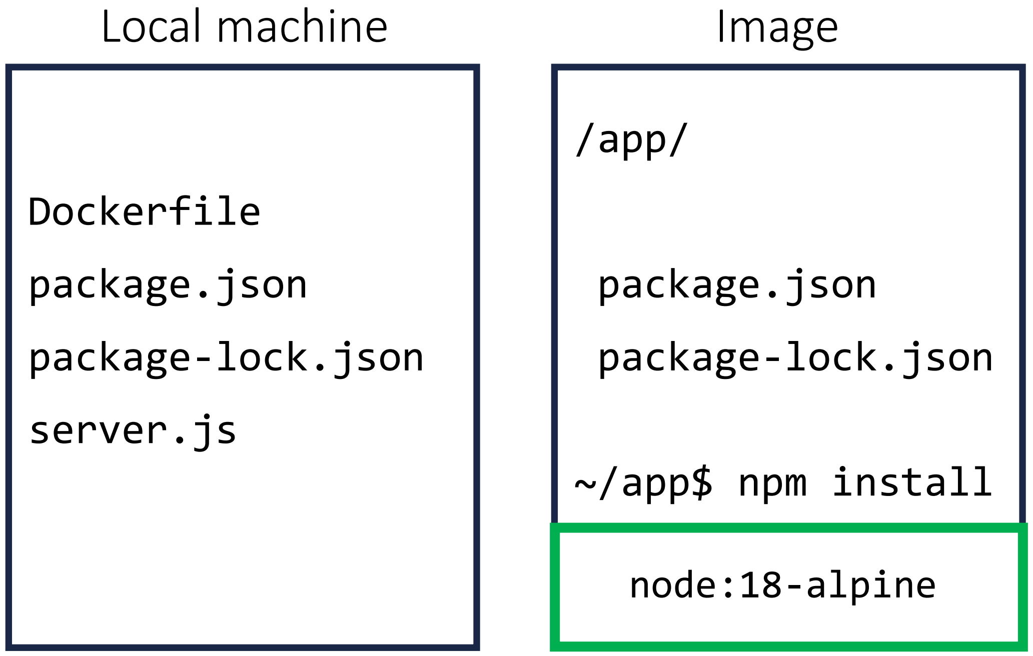
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



Local machine

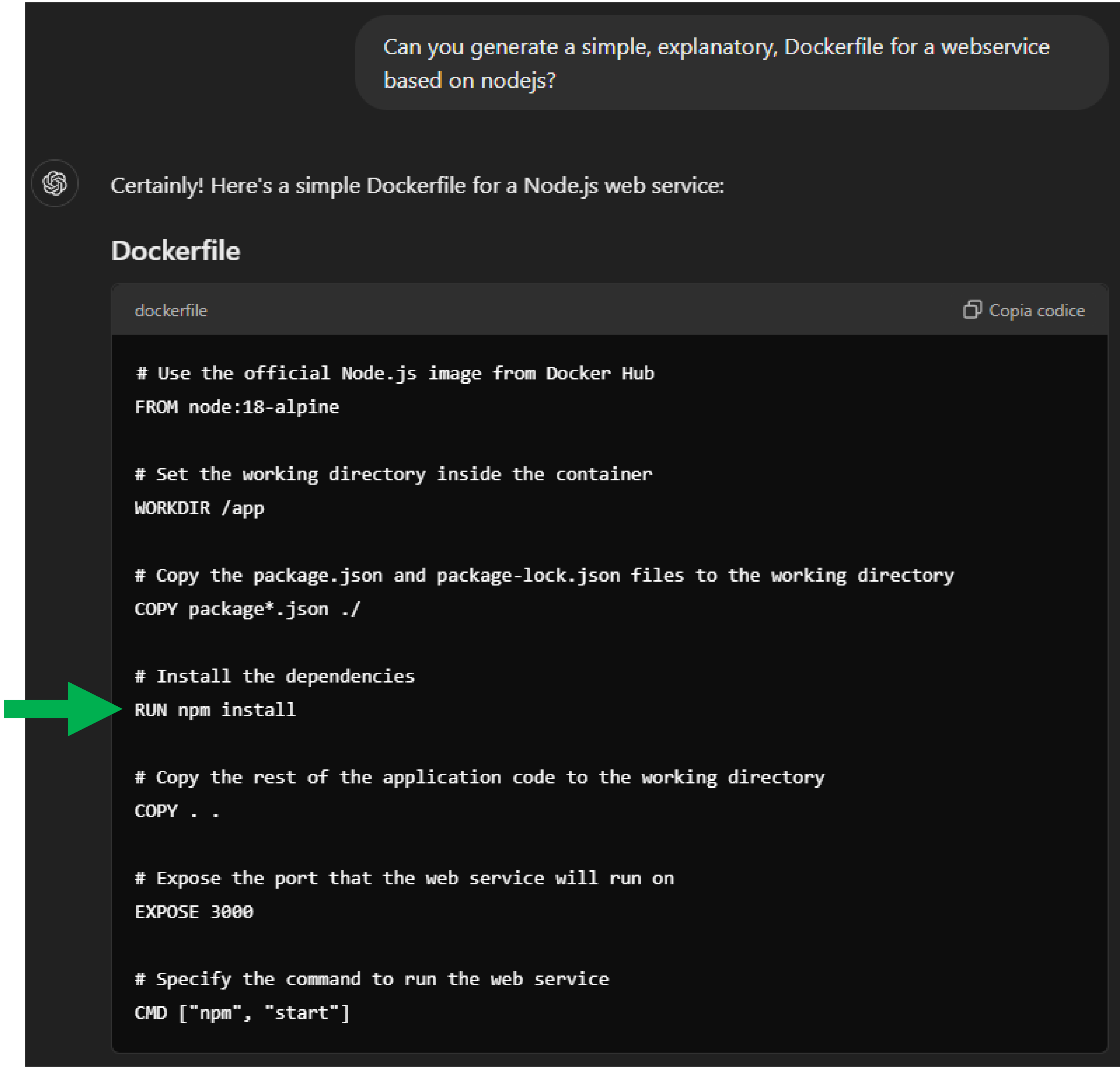
```
Dockerfile
package.json
package-lock.json
server.js
```

Image

```
/app/

 package.json
 package-lock.json

~/app$ npm install
```

```
node:18-alpine
```

Dependencies are installed in the image adding a new layer



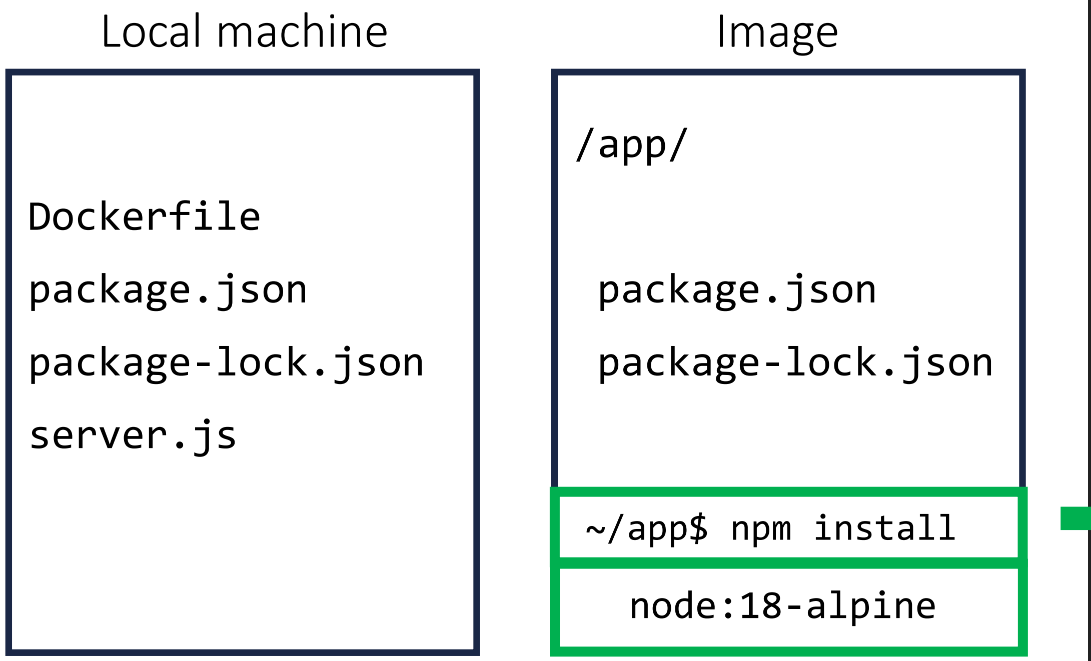
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

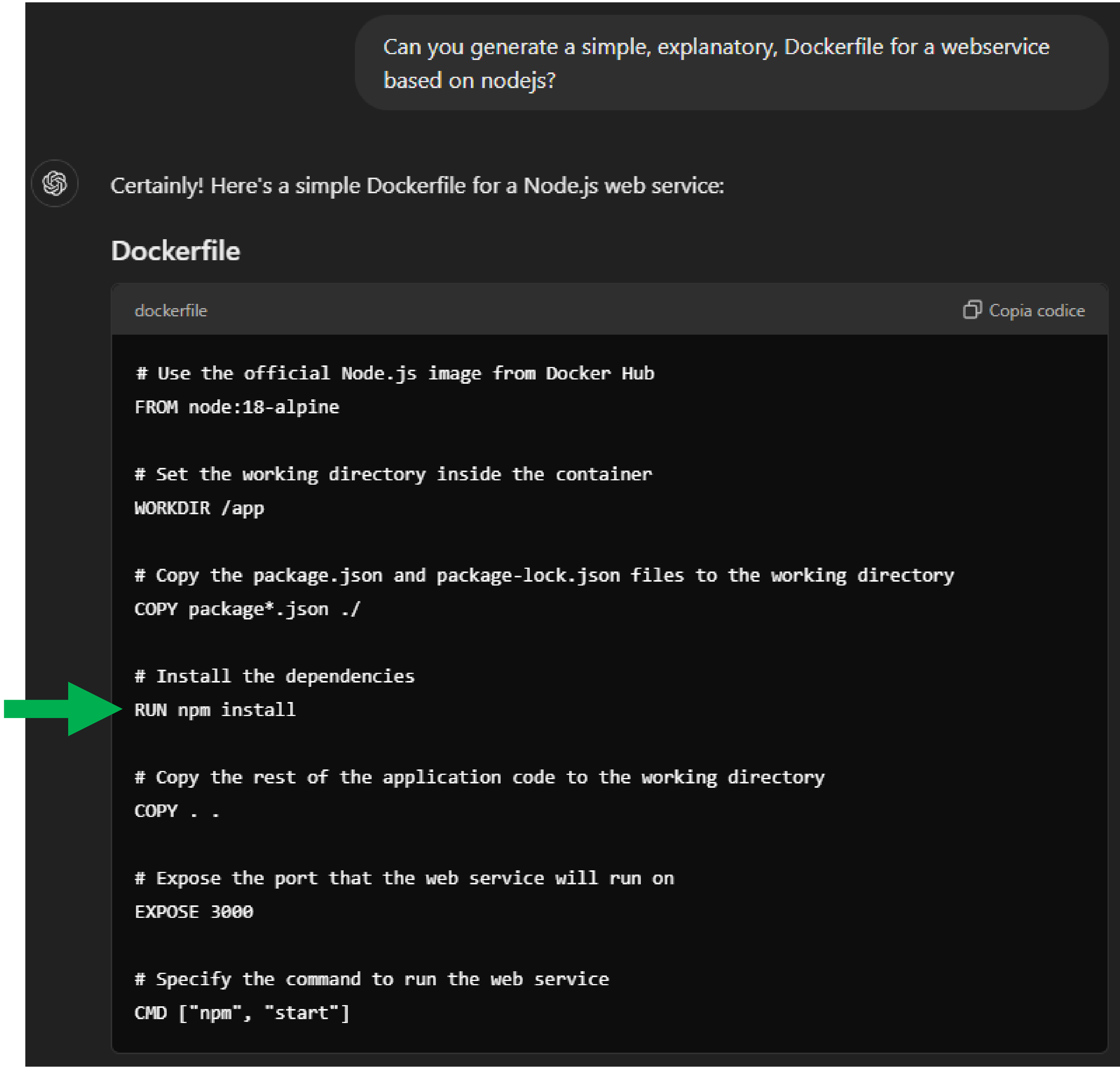
Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



Dependencies are installed in the image adding a new layer



Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

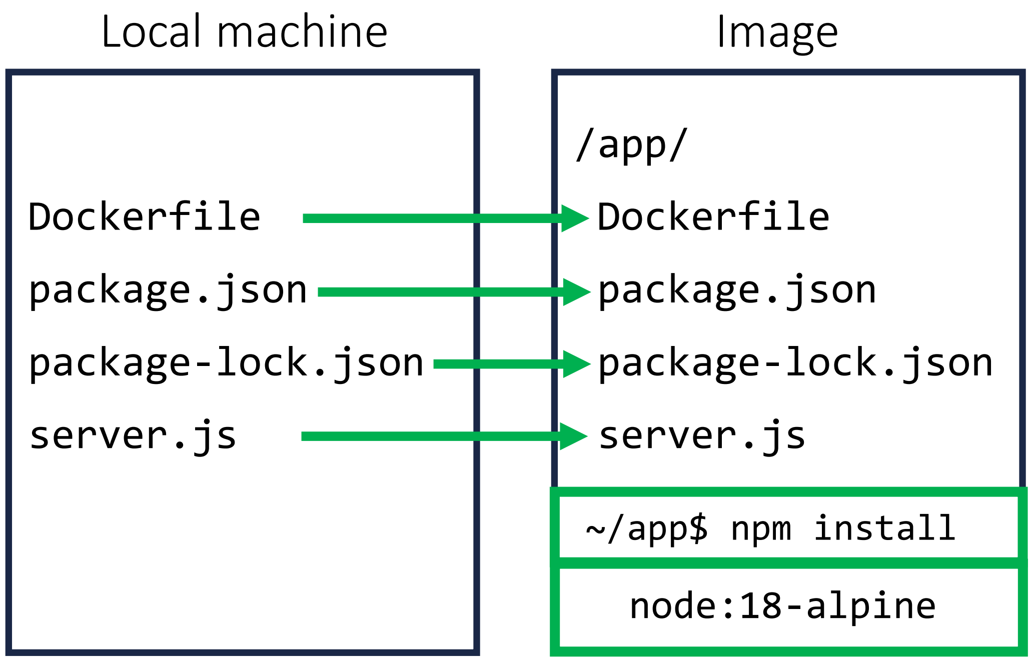
Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

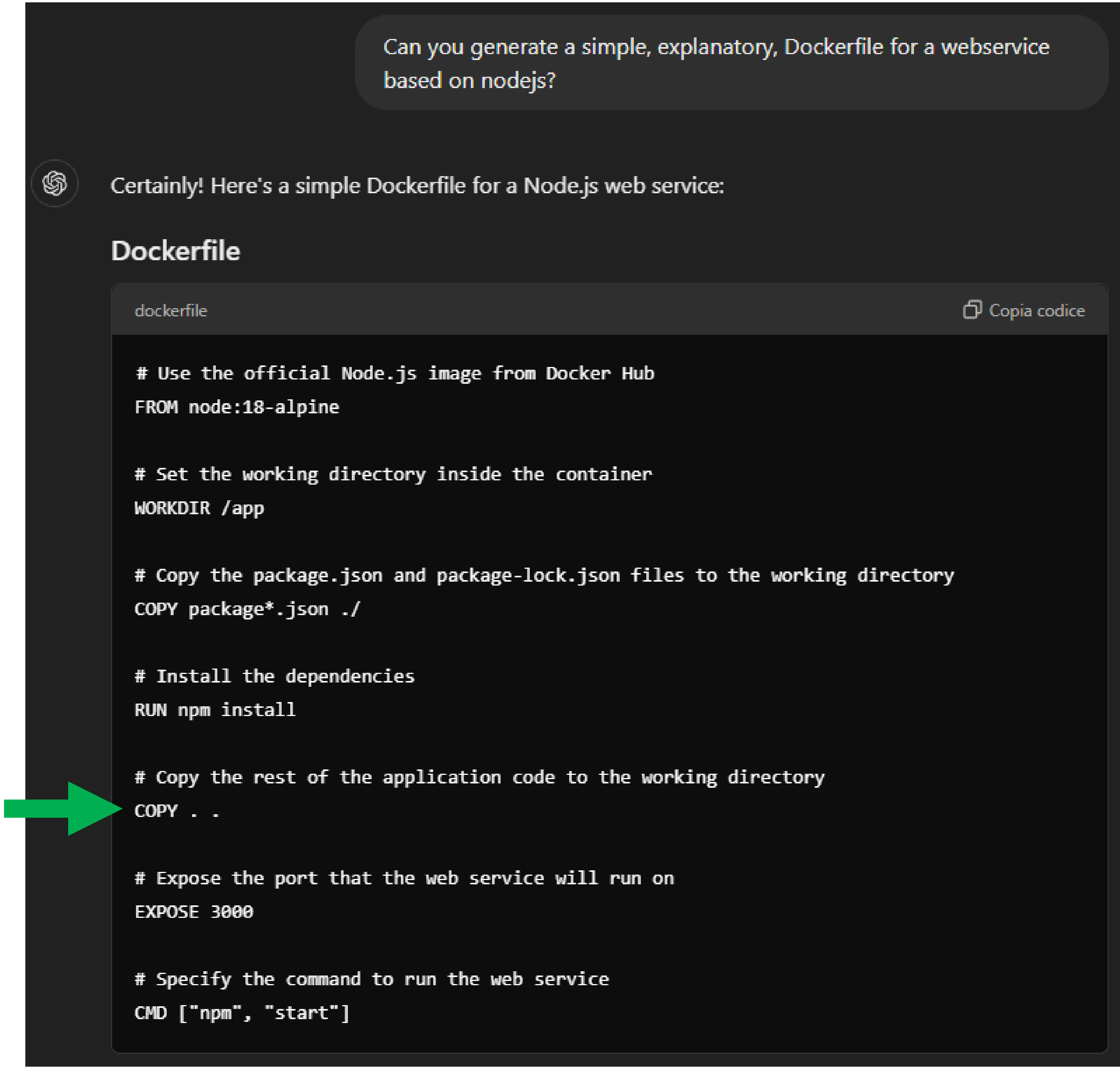
dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



Be carefull doing this, is not very safe!



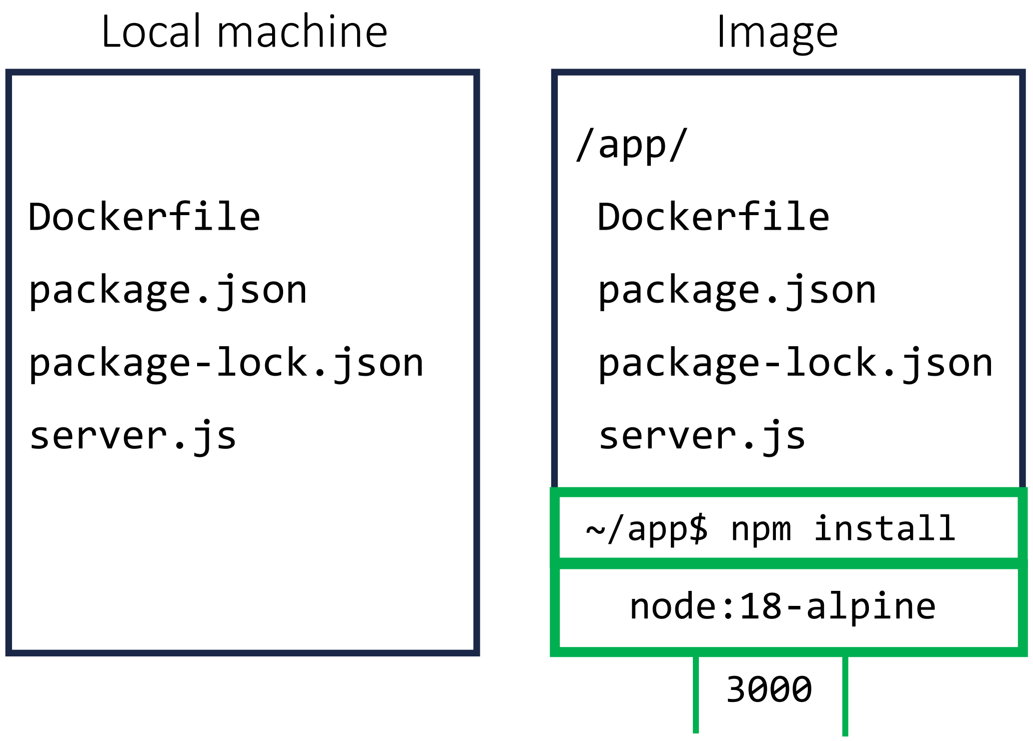
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

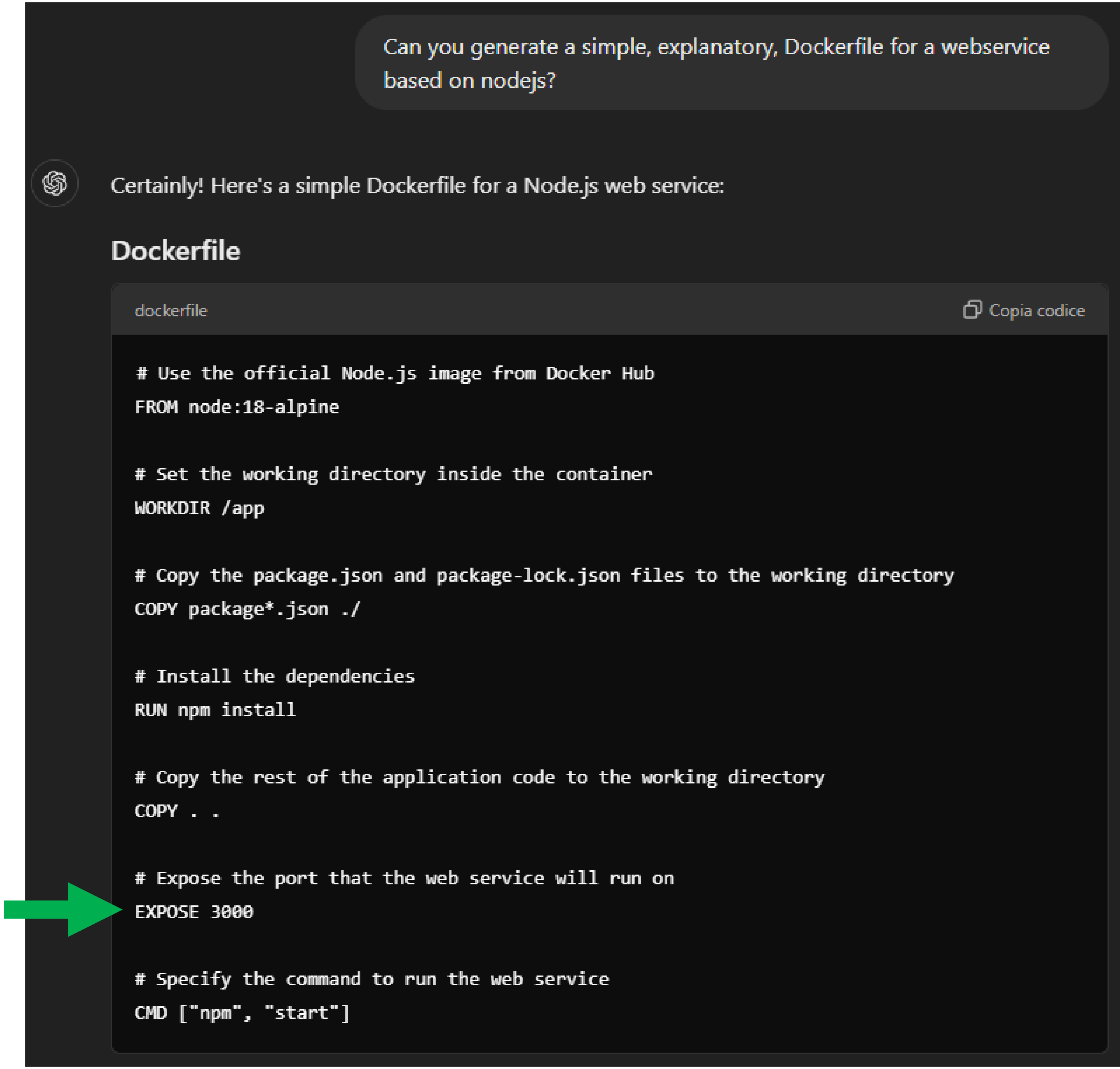
Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



When a container based on this image will be started, a local machine port must be bound with the 3000 port of the container.



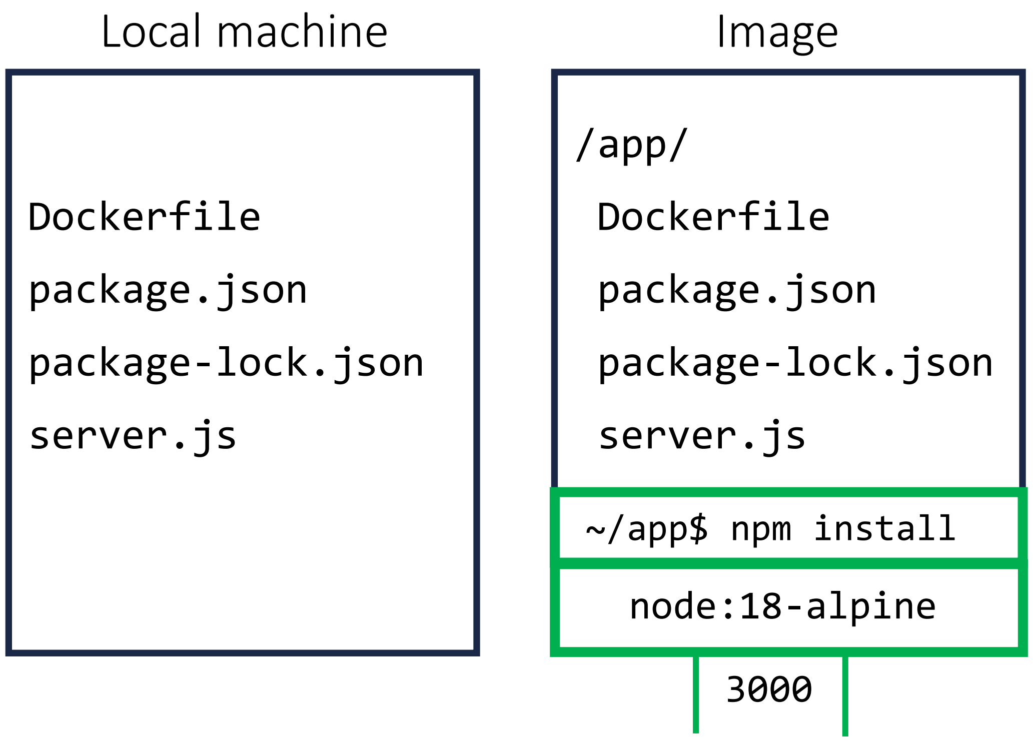
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

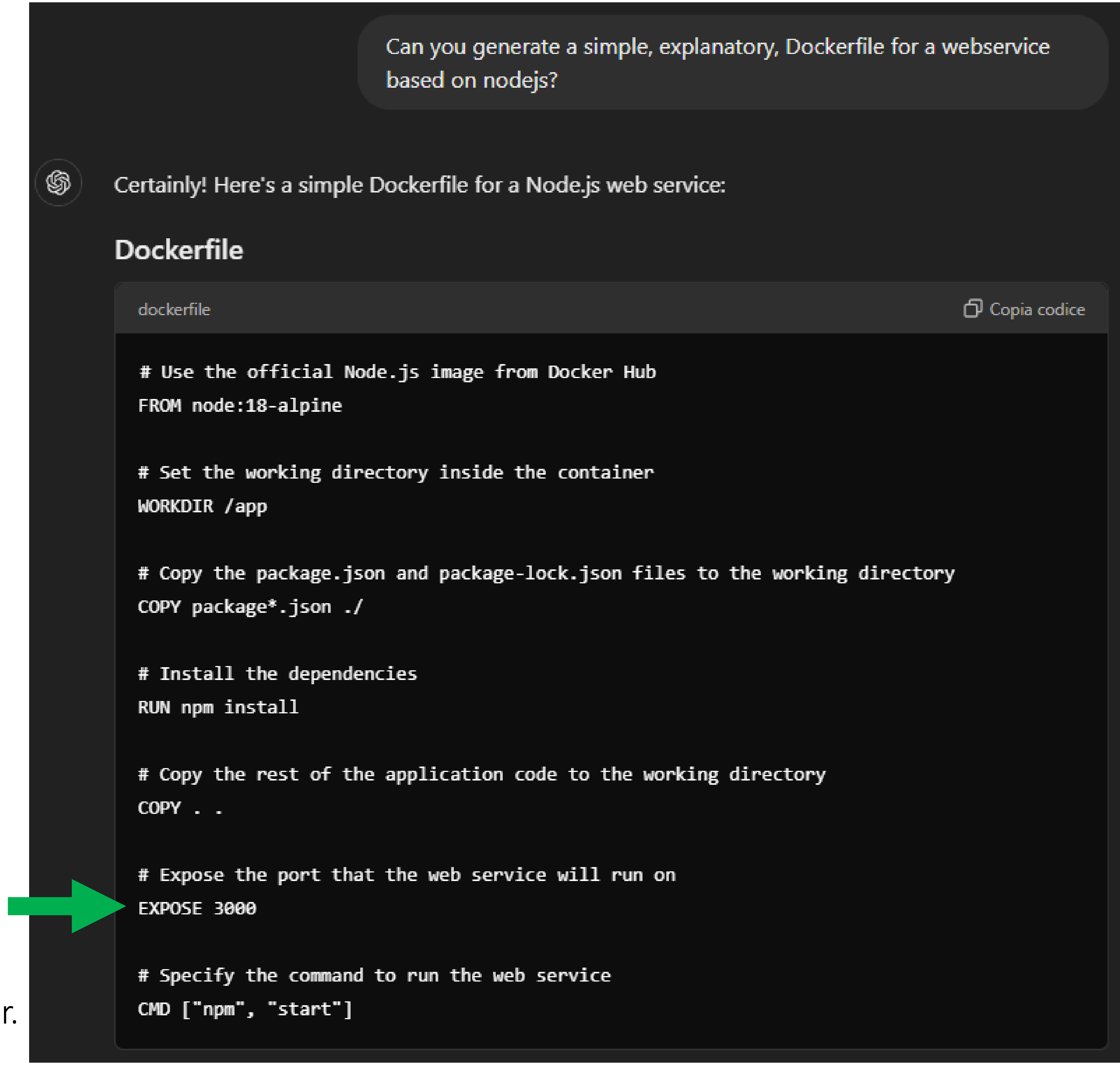
Dockerfile

```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



E.g: `-p 5000:3000`
Contacting port 5000 on the local machine will forward the request to port 3000 on the container.



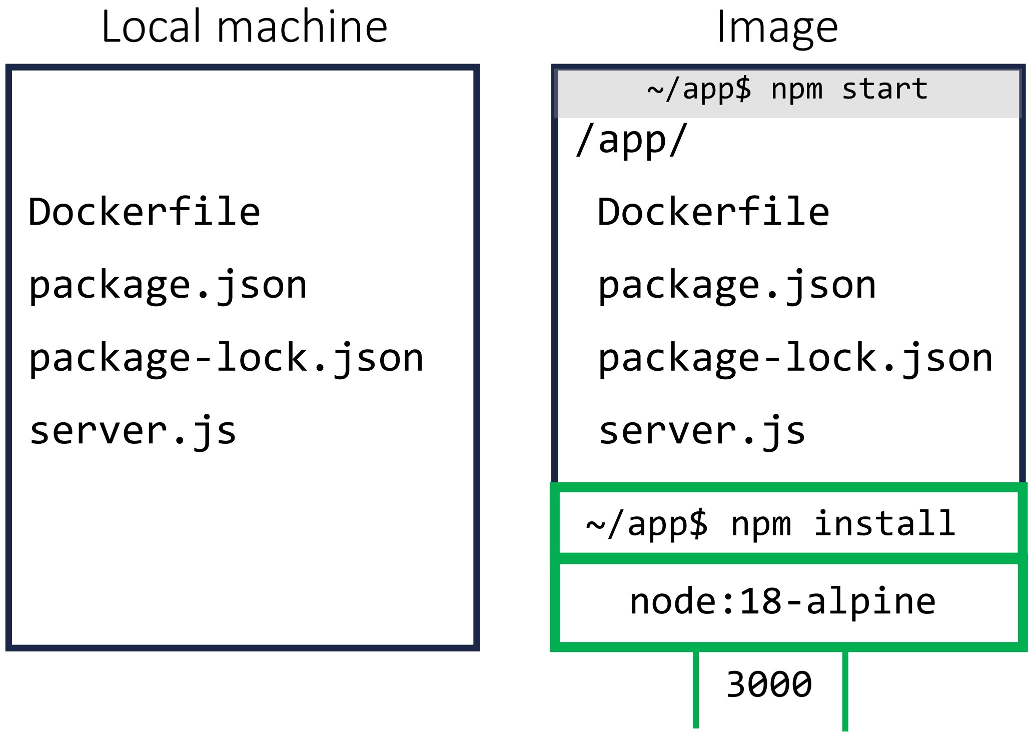
Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

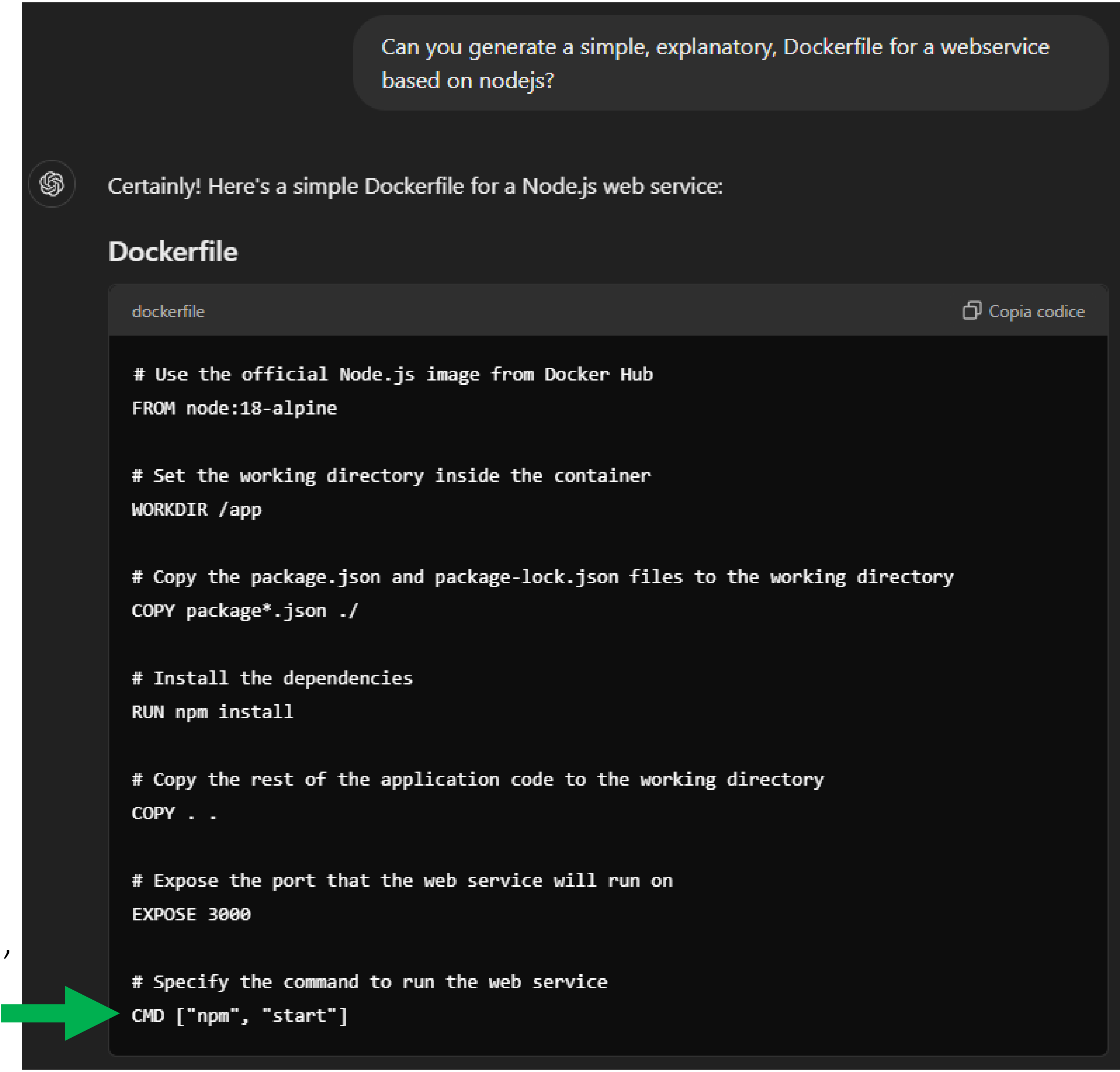
```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

Dockerfile example



When a container based on this image will be run, the command executed will be:

`npm start`



Can you generate a simple, explanatory, Dockerfile for a webservice based on nodejs?

Certainly! Here's a simple Dockerfile for a Node.js web service:

Dockerfile

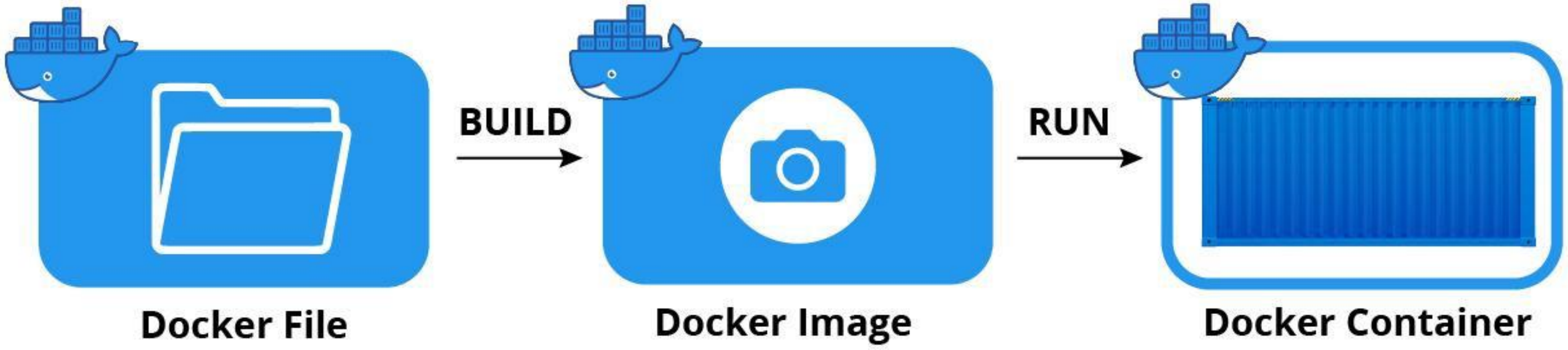
```dockerfile
# Use the official Node.js image from Docker Hub
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port that the web service will run on
EXPOSE 3000

# Specify the command to run the web service
CMD ["npm", "start"]
```

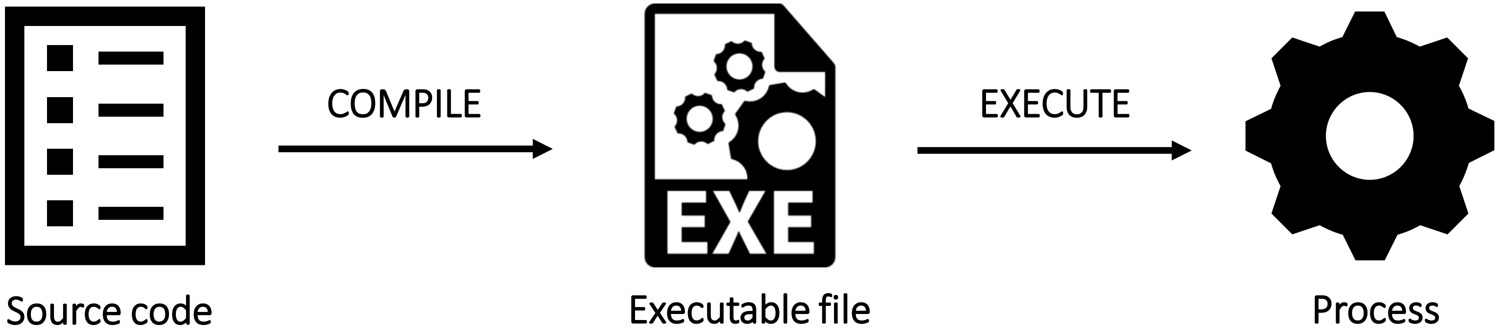
Dockerfile commands cheat sheet

command	**description**
FROM *image*	base image for the build
COPY *path dst*	copy *path* from the context into the container at location *dst*
ADD *src dst*	same as COPY but accepts archives and urls as *src*
RUN *args…*	run an arbitrary command inside the container
CMD *args…*	set the default command
USER *name*	set the default username
WORKDIR *path*	set the default working directory
ENV *name value*	set an environment variable
EXPOSE *port(s)*	allow the container to listens on the network *port(s)*
ENTRYPOINT *exec args…*	configure a container that will run as an executable

The **`Dockerfile`** is a script file having (some of) those command that are executed in order.

Analogy





Source code

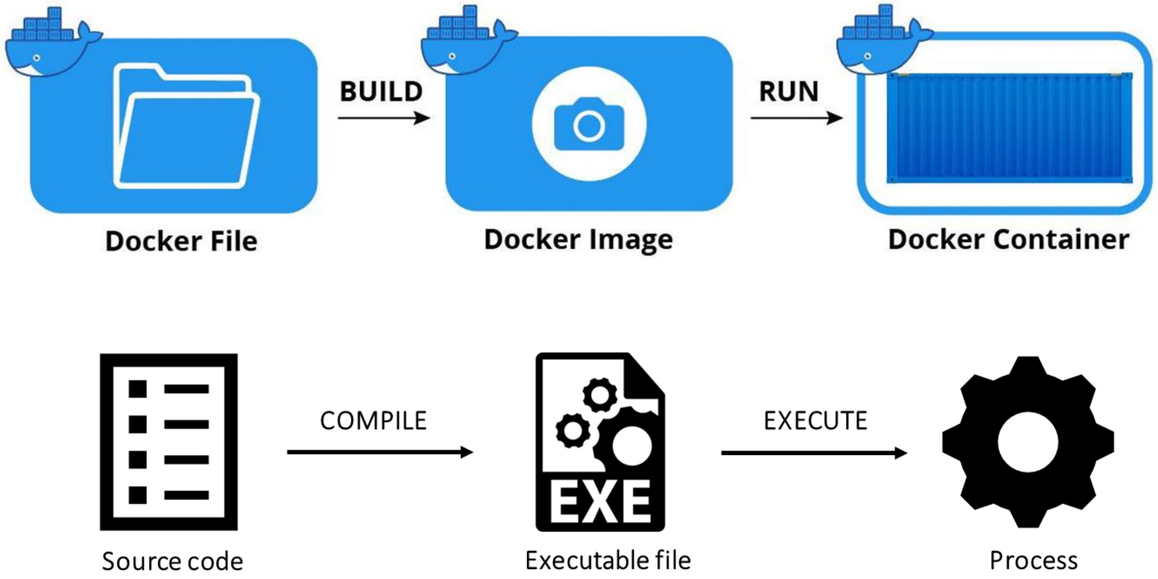
COMPILE

Executable file

EXECUTE

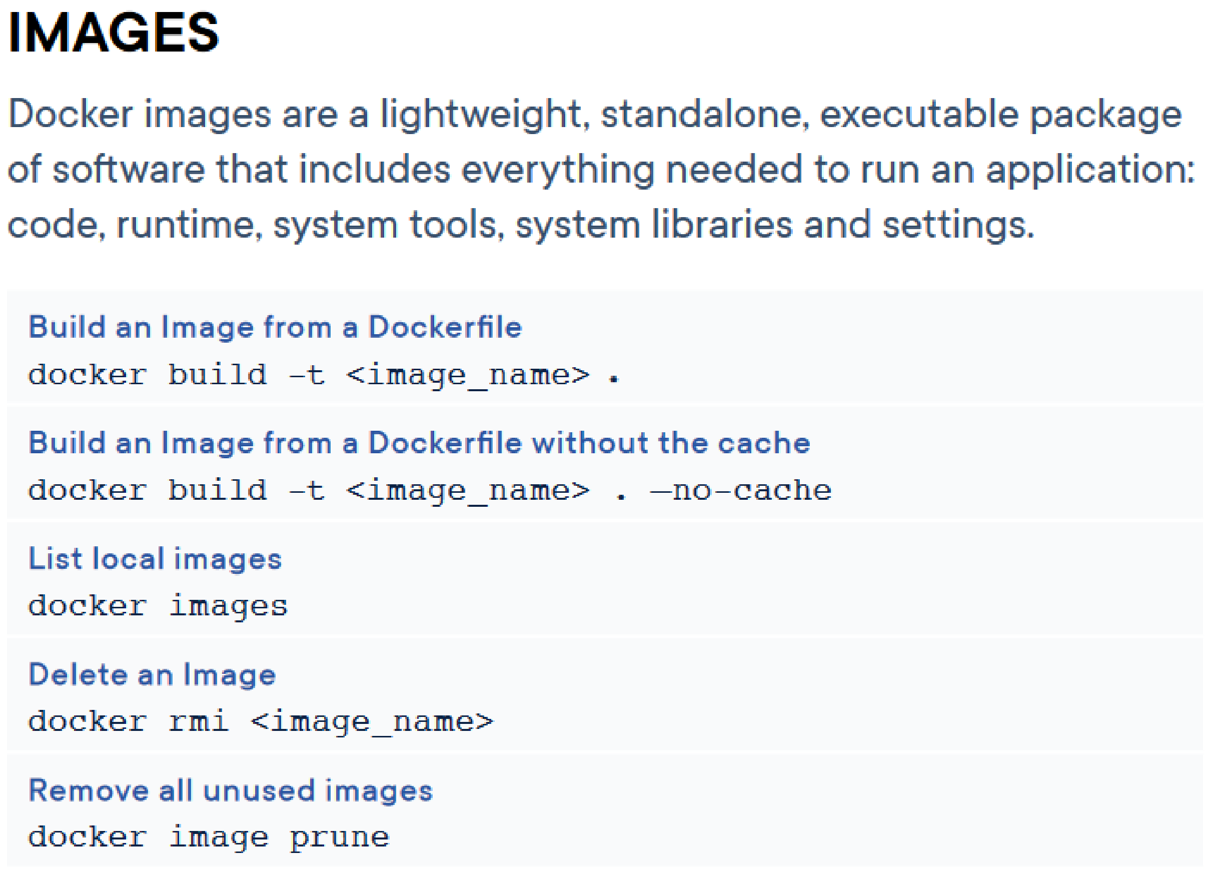
Process

Analogy



- It is possible to have multiple containers from an image.
- If I change something in the Dockerfile (or in the code copied in the image) I have to build again the image and run the container again.

Docker cheat sheet



IMAGES

Docker images are a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Build an Image from a Dockerfile

```
docker build -t <image_name> .
```

Build an Image from a Dockerfile without the cache

```
docker build -t <image_name> . —no-cache
```

List local images

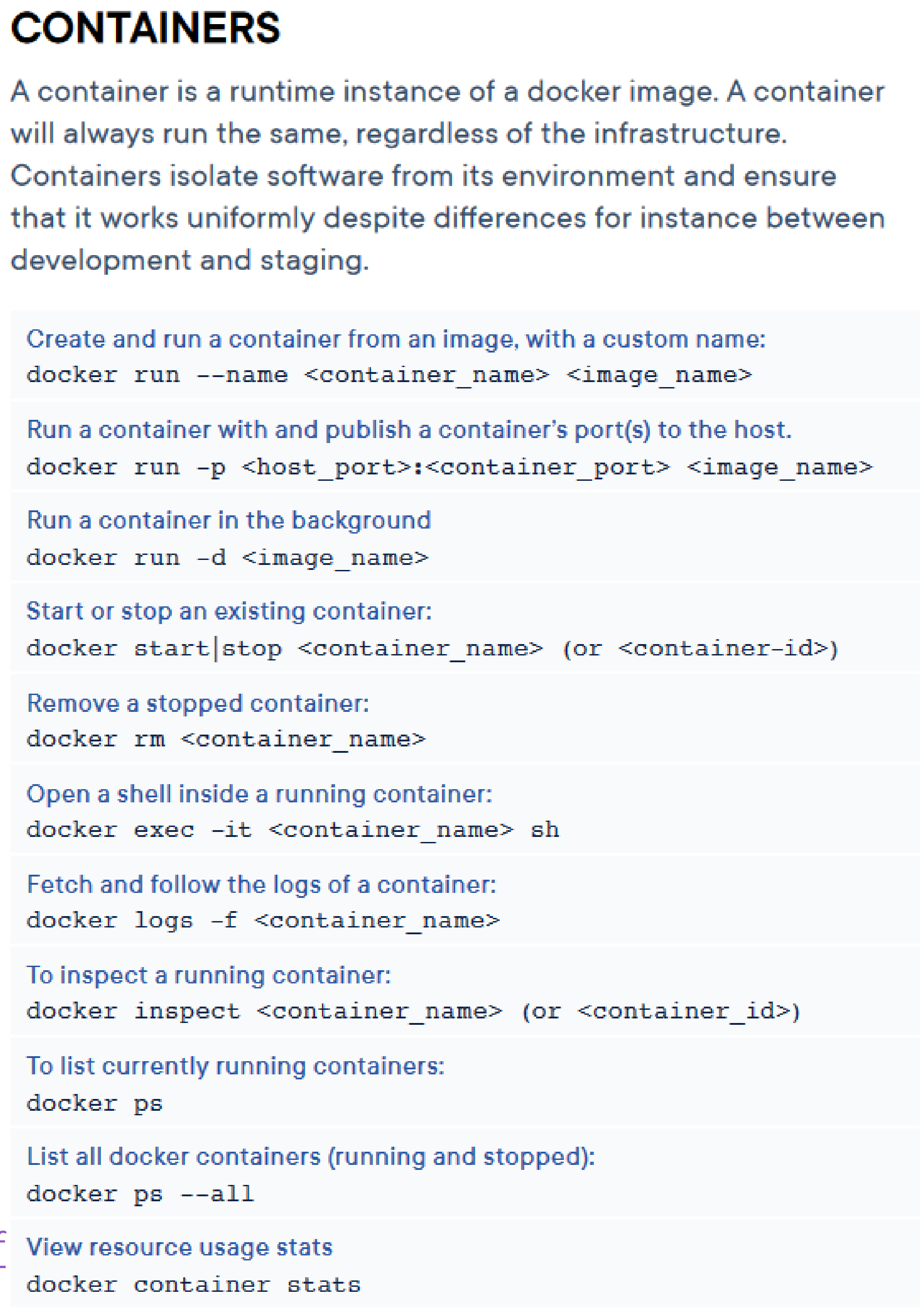
```
docker images
```

Delete an Image

```
docker rmi <image_name>
```

Remove all unused images

```
docker image prune
```



CONTAINERS

A container is a runtime instance of a docker image. A container will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Create and run a container from an image, with a custom name:

```
docker run --name <container_name> <image_name>
```

Run a container with and publish a container's port(s) to the host.

```
docker run -p <host_port>:<container_port> <image_name>
```

Run a container in the background

```
docker run -d <image_name>
```

Start or stop an existing container:

```
docker start|stop <container_name> (or <container-id>)
```

Remove a stopped container:

```
docker rm <container_name>
```

Open a shell inside a running container:

```
docker exec -it <container_name> sh
```

Fetch and follow the logs of a container:

```
docker logs -f <container_name>
```

To inspect a running container:

```
docker inspect <container_name> (or <container_id>)
```

To list currently running containers:

```
docker ps
```

List all docker containers (running and stopped):

```
docker ps --all
```

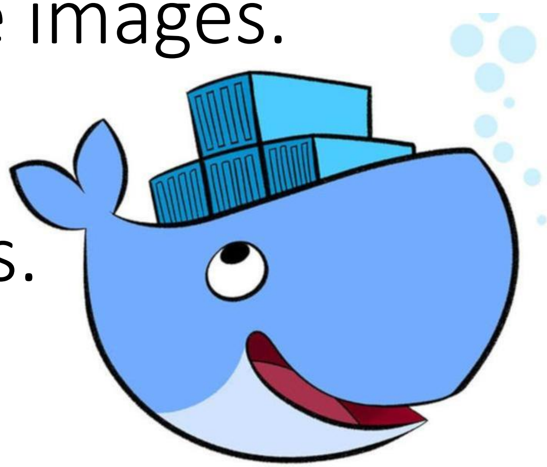
View resource usage stats

```
docker container stats
```

from https://docs.docker.com/get-started/docker_cheatsheet.pdf

With Docker

- We do not need virtual environments for executing Python code.
- Be careful with your docker environment!
  - Check running containers -> they use CPU
  - Check not used images -> they use local storage
- When you are not using Docker, stop your containers (possibly remove them), and decide if you want to remove the images.
- Use the commands of the previous slide.
- The image `python:12-slim` will be used in most Labs.



Today’s Lab

Use the code developed in the first lab including the bonus stage (or download it from the Moodle).

1. Write the `Dockerfile` related to the app.
2. Build the image.
3. Run two containers based on the image and call their API.

(Details in the next slide)

Details

In the folder with the code:

1. Create and write the `Dockerfile` for building a docker image starting from `python:3.12-slim`, move the code in the image environment, install the requirements, expose a port and set the initial command to start Flask correctly.

   (DockerFile docs @ https://docs.docker.com/engine/reference/builder/

   Flask usage @ Lab 1 slides)

2. Build the docker image. Hint: tag it with a name. (https://docs.docker.com/engine/reference/commandline/build/)

3. Use two different terminals to run two containers based on its image, changing the port binding (https://docs.docker.com/engine/reference/run/).

4. Try them! Using a browser (or an HTTP client) invoke the services sending GETs to it, e.g:

   `http://127.0.0.1:5001/add?a=2&b=1` should return a JSON with a field `s = 3`

BONUS STAGE!

Bonus stage

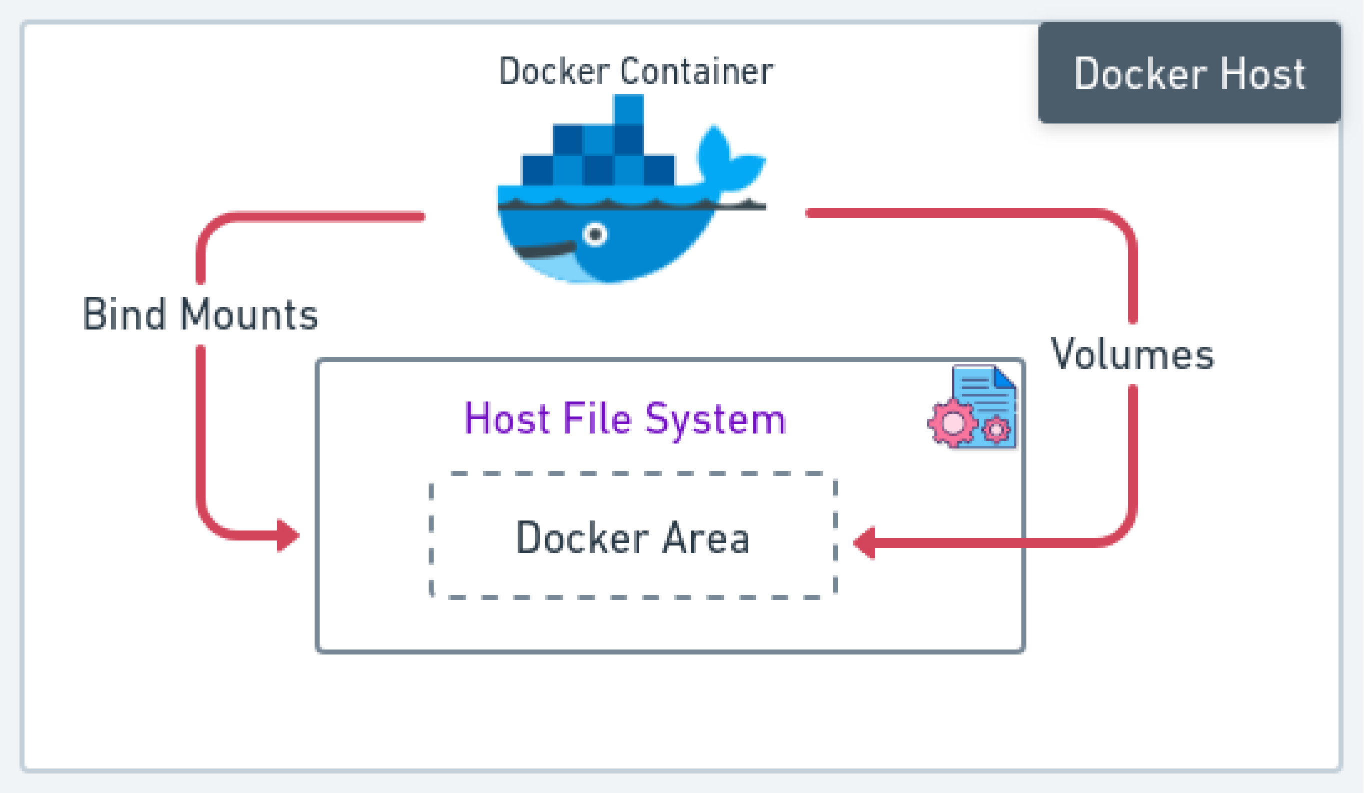
What happens if you:

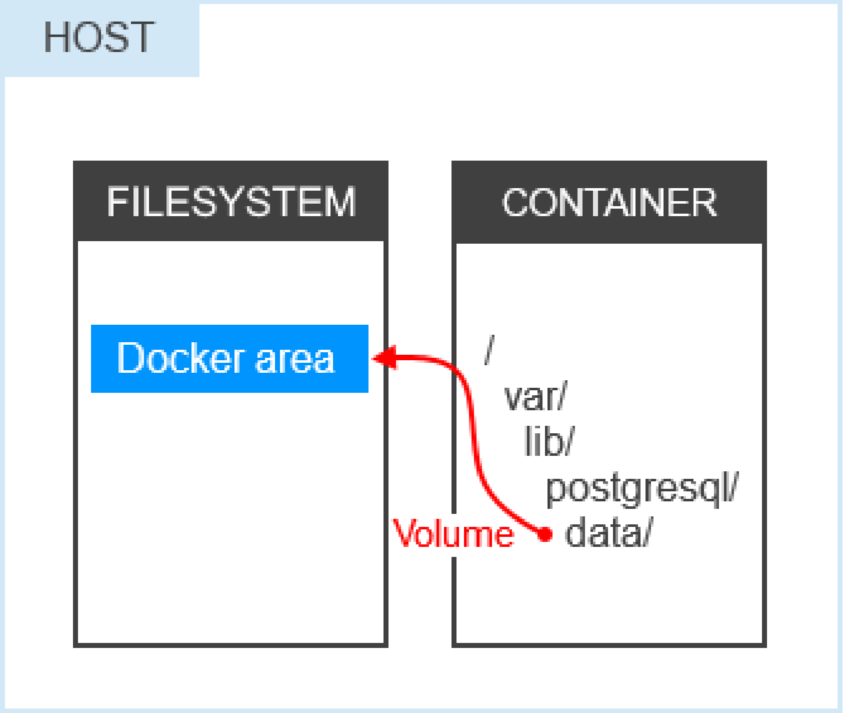
1. Run a container of our app.
2. Do an operation.
3. Stop and remove the container.
4. Start a new container.
5. Call the `/last` endpoint

?

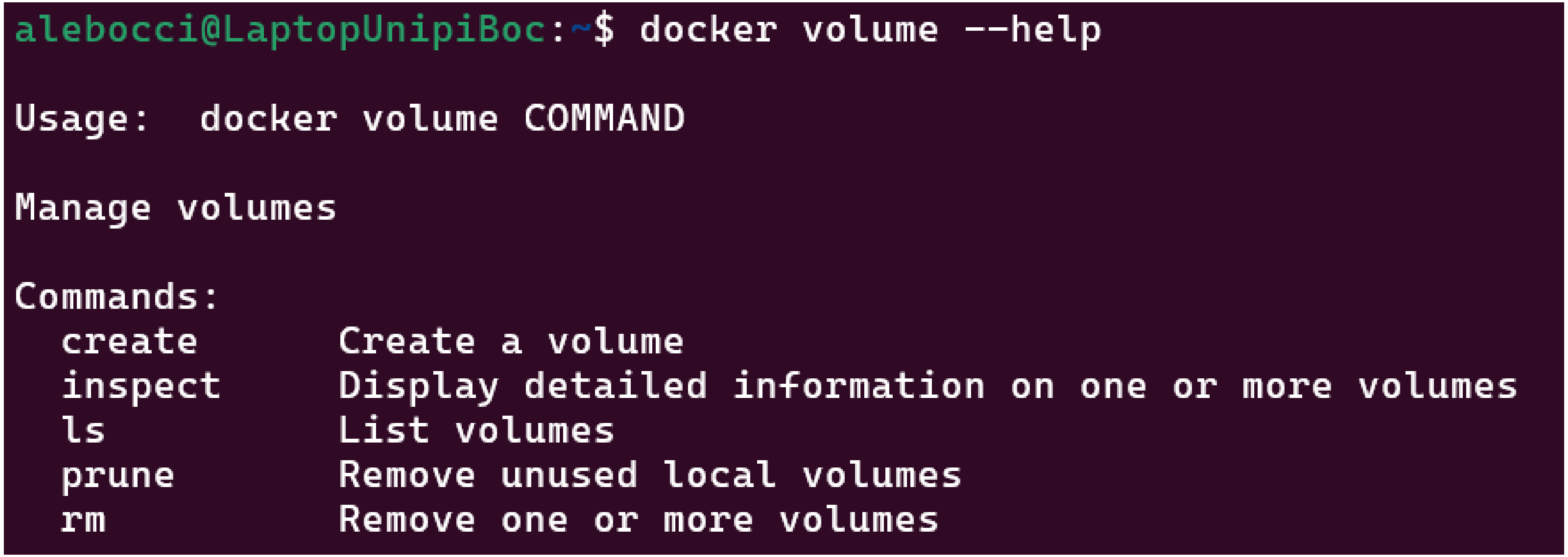
Docker Volumes

Mechanism to manage persisted data created and used by containers.





Docker Volumes - commands



```
alebocci@LaptopUnipiBoc:~$ docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove unused local volumes
  rm          Remove one or more volumes
```

After creating the volume `myvolume` to mount it at `/app`:

```
docker run -p 5000:5000 --mount source=myvolume,target=/app pycalc
```

or

```
docker run -p 5000:5000 -v myvolume:/app pycalc
```

Bonus stage

Use docker volumes to persist the `/last` endpoint file:

1. Run a container in detatched mode (`-d`) and perform an operation.
2. Stop and remove the container.
3. Run a new container and call the `/last` endpoint.
4. It should answer with the operation of point 1.

Lab take away

- ☐ Learn to write a `Dockerfile`
- ☐ Build docker images
- ☐ Run and interact with containers

A good guide to learn more:
https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction

Official Docker training website:
https://training.play-with-docker.com/

Project take away

☐ The service implemented in the project must be dockerized.

☐ You will need to write Dockerfiles for it.