docker

DISTRIBUTED DATA PROCESSING PLATFORMS

A S BHAVANI 22011102002
K R NIKHITHA 22011102036
KAPIL RAISONI A 22011102040
M R ABIRAMI 22011102049
KISHORE CHAKRABORTY 22011102042

INTRODUCTION

What is Distributed Data Processing?

- Processing large datasets across multiple machines
- Used in big data applications, cloud computing, and microservices
- Requires efficient resource management

Why Docker & Orchestration?

- Simplifies deployment and scaling
- Provides consistency across environments
- Manages dependencies efficiently

DOCKER

- A platform for developing, shipping, and running applications in containers.
- Ensures consistency across different environments.

Why Use Docker?

- Lightweight compared to virtual machines.
- Faster deployment and scaling.
- Ensures consistency across development, testing, and production.
- Platform-independent.

DOCKER

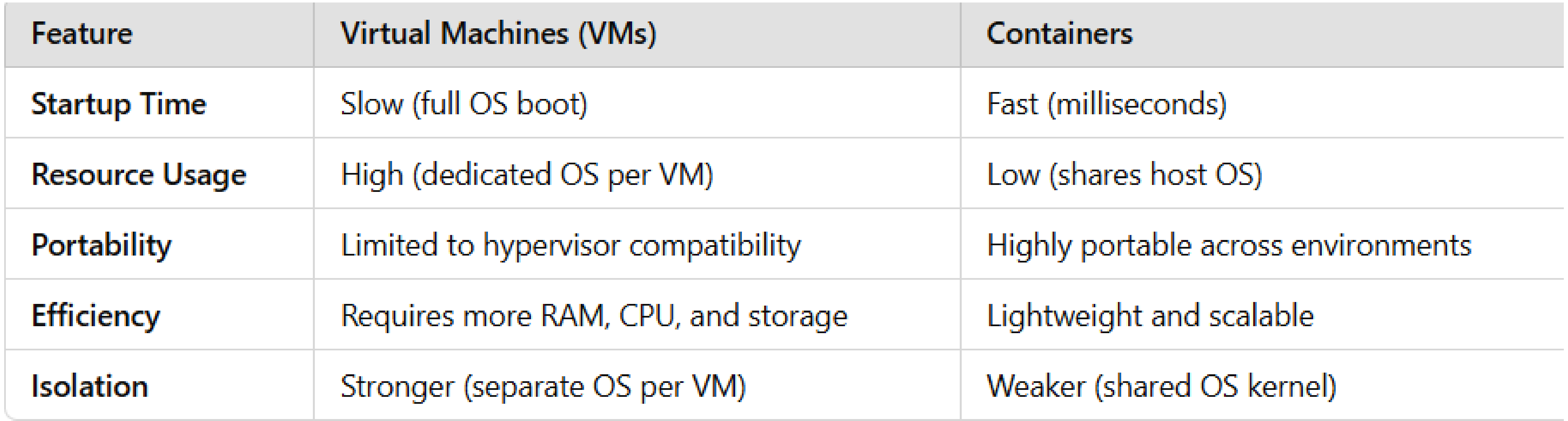
What are Virtual Machines (VMs)?

- VMs emulate a full physical computer.
- Each VM has its own OS, storage, CPU, and memory.
- Managed by a hypervisor, which enables multiple OS instances on one physical server.
- More resource-intensive, but provides strong isolation.

What are Containers?

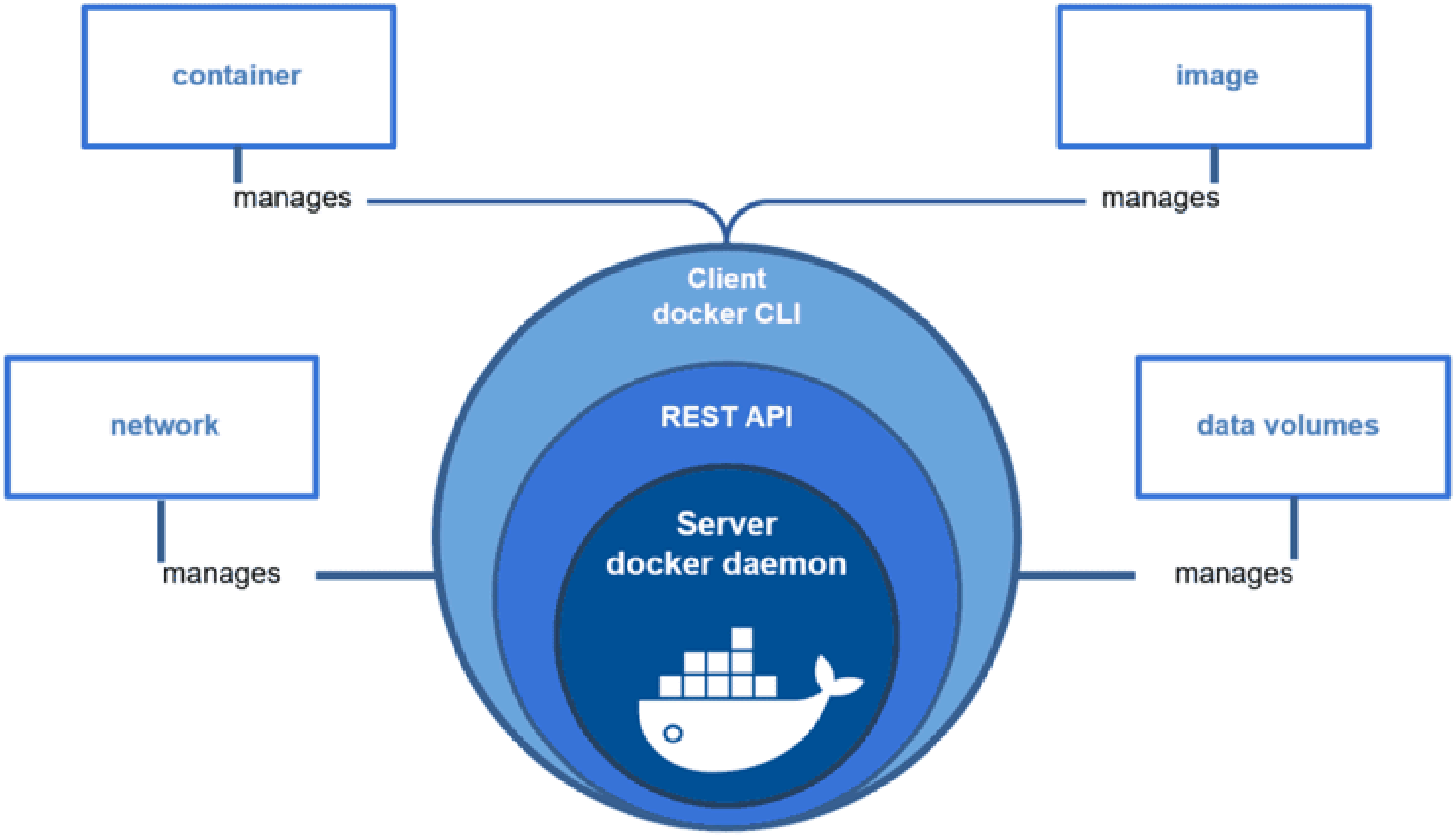
- Lightweight, efficient application environments that share the host OS kernel.
- Faster startup (milliseconds) compared to VMs.
- More portable across different environments (local, cloud, on-premises).

DOCKER



Feature	Virtual Machines (VMs)	Containers
Startup Time	Slow (full OS boot)	Fast (milliseconds)
Resource Usage	High (dedicated OS per VM)	Low (shares host OS)
Portability	Limited to hypervisor compatibility	Highly portable across environments
Efficiency	Requires more RAM, CPU, and storage	Lightweight and scalable
Isolation	Stronger (separate OS per VM)	Weaker (shared OS kernel)

DOCKER ENGINE

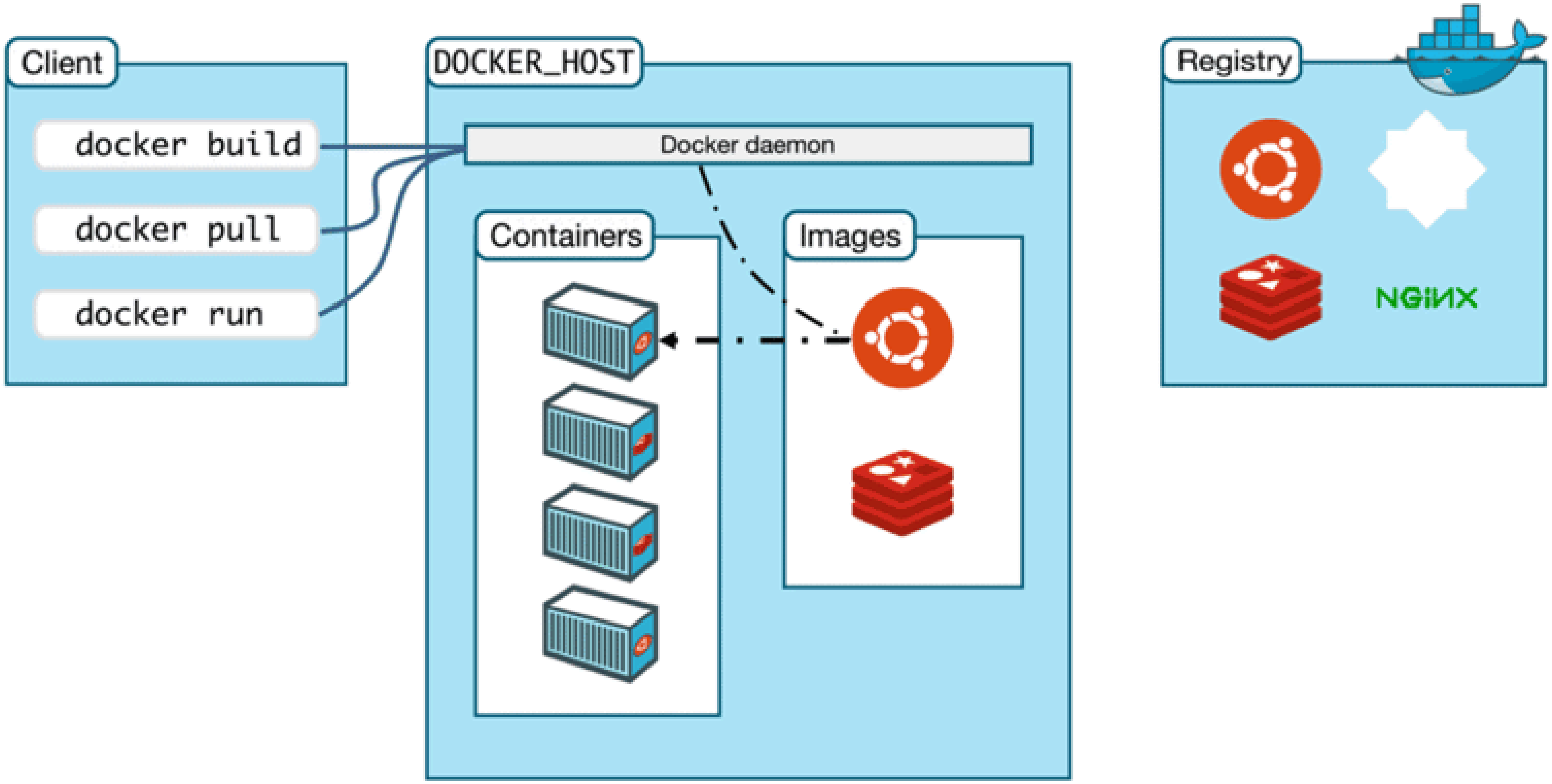


ARCHITECTURE

Docker Engine: The core of Docker, responsible for building and running containers.

- Client: The command-line interface (CLI) or API that sends requests to the Docker daemon.
- Host (Daemon): A background service that handles container management tasks.
- REST API: Enables communication between the client and daemon programmatically.

DOCKER ARCHITECTURE



ARCHITECTURE

Docker Architecture is made up of 5 major components:

1. Docker Client
2. Docker Daemon
3. Docker Host
4. Docker Registry
5. Docker Objects

ARCHITECTURE

Docker Client

- A command-line interface (CLI) tool that allows users to interact with Docker.
- Sends commands (via REST API) to the Docker Daemon, such as starting, stopping, and managing containers.
- Users can execute commands like docker run, docker stop, and docker pull to manage Docker objects.

Docker Daemon

- Runs on the host OS and is responsible for managing Docker images, containers, networks, and volumes.
- Listens for API requests and processes container-related tasks.
- Can communicate with other Docker Daemons to manage multiple services in a distributed system.
- Automates container execution, builds images, and manages lifecycle operations.

ARCHITECTURE

Docker Host

- The machine where Docker Daemon runs, serving as the execution environment for containers.
- Processes and executes the commands issued by the Docker Client.
- Can be a local machine, a virtual server, or a cloud-based instance.

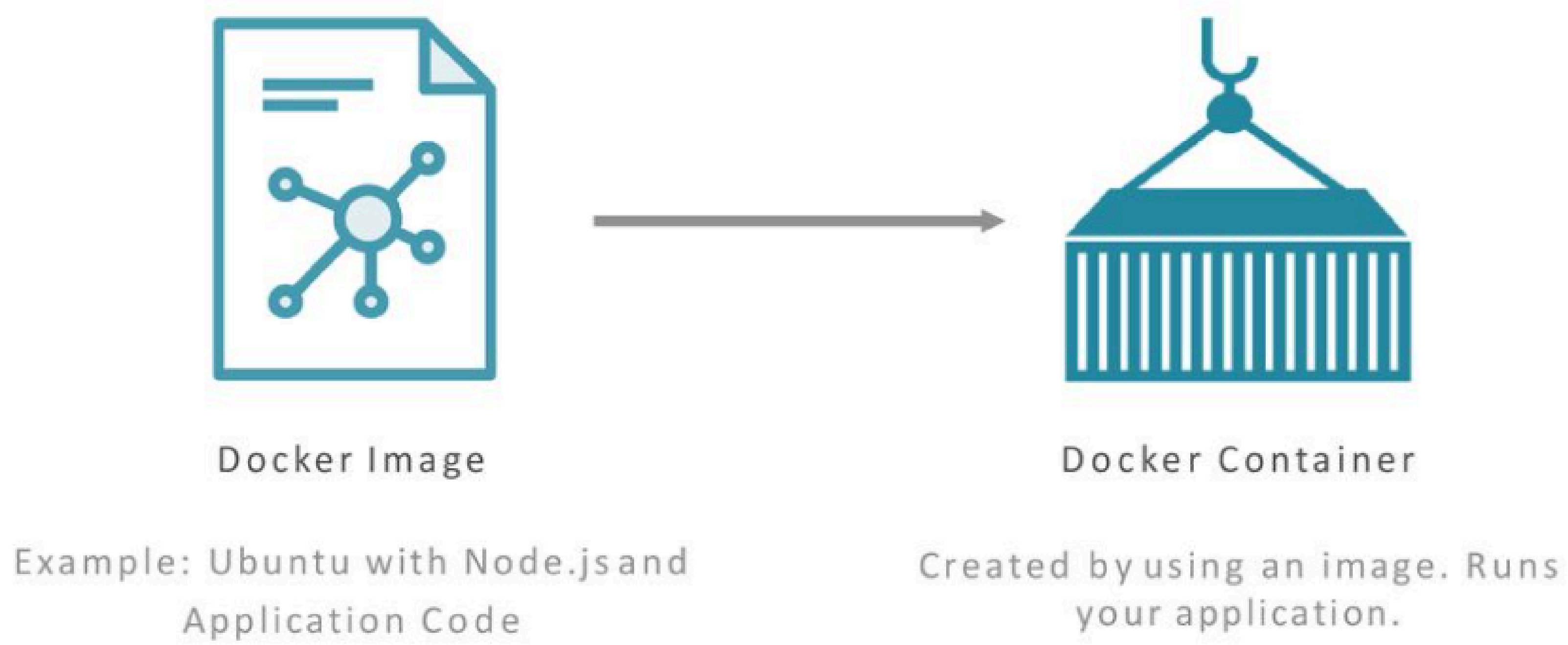
Docker Registry

- A centralized repository where Docker images are stored and distributed.
- Docker Hub (default public registry) allows users to share and pull images.
- Private registries can also be set up for internal use in organizations.

ARCHITECTURE

Docker Objects

- **Docker Images:** Read-only templates containing application code, dependencies, and configurations.
  - Can be pulled from Docker Hub or created using Dockerfiles.
  - The base layer is immutable, while the top layer can be modified.
- **Docker Containers:** Running instances of Docker images that provide isolated environments.
  - Contains all dependencies required to run an application.
  - Can be started, stopped, or deleted using Docker CLI or API.



Example: Ubuntu with Node.jsand Application Code

Created by using an image. Runs your application.

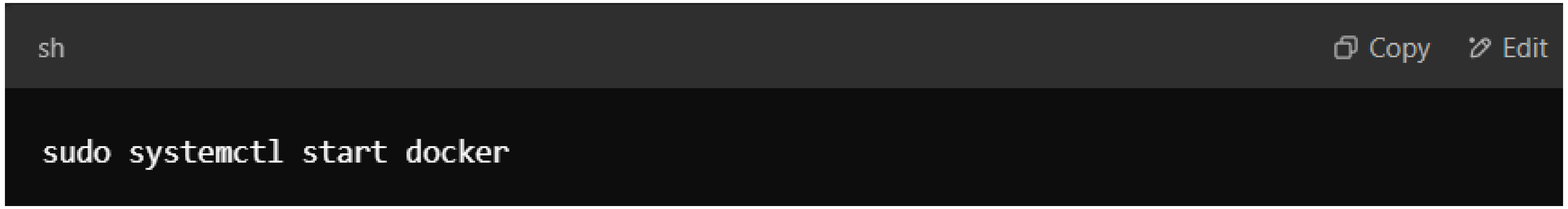
INSTALLATION

- Download Docker – Go to the official Docker website and download the appropriate version for your operating system (Windows, macOS, or Linux).

- Install Docker – Follow the installation guide for your OS:
  - Windows: Install Docker Desktop (requires Windows 10/11 Pro or WSL2 for Home edition).
  - MacOS: Install Docker Desktop for macOS.
  - Linux: Install Docker Engine via package managers like apt (Debian/Ubuntu) or yum (CentOS/RHEL).

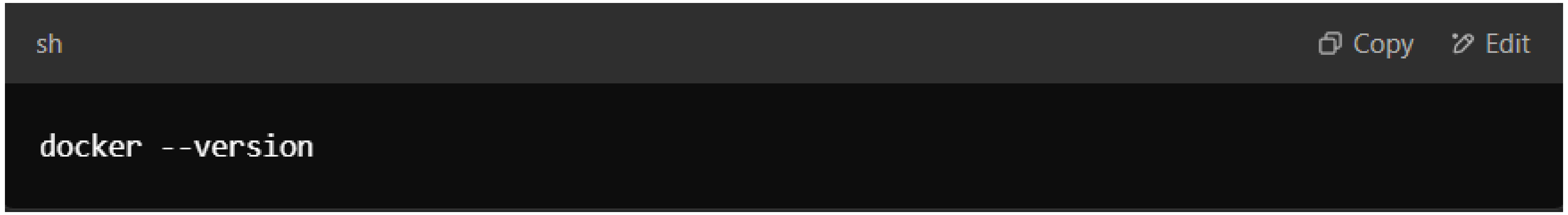
INSTALLATION

Start Docker – Once installed, launch Docker. On Windows/macOS, start Docker Desktop. On Linux, start the daemon with:



```sh
sudo systemctl start docker
```

Confirm Installation – Open a terminal or command prompt and verify that Docker is installed by running:



```sh
docker --version
```

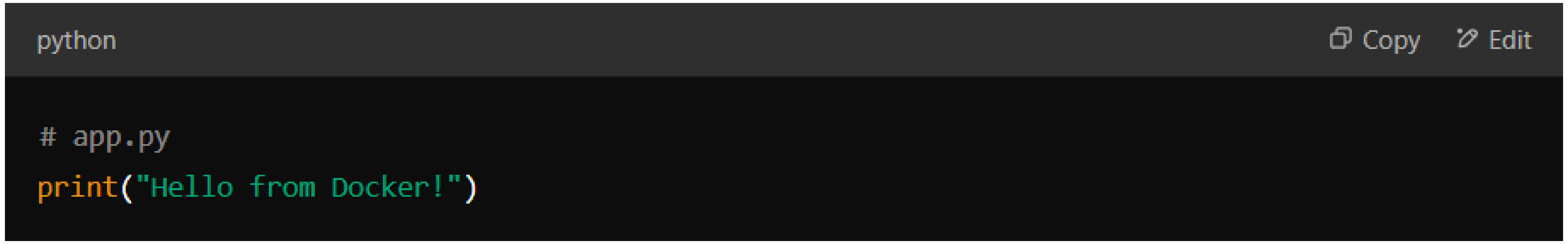
This should return the installed Docker version, confirming a successful installation.

HOW IT WORKS

Steps:

- Developer writes a Dockerfile.
- Builds an image using docker build.
- Runs a container using docker run.
- Manages containers with commands (start, stop, restart).

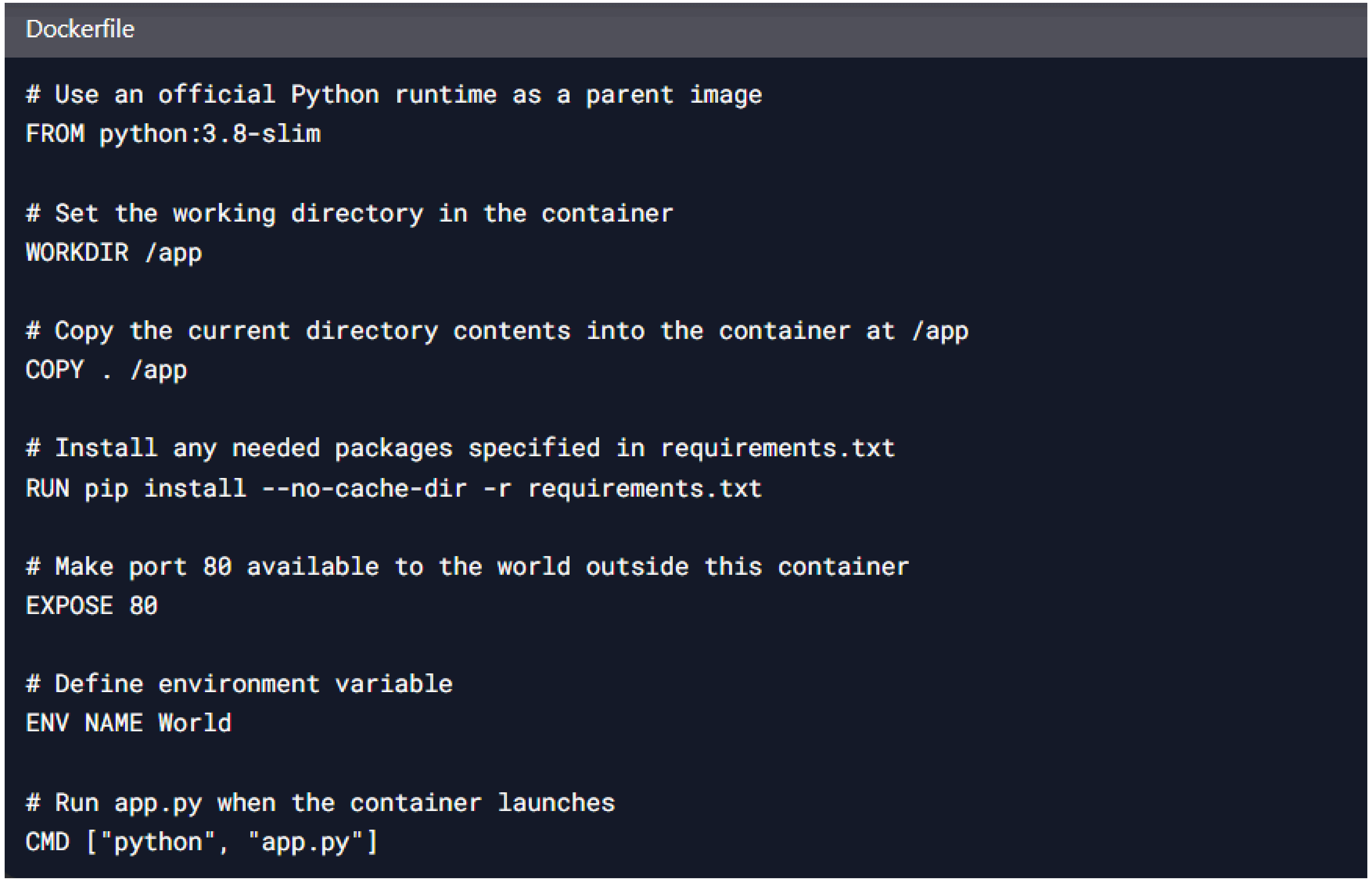
Example: Running a Simple Python App in Docker
Step 1: Create a Python Script (app.py)



```python
# app.py
print("Hello from Docker!")
```

HOW IT WORKS

Step 2: Write a Dockerfile



```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

HOW IT WORKS

A Dockerfile is a script that contains instructions on how to build a Docker image. Here are some common instructions:

FROM:

Specifies the base image to use.

Example: FROM ubuntu:20.04

RUN:

Executes a command in the shell.

Example: RUN apt-get update && apt-get install -y python3

COPY:

Copies files or directories from the host machine to the container.

Example: COPY . /app

WORKDIR:

Sets the working directory for subsequent instructions.

Example: WORKDIR /app

CMD:

Specifies the default command to run when the container starts.

Example: CMD ["python3", "app.py"]

EXPOSE:

Informs Docker that the container listens on the specified network ports at runtime.

Example: EXPOSE 80

HOW IT WORKS

Docker Compose

What is Docker Compose?

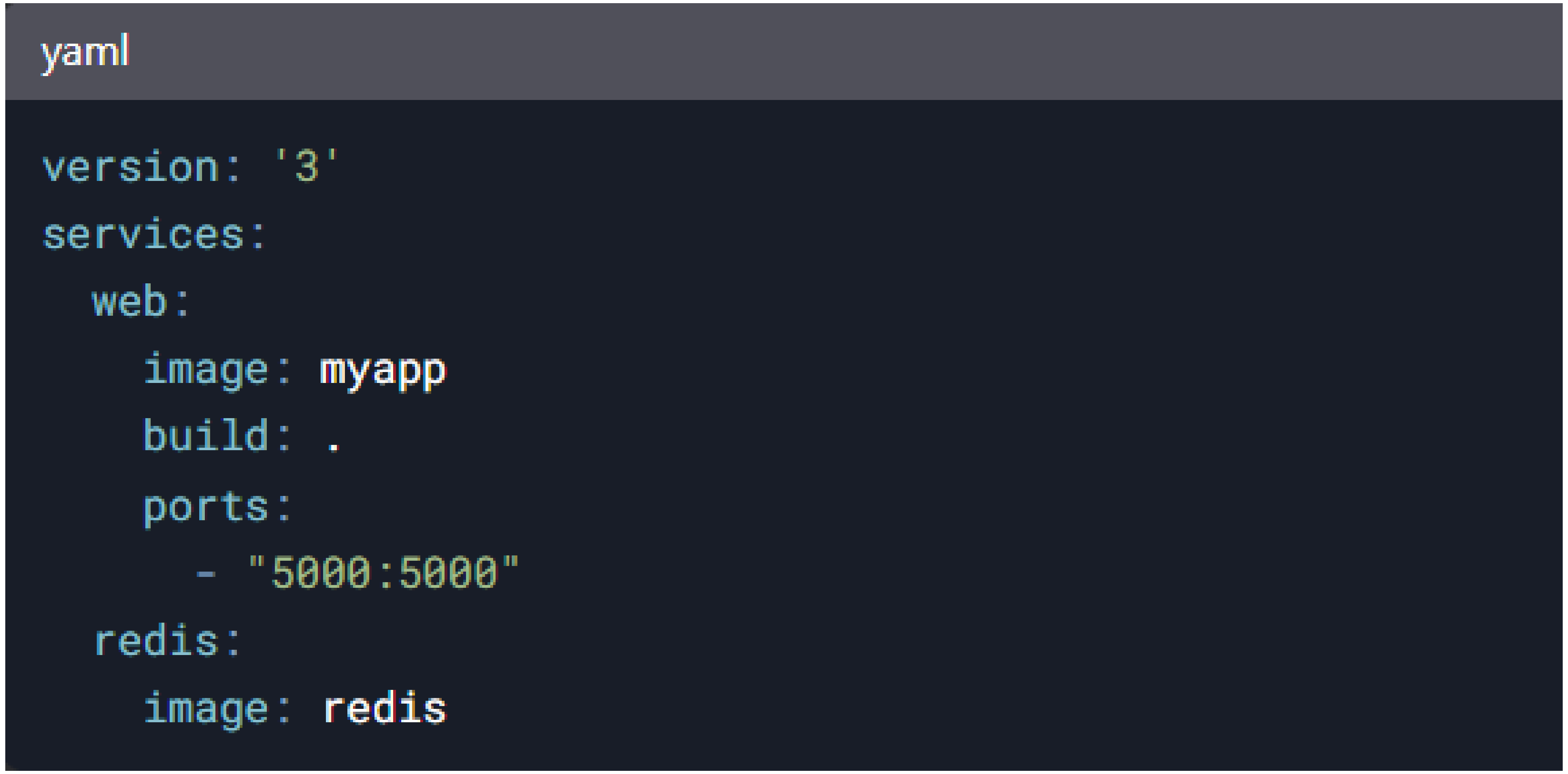
- Docker Compose is a tool for defining and running multi-container Docker applications.
- It uses a YAML file (docker-compose.yml) to configure the application's services, networks, and volumes.

- Basic Commands:

1. Starting Services:
   - **docker-compose up**: Starts all services defined in the docker-compose.yml file.
   - **docker-compose up -d**: Starts services in detached mode.
2. Stopping Services:
   - **docker-compose down**: Stops and removes containers, networks, and volumes defined in the docker-compose.yml file.
3. Viewing Logs:
   - **docker-compose logs**: Displays logs from all services.
4. Scaling Services:
   - **docker-compose scale <service_name>=<num_instances>**: Scales a service to the specified number of instances.

HOW IT WORKS

Docker Compose

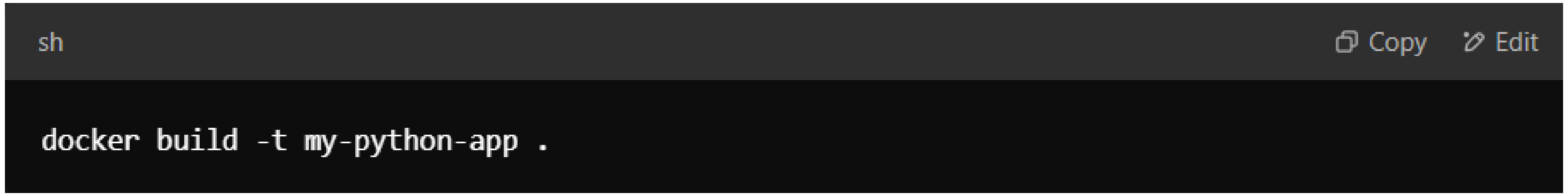


```yaml
version: '3'
services:
  web:
    image: myapp
    build: .
    ports:
      - "5000:5000"
  redis:
    image: redis
```

HOW IT WORKS

Step 3: Build the Docker Image

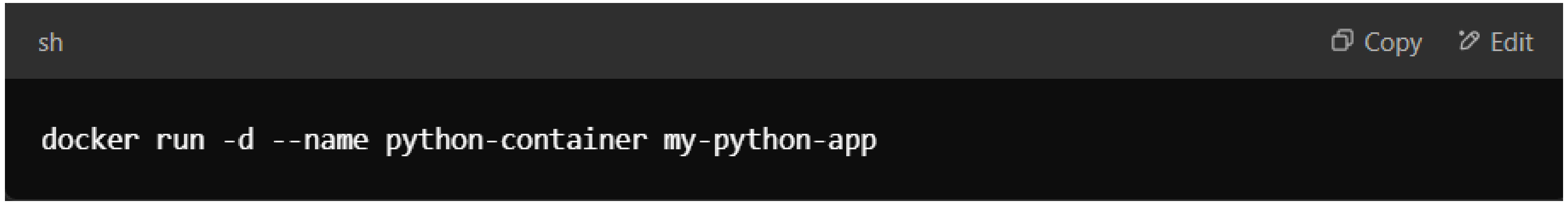
Run the following command in the terminal (inside the directory containing Dockerfile and app.py):



```sh
docker build -t my-python-app .
```

This creates an image named my-python-app.

Step 4: Run the Docker Container

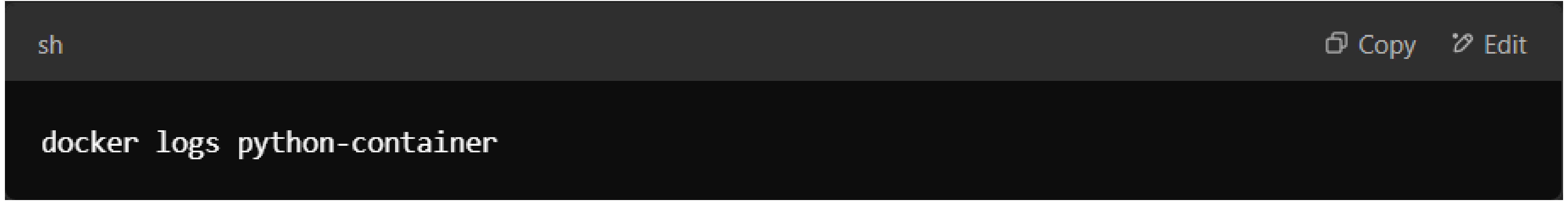


```sh
docker run -d --name python-container my-python-app
```

- -d runs the container in detached mode (in the background).
- --name python-container assigns a name to the container.

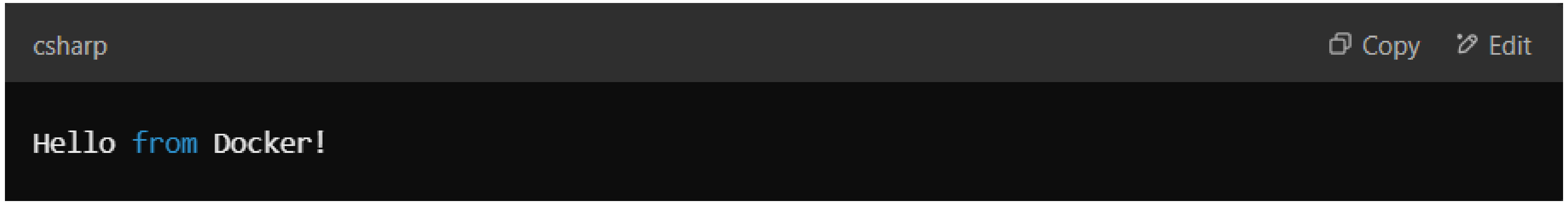
HOW IT WORKS

To see the output, use:



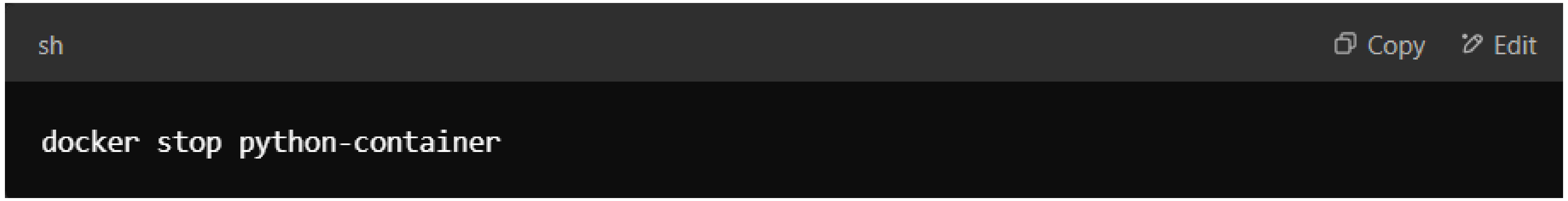
```sh
docker logs python-container
```

Output:



```csharp
Hello from Docker!
```

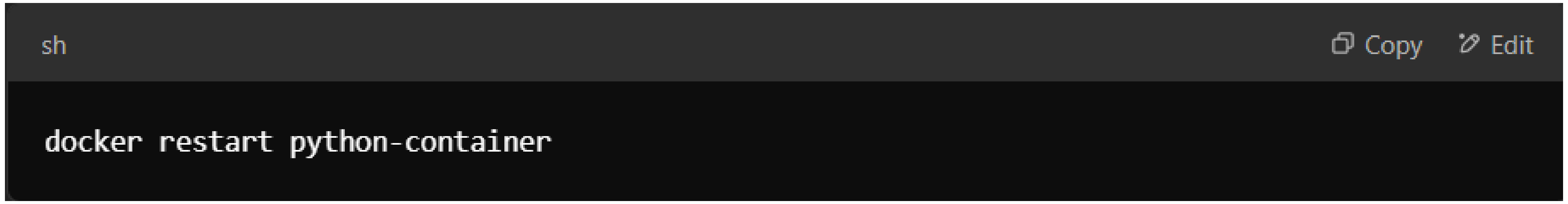
Stop the Container:



```sh
docker stop python-container
```

HOW IT WORKS

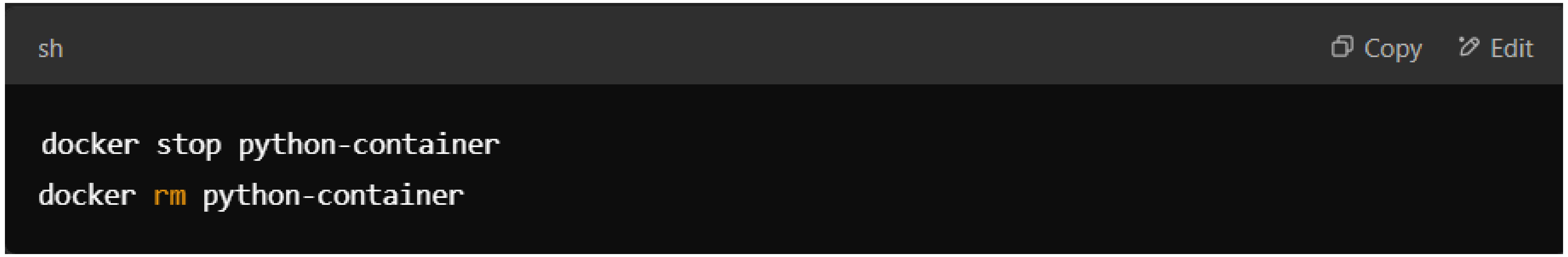
Restart the Container:



```sh
docker restart python-container
```

This will stop and start the container again.

Remove the Container:

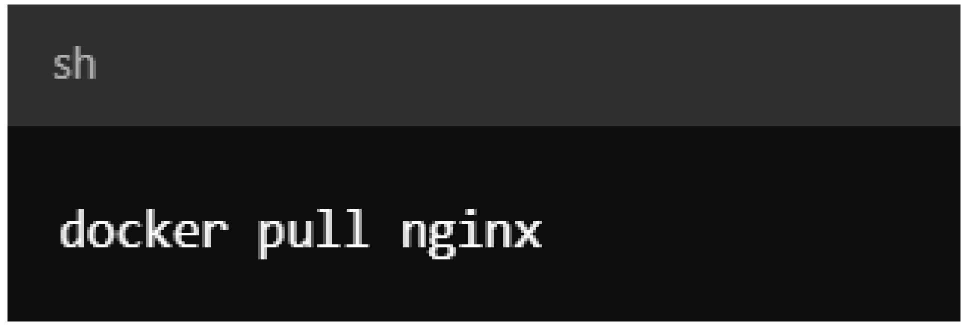


```sh
docker stop python-container
docker rm python-container
```

If you no longer need the container, stop and remove it

DOCKER COMMANDS

docker pull <image> – Download an image
This command fetches an image from Docker Hub or another container registry.
Example:



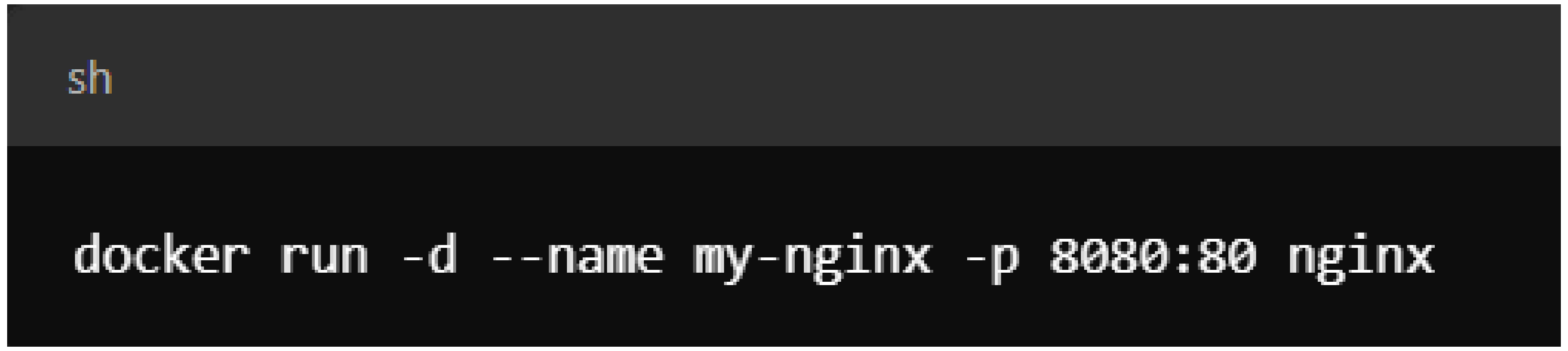
```sh
docker pull nginx
```

This downloads the latest Nginx image.

Nginx (pronounced "engine-x") is a high-performance web server that can also function as a reverse proxy, load balancer, and HTTP cache. It is widely used for hosting websites, serving static content, and handling traffic efficiently.

DOCKER COMMANDS

docker run <image> – Start a container
This creates and runs a container from the specified image.
Example:

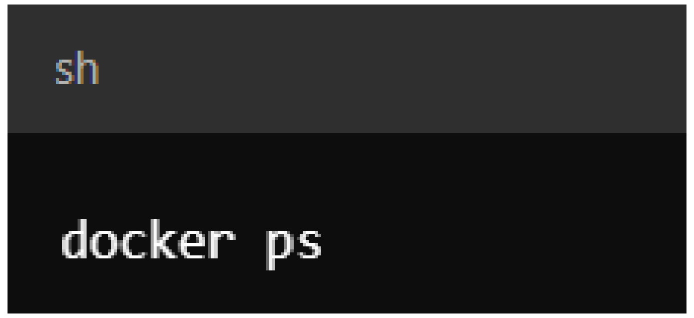


```sh
docker run -d --name my-nginx -p 8080:80 nginx
```

This runs an Nginx container, mapping port 80 inside the container to port 8080 on the host.

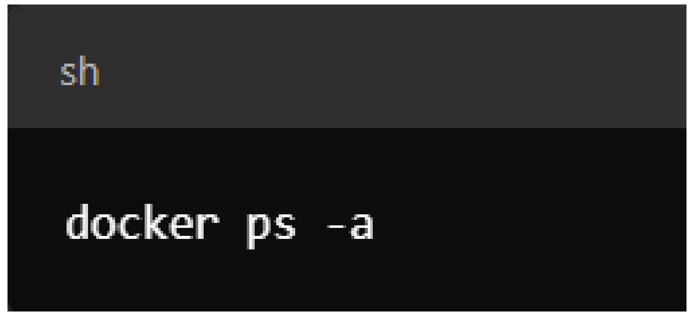
DOCKER COMMANDS

docker ps – List running containers
This command shows all currently running containers.
Example:



```sh
docker ps
```

To list all containers, including stopped ones, use:



```sh
docker ps -a
```

DOCKER COMMANDS

Pushing an Image to Docker Hub:

docker push <username>/<image_name>: Pushes an image to Docker Hub.

Example: docker push myusername/myapp

Pulling an Image from Docker Hub:

docker pull <image_name>: Pulls an image from Docker Hub.

Example: docker pull ubuntu

Viewing Logs:

docker logs <container_id>: Displays the logs of a container.

Executing Commands in a Running Container:

docker exec -it <container_id> <command>: Executes a command in a running container.

Example: docker exec -it mycontainer bash

DOCKER COMMANDS

summary of the commands:

- **docker build <path to docker file>**
- This command is used to build an image from a specified docker file
- **docker -version**
- This command is used to get the currently installed version of docker
- **docker run -it -d <image name>**
- This command is used to create a container from an image
- **docker ps**
- This command is used to list the running containers
- **docker ps -a**
- This command is used to show all the running and exited containers
- **docker stop <container id>**
- This command stops a running container
- **docker kill <container id>**
- This command kills the container by stopping its execution immediately
- **docker pull**
- This command is used to pull images from the docker repository
- **docker push <username/image name>**
- This command is used to push an image to the docker hub repository
- **Docker rmi <image-id>**
- This command is used to delete an image from local storage
- **docker rm <container id>**
- This command is used to delete a stopped container

ADVANTAGES OF DOCKER

🚀 Rapid Application Deployment – Quickly package and deploy applications across various environments.

🔥 Efficient Resource Utilization – Uses fewer resources than traditional virtual machines.

🌍 Scalability & Portability – Easily scale applications and run them anywhere (cloud, on-premises, or hybrid).

⚙️ Simplified Dependency Management – Ensures all dependencies are included within containers, eliminating "works on my machine" issues.

USE CASES

- Microservices Architecture – Docker allows developers to break applications into smaller, independent services that can run in isolated containers, making deployment and scaling easier.
- Continuous Integration & Deployment (CI/CD) – Docker helps automate the testing and deployment of applications, ensuring consistency across different environments.
- Cloud Computing – Many cloud providers support Docker, making it easy to deploy and manage containerized applications on platforms like AWS, Azure, and Google Cloud.
- Development and Testing Environments – Developers can create uniform, reproducible environments that match production, reducing the "works on my machine" problem.

SUMMARY

- Docker revolutionizes software deployment by providing a lightweight, scalable, and efficient containerization platform.
- Helps developers build, share, and run applications seamlessly across different environments.
- Enhances portability, security, and resource efficiency compared to traditional virtualization.
- Widely adopted in modern DevOps workflows, supporting CI/CD, microservices, and cloud deployments.

Thank You