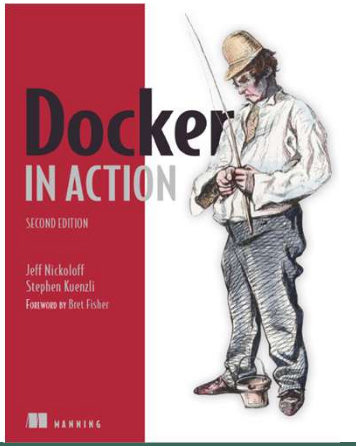
MEETING DOCKER



DOCKER ARCHITECTURE

- **Docker Host** = The computer (physical or virtual) where containers run.
- **Docker Daemon** = The background process that manages Docker containers, images, and networks. It executes the commands sent to it.
- **Docker CLI** = The Command Line Interface that allows the user to send commands to the Docker Daemon.

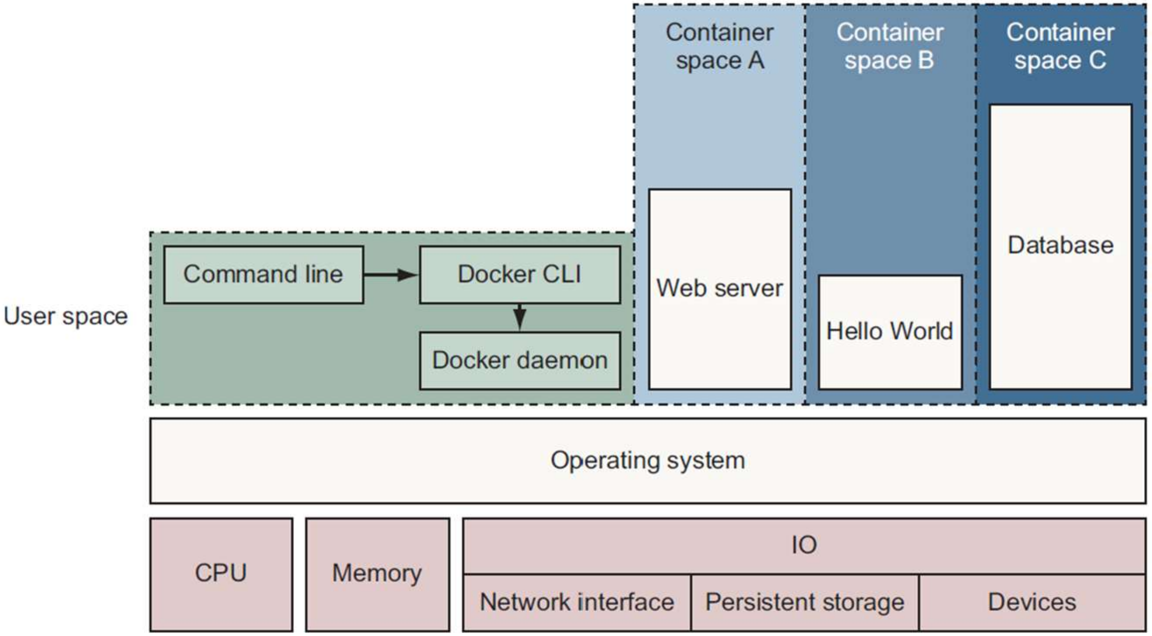
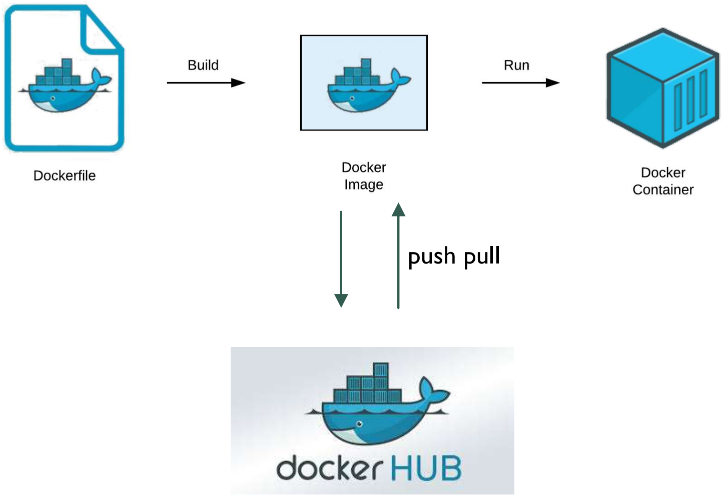


Figure 1.4 **Docker running three containers on a basic Linux computer system**

Credits: Docker in action

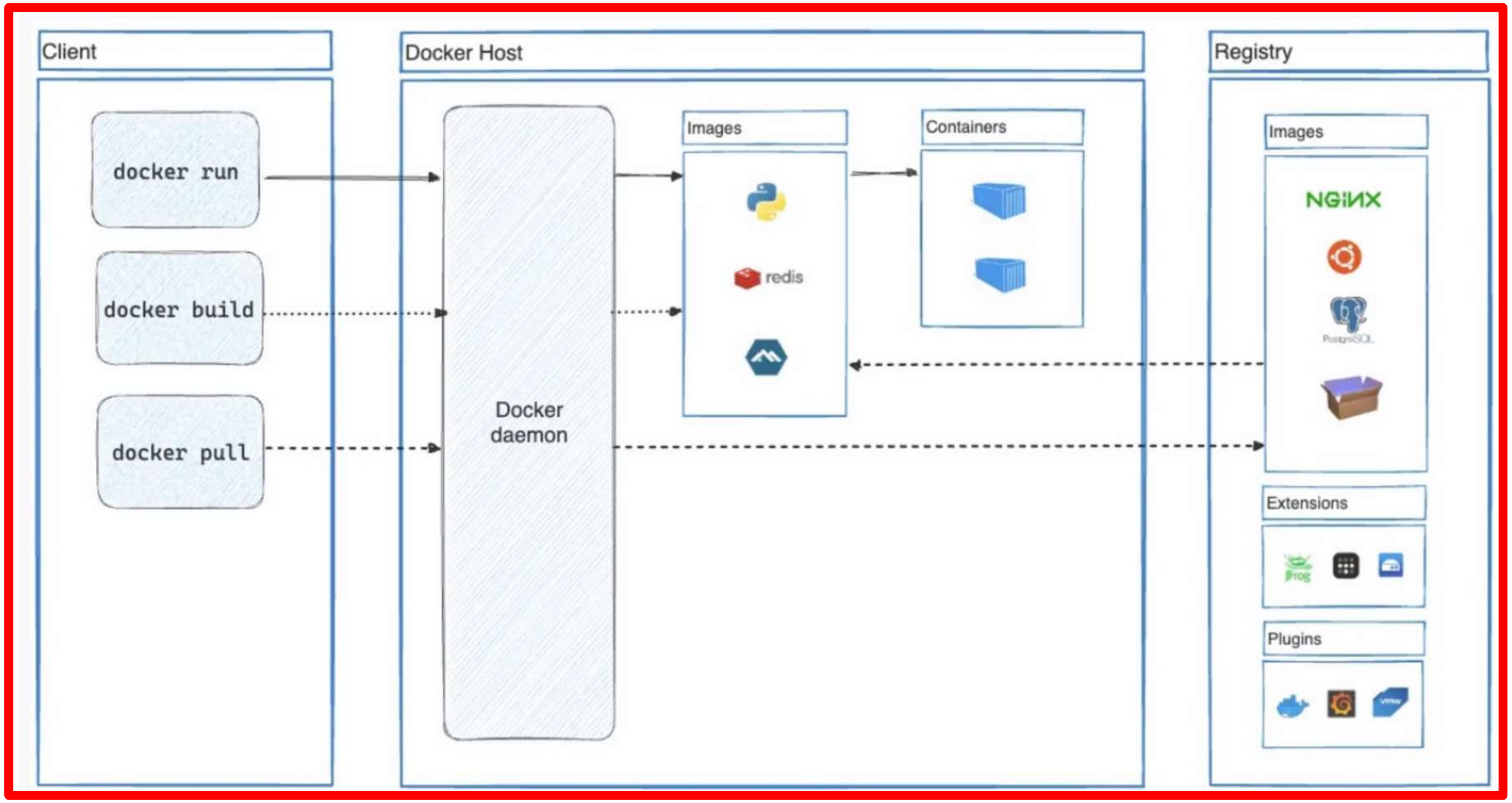
DOCKER WORKFLOW

- Docker activities can be classified as
- **build**, **run**, **push** and **pull**
- A container is a running image
- container : image = process : program
- Registry = place where images are stored, can be public or private
- Docker hub = registry maintained by Docker Inc.

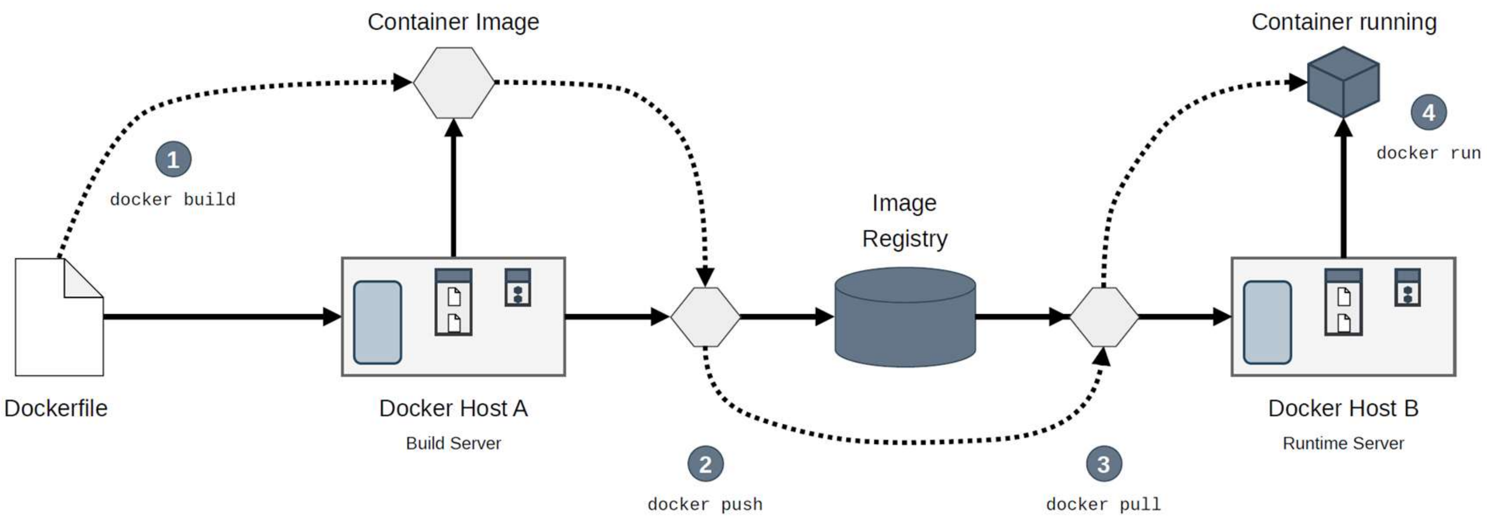


DOCKER ARCHITECTURE

- Plugins: add functionalities of the (core) daemon, e.g. buildx
- Extensions: functionalities seen by users



DOCKER WORKFLOW

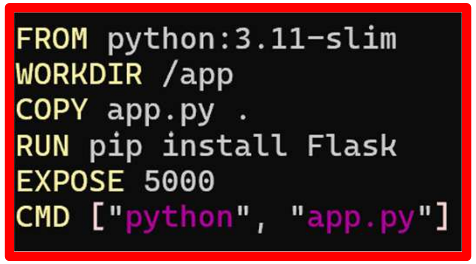


1. Building Docker image from Dockerfile on Host A (Build Server)
2. Pushing image to registry after successful build
3. Docker Host B (Runtime Server) pulls image form registry
4. Docker Host B runs container

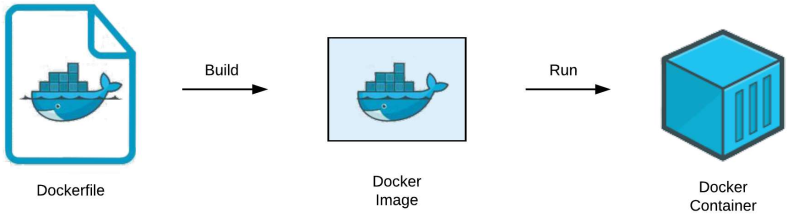
Credits: Henry-Norbert Cocos, Cloud Computing Technological foundations of Cloud Computing

DOCKER IMAGES

- Read-only template used to create a docker image
- Build means to create an image that includes all the dependencies of an app
- **$docker build**: command that reads instructions from a **docker file**
- Docker file is a list of instructions, each that follows a simple syntax
- INSTRUCTION argument



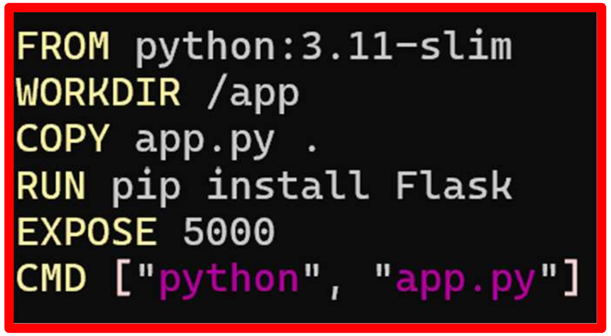
```
FROM python:3.11-slim
WORKDIR /app
COPY app.py .
RUN pip install Flask
EXPOSE 5000
CMD ["python", "app.py"]
```



FROM specify parent image (mandatory)
WORKDIR specify working directory
COPY copy files from host and put them into image
RUN to execute specified command
EXPOSE to set specified network port exposed by container
CMD ["", ""]: to provide default command the container will run

https://docs.docker.com/get-started/docker-concepts/buildingimages/writing-a-dockerfile/

DOCKER FILE



```
FROM python:3.11-slim
WORKDIR /app
COPY app.py .
RUN pip install Flask
EXPOSE 5000
CMD ["python", "app.py"]
```

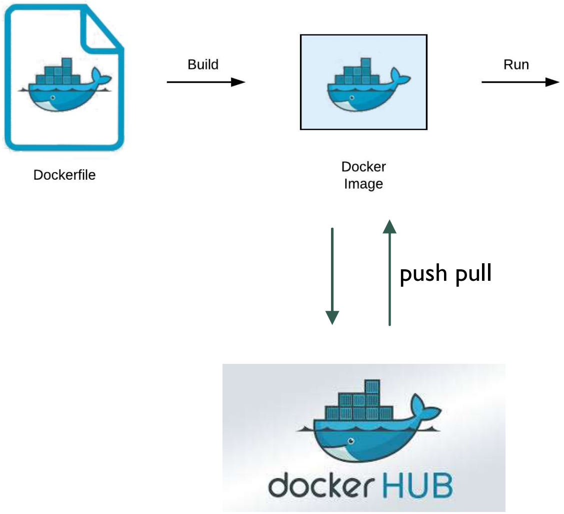
1. parent image (python:3.11-slim) pulled from Docker Hub
2. create and make current it the current directory in the container (/app)
3. copy the file app.py from the host into the current directory (a/pp)
4. run the command pip install Flask
5. make port 5000 accessible (just information)
6. execute this command after the build

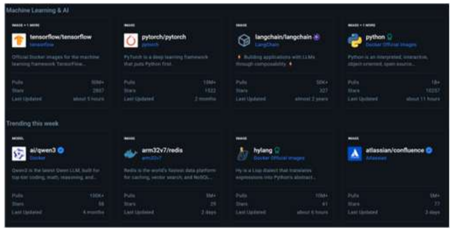
2.,3.,4. add a layer to the base image file identified by 1. (from) [see next slides]
5. Just information that are included in the image
6. Executed only after the build, it doesn't add any layer, modifies metadata

[REGISTRYHOST:PORT/][USERNAME/]NAME[:TAG]

REGISTRY

- Images can also be pulled from a registry or pushed to a registry
- A *named repository* is a named bucket of images. The name is similar to a URL.
- For example, AWS has a public registry : : https://gallery.ecr.aws
- Docker hub: https://hub.docker.com/





Machine Learning & AI

IMAGE • 1 MORE
tensorflow/tensorflow
tensorflow
Official Docker images for the machine learning framework TensorFlow...
Pulls 50M+
Stars 2807
Last Updated about 5 hours

IMAGE
pytorch/pytorch
pytorch
PyTorch is a deep learning framework that puts Python first.
Pulls 10M+
Stars 1522
Last Updated 2 months

IMAGE
langchain/langchain
LangChain
🦜 Building applications with LLMs through composability 🦜
Pulls 50K+
Stars 327
Last Updated almost 2 years

IMAGE • 1 MORE
python
Docker Official Images
Python is an interpreted, interactive, object-oriented, open-source...
Pulls 1B+
Stars 10237
Last Updated about 11 hours

Trending this week

MODEL
ai/qwen3
Docker
Qwen3 is the latest Qwen LLM, built for top-tier coding, math, reasoning, and...
Pulls 100K+
Stars 58
Last Updated 4 months

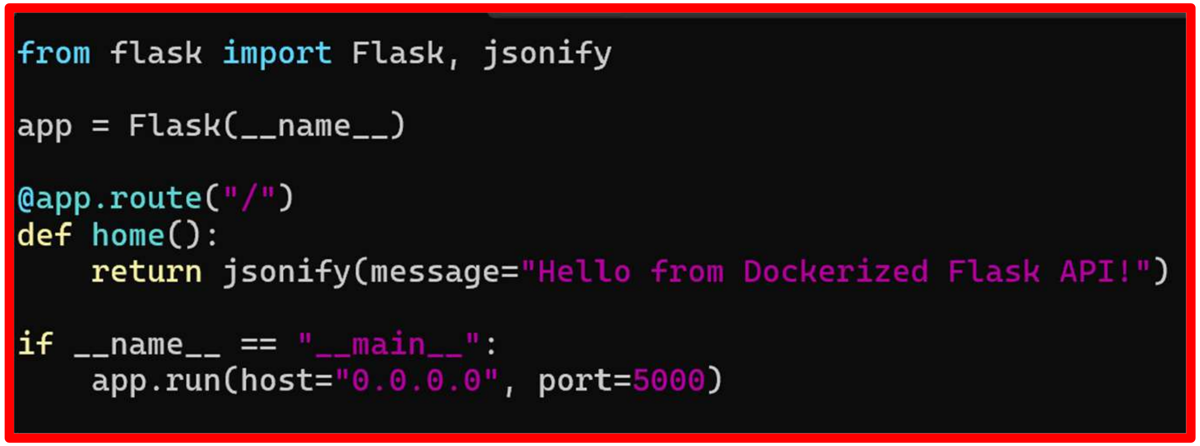
IMAGE
arm32v7/redis
arm32v7
Redis is the world's fastest data platform for caching, vector search, and NoSQL...
Pulls 5M+
Stars 25
Last Updated 2 days

IMAGE
hylang
Docker Official Images
Hy is a Lisp dialect that translates expressions into Python's abstract...
Pulls 10M+
Stars 41
Last Updated about 6 hours

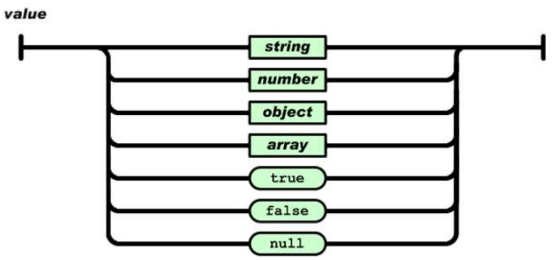
IMAGE
atlassian/confluence
Atlassian
Pulls 5M+
Stars 77
Last Updated 3 days

APP.PY

- **Flask** = open-source micro-framework for web application and web-api
- **jsonify** = function to convert Python objects (usually dictionaries) into JSON-formatted responses
- **Annotation**: app.route is an ‘annotation’ that configure the server to route a HTTP get request on /, to the following method (get_data in this example)

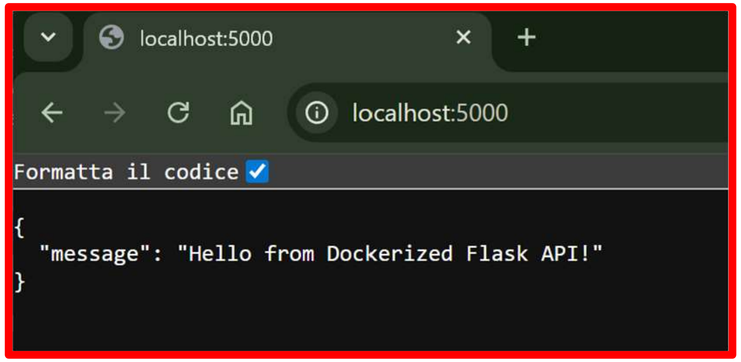


```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/")
def home():
    return jsonify(message="Hello from Dockerized Flask API!")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```



BUILD THE IMAGE AND RUN IT

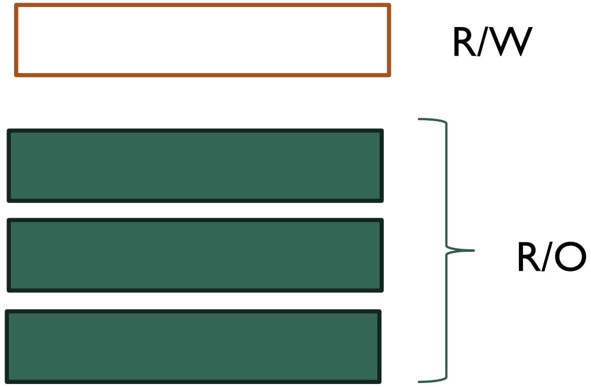
- **docker buildx build --load -t simple-flask-api .**
- **docker run -d -p 5000:5000 simple-flask-api:latest**
- **docker ps**



```
{
  "message": "Hello from Dockerized Flask API!"
}
```

IMAGE FILE

- Each image consists of a stack of layers
- The first set of layers is read/only, which allows efficient sharing among other images
- The last layer is R/W and created at each run (containers are then stateless)



- Layers are created by commands in the docker file
- CMD doesn’t create a layer; it specifies what command to run within the container
- One R/W layer created after RUN (doesn’t persist)

LIFECYCLE OF A CONTAINER

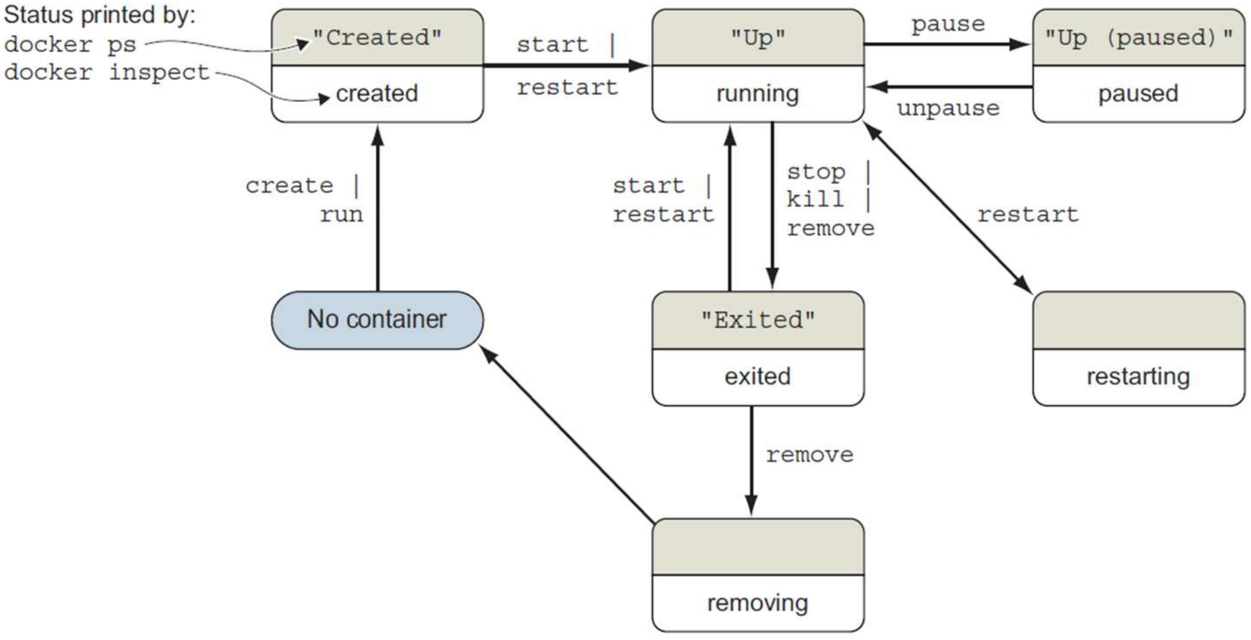


Figure 2.3 The state transition diagram for Docker containers

IMAGE HANDLING

- List the image on the host (local repository)
- **$ docker images**
- **$ docker image ls**
- **$ docker rmi *imageid***

- Remove an image (by id or name)

CONTAINER MANAGEMENT

- **$ docker ps**
- **$ docker stop containerid**
- **$ docker start containerid**
- **$ docker kill containerid**
- **$ docker rm containerid**

DOCKER NETWORKING

- Communication with other containers in the same host or to non-Docker processes
- By default, Docker includes three networks, and each is provided by a different driver.
- The network named bridge is the default network provided by a bridge driver.
- The bridge driver provides inter container connectivity for all containers running on the same machine.

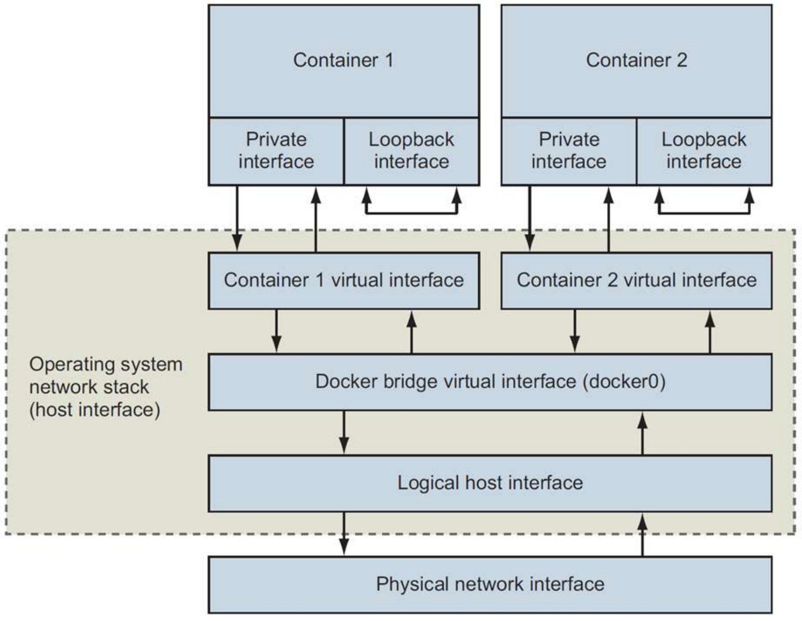
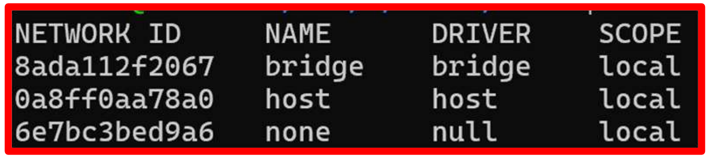

Container 1
Container 2
Private interface
Loopback interface
Private interface
Loopback interface
Container 1 virtual interface
Container 2 virtual interface
Operating system network stack (host interface)
Docker bridge virtual interface (docker0)
Logical host interface
Physical network interface

Figure 5.4 The default local Docker network topology and two attached

DOCKER NETWORKING

- The host driver allows containers interact with the host’s network stack like uncontained processes.
- **$docker run the -p** flag (publish) makes a port available to outside
- The none network uses the null driver.
- Containers attached to the none network will not have any network connectivity outside themselves.

$ docker network ls



```
NETWORK ID     NAME      DRIVER    SCOPE
8ada112f2067   bridge    bridge    local
0a8ff0aa78a0   host      host      local
6e7bc3bed9a6   none      null      local
```

DOCKER STORAGE

- Recall in linux the file system Is a tree and some point of the tree one can mount a device
- To persist data, a container can mount part of the host file system tree to the container tree (bind-mount)
- This however ties the container to a specific host configuration
- The preferred option is to use docker volumes generated and managed by docker
- Volumes provide container-independent data management
- The last option is to use in-memory fs (cache)

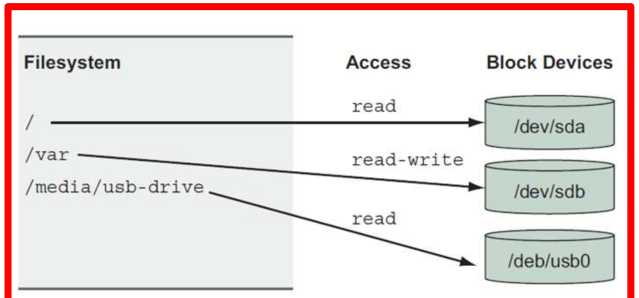
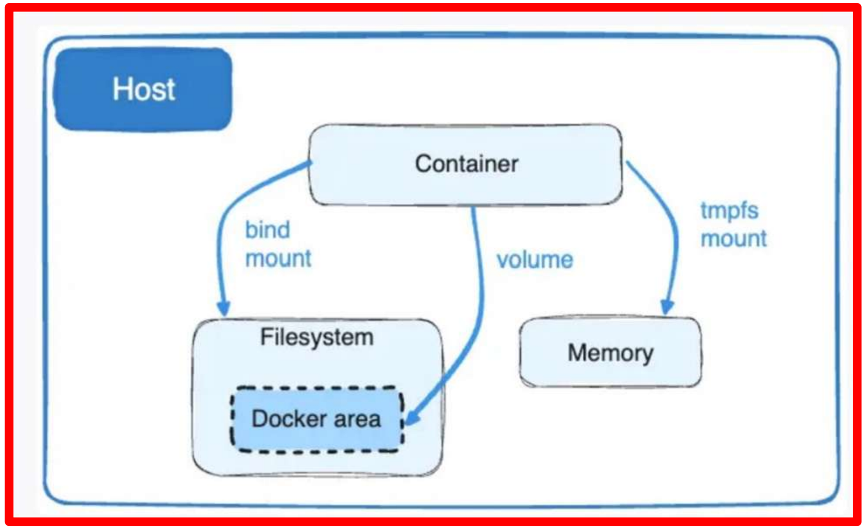


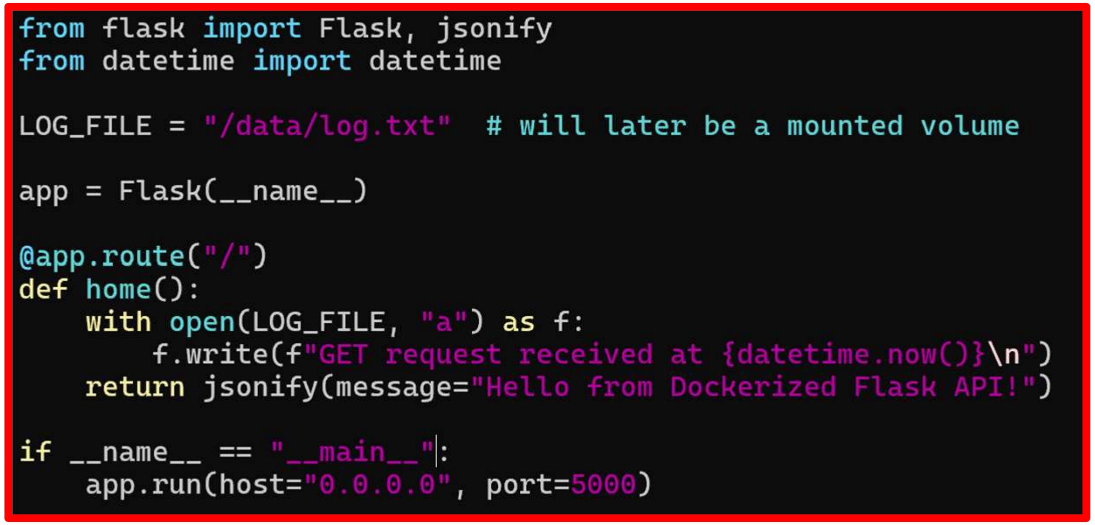
Figure 4.1 Storage devices attached to a filesystem tree at their mount point



DOCKER VOLUME

- Commands to manage a container
- Create a volume: **$ docker volume create *name***
  - will create a directory to store the contents of a volume somewhere in a part of the host filesystem under control of the Docker engine.
  - List volumes: **$ docker volume ls**
- Inspect volume: **$ docker volume inspect *name***
- Remove volume: **$ docker volume rm *name***
- Mount a volume: **$ docker run … -v *source* : *destination***

HANDS-ON



```python
from flask import Flask, jsonify
from datetime import datetime

LOG_FILE = "/data/log.txt"  # will later be a mounted volume

app = Flask(__name__)

@app.route("/")
def home():
    with open(LOG_FILE, "a") as f:
        f.write(f"GET request received at {datetime.now()}\n")
    return jsonify(message="Hello from Dockerized Flask API!")

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

-v mount volume

-t image name

-d image runs detached (bg)

-p port mapping

docker buildx build -t simple-flask . –load
docker run -d -p 5000:5000 -v $(pwd)/data:/data simple-flask

MULTI-CONTAINER APPLICATIONS

- **Docker Compose**: tool to manage multiple containers on the same host
  - Coordinate a composition of containers on the same host, communicating via bridge (private network among containers)
  - The composition is defined into a text file (with YAML formatting)
- **Docker swarm** or **Kubernetes** (**K8**): container management on multiple hosts

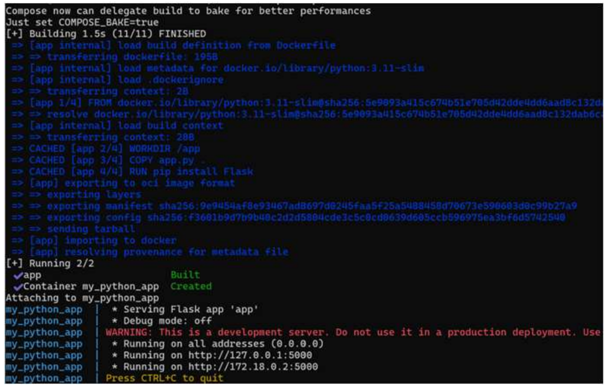
DOCKER COMPOSE

- Tool for defining and running multi-container Docker applications
  - https://docs.docker.com/compose/
- Allows us to coordinate a composition of multiple containers running on a single host (i.e., single Docker engine)
- User expresses the containers to be instantiated at once and their relationships

DOCKER SERVICE

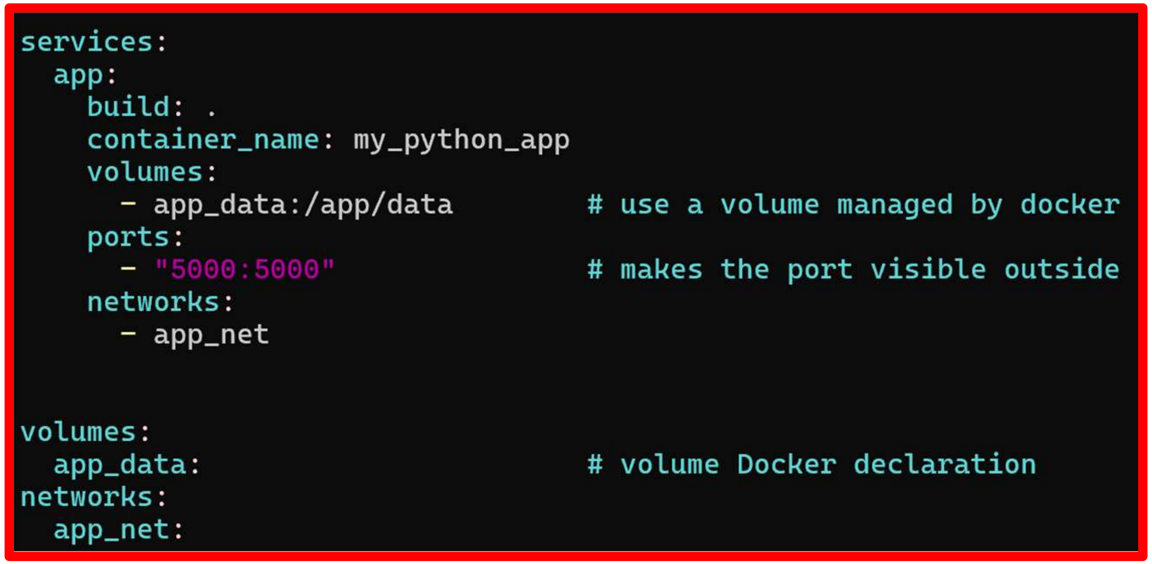
- **Service**: abstract definition of computing resources within application
- Build: means used the Docker file in the current directory to build the image
- Call the container my_python_app

- **$ docker compose up --build**



```
Compose now can delegate build to bake for better performances
Just set COMPOSE_BAKE=true
[+] Building 1.5s (11/11) FINISHED
 => [app internal] load build definition from Dockerfile
 => => transferring dockerfile: 195B
 => [app internal] load metadata for docker.io/library/python:3.11-slim
 => [app internal] load .dockerignore
 => => transferring context: 2B
 => [app 1/4] FROM docker.io/library/python:3.11-slim@sha256:5e9893a415c674b51e705d42dde4dd6aad8c132d
 => => resolve docker.io/library/python:3.11-slim@sha256:5e9093a415c674b51e705d42dde4dd6aad8c132dab6c
 => [app internal] load build context
 => => transferring context: 28B
 => CACHED [app 2/4] WORKDIR /app
 => CACHED [app 3/4] COPY app.py .
 => CACHED [app 4/4] RUN pip install Flask
 => [app] exporting to oci image format
 => => exporting layers
 => => exporting manifest sha256:9e9454af8e93467ad8697d0245faa5f25a5488458d70673e590603d0c99b27a9
 => => exporting config sha256:f3601b9d7b9b40c2d2d5804cde3c5c0cd0639d605ccb596975ea3bf6d5742540
 => => sending tarball
 => [app] importing to docker
 => [app] resolving provenance for metadata file
[+] Running 2/2
 ✔app                           Built
 ✔Container my_python_app       Created
Attaching to my_python_app
my_python_app  |  * Serving Flask app 'app'
my_python_app  |  * Debug mode: off
my_python_app  | WARNING: This is a development server. Do not use it in a production deployment. Use
my_python_app  |  * Running on all addresses (0.0.0.0)
my_python_app  |  * Running on http://127.0.0.1:5000
my_python_app  |  * Running on http://172.18.0.2:5000
my_python_app  | Press CTRL+C to quit
```

docker-compose.yaml



```
services:
  app:
    build: .
    container_name: my_python_app
    volumes:
      - app_data:/app/data          # use a volume managed by docker
    ports:
      - "5000:5000"                 # makes the port visible outside
    networks:
      - app_net


volumes:
  app_data:                         # volume Docker declaration
networks:
  app_net:
```

DOCKER COMPOSE

- To start Docker composition (background -d): **$ docker compose up -d**
- By default, Docker Compose looks for compose.yaml in working directory
- To stop running containers: **$ docker compose stop**
- To bring composition down, removing everything $ **docker compose down**

Play with Docker

A simple, interactive and fun playground to learn Docker