Container & Docker 101

Ittipon (Art) Rassameeroj

Faculty of ICT, Mahidol University

This is for the training by MUIT on 27 Nov 2025 only.

มหาวิทยาลัยมหิดล
Mahidol University

ห้ามมิให้ผู้เรียนทำซ้ำ ดัดแปลง หรือใช้ประโยชน์จากงานอันมีลิขสิทธิ์ปรากฏอยู่ในระบบการศึกษาอิเล็กทรอนิกส์ (E-Learning) ของมหาวิทยาลัย ไม่ว่าจะทั้งหมดหรือบางส่วน โดยไม่ได้รับอนุญาตจากมหาวิทยาลัย นอกเหนือจากการศึกษาส่วนบุคคล ทั้งนี้ การทำซ้ำ ดัดแปลง หรือเผยแพร่ต่อสาธารณชนซึ่งงานอันมีลิขสิทธิ์ **จะมีโทษปรับตั้งแต่ 20,000 บาท ถึง 200,000 บาท และหากเป็นการกระทำเพื่อการค้า จะมีโทษจำคุกตั้งแต่ 6 เดือน ถึง 10 ปี หรือปรับตั้งแต่ 100,000 บาท ถึง 800,000 บาท หรือทั้งจำทั้งปรับ**

Unless allowed by Mahidol University, the User shall not copy, modify, or exploit in part or in whole of the copyrighted materials on the Platform, other than for your own individual study. The copy, modify or communication to public of the copyrighted materials shall be inflicted with a fine from 20,000 Baht up to 200,000 Baht. If the offence is committed with the commercial purpose, the offender shall be inflicted with imprisonment for a term from 6 months up to 10 years or a fine from 100,000 Baht up to 800,000 Baht or both imprisonment and fine.

Instructor: อิทธิพล (อาร์ท) รัศมีโรจน์

- https://ittiponr.github.io
- Lecturer and Assistant Dean for Technology Infrastructure at Faculty of ICT, Mahidol University
- Education
  - Ph.D. in Computer Science, University of California, Davis, USA
  - M.S. and B.S. in Computer Science, Mahidol University
- Certificates
  - Certified Information Systems Security Professional (CISSP)
  - Certified Kubernetes Administrator (CKA)
  - Certificate of Cloud Security Knowledge (CCSK)
  - Certified Ethical Hacker (CEH)
  - AWS Certified Solutions Architect - Associate
  - AWS Certified Cloud Practitioner
  - CompTIA Security+
  - Microsoft Certified: Security, Compliance, and Identity Fundamentals (SC-900)
  - CQI and IRCA Certified ISO/IEC 27001:2022 Lead Auditor
  - Certified Data Protection Officer (C-DPO)
- Research areas
  - Computer and networked systems
  - Cyber security
  - Operating/distributed systems, cloud/parallel/high-performance computing
- Teaching classes
  - Computer and Network Security
  - Parallel and High-Performance Computing
  - Cloud Security
  - Cloud Computing and Service-Oriented
  - Introduction to Linux
  - Penetration Testing and System Hardening
  - DevOps

Agenda

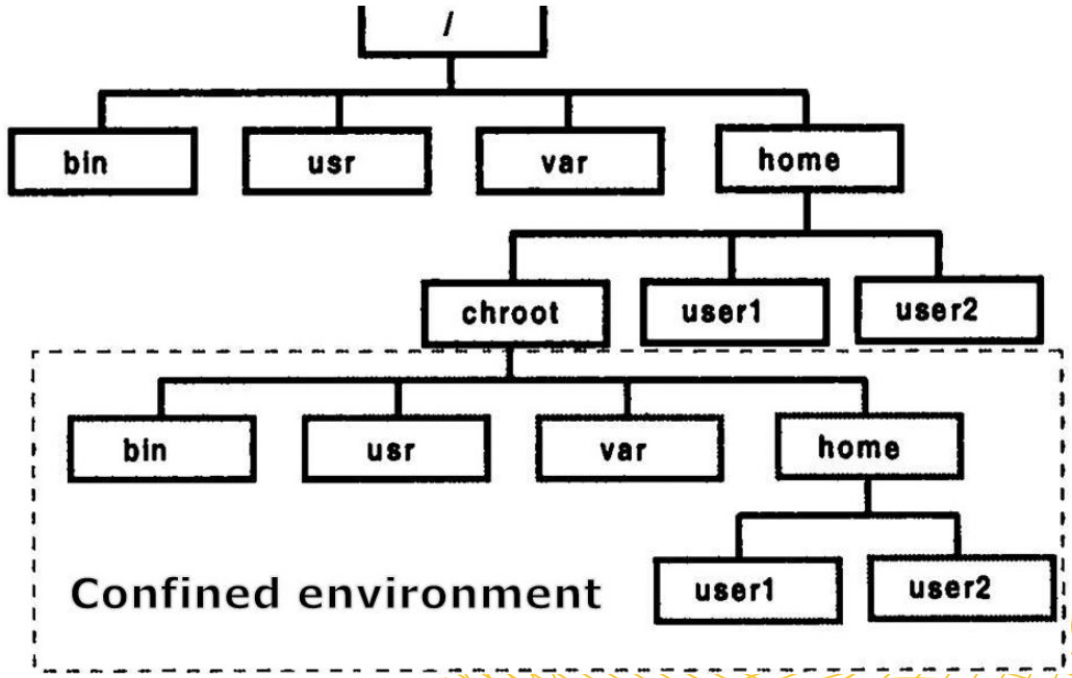
- Virtualization evolution
- Introduction to container technology
- Introduction to Docker
- Docker components & lifecycle
- Docker architecture and concepts
- Basic commands
- Dockerfile
- Docker network
- Layered architecture
- File system in Docker
- Volumes
- Docker Compose
- Containerize a single-container app
- Docker and AI

Lab Preparation

- Install Docker Desktop: https://www.docker.com/products/docker-desktop/
  - For Windows, install `WSL2` manually (if needed)
    - As admin, run this command in PowerShell: **`wsl --install`** then reboot
    - Run this command in PowerShell: **`wsl --set-default-version 2`**
- Signup at https://www.docker.com

Virtualization Evolution

- 1960s: Virtualization technology in mainframe.
- 1979: Chroot in V7 Unix and BSD provided file system mail in 1982
- 2001: Linux Vserver
- 2004: Solaris Containers
- 2005: OpenVZ
- 2006: Control Groups (cgroups)
- 2008: LinuX Containers (LXC)
- 2013: Docker
- 2017: Kubernetes

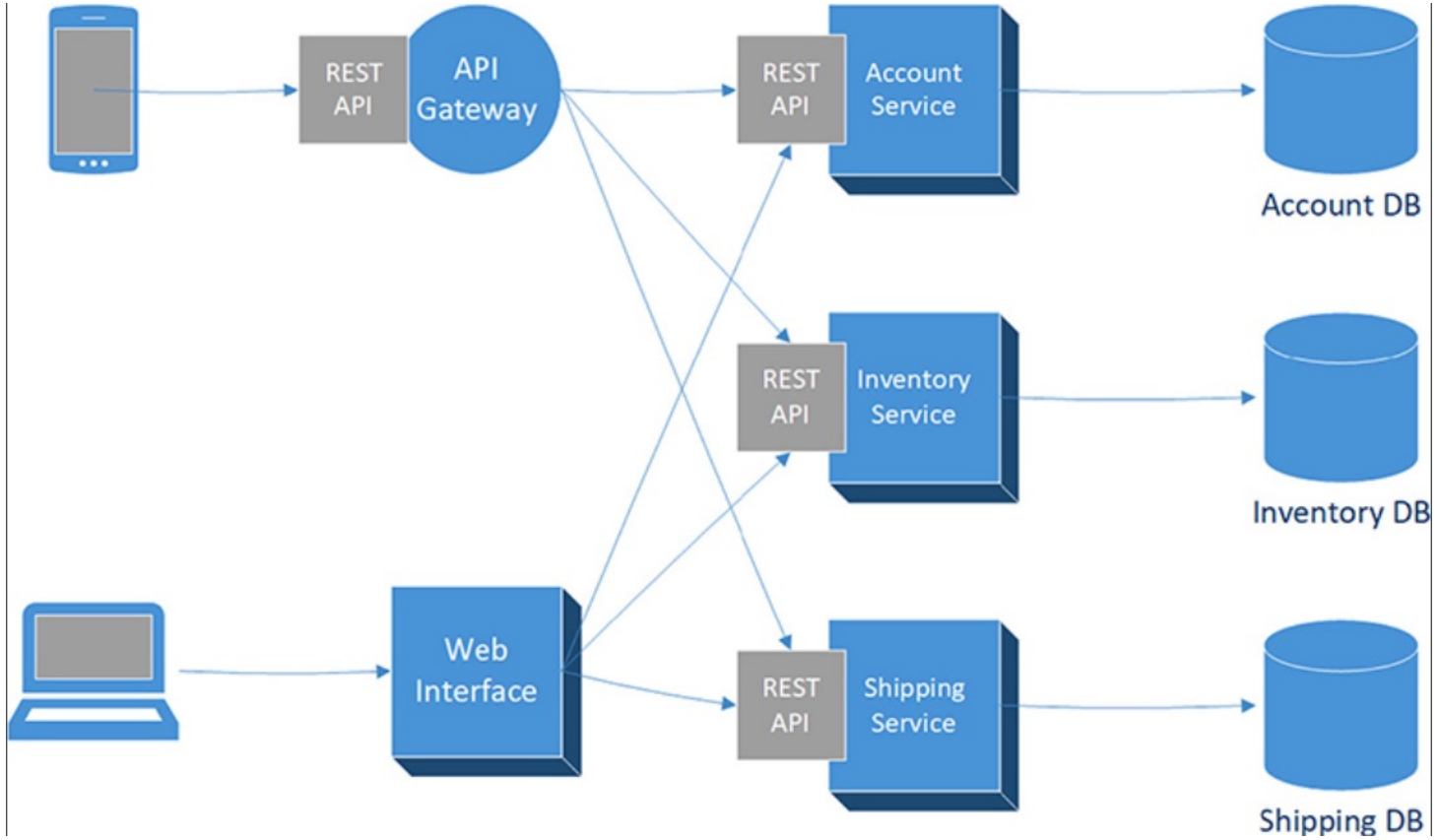


Credit: Nutanix

Why Container Is Important?

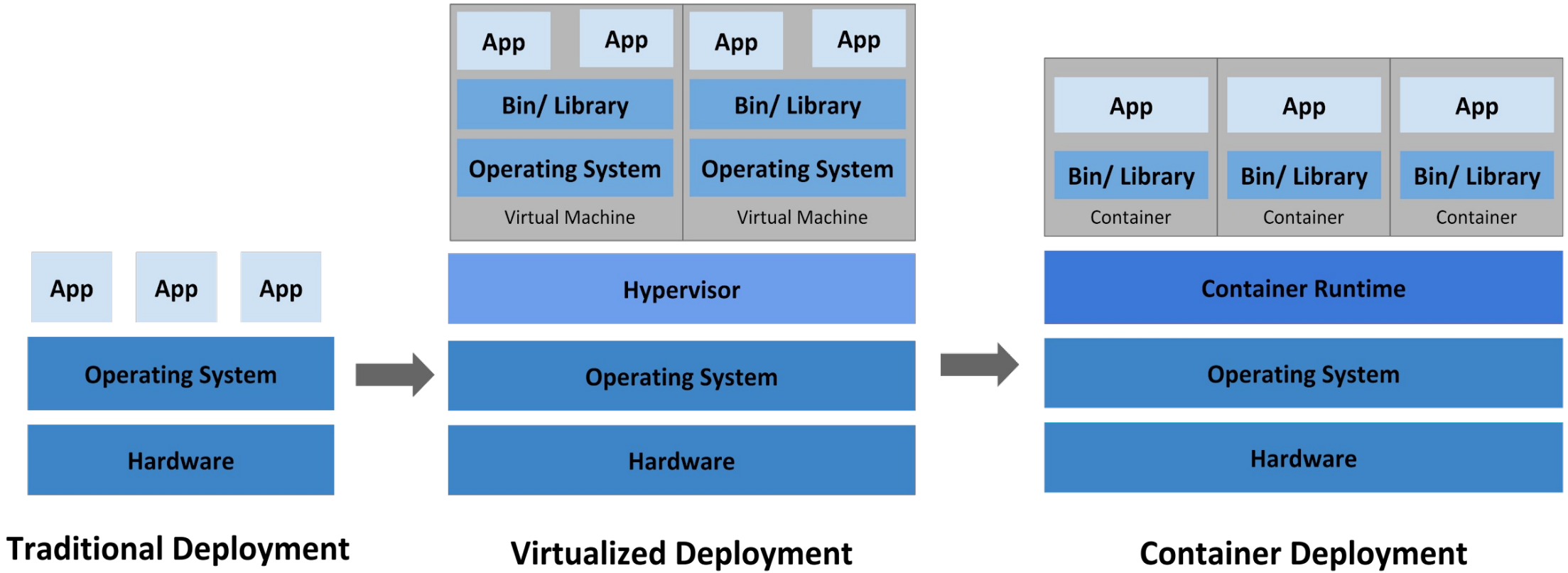
- The end of ”It works on my machine!”
  - Consistency and portability
- Faster development and deployment
- Isolation
- Resource efficiency
- Scalability and cloud readiness

Microservices

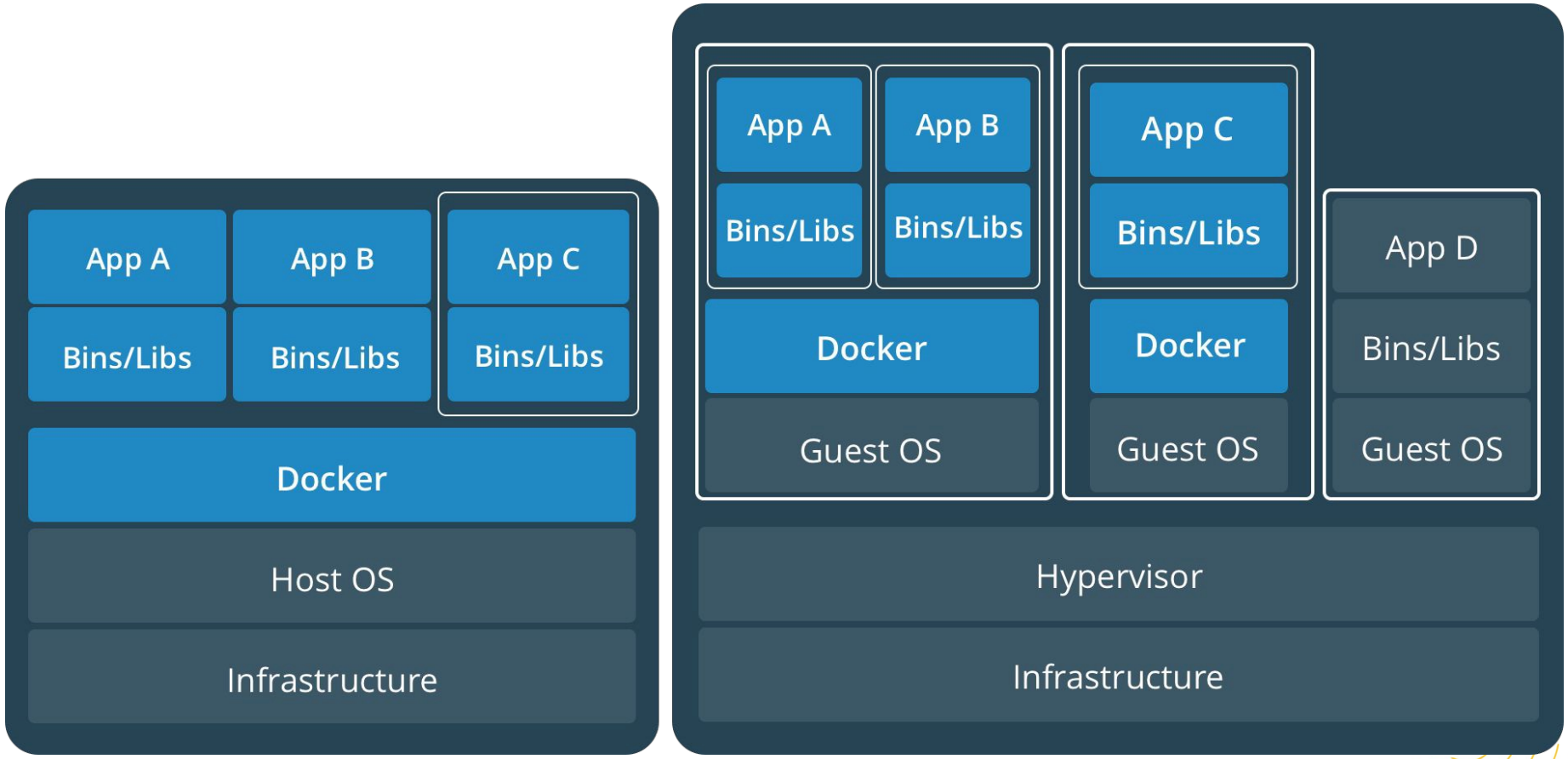


Source: Graham Thompson

VM vs Container

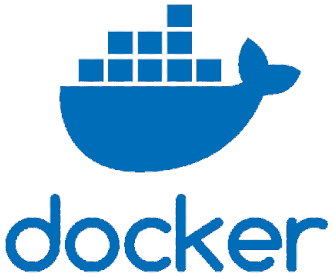


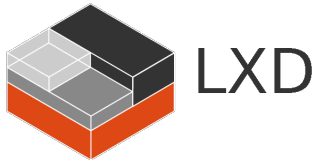
VM and Container Together



Credit: Jirayut Nimsaeng, Opsta Co. Ltd.

Container Platforms





- Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.
- Docker is a platform designed to help developers build, share, and run container applications.
  - ”We handle the tedious setup, so you can focus on the code.”
- The service has both free and premium tiers.
- The software that hosts the containers is called Docker Engine.
- It was first started in 2013 and is developed by Docker, Inc.

Docker

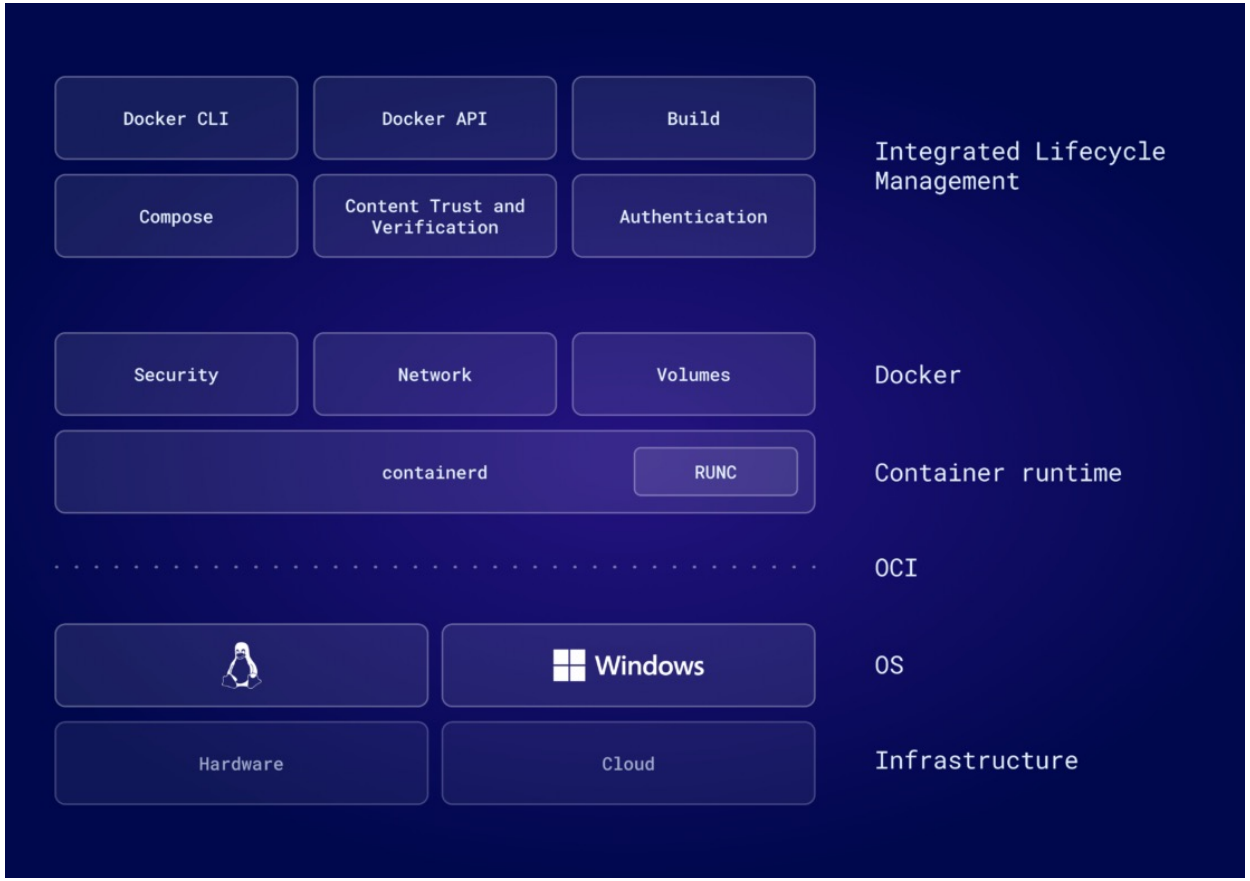
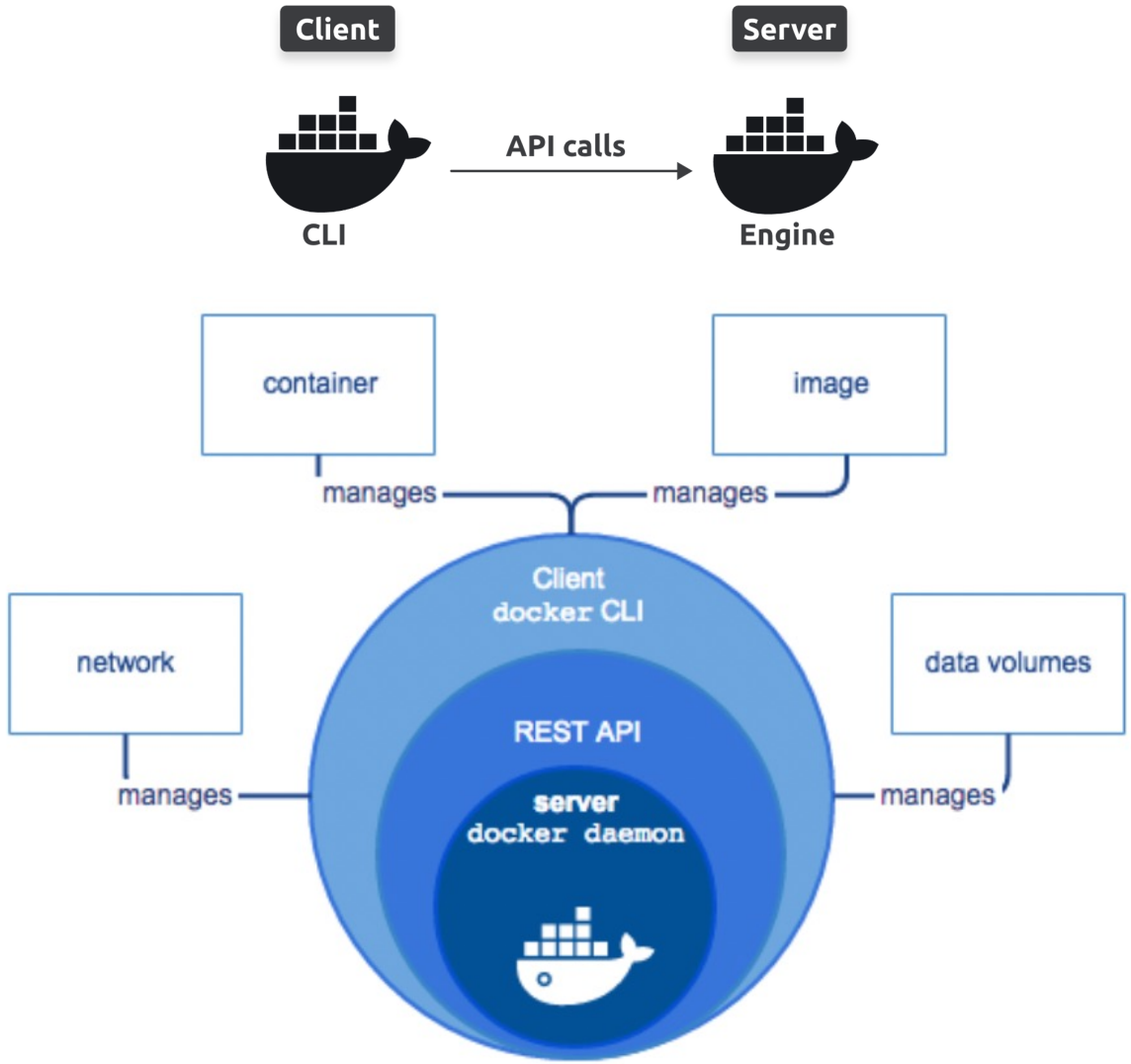


Image source: https://www.docker.com/blog/containerd-vs-docker/
OCI = Open Container Initiative: https://opencontainers.org/

Docker Components

- Docker Engine
  - Docker Daemon
- REST API
- Docker client (CLI)
- Image Registry
  - Public
  - Private



Docker Concept

Image

The basis of a Docker container. The content at rest. build from Dockerfile.

Container

The image when it is ‘running.’ The standard unit for app service

Engine

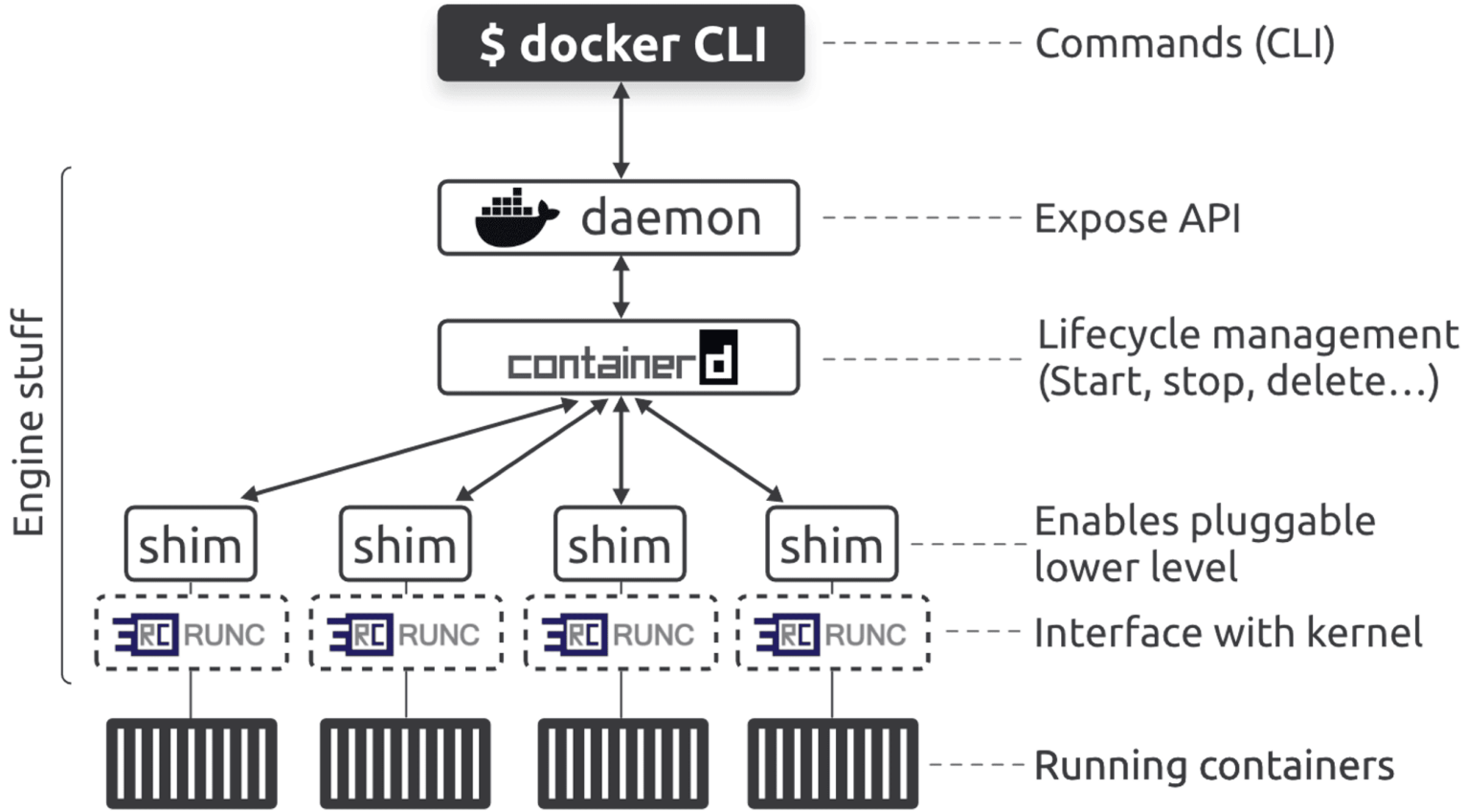
The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry

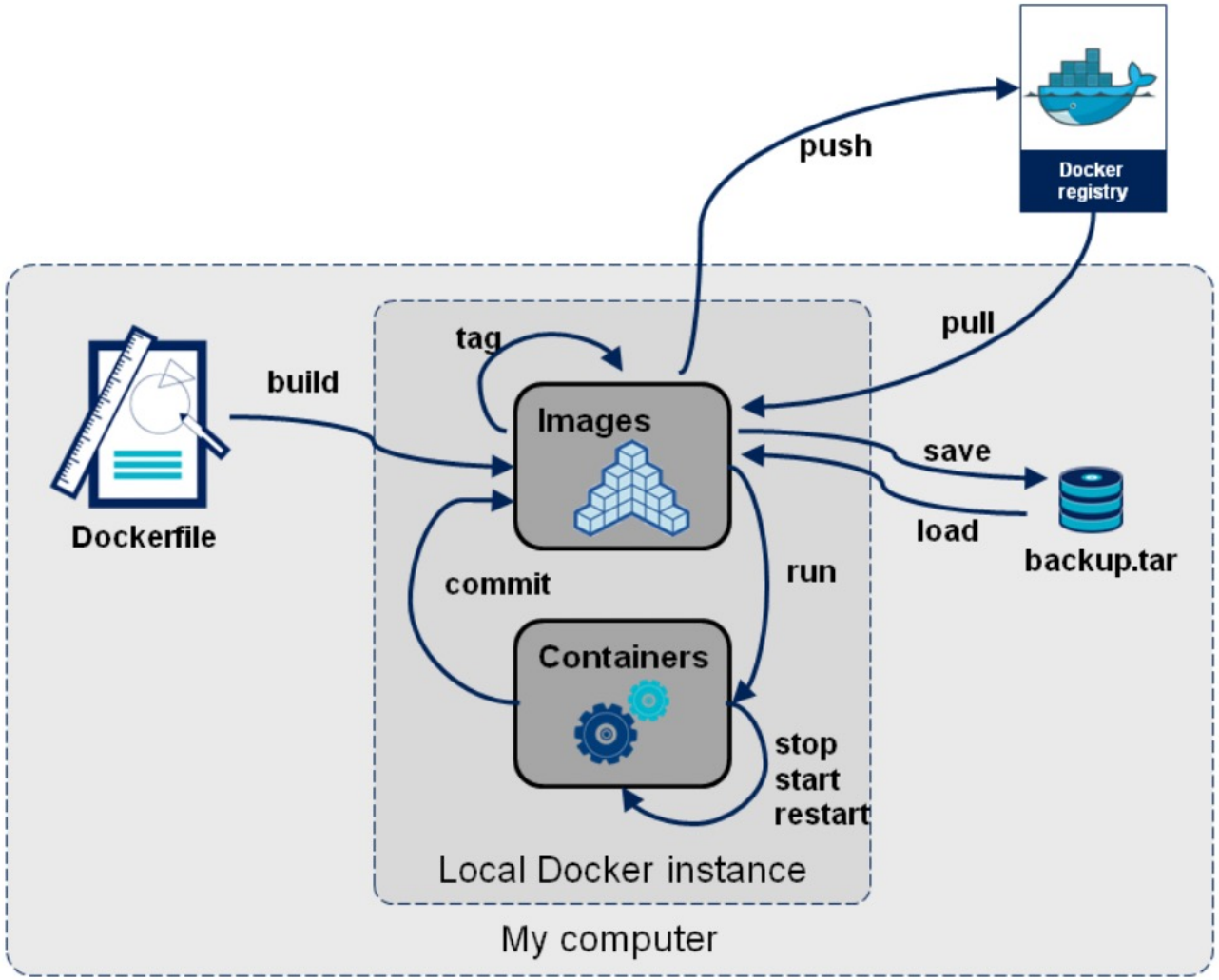
Stores, distributes and manages Docker images

Credit: Opsta

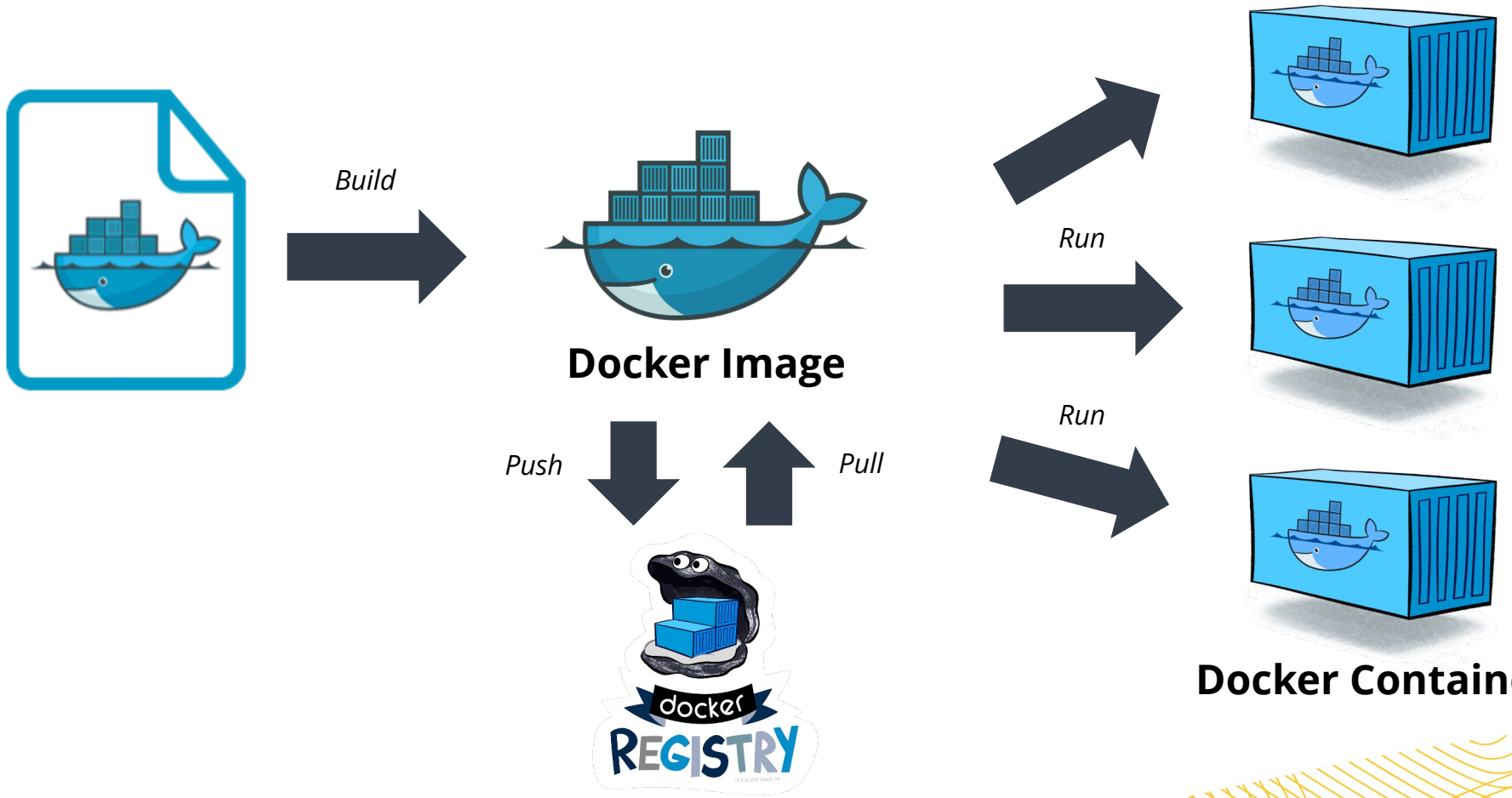
Engine components and responsibilities



Docker Lifecycle

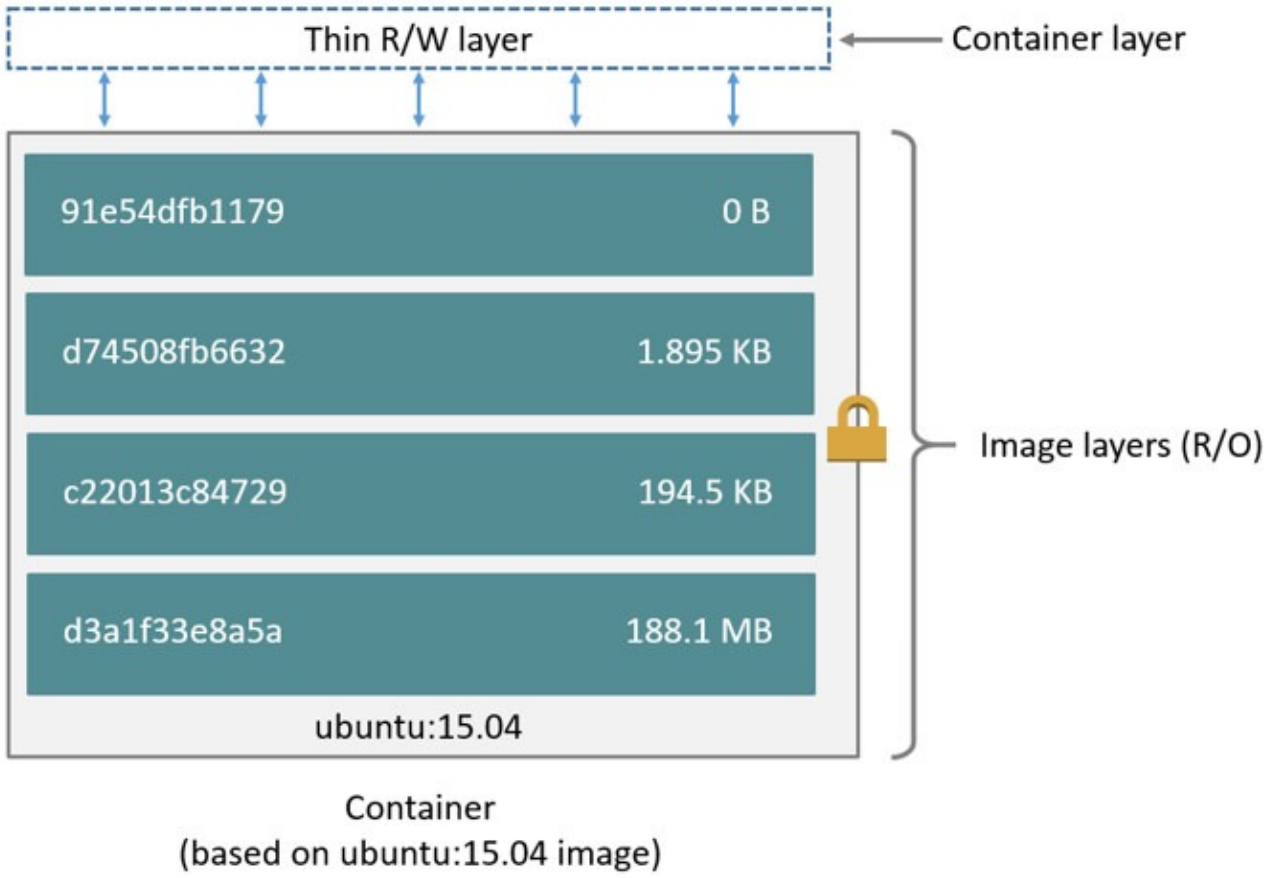


Docker Lifecycle



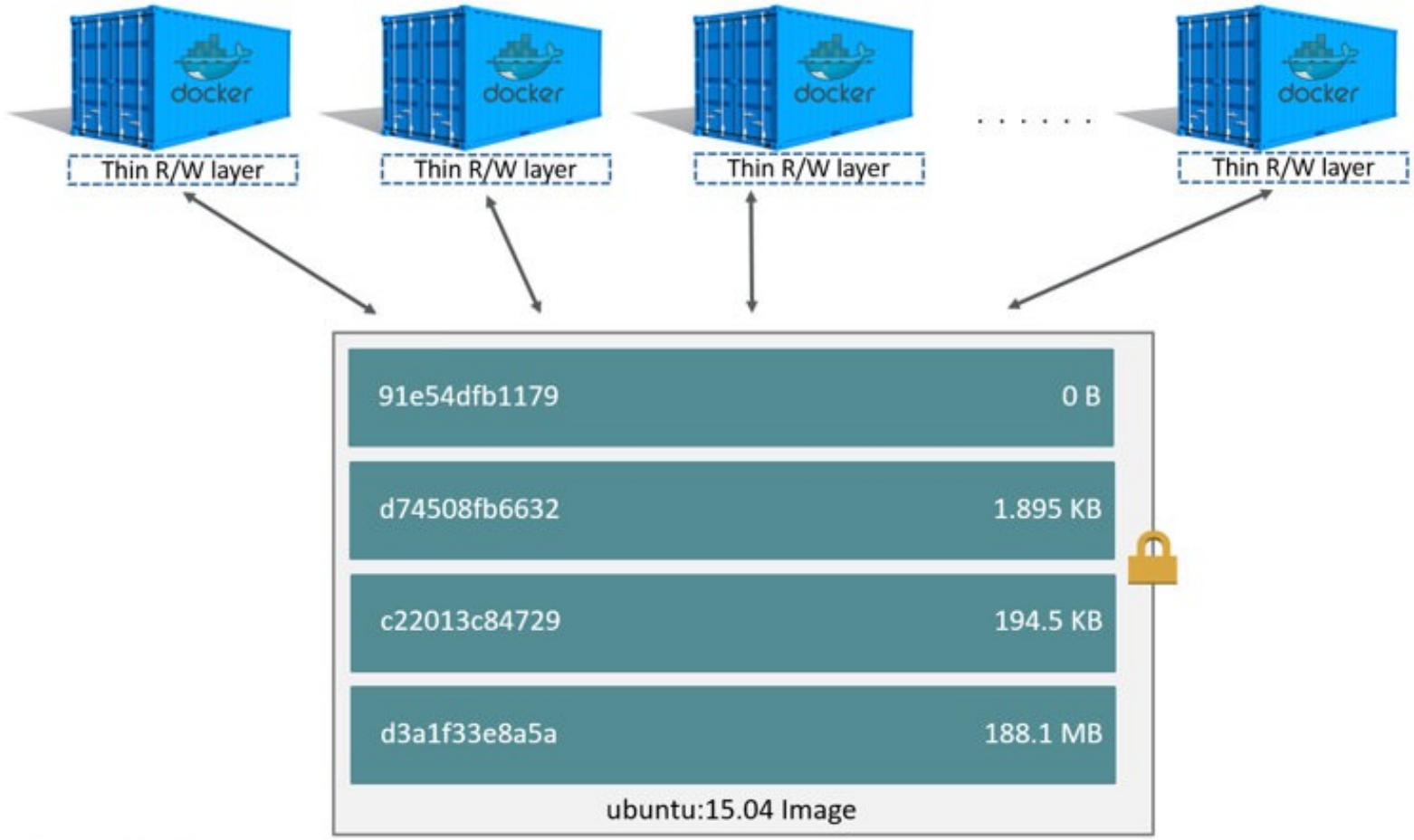
Credit: Jirayut Nimsaeng, Opsta Co. Ltd.

Container Layers



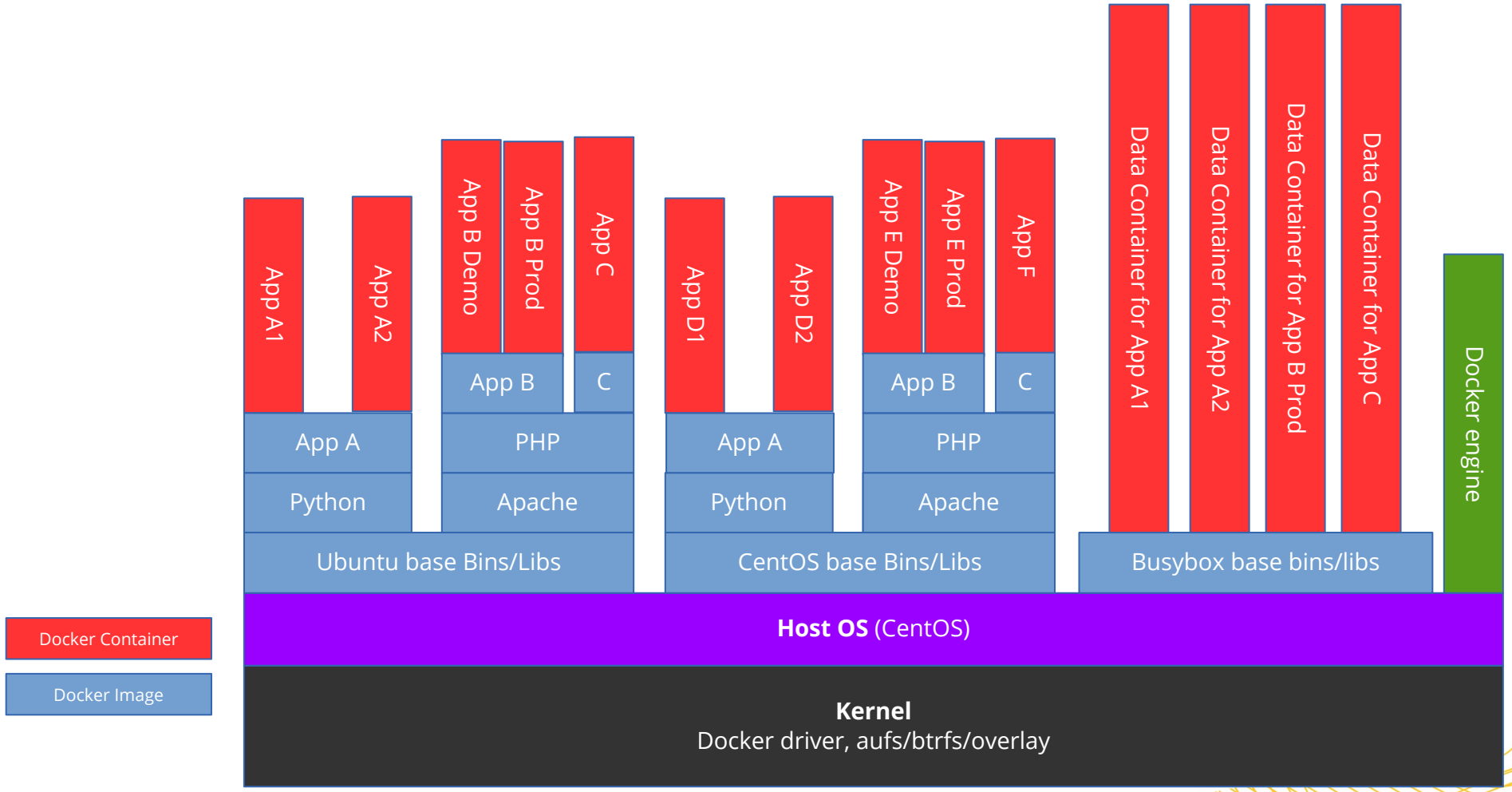
Source: https://docs.docker.com/storage/storagedriver

Container Layers



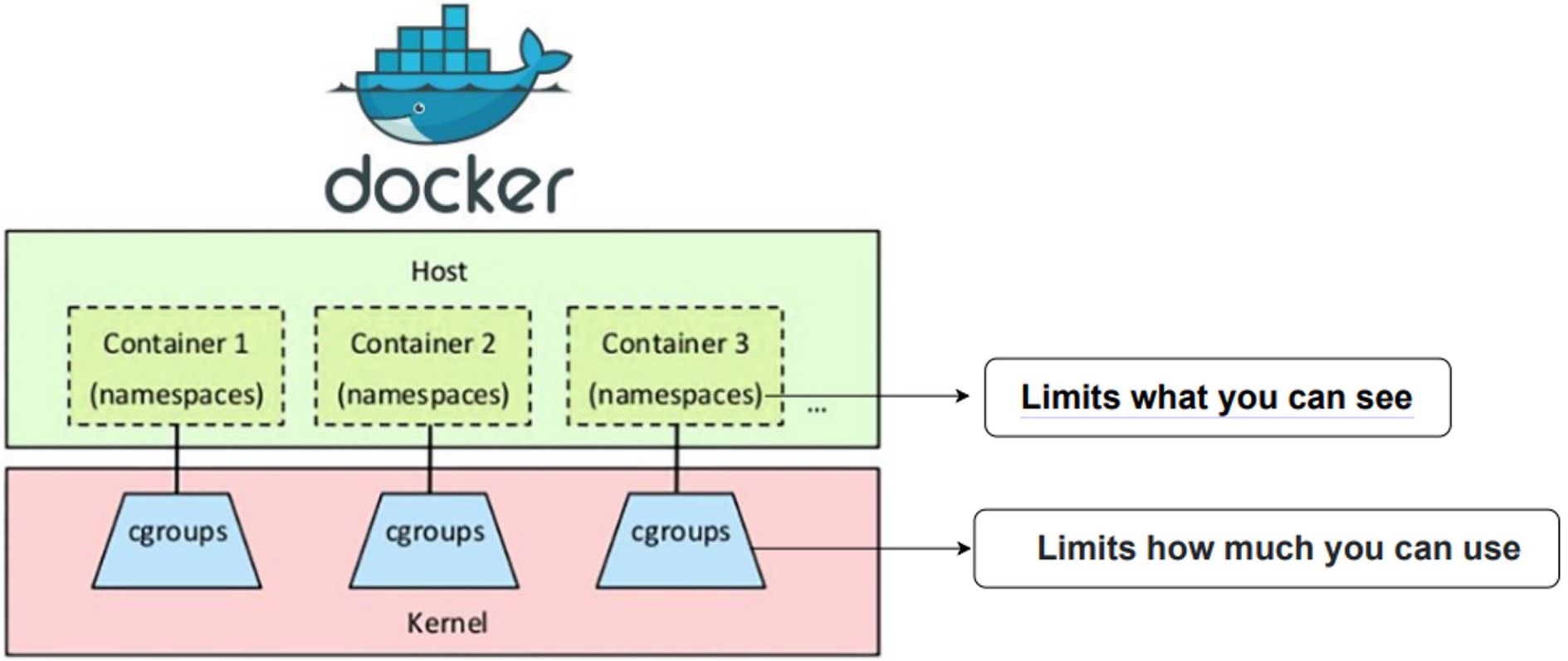
Source: https://docs.docker.com/storage/storagedriver

Docker Layers



Credit: Jirayut Nimsaeng, Opsta Co. Ltd.

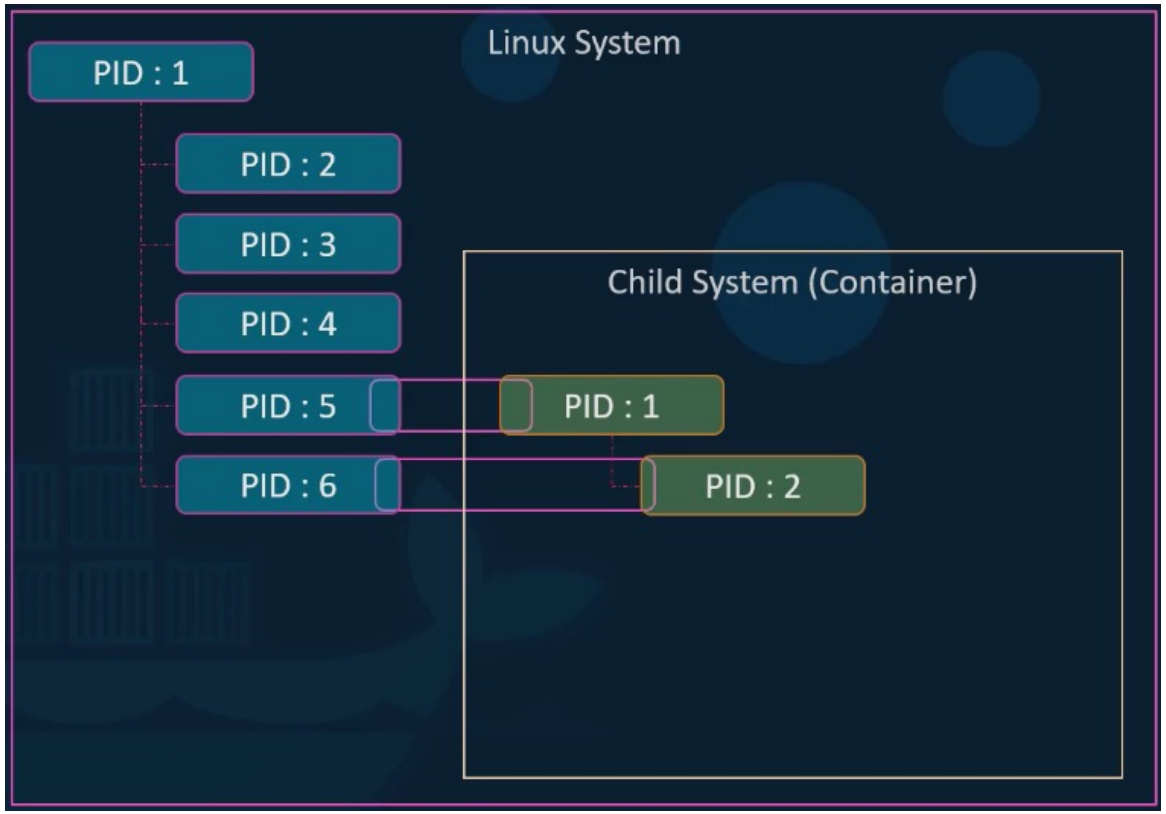
Kernel Namespaces and cgroups



Kernel Namespaces

- Namespaces provide a layer of isolation for containers.
- Each aspect of a container runs in a separate namespace and its access is limited to that namespace.
- When you run a container, Docker creates a set of namespaces for that container.
- Namespace makes processes running inside that namespace believe they have their own instance of that resource.
- A namespace can limit visibility to certain process trees, network interfaces, user IDs, or filesystem mounts.

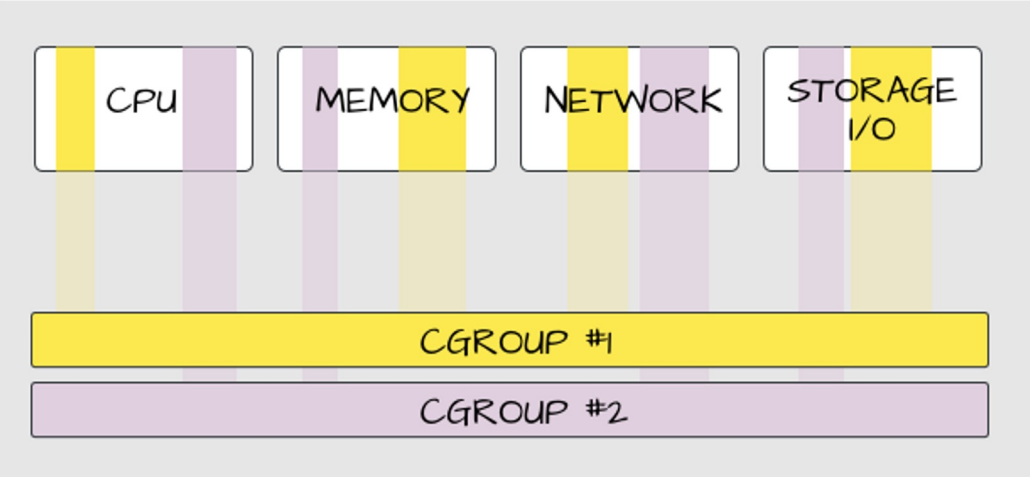
Namespace – PID



Source: Mumshad Mannambeth, KodeKloud

Control Groups (cgroup)

- It is a Linux kernel feature that limits an application to a specific set of resource usage (CPU, memory, disk I/O, network, and so on).
- Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints.




```
docker run --cpu=.5 ubuntu
docker run --memory=100m ubuntu
```

https://bikramat.medium.com/namespace-vs-cgroup-60c832c6b8c8

Installation

- Docker Engine: https://docs.docker.com/engine/install/ubuntu
- Docker Desktop:
  - Ubuntu: https://docs.docker.com/desktop/install/ubuntu
  - Windows: https://docs.docker.com/desktop/install/windows-install
  - macOS: https://docs.docker.com/desktop/install/mac-install
- Testing Docker

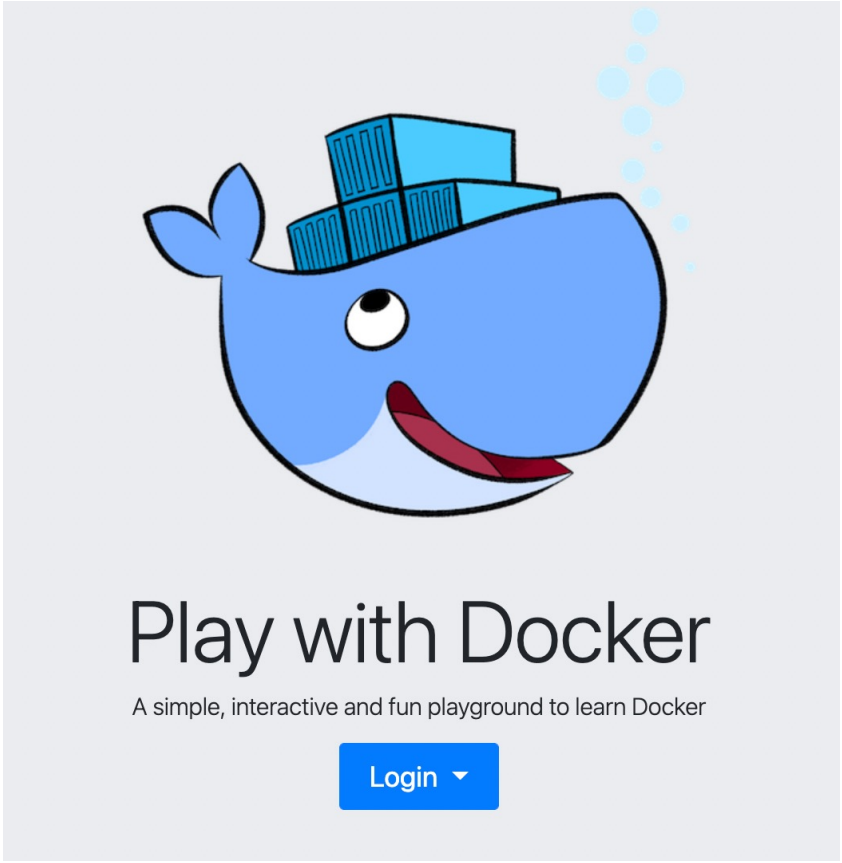
```
$ docker version
$ docker info
$ systemctl status docker
```

Software Components

- docker.io (on Debian/Ubuntu) and docker (on Fedora) packages are maintained by their respective Linux distributions.
  - They're available to be installed without adding any additional package repositories.
- containerd is a runtime built to run containers.
  - This open source tool builds on top of OS kernel features and improves container management with an abstraction layer, which manages namespaces, cgroups, union file systems, networking capabilities, and more.
- docker-ce is a package provided by Docker.
  - The package is available through a third-party package repository provided for major Linux distributions.
- Docker Desktop is the only way to install the Docker Engine on Windows and macOS.
  - Docker Desktop is also available for Linux, although Linux users are free to install the Docker Engine separately.

Docker Sandbox

- https://labs.play-with-docker.com
- It is free!
- Clean up every 4 hours.



Basic Commands Summary [1/2]

Commands	Description
docker search <image>	Search images from Docker Hub.
docker pull <image>	Pull an image from Docker Hub.
docker run --name <container> -d <image>	Create a container from an image by background.
docker exec -it container <command/shell>	Execute a command in a running container.
docker <images>	List all images.
docker container ls	List all containers.
docker ps	List all containers.
docker inspect <container>	Show details of a container.
docker [start|stop|restart|pause|unpause] <container>	Start, stop, restart, pause, or unpause a container.
docker rmi <image>	Remove an image.
docker rm <container>	Remove a container.
docker logs <container>	Show logs of a container.
docker tag <src-img>:<src-tag> <dst-img>:<dst-tag>	Put a tag on destination image from source image.

Basic Commands Summary [2/2]

Commands	Description
docker build -t <image> -f <dockerfile>	Build an image from a docker file.
docker commit <container> <image>:<tag>	Commit or save a container as an image file.
docker image save <image>:<tag> -o <backup-file>	Backup an image file as another file, e.g., tar file.
docker image load -i <backup-file>	Import a backup file into an image file.
docker run -d -p 5000:5000 --restart-always --name registry registry:2	Create our own Registry.
docker attach <container>	Attach to a container.
docker network create --drive bridge <bridge>	Create a bridge network for containers.
docker network connect <network> <container>	Make a connection from a container to a network.
docker network rm <network>	Remove a network.
docker stats	Display a live stream of containers resource usage statistics.
docker top <container>	Display the running processes of a container.
docker volume create <data_volume>	Create data volume for a container
docker run --mount source=<path>,target=<path>	Mount from source path on Docker host to destination path on a container

Mahidol University
Wisdom of the Land

- https://hub.docker.com
- Commands:

```
$ docker search ubuntu
$ docker search --filter is-official=true ubuntu
$ docker search --filter stars=100 ubuntu
```

- More info:

  https://docs.docker.com/engine/reference/commandline/search/

Basic Docker Commands

```
$ docker pull ubuntu
$ docker run -it ubuntu
# id
<control+p><control+q>
$ docker ps
$ docker container ls
$ docker images
$ docker attach <container>
# <control+p><control+q>
```

```
$ docker run -it -d --name=ubuntu2
ubuntu
$ docker exec -it ubuntu2 bash
#
<control+p><control+q>
$ docker ps
$ docker images
$ docker inspect ubuntu2
$ docker attach ubuntu2
# exit
```

Basic Docker Commands

- Docker exec
  - Execute a command in a running container.
  - We use it for testing, installing, and modifying something in the container.
- Docker attach
  - It is to connect stdin, stdout, stderr with a running container.
  - When you run an exec, it will basically spin up a new process inside the container whereas attach basically lets you attach to an existing process inside the container
- If we use docker attach, we can use only one instance of shell. So, if we want to open new terminal with new instance of container's shell, we just need to run docker exec.
- For example, if the docker container was started using /bin/bash command, you can access it using attach, if not then you need to execute the command to create a bash instance inside the container using exec.

Example: MySQL Container

```
$ docker pull mysql/mysql-server
$ docker run --name=mysql1 -d mysql/mysql-server
$ docker logs mysql1
$ docker logs mysql1 2>&1 | grep GENERATED
[Entrypoint] GENERATED ROOT PASSWORD: xxxxxxxxxxx
$ docker exec -it mysql1 mysql -uroot -p
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';

$ docker exec -it mysql1 /bin/bash
$ docker inspect mysql1
```

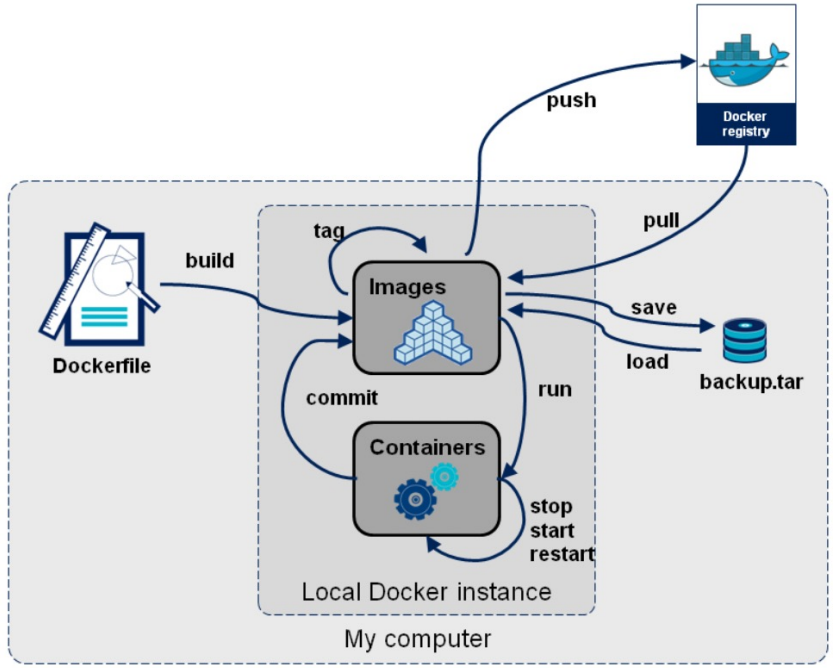
https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/docker-mysql-getting-started.html

Basic Commands

```
$ docker stop <container>
$ docker start <container>
$ docker restart <container>
$ docker pause <container>
$ docker unpause <container>
$ docker rm <container>
$ docker images
$ docker rmi <image>
```

Image Backup

```
$ docker commit <container> <image>:<tag>
$ docker image save <image>:<tag> -o image.tar
$ docker image load -i ./image.tar

$ docker run -it --name ubuntu1 -d ubuntu
$ docker exec -it ubuntu1 /bin/bash
# <control+p><control+q>
$ docker commit utuntu1 my-ubuntu2:1.1
$ docker image save my-utuntu2:1.1 -o my-ubuntu2.tar
$ docker rmi my-utuntu2:1.1
$ docker image load -i ./my-ubuntu2.tar
$ docker images
```



Docker Tag

- If tag is unidentified, latest will be default tag.
- “latest” may not be the latest version.
- `docker tag source[:tag] target[:tag]`

```
$ docker pull ubuntu:plucky
$ docker tag ubuntu:plucky plucky
$ docker images
REPOSITORY     TAG             IMAGE ID
plucky         latest          7fab1c5df1d0
ubuntu         plucky          7fab1c5df1d0
$ docker rmi 7fab1c5df1d0
Error response from daemon: conflict: unable to detect bafda420a37f (must be forced)
- image is referenced in multiple repositories
```

How to delete both images?

Deleting Tagged Images

- Option 1

```
$ docker rmi ubuntu:plucky plucky:latest
```

- Option 2

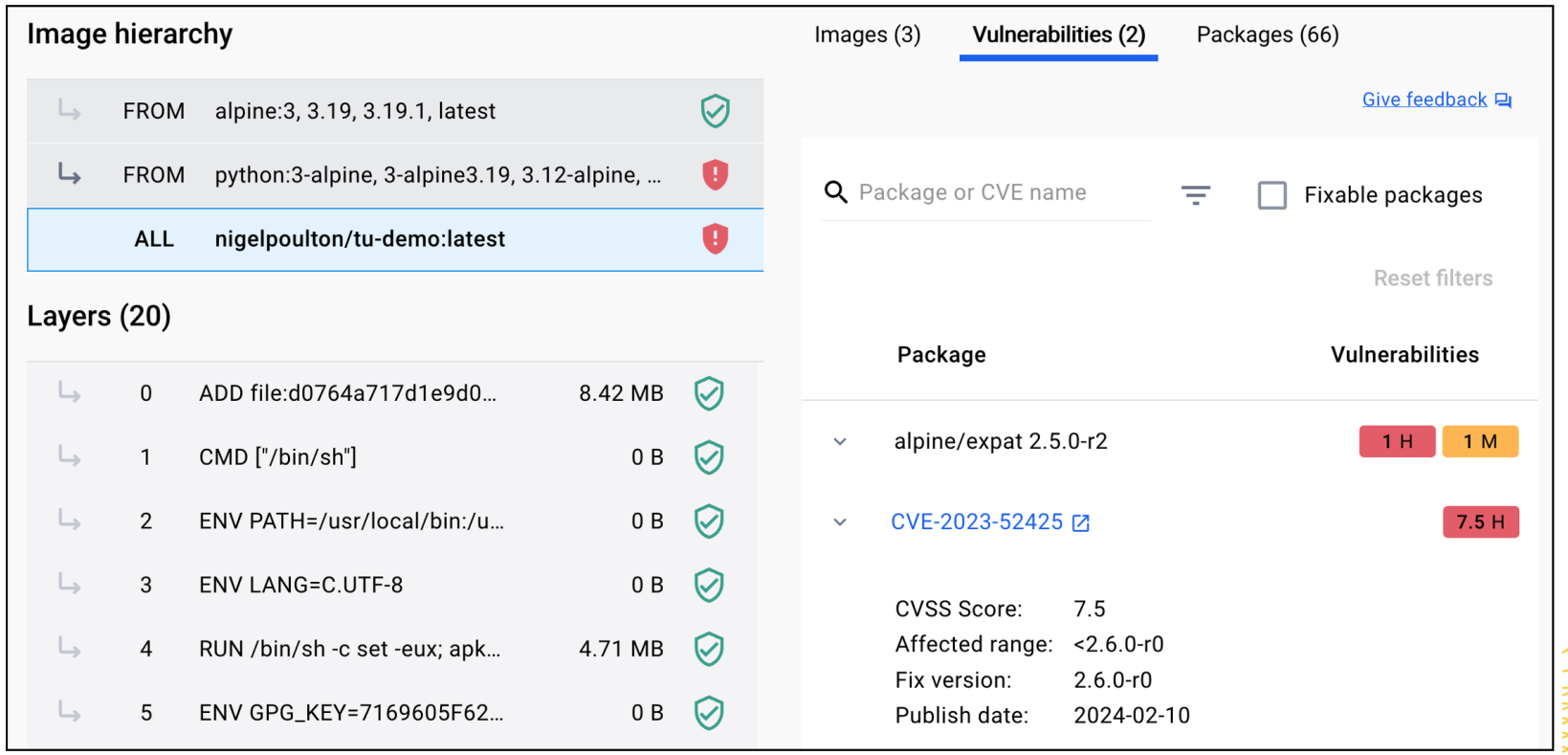
```
$ docker rmi plucky:latest
$ docker rmi ubuntu:plucky
```

- Option 3

```
$ docker rmi -f 7fab1c5df1d0
```

Vulnerability scanning with Docker Scout

```
$ docker scout quickview nigelpoulton/tu-demo:latest
$ docker scout cves nigelpoulton/tu-demo:latest
```



Privileged Mode

```
$ docker run -it --rm ubuntu bash
# mount -t tmpfs none /mnt
mount: /mnt: access denied
$ docker run -it --privileged ubuntu bash
# mount -t tmpfs none /mnt
# df -h
File System  Size   Used   Avail  Use%   Mounted on
none         1.9G   0      1.9G   0%     /mnt
```

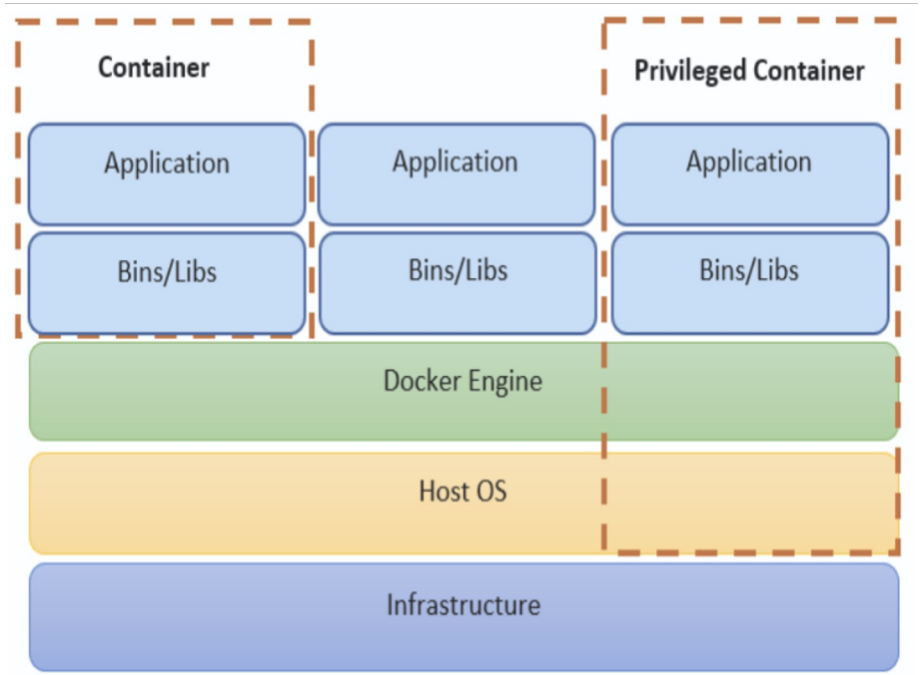
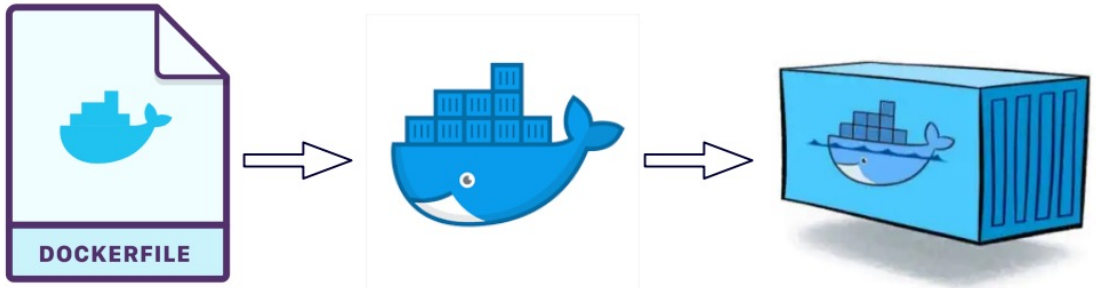


Image Source: Opsta Co. Ltd.

Dockerfile

A Dockerfile is a text file that contains a set of instructions which are used by docker build for building a custom docker image.

```
FROM image:tag
WORKDIR path
ENTRYPOINT command, param1, param2
RUN command
ADD src dst
COPY src dst
EXPOSE port
CMD command parameters
```



Reference: https://docs.docker.com/reference/dockerfile/

Dockerfile Example

```
# Dockerfile
FROM alpine
RUN apk add nginx
CMD [“echo”, “Image created”]

$ docker build -t mytest .
$ docker images
REPOSITORY  TAG          IMAGE ID          SIZE
mytest      latest       0bb33bc8d85e      4.41 MB
alpine      latest       a24bb4013296      5.57 MB
```

Dockerfile Example with Tag

```
$ docket build -t mytest -f mydockerfile
$ docker tag mytest:latest mytest:1.0.0
REPOSITORY   TAG          IMAGE ID          CREATED      SIZE
mytest       1.0.0        4f59b4e353ca      1 min ago    8.76MB
mytest       latest       4f59b4e353ca      1 min ago    8.76MB
alpine       latest       adafef2e596e      a day ago    5.57MB
```

Dockerfile: CMD

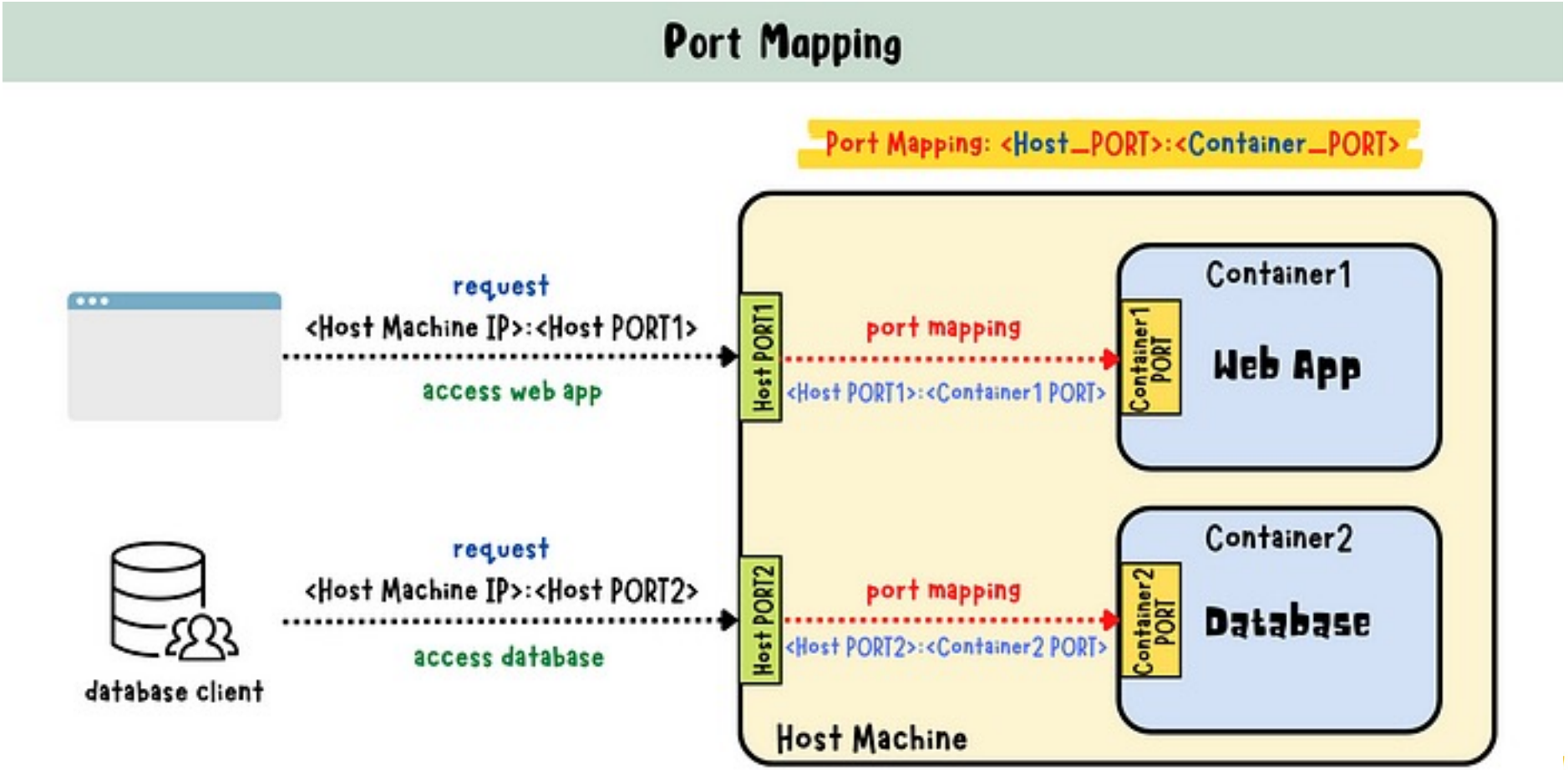
- CMD ["*executable*", "*param1*", "*param2*"]
- CMD ["*param1*", "*param2*"]
  - ENTRYPOINT["/bin/echo","Hi"]
  - CMD["Hello World"]
- CMD *command param1 param2*
  - E.g., CMD echo HELLO

We can have only one CMD

Dockerfile Lab

```
$ git clone https://github.com/nigelpoulton/psweb.git
$ cd psweb
$ ls -l
$ cat Dockerfile
$ docker build -t test:latest .
$ docker images
$ docker run -d --name web1 --publish 8080:8080 test:latest
$ curl localhost:8080
$ docker rm web1 -f
```

Port Mapping



Source: https://yuminlee2.medium.com/docker-port-mapping-12dd1d94fa54

Port Mapping

```
$ docker run -d --name nginx1 nginx
$ docker run -d -p 80 --name nginx2 nginx
$ docker inspect nginx2 | grep HostPort
      ”HostPort”:””
      “HostPort”:”32768”
$ curl localhost:32768
...
$ docker stop nginx1 nginx2
$ docker rm nginx1 nginx2
$ docker run -d -p 80:80 --name nginx3 nginx
$ curl localhost
```

Useful Docker Commands

- Stop all running containers

```
$ docker stop $(docker ps -a -q)
```

- Display all dockers

```
$ docker ps
$ docker ps -a # all running and stopped containers
```

- Delete all images

```
$ docker rmi $(docker images -q) # -q = container ID
```

- Display container detail

```
$ docker inspect NAME|ID
```

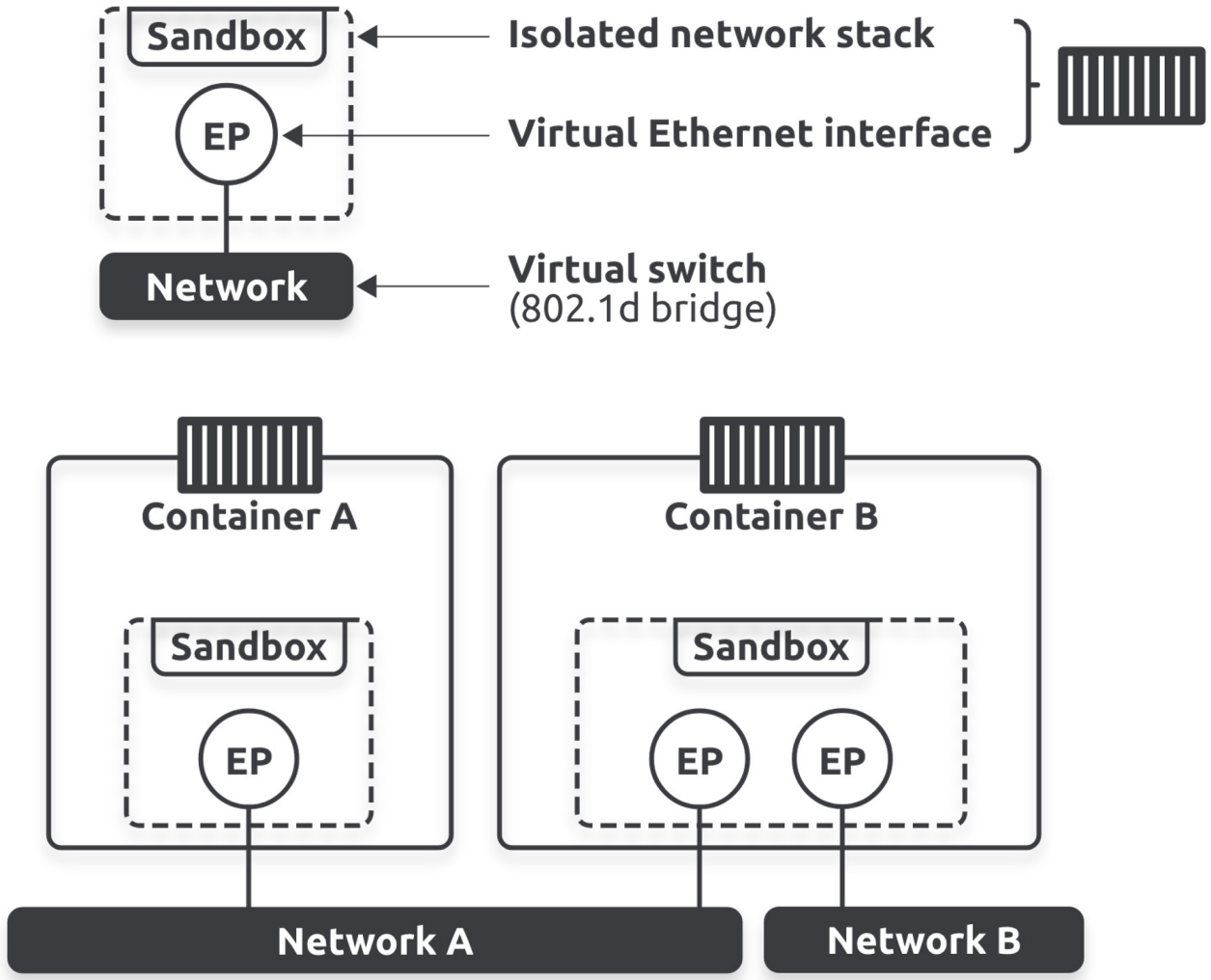
- Monitoring containers

```
$ docker stats
$ docker top <container_id>
```

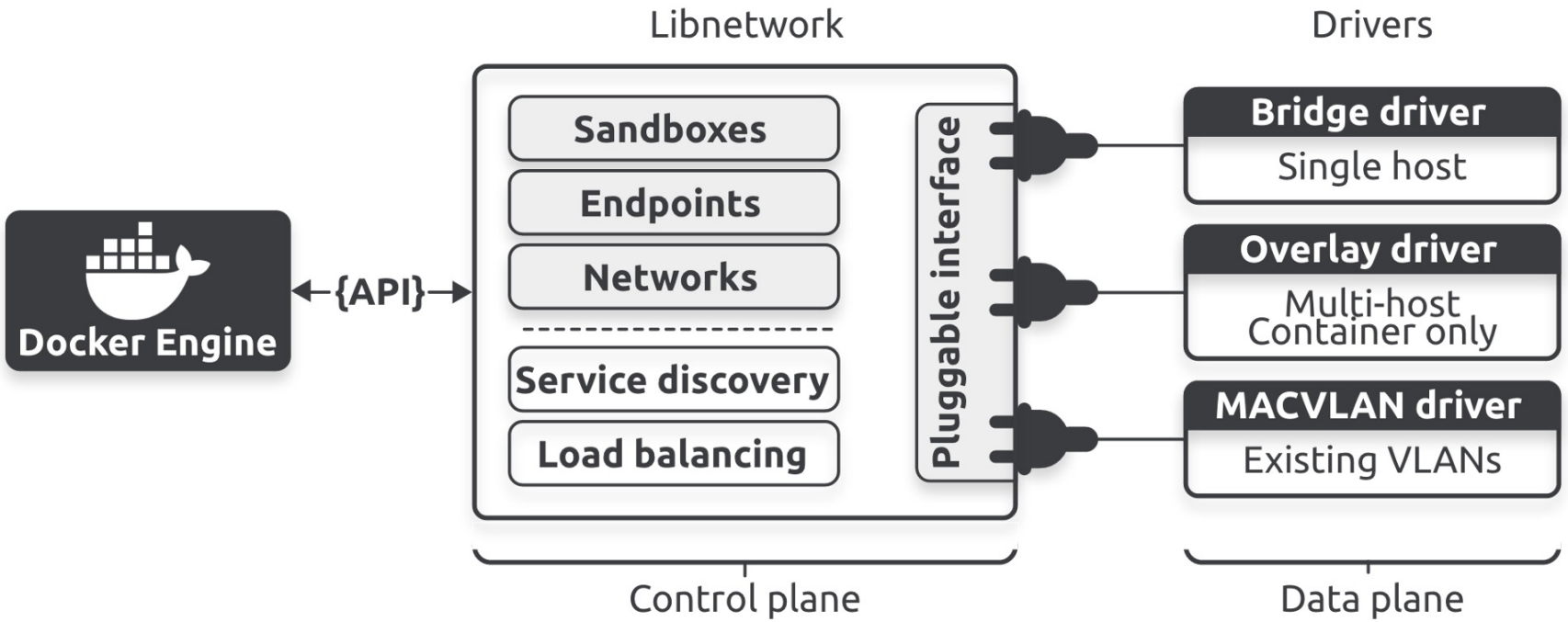
Docker Network

- The Container Network Model (CNM) is the design specification and outlines the fundamental building blocks of a Docker network.
- Libnetwork is a real-world implementation of the CNM.
  - It’s open-sourced as part of the Moby project18 and used by Docker and other platforms.
- Drivers extend the model by implementing specific network topologies such as VXLAN overlay networks.

Container Network Model (CNM)



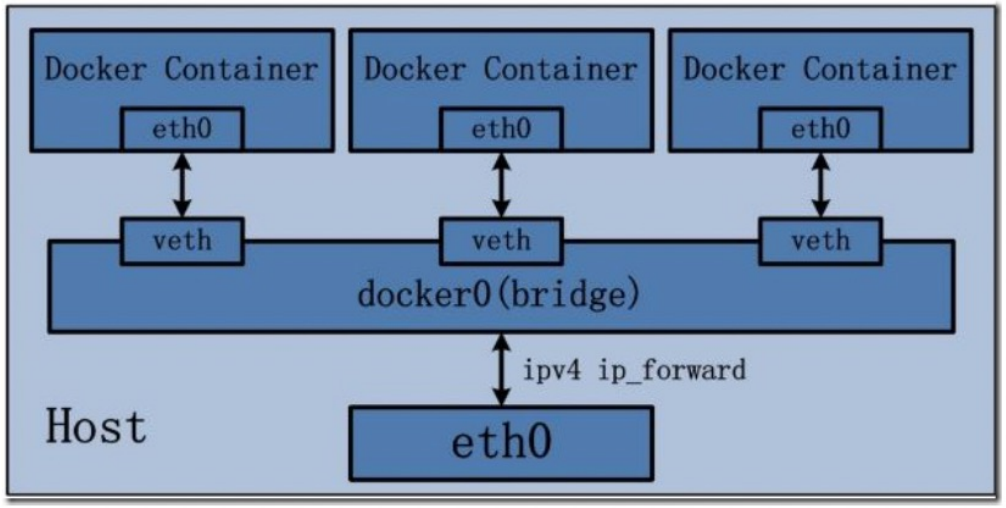
Docker Network



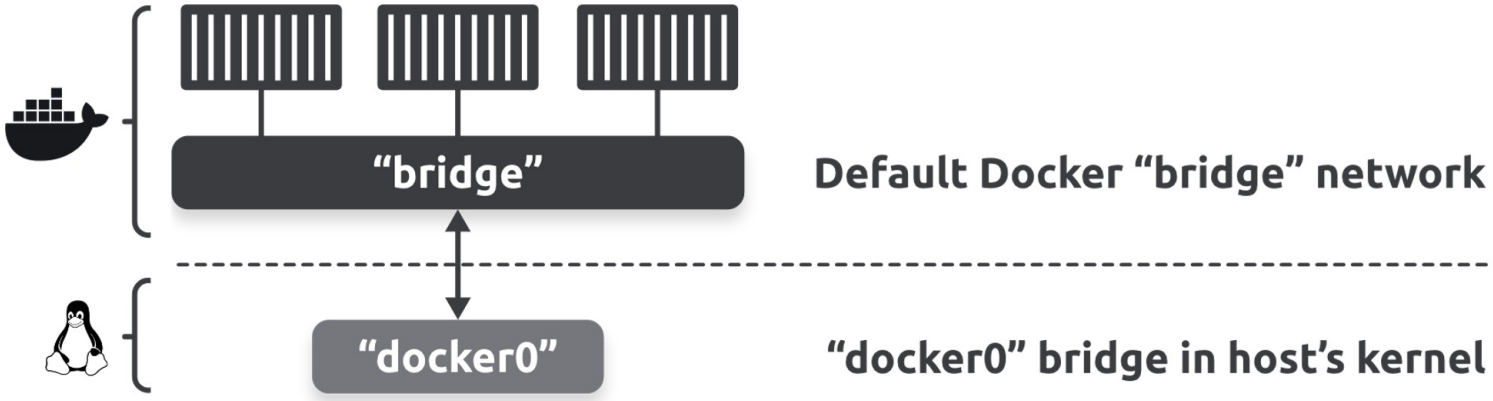
Bridge [1/3]

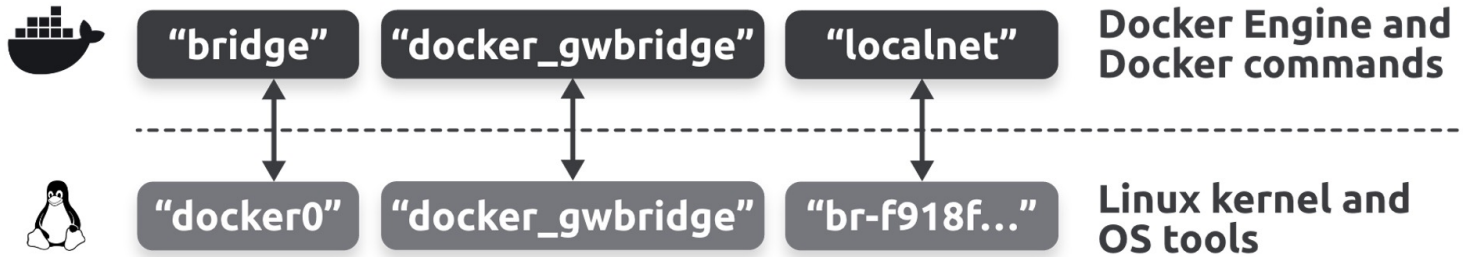
- Docker uses bridge network to connect among containers on the same host.
- Docker creates a bridge by default

```
$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
62aa91d5a47d   bridge        bridge    local
f325bf1ca51e   host          host      local
1743c3db6711   none          null      local

$ docker network inspect bridge
```

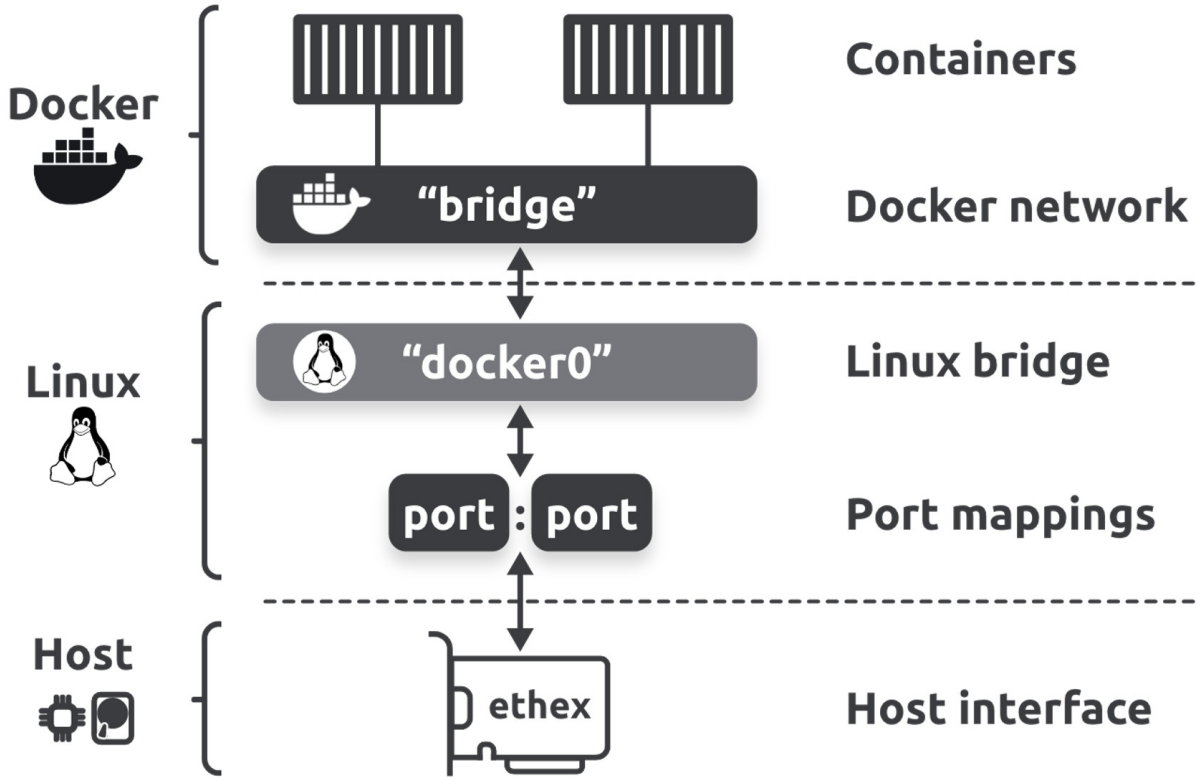


Bridge [2/3]


"bridge"
Default Docker "bridge" network
"docker0"
"docker0" bridge in host's kernel

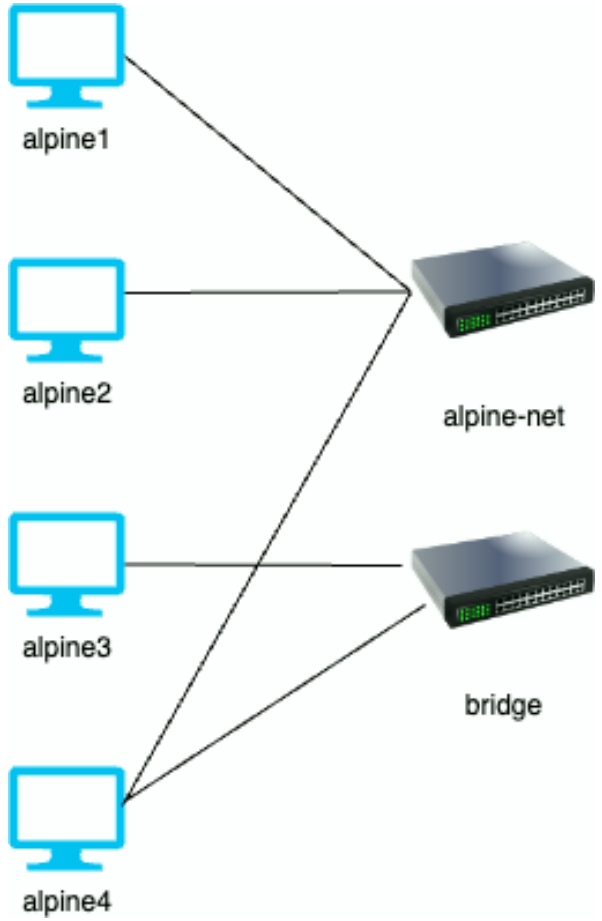

"bridge"
"docker_gwbridge"
"localnet"
Docker Engine and Docker commands
"docker0"
"docker_gwbridge"
"br-f918f..."
Linux kernel and OS tools

Bridge [3/3]

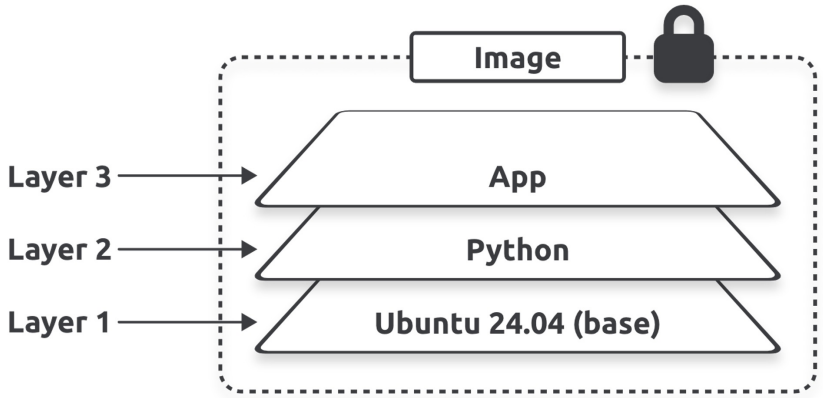

Docker
Containers
"bridge"
Docker network
Linux
"docker0"
Linux bridge
port : port
Port mappings
Host
ethex
Host interface

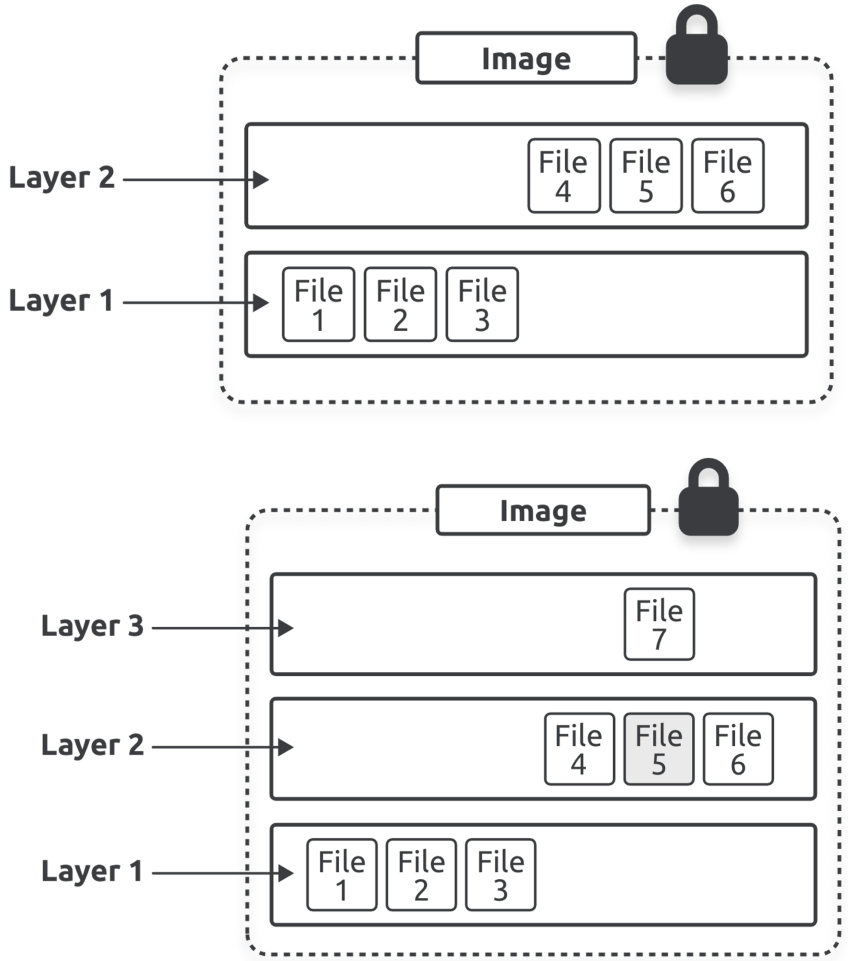
User Defined Bridge Network

```
$ docker network create --driver bridge <bridge_name>

$ docker run -dit --name <name> --network <network> \
<image> <command>

$ docker network connect <network> <container>

$ docker network rm <network>

$ docker network disconnect <network> <container>
```



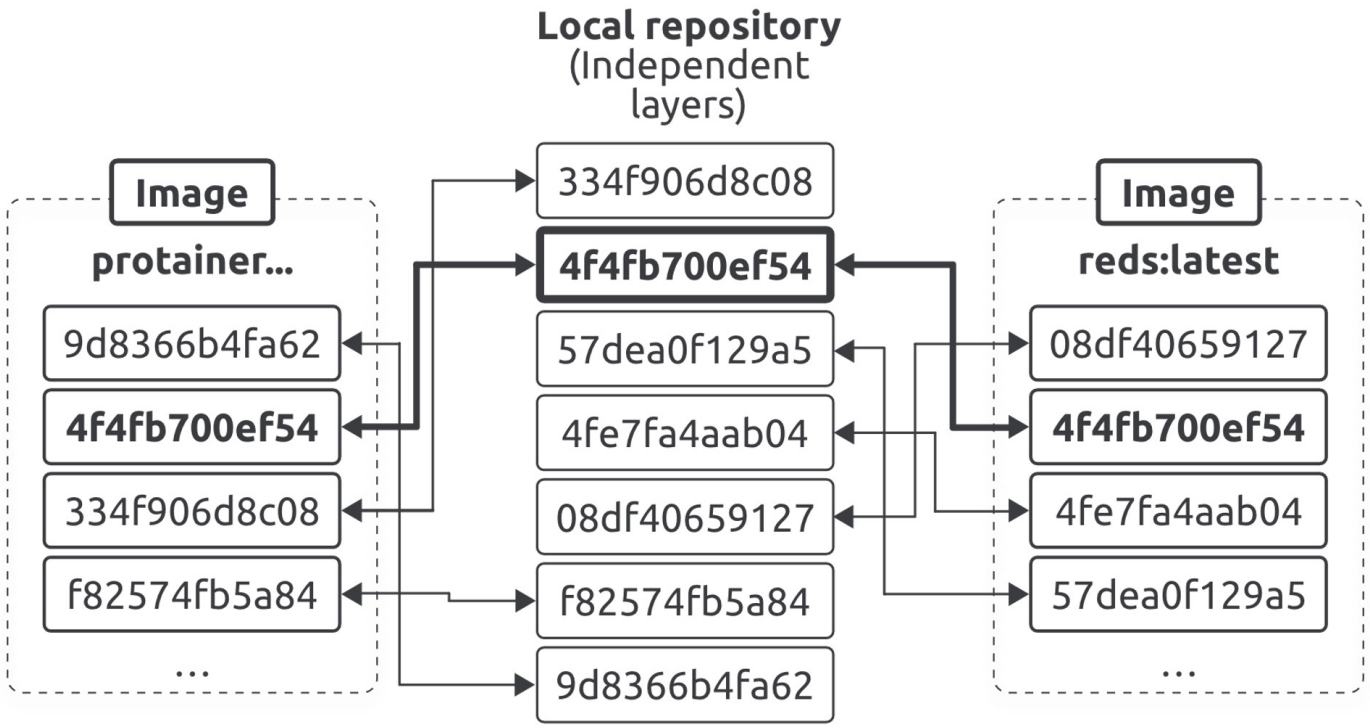
Layered Architecture [1/5]






```
$ docker inspect <image>
$ docker history <image>
```

Layered Architecture [2/5]



Layered Architecture [3/5]

#Dockerfile

```
FROM Ubuntu
RUN apt-get update && apt-get -y install python
RUN pip install flask flask-mysql
COPY . /opt/source-code
ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask

$ docker build Dockerfile -t my-custom-app
```

1. Base Ubuntu layer = 120 MB
2. Changes in apt package = 306 MB
3. Changes in pip package = 6.3 MB
4. Source code = 229 B
5. Update entrypoint = 0 B

Layered Architecture [4/5]

#Dockerfile2

```
FROM Ubuntu
RUN apt-get update && apt-get -y install python
RUN pip install flask flask-mysql
COPY app2.py /opt/source-code
ENTRYPOINT FLASK_APP=/opt/source-code/app2.py flask

$ docker build Dockerfile2 -t my-custom-app2
```

1. Base Ubuntu layer = 0 MB
2. Changes in apt package = 0 MB
3. Changes in pip package = 0 MB
4. Source code = 229 B
5. Update entrypoint = 0 B

Layered Architecture [5/5]

- docker build Dockerfile -t my-custom-app
  - Layer 1 - 5 are image layers, which is read only
- docker run my-custom-app
  - Layer 6: container layer, which is read + write
- Can we modify app.py?
  - COPY-ON-WRITE

File System

- /var/lib/docker
  - aufs
  - containers
  - image
  - volumes

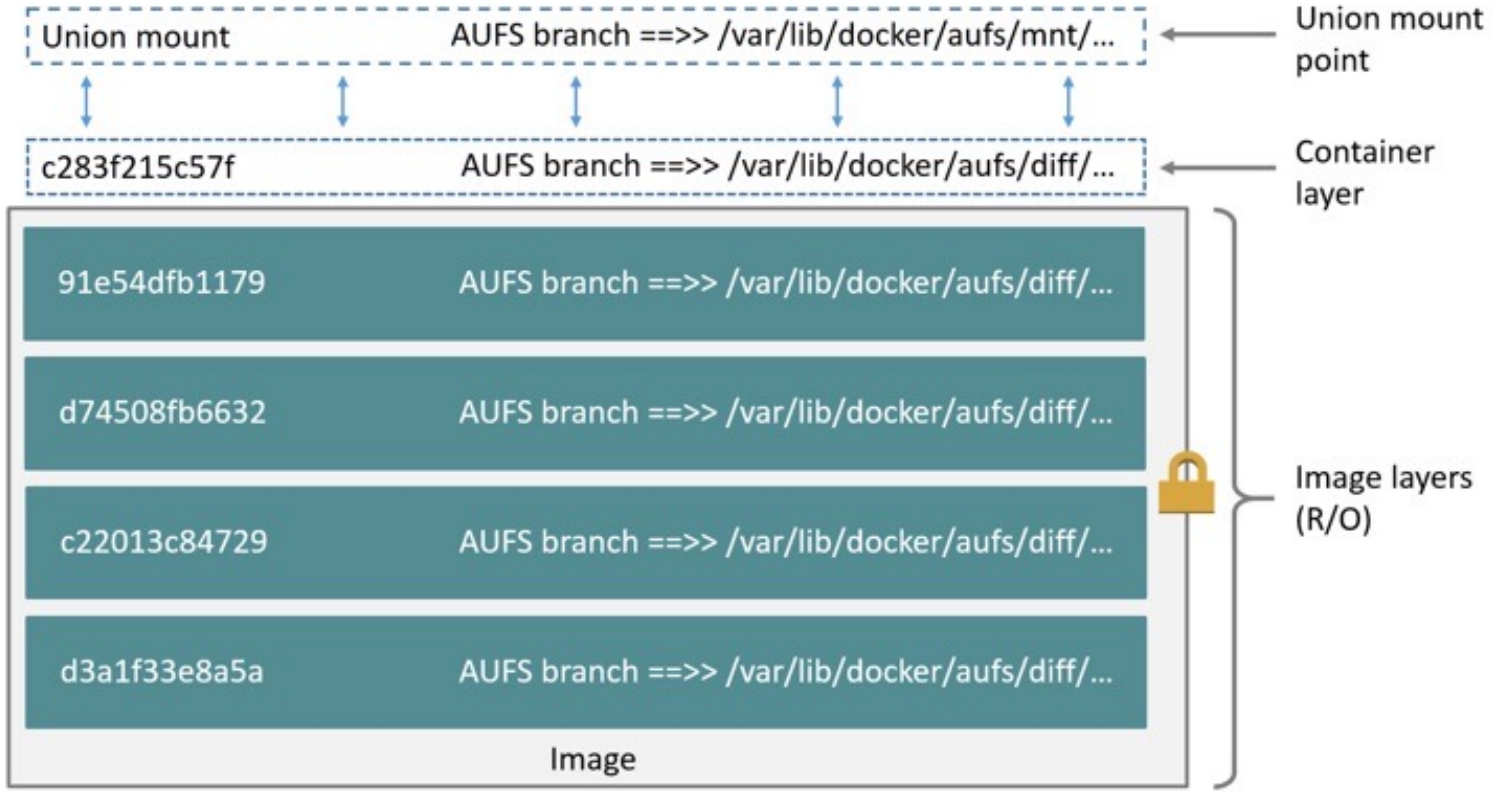
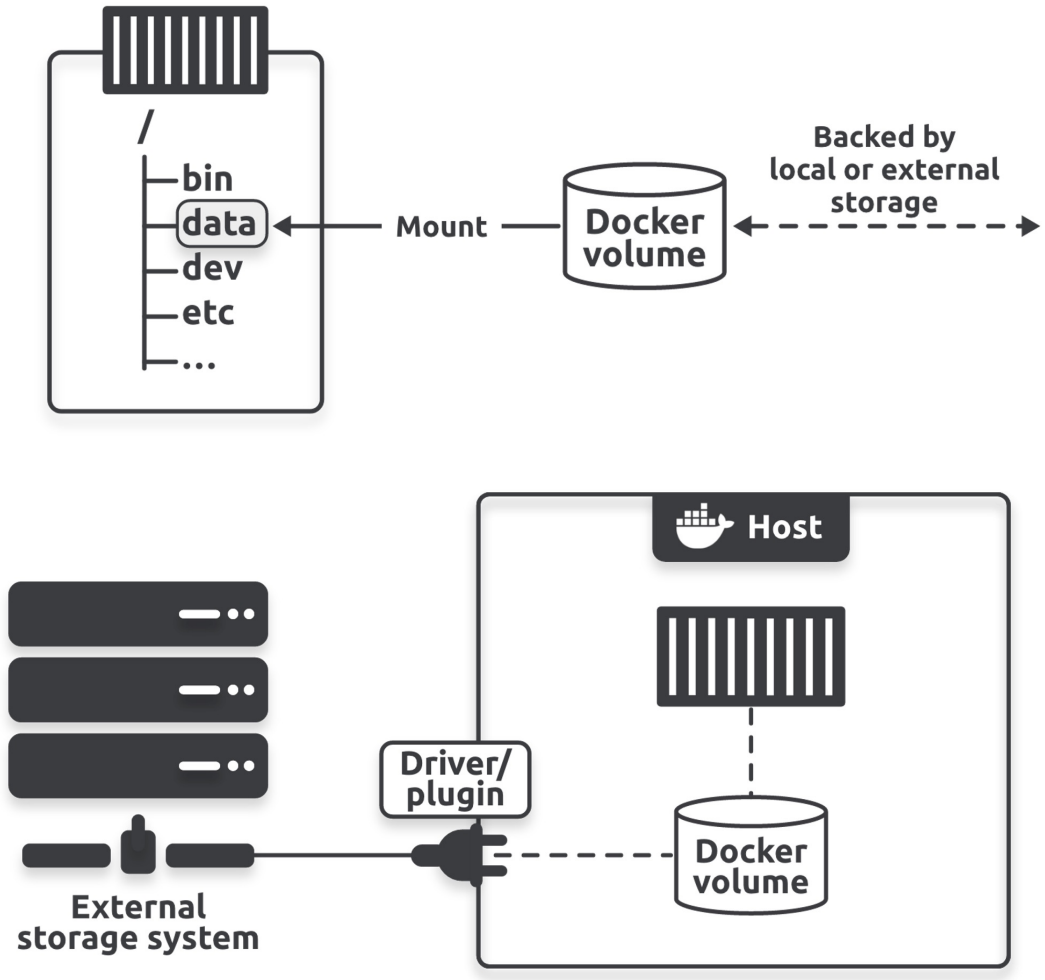


Image source: https://gdevillele.github.io/engine/userguide/storagedriver/aufs-driver/

Volumes


/
bin
data
dev
etc
...
Mount
Docker volume
Backed by local or external storage
Host
Driver/ plugin
Docker volume
External storage system

Storage Drivers

AUFS (advanced multi-layered unification filesystem)	• Older linux systems or distributions that natively support AUFS.
ZFS (Zettabyte file system)	• A robust filesystem and volume manager originally from Solaris • High-performance environments needing advanced features like snapshots and replication
Btrfs (B-tree filesystem)	• A modern copy-on-write filesystem with built-in support for features like snapshots and subvolumes.
Device mapper	• Best for legacy system
Overlay	• A lightweight storage driver that uses a union filesystem.
Overlay2	• Most modern Linux distributions and production workloads.

Volumes

```
$ docker volume create data_volume
$ docker run -v data_volume:/var/lib/mysql mysql          → volume mounting
$ docker run -v data_volume2:/var/lib/mysql mysql
$ docker run -v /data/mysql:/var/lib/mysql mysql          → bind mounting
$ docker run --mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql
```

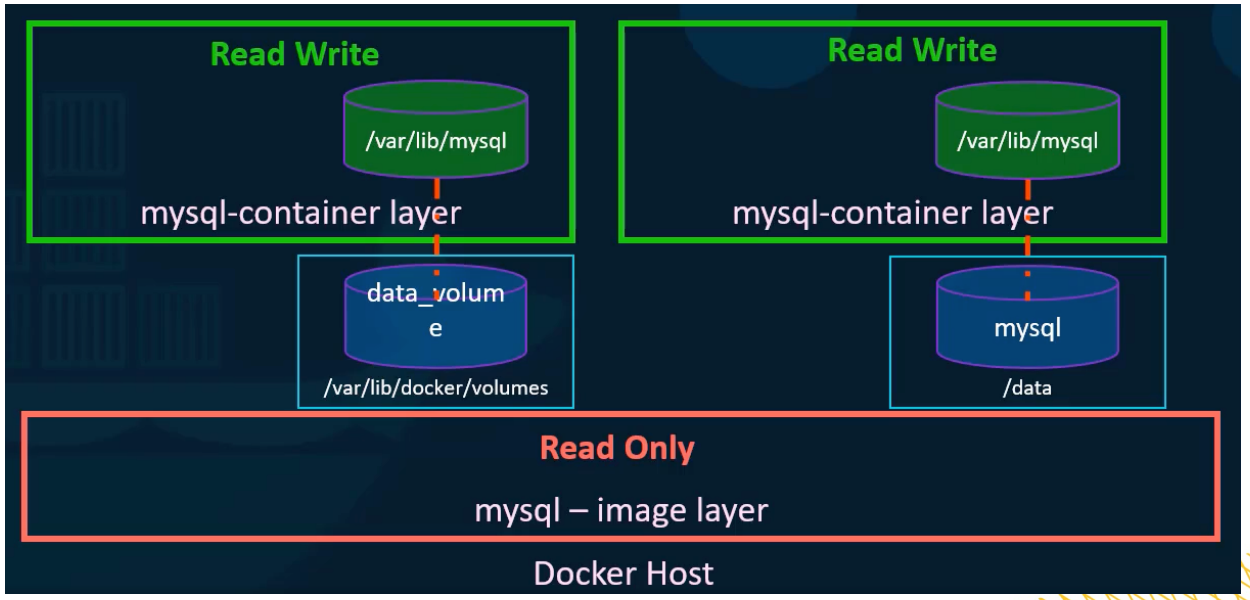


Image: Mumshad Mannambeth at KodeKloud, Packt Publishing

Volume Lab

```
$ docker run -it --name voltainer --mount type=bind,source=bizvol,target=/vol alpine
# <Ctrl-PQ>
$ docker volume ls
$ docker volume rm bizvol
$ docker exec -it voltainer sh
# echo ”Hello world!" > /vol/file1
# ls -l /vol
# cat /vol/file1
# exit
$ docker rm voltainer -f
$ docker ps -a
$ docker volume ls
$ ls -l /var/lib/docker/volumes/bizvol/_data/
$ cat /var/lib/docker/volumes/bizvol/_data/file1
```

Volume Commands

- `docker volume create` creates new volumes. By default, it creates them with the local driver, but you can use the -d flag to specify a different driver.
- `docker volume ls` lists all volumes on your Docker host.
- `docker volume inspect` shows you detailed volume information. You can use this command to see where a volume exists in the Docker host's filesystem.
- `docker volume prune` deletes all volumes not in use by a container or service replica. Use with caution!
- `docker volume rm` deletes specific volumes that are not in use.

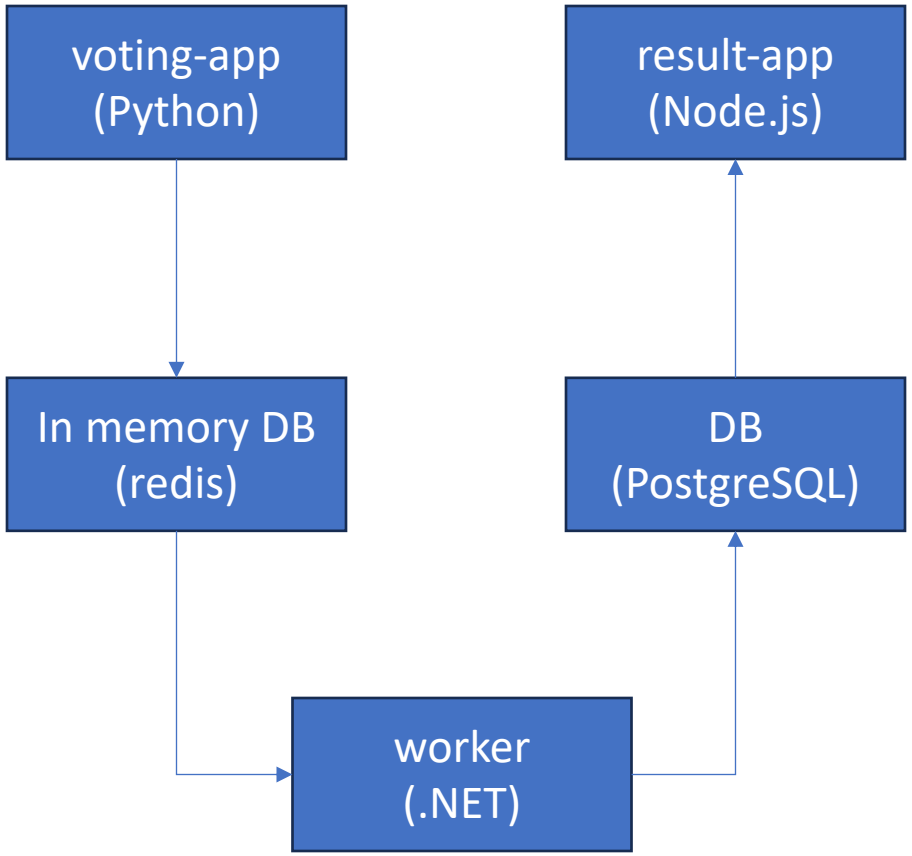
Docker Compose

- It is a tool to create an application from many containers
- It uses YAML or JSON to define.
- There are three main steps
  - Create Dockerfile to create your container.
  - Define services as your application by `compose.yml`.
  - Run `docker compose up` command to start your application.
- Docker compose is very good at network and container management.
- More info: https://docs.docker.com/compose/

Docker Compose

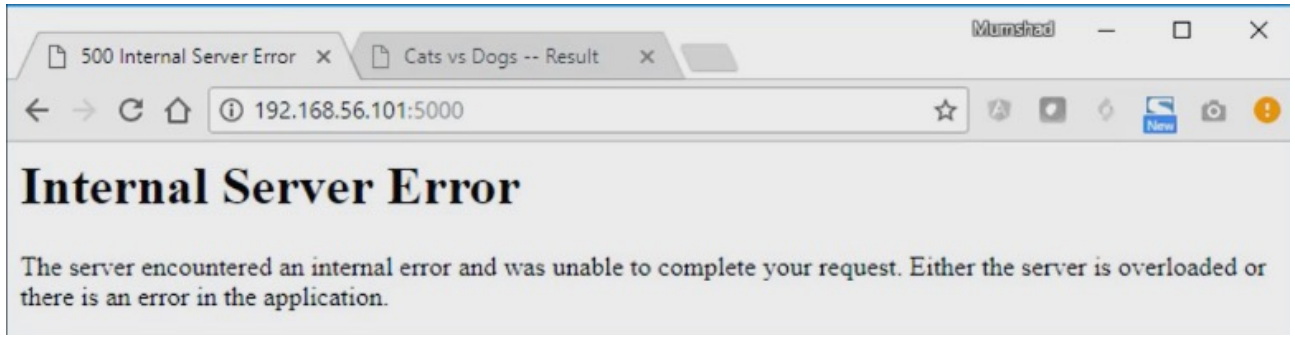
- Dockerfile is not replaced by docker compose
  - Dockerfile is to assemble an image
  - Docker Compose is a tool for defining and running multi-container Docker applications.
  - Docker Compose define the services that make up your app in docker-compose.
- Install Docker Compose: https://docs.docker.com/compose/install/

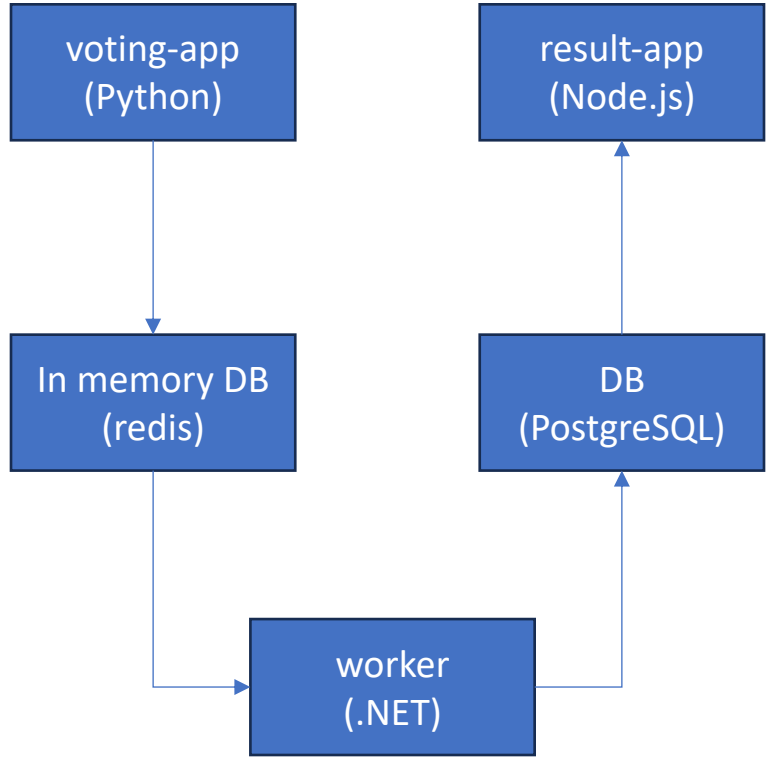
Docker Compose Scenario



Docker Compose Scenario

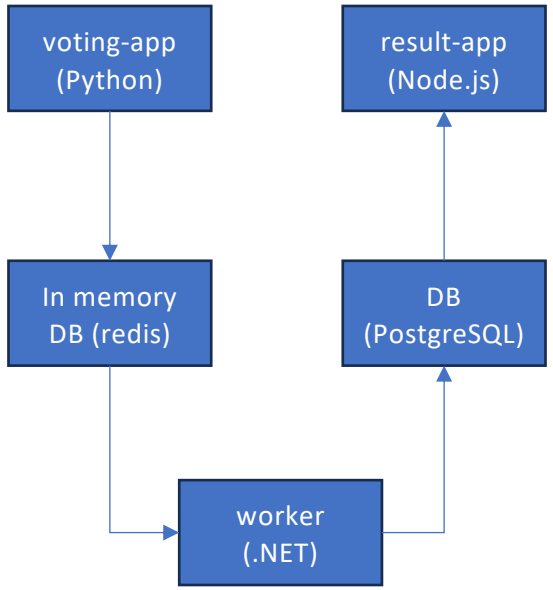
```
docker run -d --name=redis redis
docker run -d --name=db postgres
docker run -d --name=vote -p 5000:80 voting-app
docker run -d --name=result -p 5001:80 result-app
docker run -d --name=worker worker
```





Docker Compose Scenario [1/3]

```
docker run -d --name=redis redis
docker run -d --name=db postgres
docker run -d --name=vote --link redis:redis -p 5000:80 voting-app
docker run -d --name=result --link db:db -p 5001:80 result-app
docker run -d --name=worker --link db:db --link redis:redis worker
```



Let's note that the --link flag is a legacy feature of Docker.

Docker Compose Example [2/3]

```
docker run -d --name=redis redis

docker run -d --name=db

docker run -d --name=vote --link redis:redis -p 5000:80
voting-app

docker run -d --name=result --link db:db -p 5001:80
result-app

docker run -d --name=worker --link db:db --link
redis:redis worker
```

```yaml
redis:
    image: redis
db:
    image: postgres:9.4
vote:
    image: voting-app
    ports:
     - 5000:80
    links:
     - redis
result:
    image: result-app
    ports:
     - 5001:80
    links:
     - db
worker:
    image: worker
    links:
     - redis
     - db
```

Let's note that the --link flag is a legacy feature of Docker.

Docker Compose Example [3/3]

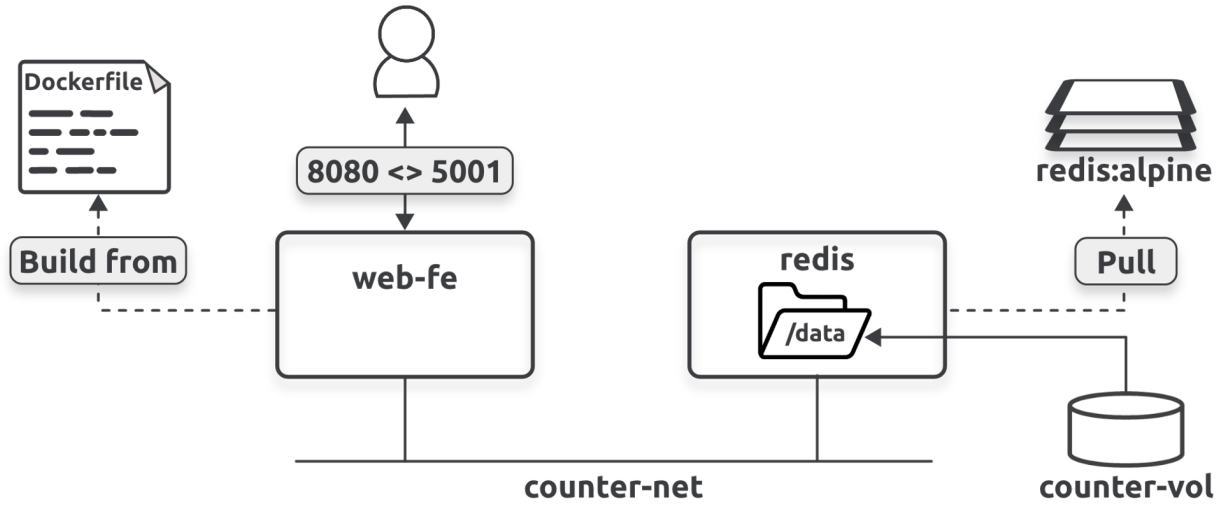
```yaml
services:
  redis:
    image: redis
  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
  vote:
    image: voting-app
    ports:
      - 5000:80
  results:
    image: result-app
    ports:
      - 5001:80
  worker:
    image: worker
```

Basic commands

- `docker compose up` = docker run <myimage> (look for docker-compose.yaml or compose.yaml)
- `docker compose up --build`
- `docker compose up -d`
- `docker compose down`

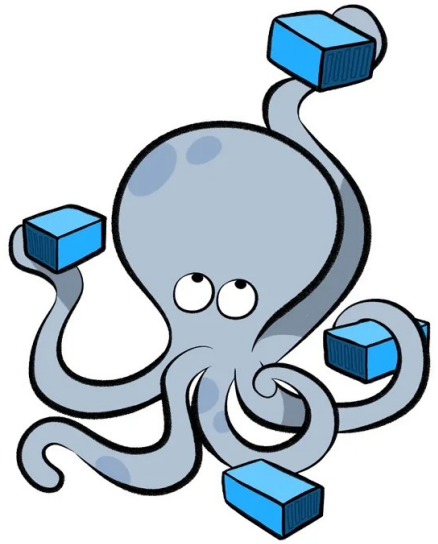
Docker Compose Lab

```
git clone https://github.com/nigelpoulton/ddd-book.git
cd ddd-book/multi-container/
docker compose up --detach
docker images
docker ps
docker network ls
docker volume ls
curl http://localhost:5001
docker compose ps
docker compose down
docker volume ls
```

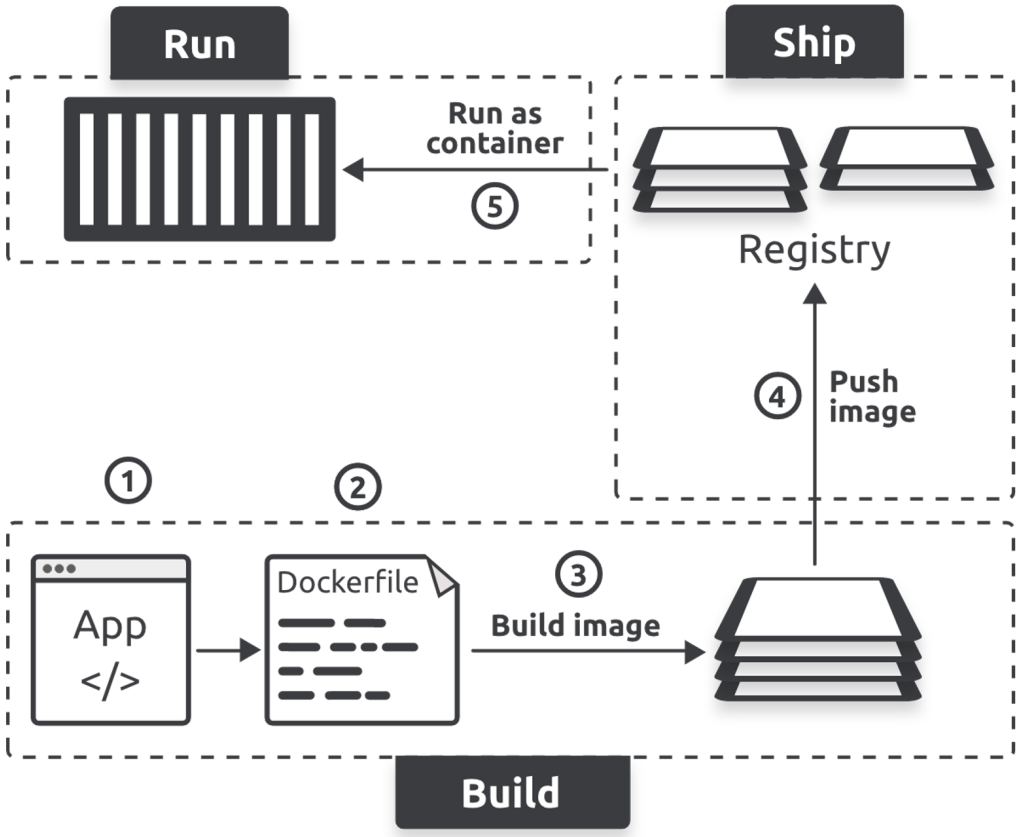


Docker Compose

- `docker compose up`
- `docker compose stop`
- `docker compose restart`
- `docker compose ps`
- `docker compose down`
- `docker compose top`
- `docker compose down --volumes --rmi all`



Basic Flow of Containerizing an App



Containerize a single-container app [1/3]

```
$ git clone https://github.com/nigelpoulton/ddd-book.git
$ cd ddd-book/node-app
$ ls -l
$ docker init
Welcome to the Docker Init CLI!
<Snip>
? What application platform does your project use? Node
? What version of Node do you want to use? 20.8.0 <<---- Newer versions are OK
? Which package manager do you want to use? npm
? What command do you want to use to start the app? node app.js
? What port does your server listen on? 8080
CREATED: .dockerignore
CREATED: Dockerfile
CREATED: compose.yaml
CREATED: README.Docker.md
Your Docker files are ready!
```

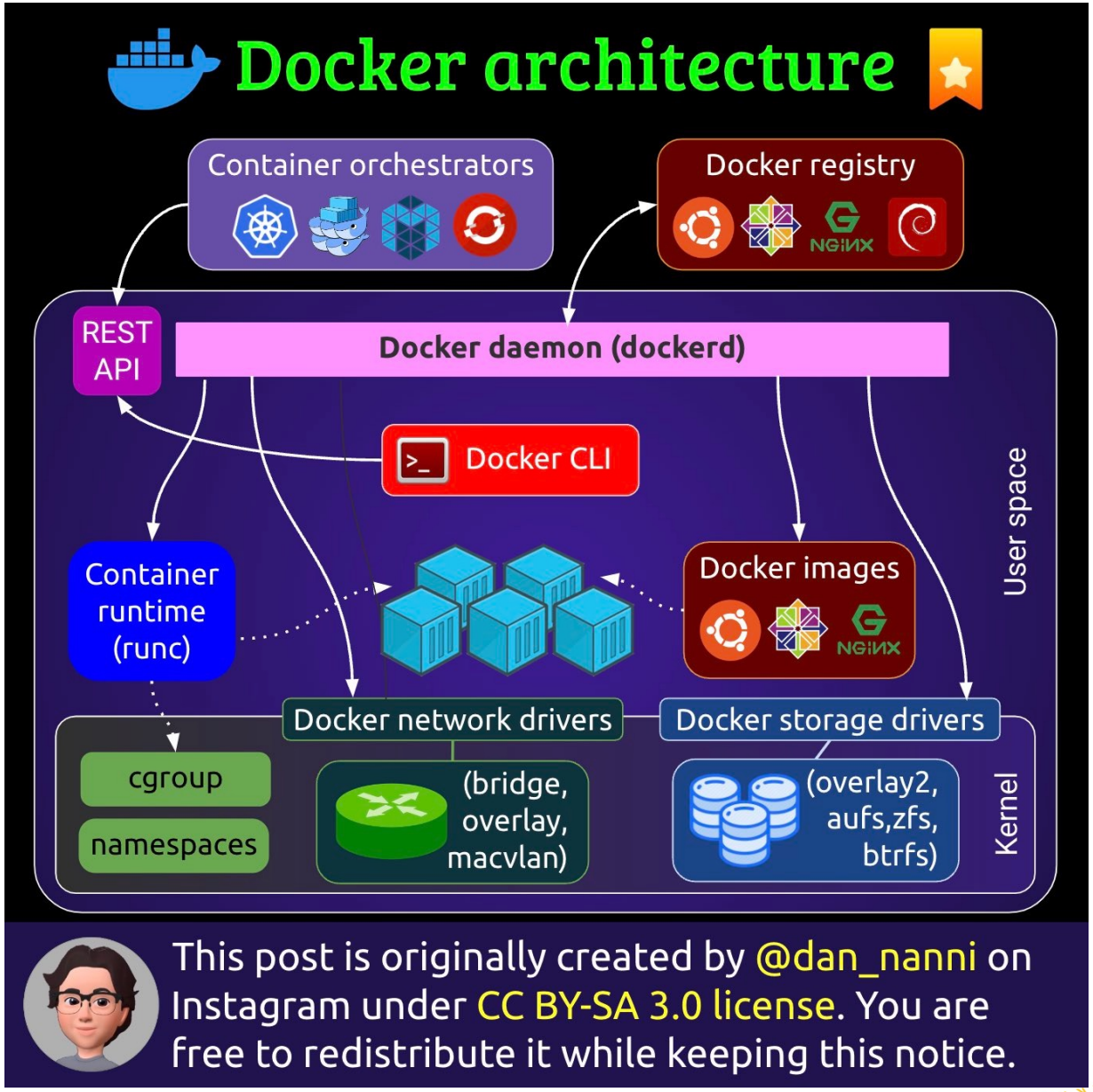
Containerize a single-container app [2/3]

```
1. ARG NODE_VERSION=20.8.0
2. FROM node:${NODE_VERSION}-alpine
3. ENV NODE_ENV production
4. WORKDIR /usr/src/app
5. RUN --mount=type=bind,source=package.json,target=package.json \
   --mount=type=bind,source=package-lock.json,target=package-lock.json \
   --mount=type=cache,target=/root/.npm \
   npm ci --omit=dev
6. USER node
7. COPY . .
8. EXPOSE 8080
9. CMD node app.js
```

Containerize a single-container app [3/3]

```
$ docker build -t ddd-book:test.node .
$ docker images
$ docker inspect ddd-book:test.node
$ docker login -u username
$ docker tag ddd-book:test.node username/ddd-book:test.node
$ docker push username/ddd-book:test.node
$ docker run -d --name c1 -p 5005:8080 username/ddd-book:test.node
$ docker ps
```

You can test by: http://localhost:5005

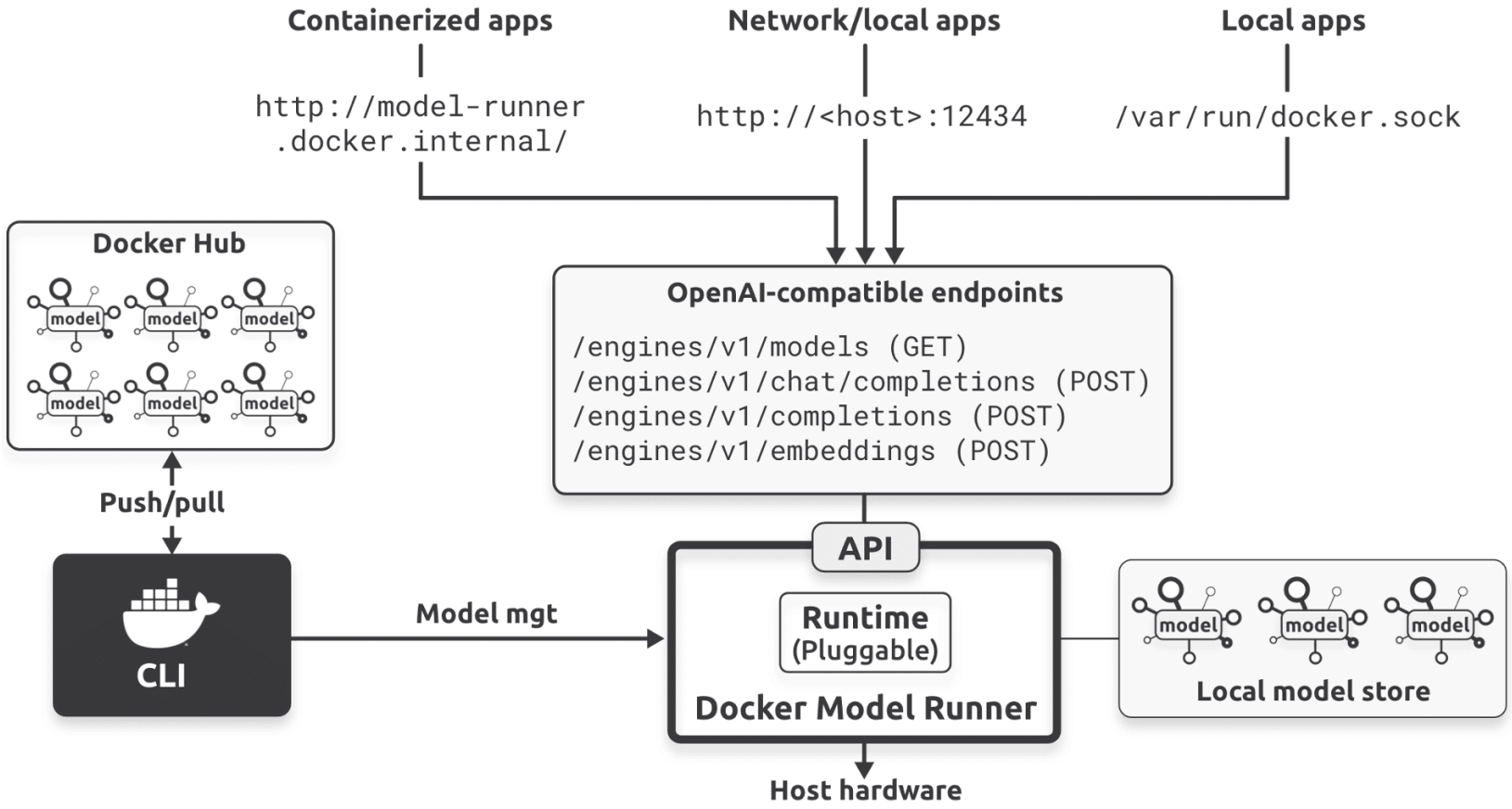


https://twitter.com/xmodulo/status/1705377071650992342/photo/1

Docker Model Runner (DRM)

- DMR is a new technology, fully integrated with the Docker toolchain, that executes AI models directly on host machines rather than inside containers.
- Containers can access modern NVIDIA GPUs if you install the NVIDIA Container Toolkit.
- By executing models outside of containers, Docker Model Runner gives you the best of both worlds
  - Integration with Docker tools and the wider cloud native ecosystem
  - Easier access to AI acceleration hardware
- DMR executes models directly on host hardware, exposes them via OpenAI-compatible endpoints.
- If you're already running local models, you'll recognize similarities with tools like LM Studio and Ollama.
  - For example, they can all use **`llama.cpp`** as their core inference engine and can expose models via OpenAI-style endpoints.

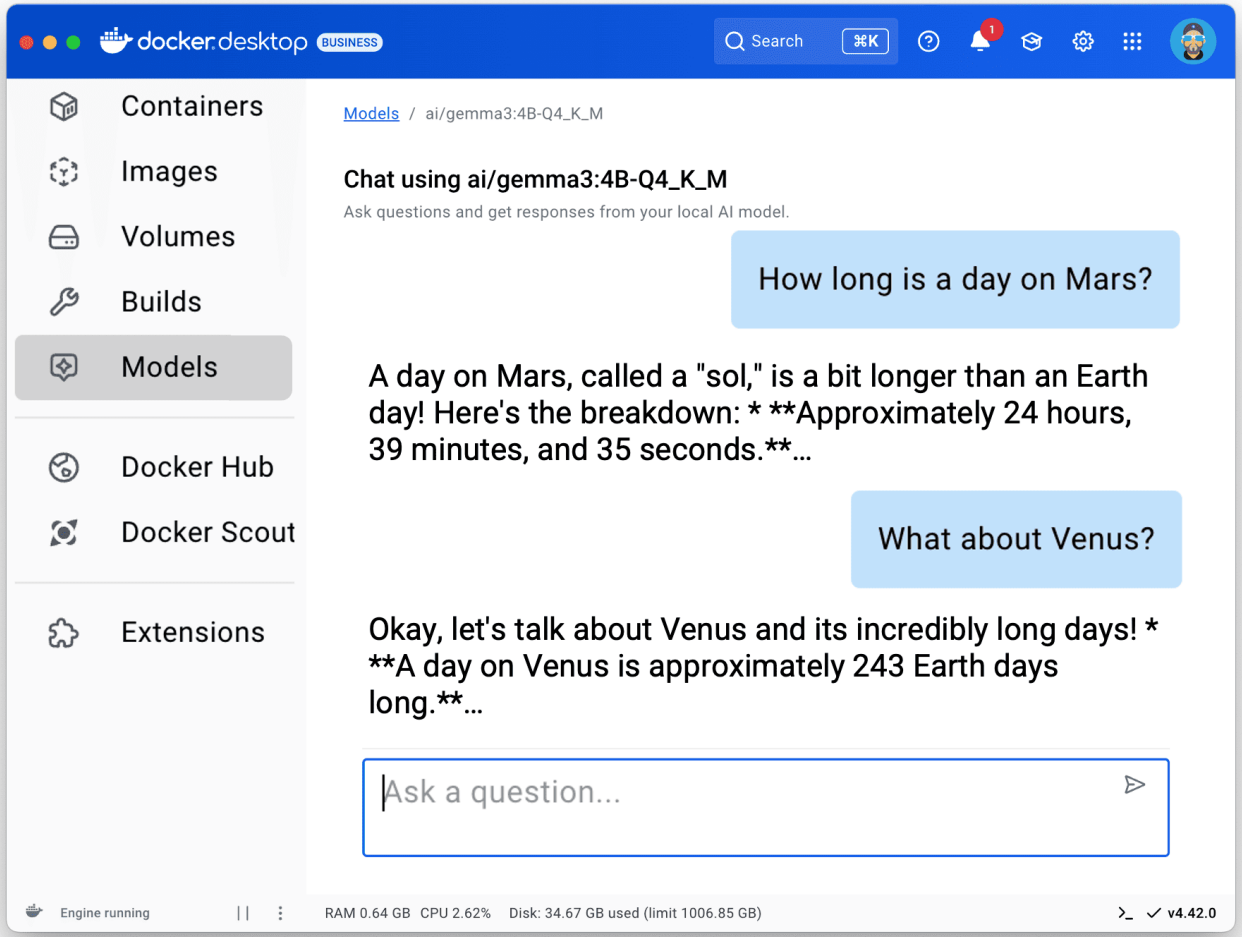
Docker Model Runner (DRM)



Docker Model Runner (DRM)

- Docker maintains a catalog of models below the ai namespace: https://hub.docker.com/u/ai
- https://hub.docker.com/catalogs/models
- `git clone https://github.com/nigelpoulton/ddd-book.git`
- `cd ddd-book/dmr`
- `docker model pull ai/gemma3:4B-Q4_K_M`
- `docker model ls`
- `docker manifest inspect ai/gemma3:4B-Q4_K_M | jq`
- `docker model run ai/gemma3:4B-Q4_K_M`

Docker Desktop's Model



Inspect the DMR API

- Docker Model Runner exposes a set of native endpoints for model management and a set of OpenAI-compatible endpoints for model interaction.
- DMR endpoints
  - `/models` <<---- GET
  - `/models/create` <<---- POST
  - `/models/{namespace}/{name}` <<---- GET and DELETE
- OpenAI-compatible endpoints
  - `/engines/llama.cpp/v1/models` <<---- GET
  - `/engines/llama.cpp/v1/chat/completions` <<---- POST
  - `/engines/llama.cpp/v1/completions` <<---- POST
  - `/engines/llama.cpp/v1/embeddings` <<---- POST

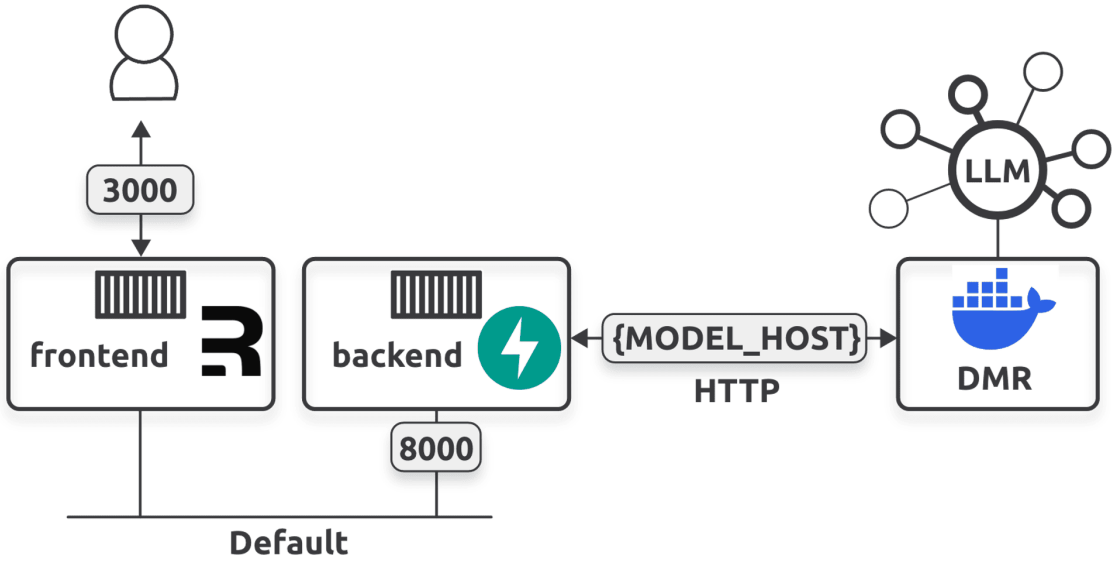
Inspect the DMR API

```
$ curl -s localhost:12434/engines/v1/models | jq
{
  "object": "list",
  "data": [
    {
      "id": "ai/gemma3:4B-Q4_K_M",
      "object": "model",
      "created": 1742979452,
      "owned_by": "docker"
    }
  ]
}
```

```
$ curl -s http://localhost:12434/engines/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "ai/gemma3:4B-Q4_K_M",
    "messages": [
      {
        "role": "system",
        "content": "Keep your responses to one sentence only."
      },
      {
        "role": "user",
        "content": "How long is a day on Mars?"
      }
    ],
    "temperature": 0.7,
    "max_tokens": 500
  }' | jq -r '.choices[0].message.content'

A day on Mars, also known as a sol, is approximately 24.6 hours long.
```

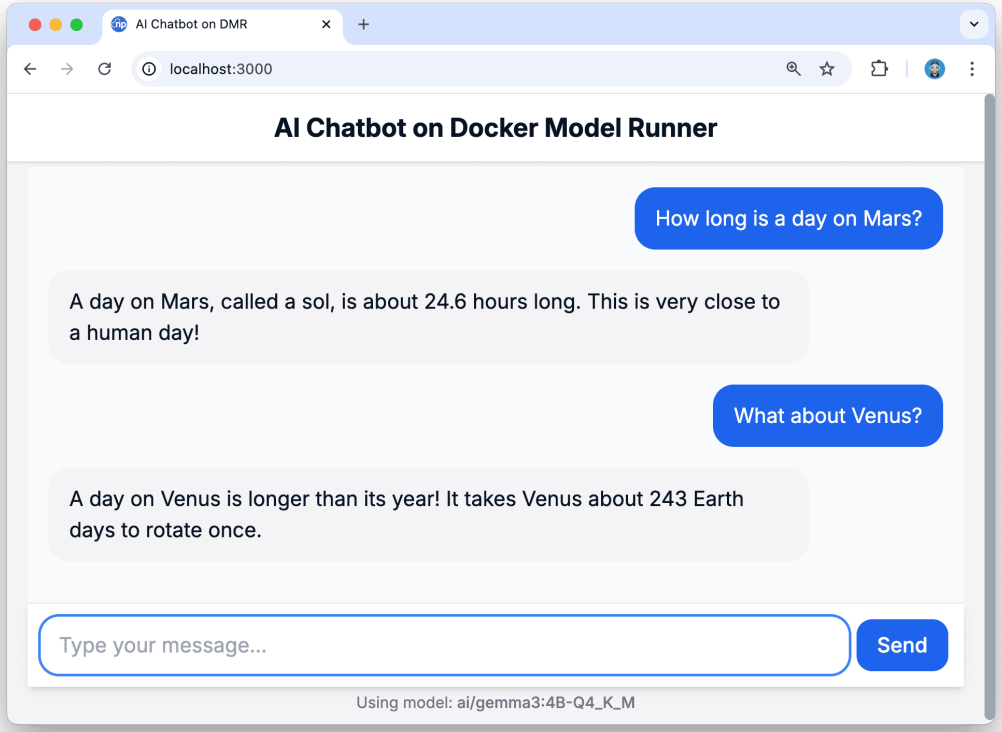
Use Docker Model Runner with Compose

```
services:
  frontend:                          ----┐
    build: ./frontend                    |
    ports:                               |
      - "3000:3000"                      | Frontend
    env_file:                            | service
      - .env                             |
    depends_on:                          |
      - backend                      ----┘
  backend:                           ----┐
    build: ./backend                     |
    ports:                               |
      - "8000:8000"                      | Backend
    env_file:                            | service
      - .env                             |
    depends_on:                          |
      - dmr                          ----┘
  dmr:                               ----┐
    provider:                            |
      type: model                        | DMR
      options:                           |
        model: ${LLM_MODEL_NAME}     ----┘
```



Use Docker Model Runner with Compose

```
docker compose up --build --detach
```



Use Docker Model Runner with Open WebUI

- `cd ddd-book/openwebui`
- `docker model status`
- `docker model pull ai/qwen3:0.6B-Q4_K_M`
- `docker model ls`
- `docker compose up`