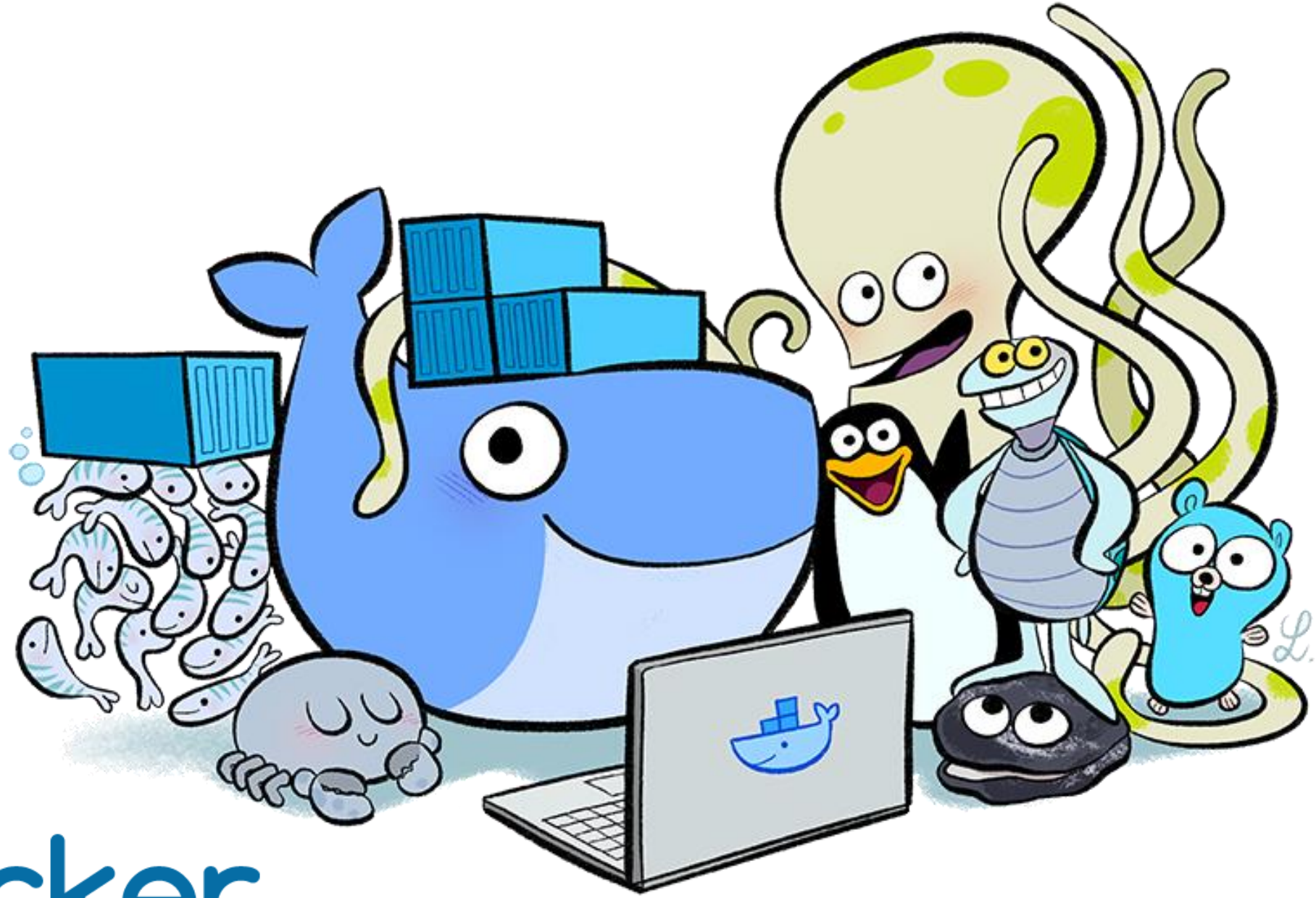
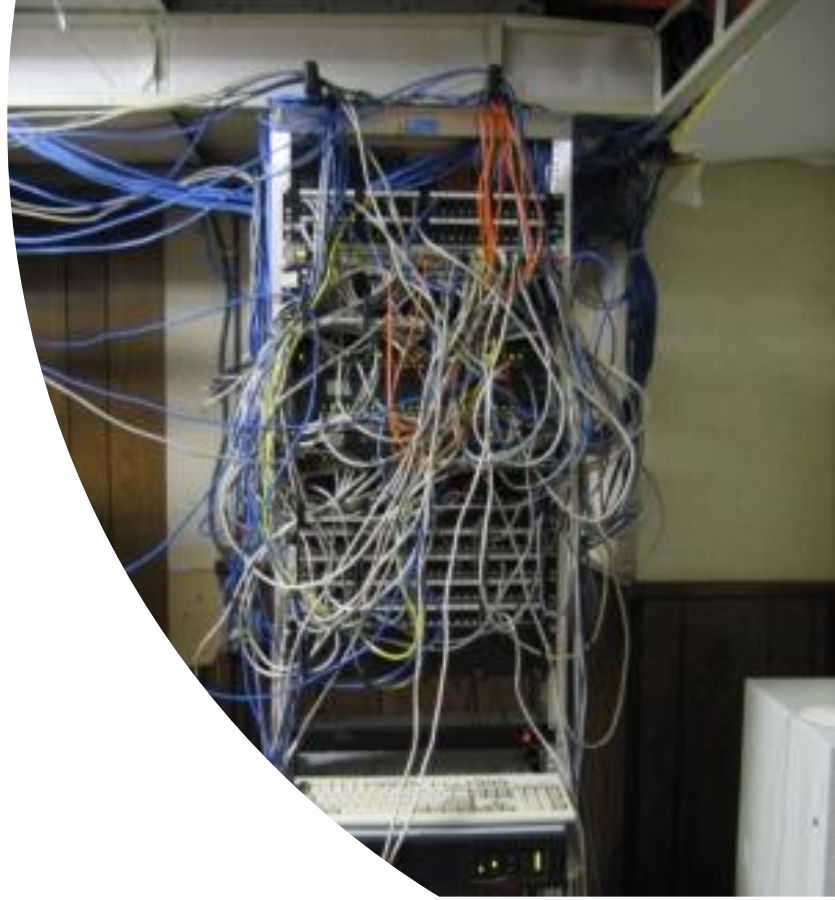


docker

History of containers

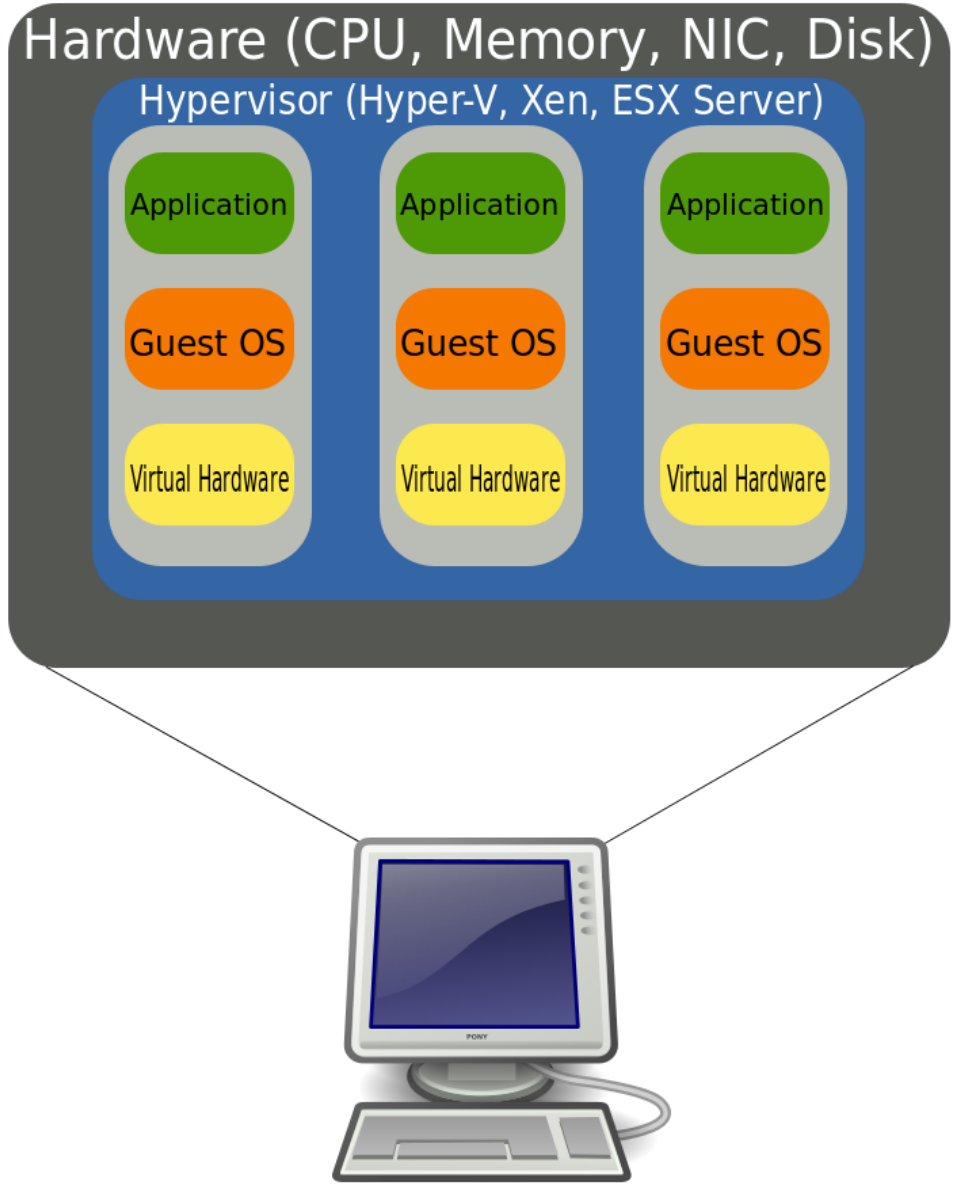
One Server – One App

- No technologies to run several apps at one server safely
- New app – need to by new server
- Need to guess in prior power of servers
- Some servers are under-powered
- Some servers are over-powered
- Waste of money



Virtual Machines

- Good technology to run several applications at one server
- Efficient resources consumptions
- No waste money
- Cloud solutions



vmware®

VirtualBox

Parallels™

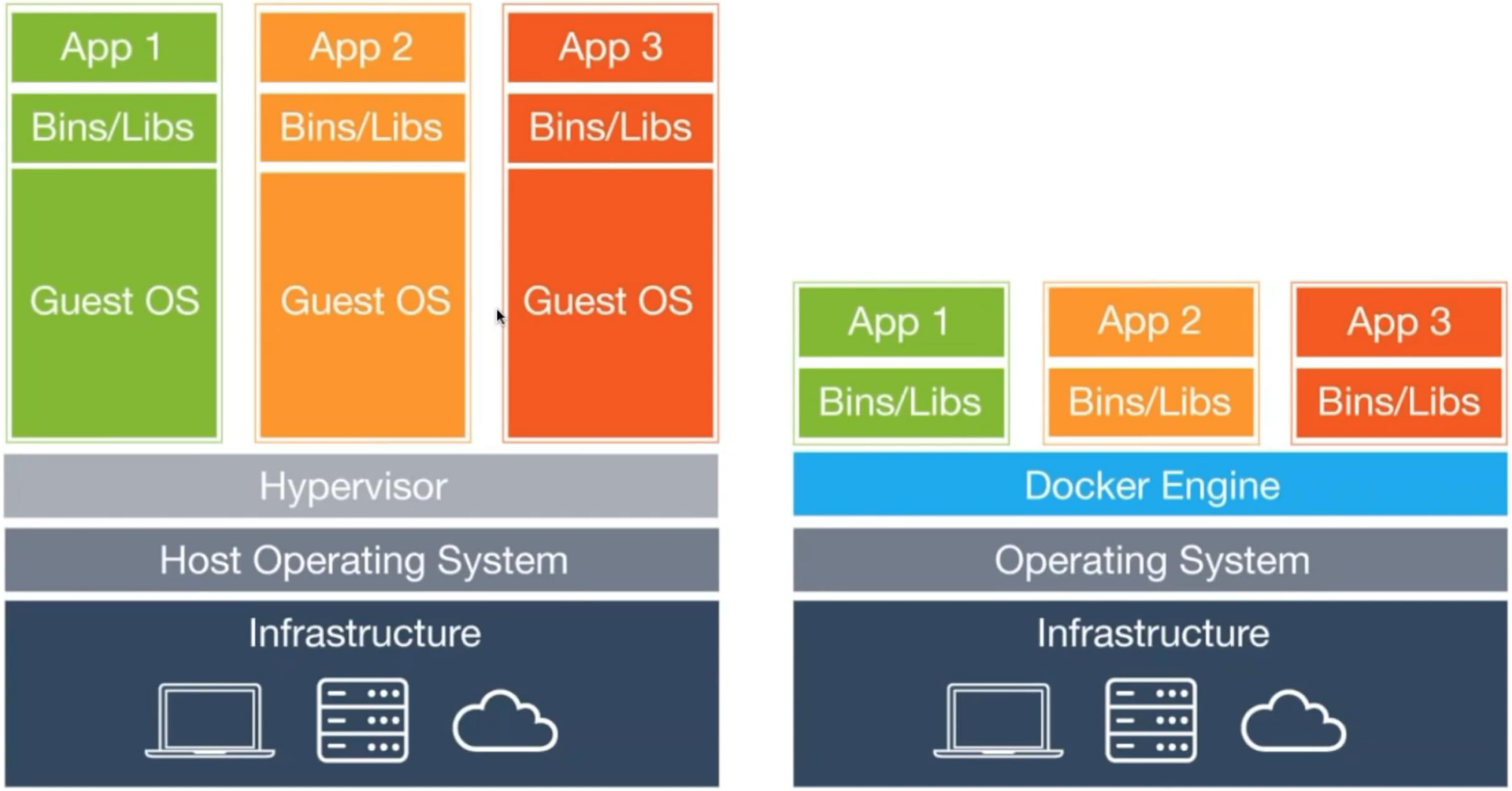
Virtual Machines disadvantages

- Each OS itself consumes resources
- Need license for each OS
- Need patching of each OS
- Slow to boot
- Big size

Containerization

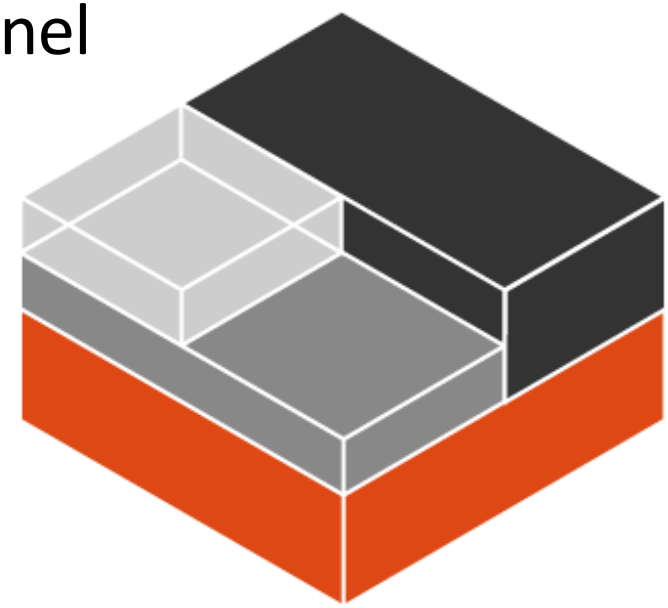
- **Containerization** - Technology for providing multiple isolated environments on a single host

Container vs VM



Linux containers (LSX)

- All Linux Family OS are build on the same kernel
- https://dzone.com/articles/evolution-of-linux-containers-future
- Control Groups (**cgroups**): isolating resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes
- A lot of isolation contributions were made to Linux kernel



Windows containers (2016)

- Changes in Windows kernel were made
- Supported started from Windows 10 and Windows Server 2016
- Windows Container
- Hyper V

Windows containers vs Linux containers

- Windows containers are based on windows kernel
- Linux containers are based on Linux kernel
- App that is designed to run on windows can’t be run inside Linux container and vise versa

What is Docker

1. **Docker Inc** – company
2. Docker is a **software** that performs containerization (operating-system-level virtualization)
3. Open source project (Moby) 

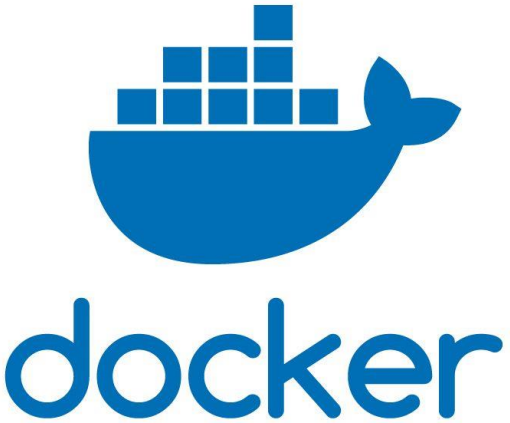
Docker Inc history

- Founded in San Francisco as **dotCloud** in 2010 by **Solomon Hykes**
- Started as **PaaS** provides
- **dotCloud** was build using Linux containers
- To create and manage containers they created tool **Docker**
- In 2013 they rebranded company to Docker Inc, stopped PaaS business and focused on Docker tool

Docker Inc today

- Valuation is 1BN
- Was invested by many big names of ventral capital
- Holds **Dockercon** 

- 300-400 employees
- Docker, Inc is Dead - Chris Short
- Solomon Hykes left docker



Docker Captain

- https://www.docker.com/docker-captains

Docker Engine

The Docker Engine - lightweight and powerful open source containerization technology combined with a workflow for building and containerizing and orchestrating your applications.

- Enterprise Edition (supported for 12 month)
- Community Edition (supported for 4 month)

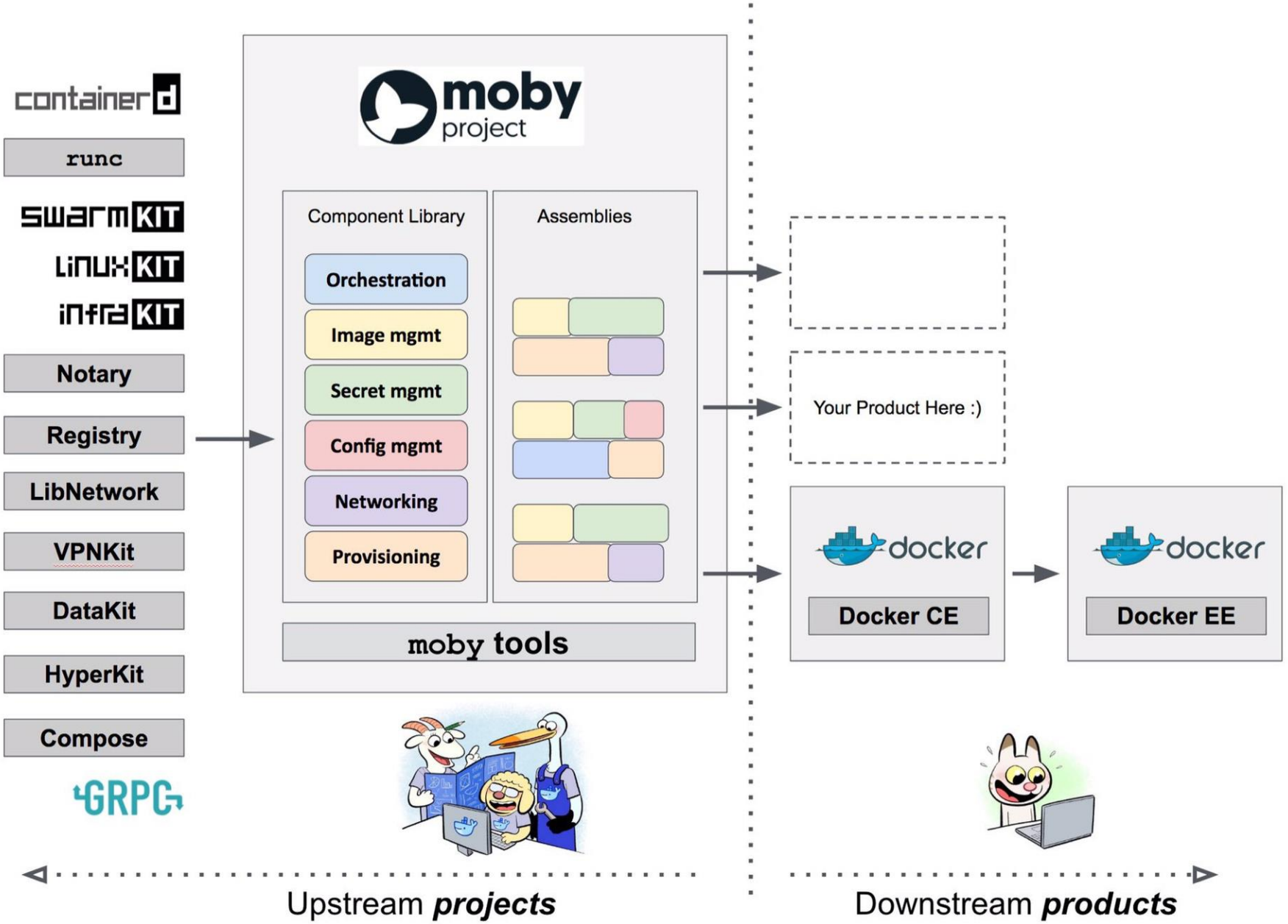
the YY.MM-xx versioning scheme

Moby (Docker opensource project)

Moby project has a goal is to be the upstream for Docker, and to break Docker down into more modular components

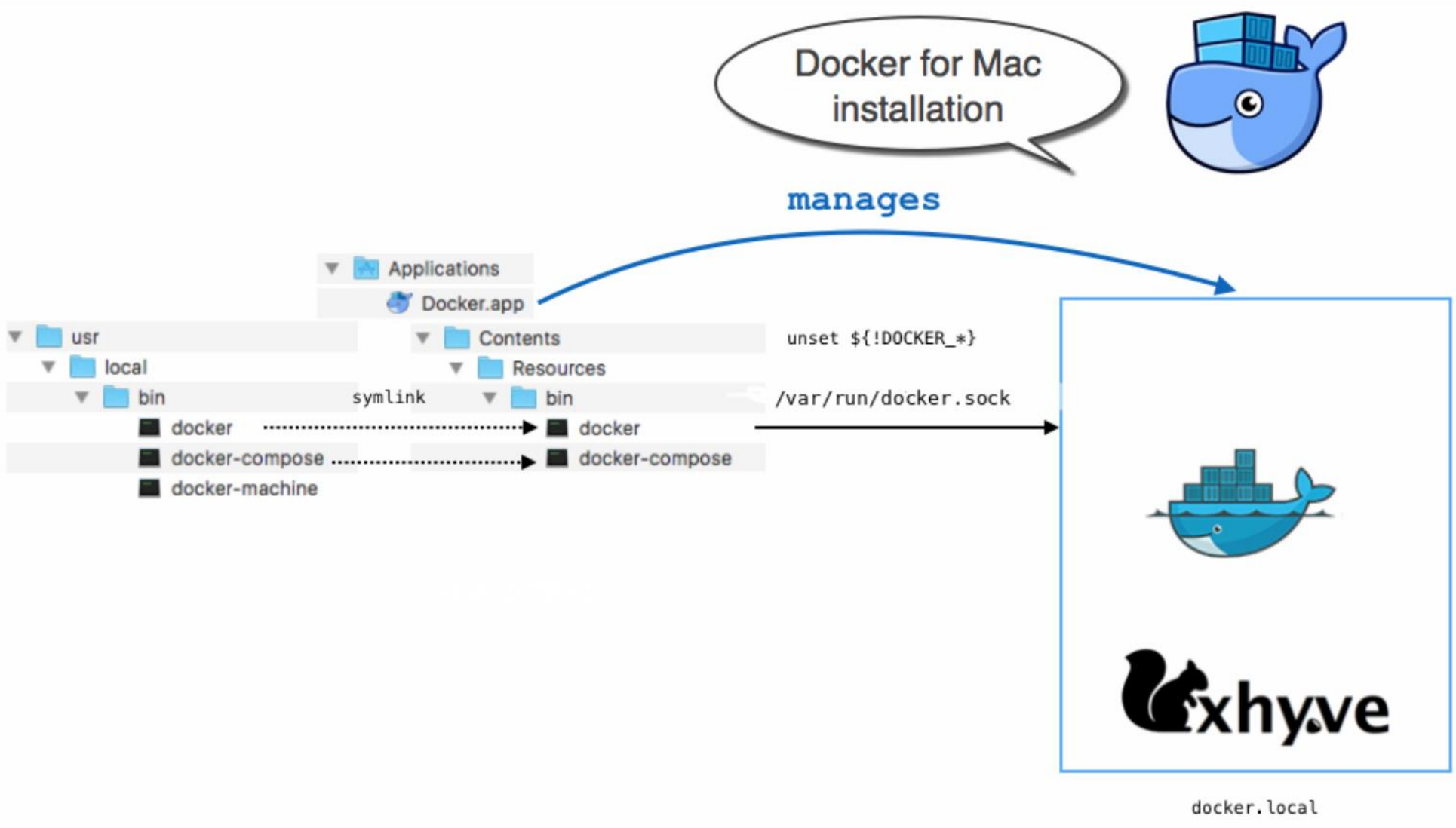
- Announced at DockerCon 2017
- Apache 2.0 license
- https://github.com/moby/moby
- Written in Go 
- Many committers from big companies (RedHat, Microsoft, Cisco, IBM, etc.)

https://boxboat.com/2017/04/28/moby-project-explained

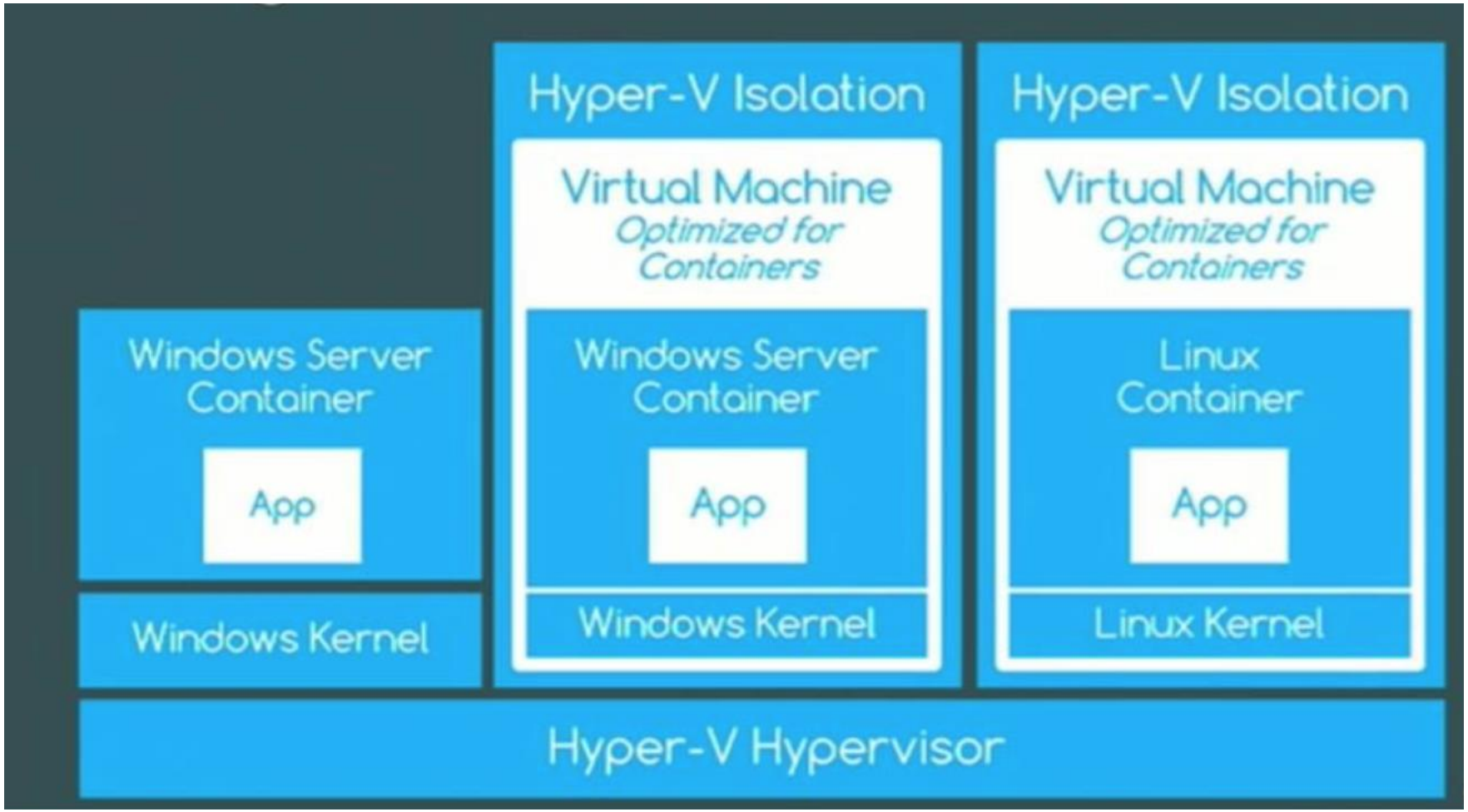


Install Docker

Docker on Mac



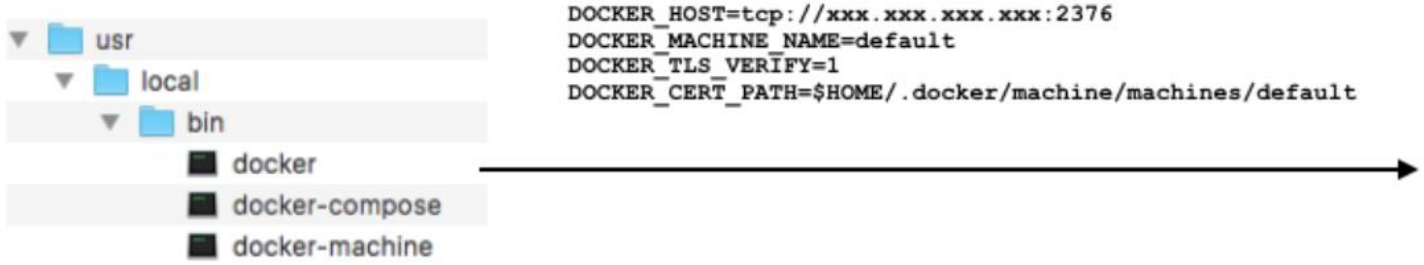
Docker on Windows 10



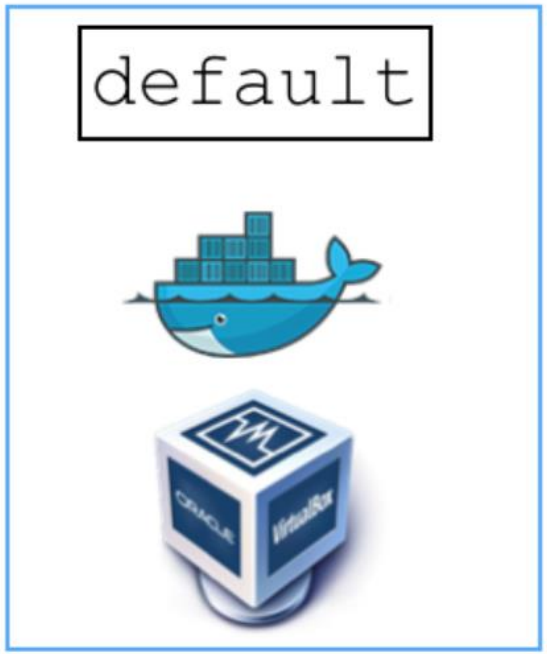
Microsoft
Hyper-V

Docker Toolbox (for old systems)

Docker Toolbox installation



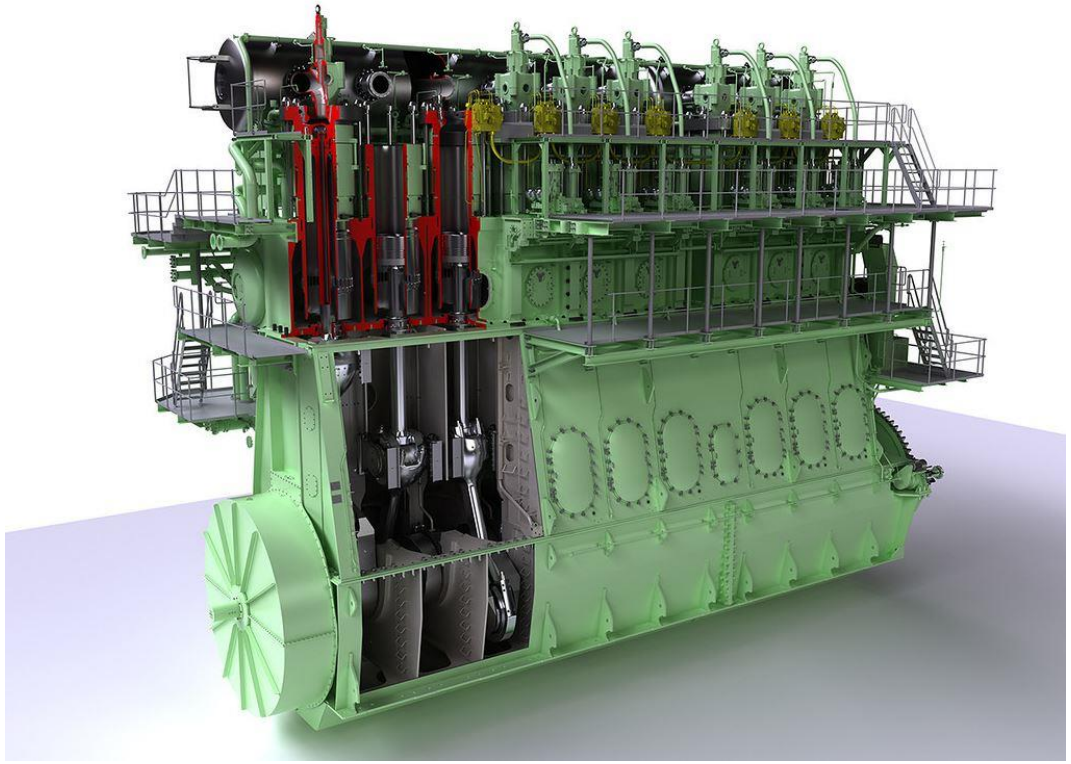

```
DOCKER_HOST=tcp://xxx.xxx.xxx.xxx:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
DOCKER_CERT_PATH=$HOME/.docker/machine/machines/default
```



manages

IP xxx.xxx.xxx.xxx

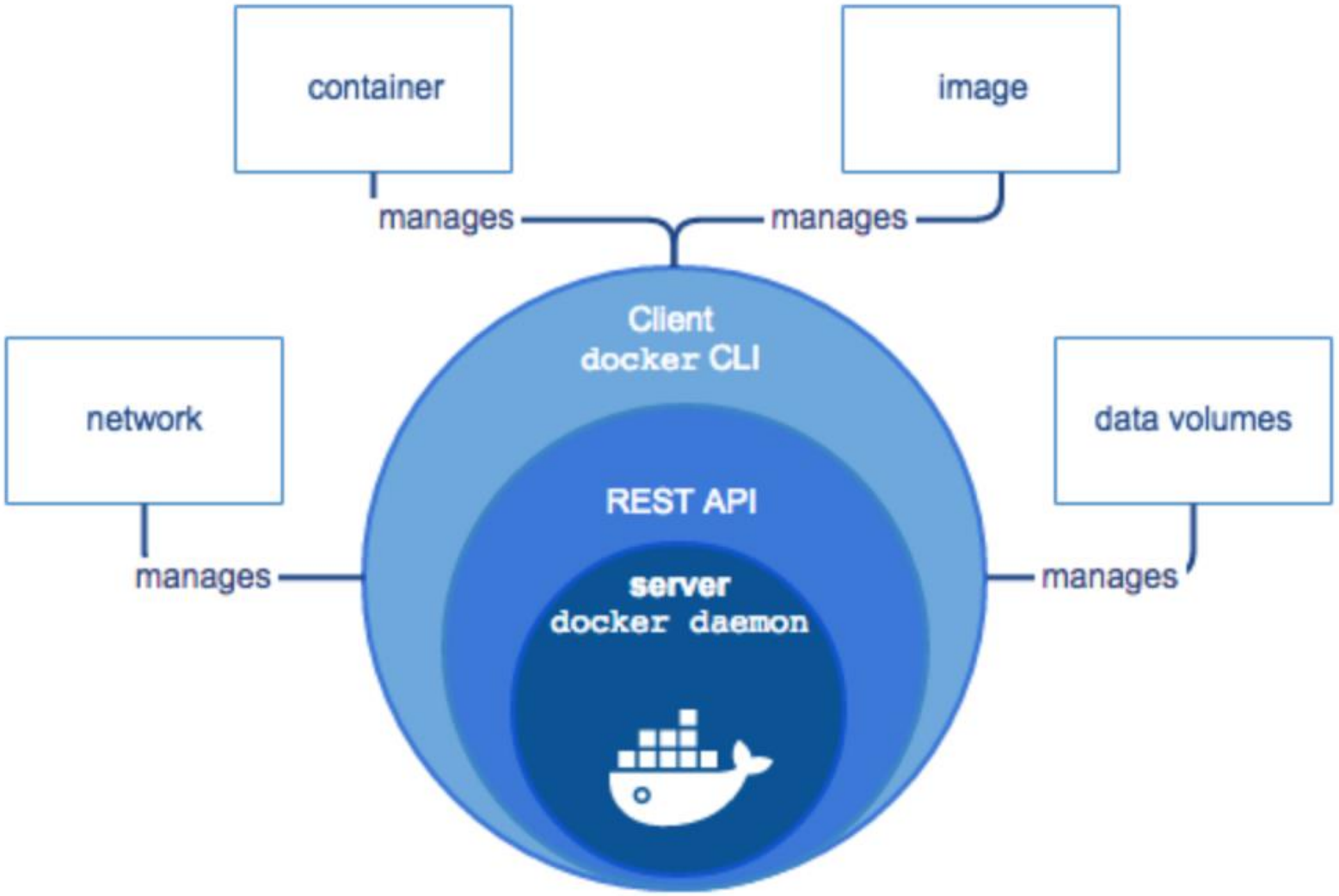
Docker Engine



Definition

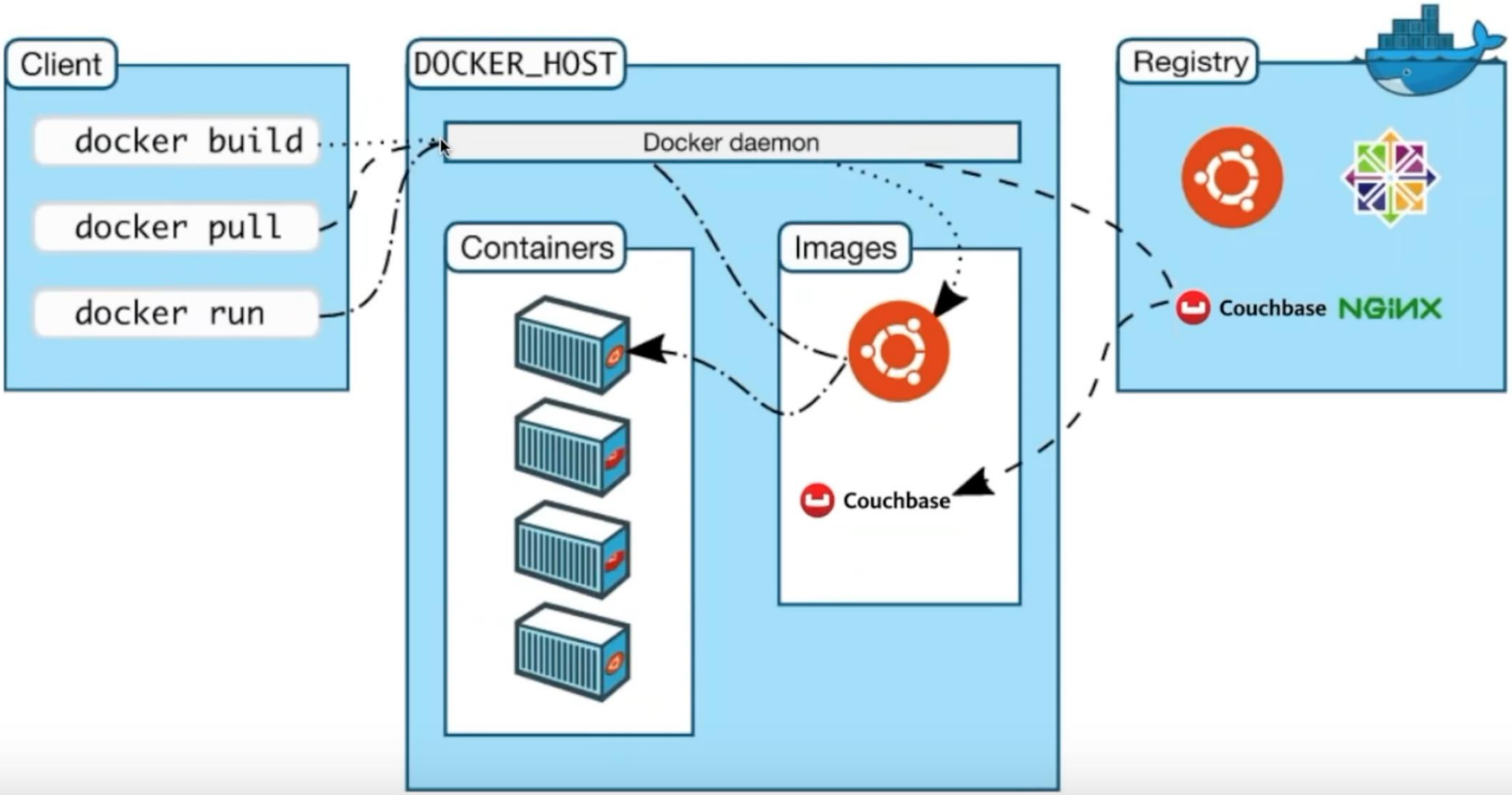
Docker Engine is a client-server application with 2 major components:

- Docker client
- Docker demon (sometimes called “daemon” or “engine”)



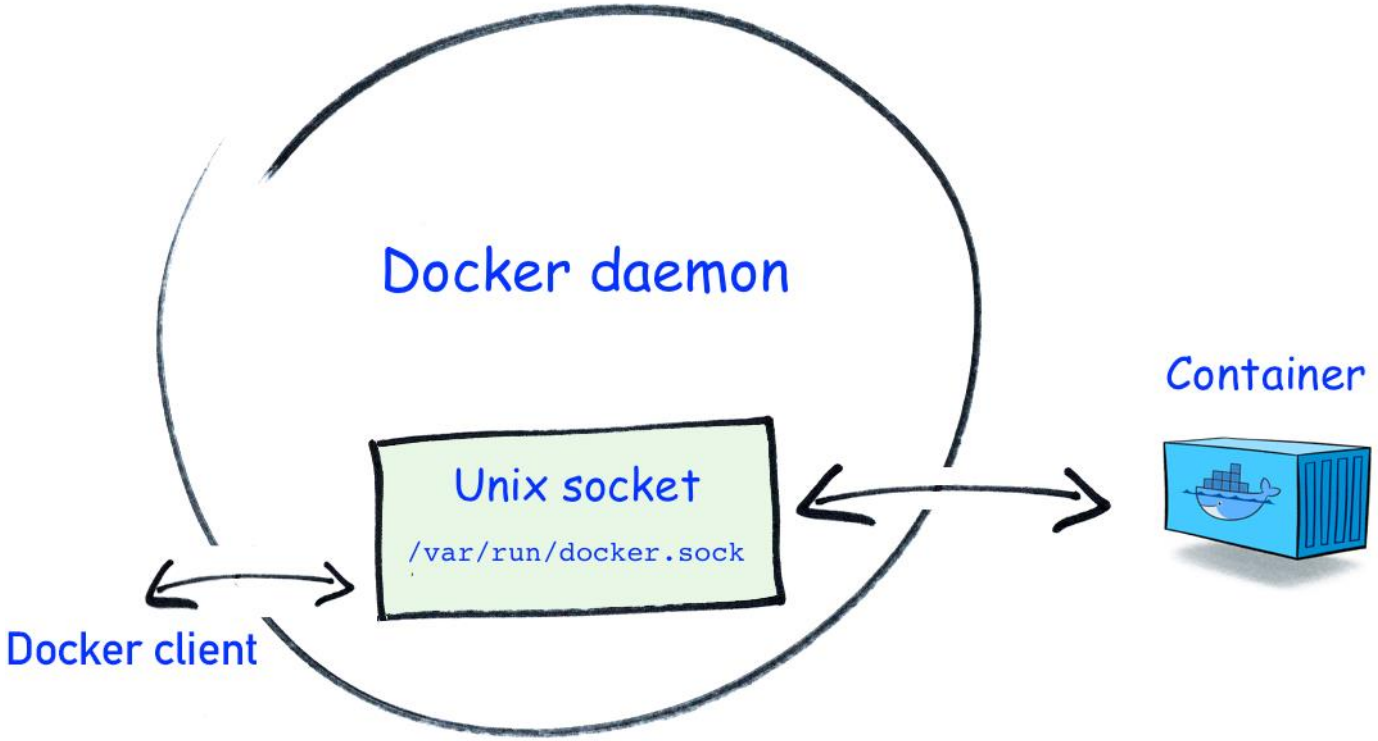
Docker Engine API

Docker Workflow



Docker Sock

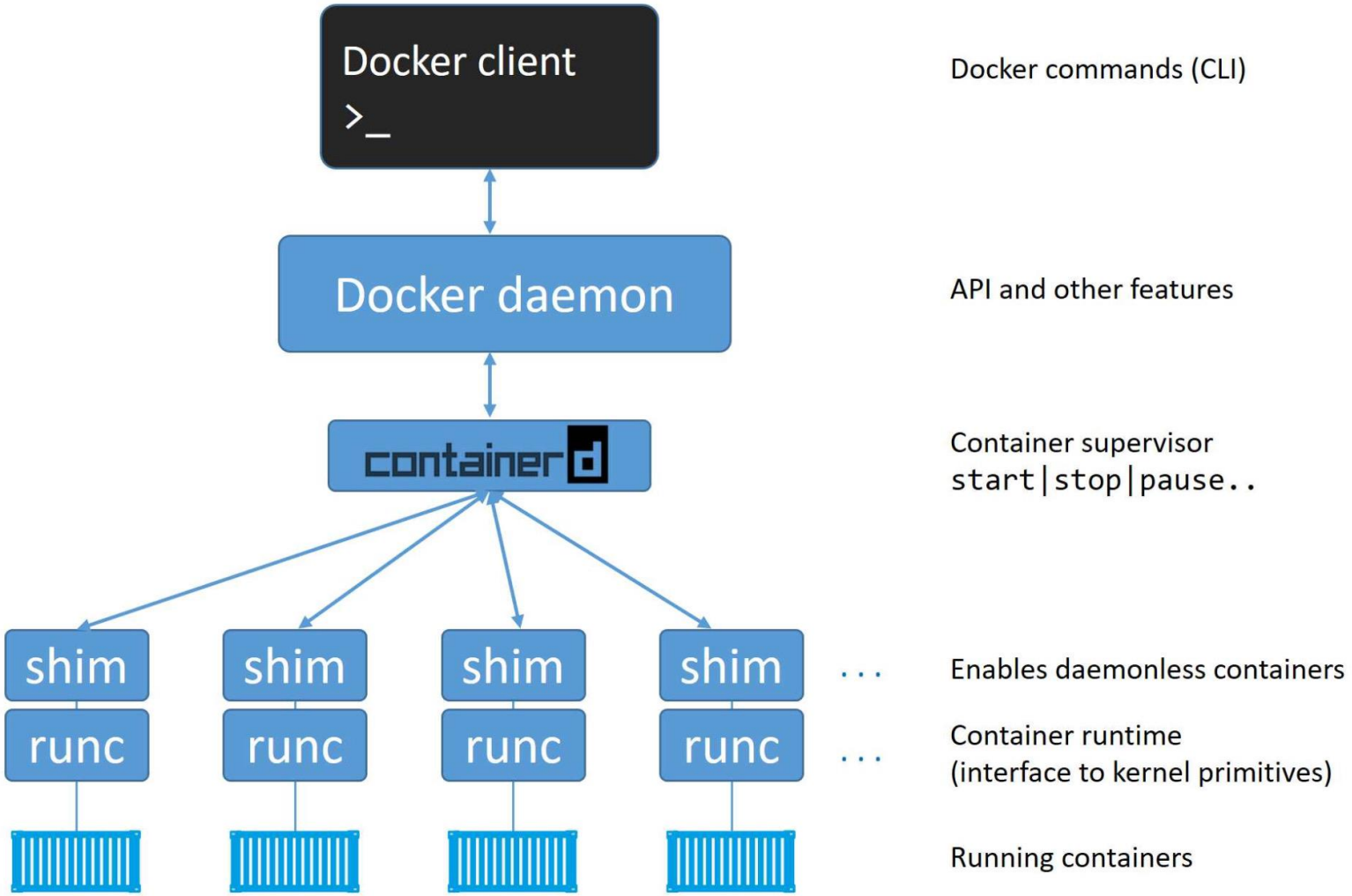
- client talks to the daemon via a local IPC/ Unix socket at **/var/run/ docker.sock**.
- via named pipe at **npipe:////./ pipe/ docker_engine** (Windows)



Docker version information

docker version allows to check version of client and daemon

Docker engine architecture



Docker commands (CLI)

API and other features

Container supervisor
`start|stop|pause..`

Enables daemonless containers

Container runtime
(interface to kernel primitives)

Running containers

is a project to design open standards around container standards and runtime

It defines two standarts:

- The image-spec - *how to deliver containers*
- The runtime-spec - *how to run containers*

runc

- reference implementation of the OCI c**ontainer-runtime-spec**
- lightweight CLI wrapper for libcontainer
- CLI tool for creating and running containers

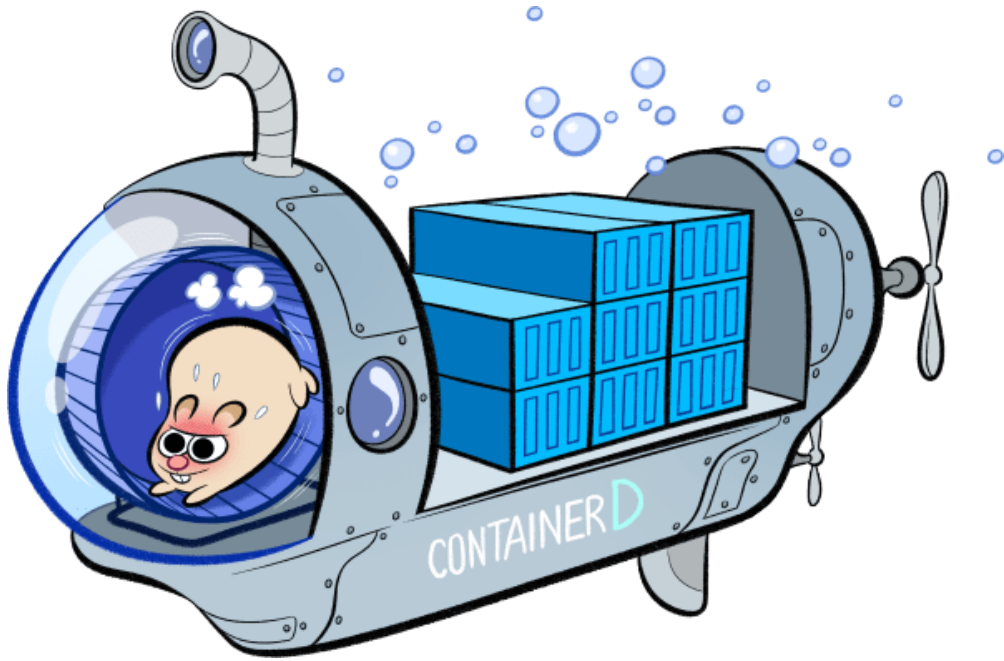
containerd

Tool that handles container lifecycle operations

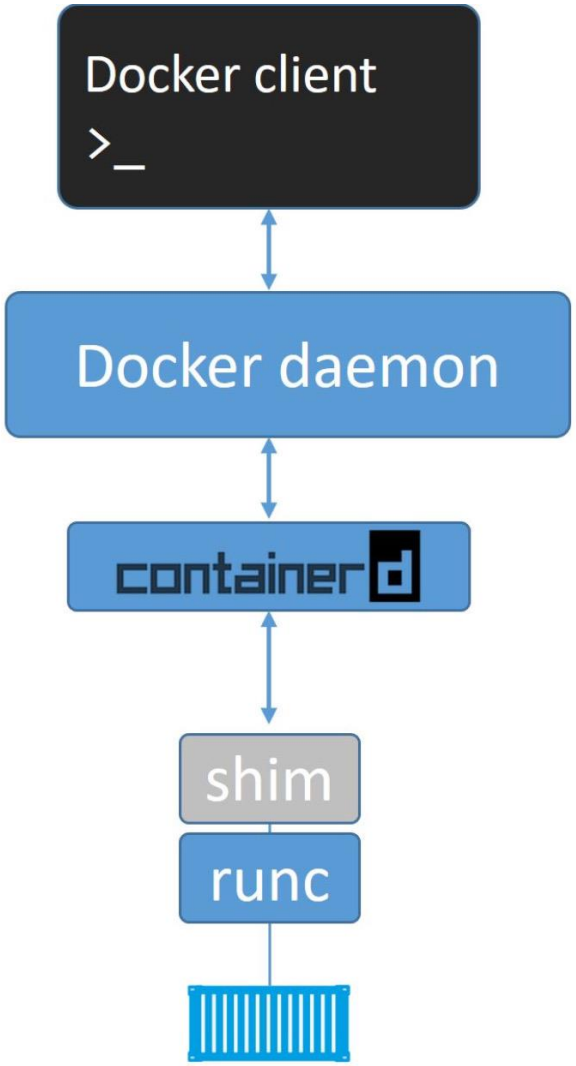
start | stop | pause | rm

Was donated to **Cloud Native Computing Foundation**

Provides gRPC API



docker run ubuntu



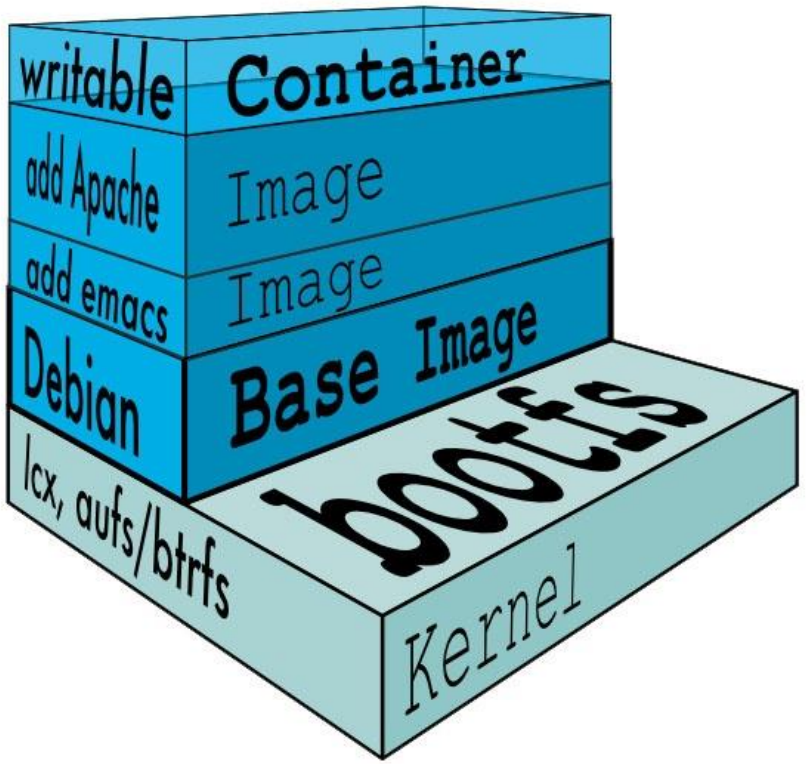
Issue `**docker container run**` command to **Docker API**.

Receive instruction at API endpoint.
Instruct **containerd** to start new container.

Give **runc** OCI bundle (image).
Instruct **runc** to create container.

Create container

Docker Images

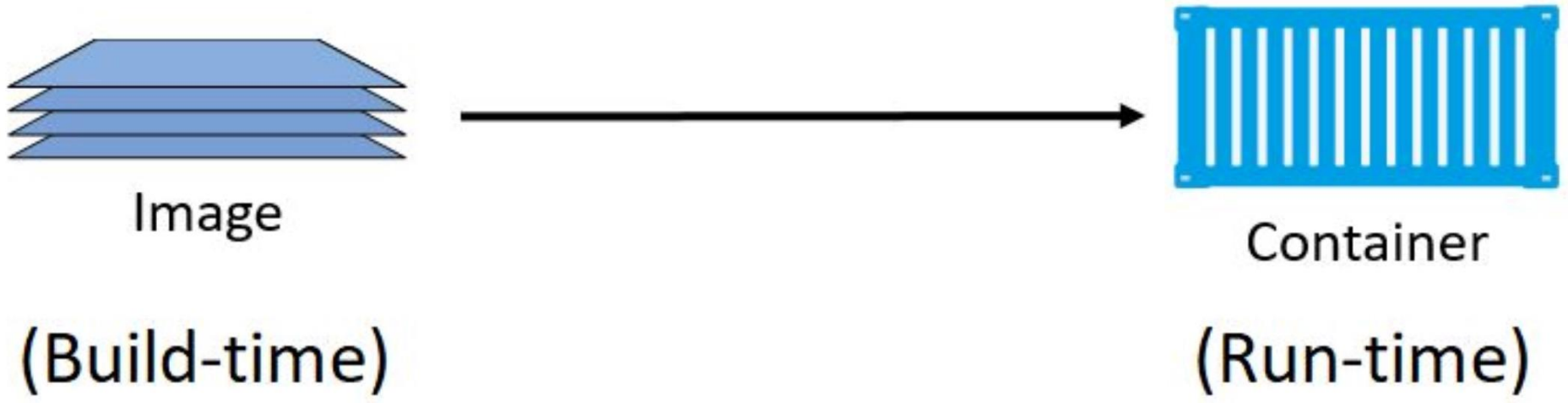


Image

- Image like a **stopped container**
- Image like a **class** (container is like an **object** that can be created form image)
- Images are made up of **multiple layers** that get stacked on top of each other and represented as a single object.
- Inside of the image is a **OS** and all of the **files** and **dependencies** required to run an application.

Images and containers

- docker run <image_name> - can be used to run one or **more** containers from **single image**
- Image can’t be deleted until the latest container is destroyed



Images should be small

- Container should be lightweight
- Images contains only essentials parts
- Do not contain kernel (use kernel of host)

Docker Alpine Linux

- A minimal Docker image based on Alpine Linux
- only 5 MB in size
- To compare:
  - ubuntu image (**110MB**)
  - microsoft/nanoserver (over 1GB)

Image Registry

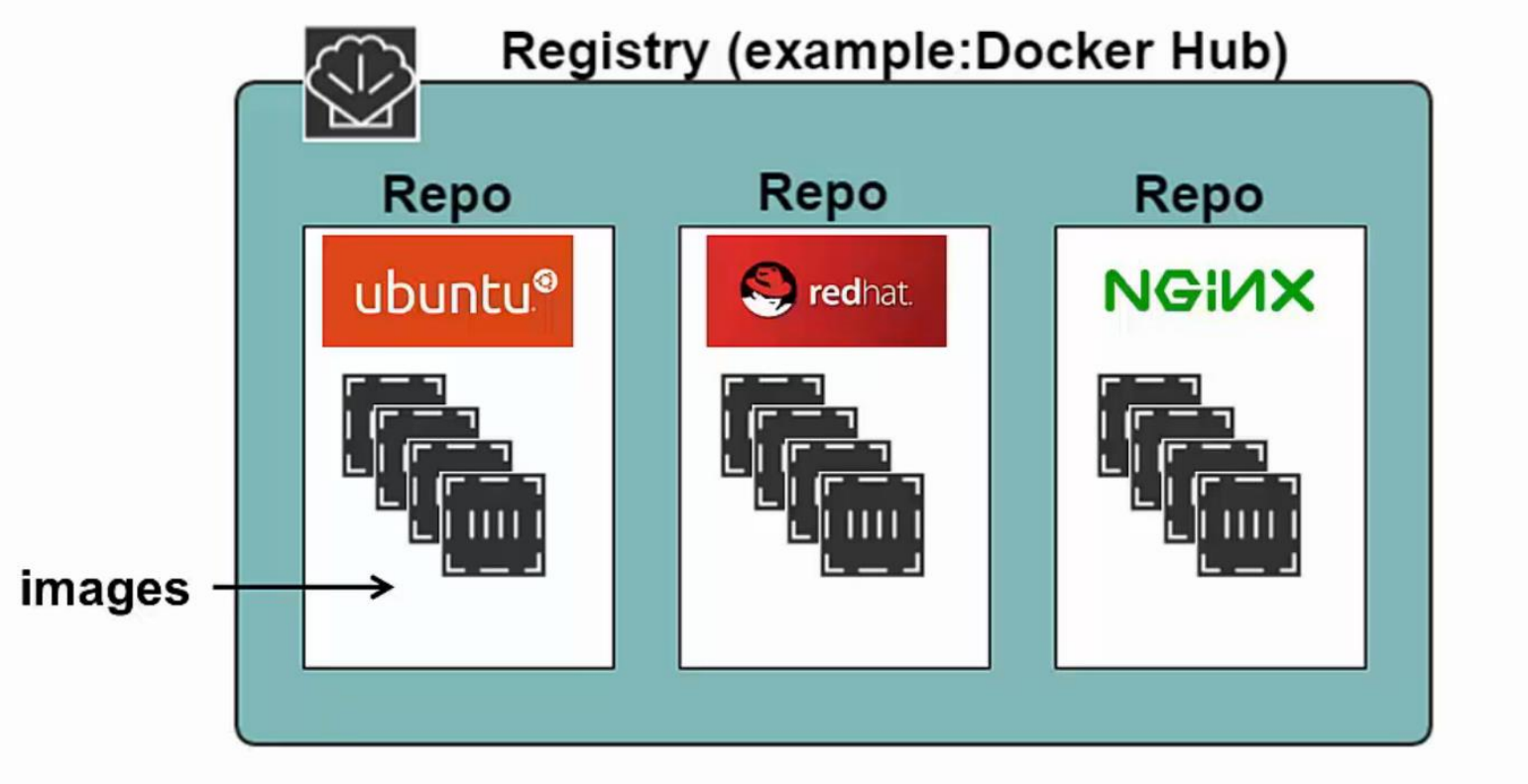
- Images are pulled from **image registries**
- The most common registry is **Docker Hub** (by default)
- Google Container Registry, Amazon ECR

Docker Hub

- Docker Hub (https://hub.docker.com)
- Some repositories are public some are private
- The Docker Hub Registry is free to use for public repositories (+ free **1 Private** Repository)
- Contains **official** and **unofficial** repositories

Image Repository

- Image Registry contains multiple **Image Repositories**
- Each Image Repository contain one ore more images



Pull image

docker pull <image_name>:<tag_name> - pull image from registry

After pulling image presents on the host local repository

List pulled images

docker image ls

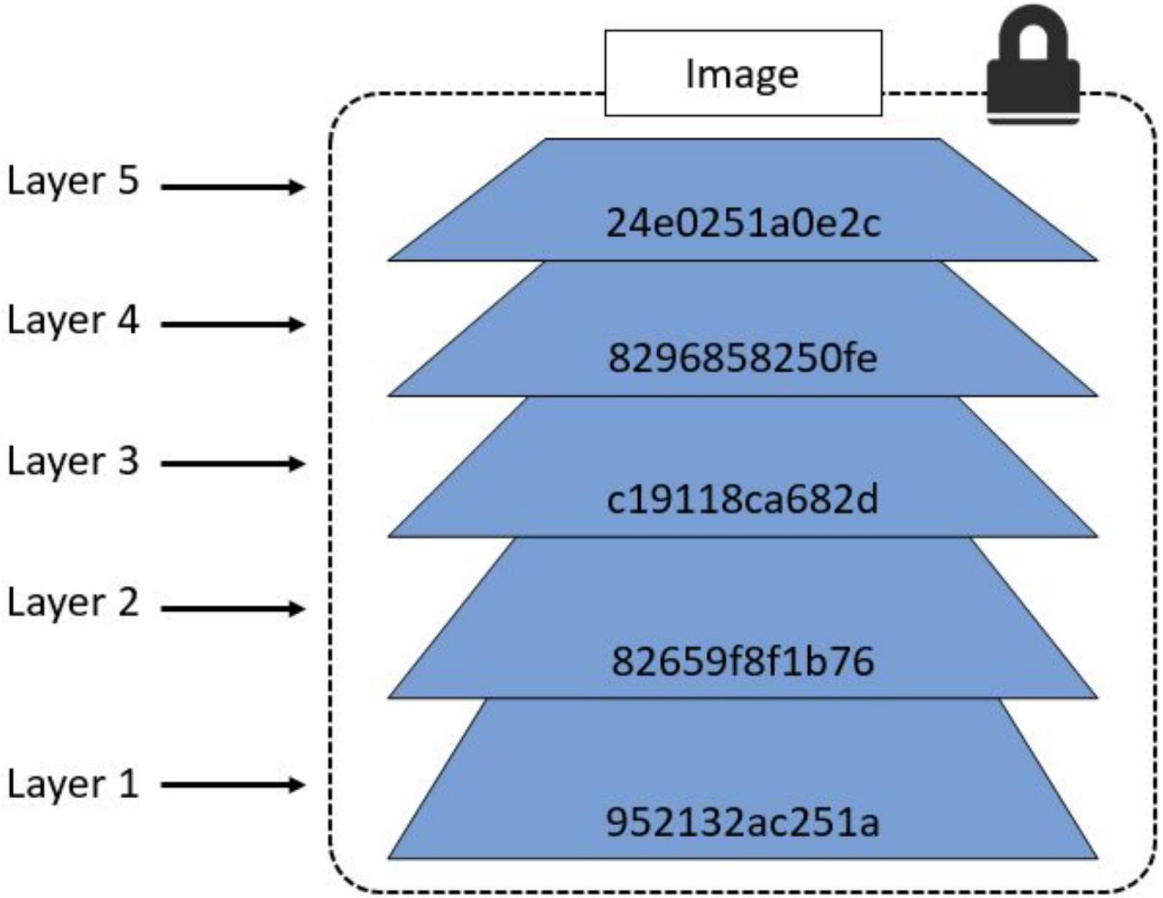
Search Repositories

docker search ubuntu

docker search ubuntu --filter “is-official=true”

Image and layers

Image is an object that represents a merge of those layers



Digest

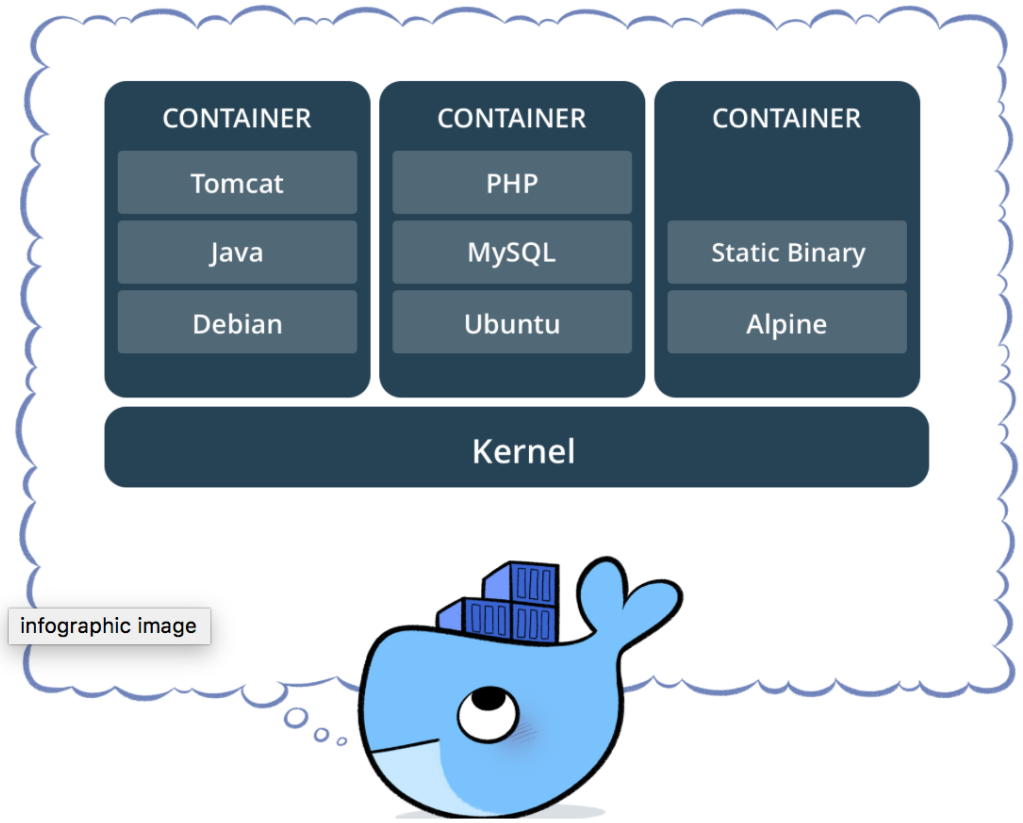
- Docker uses content addressable file system
- Digest (signature) for each layer is used to verify that layer is correct
- Digest of image is used to verify that the image (image is just configuration that lists layers and some metadata) is correct

Delete image

docker image rm <image_name> - deletes image and all of its layers from docker host

- if a layer is shared by some other images it will not be removed
- If the image you are trying to delete is in use by a running container you will not be able to delete it.
- **docker image prune -a**

Docker containers

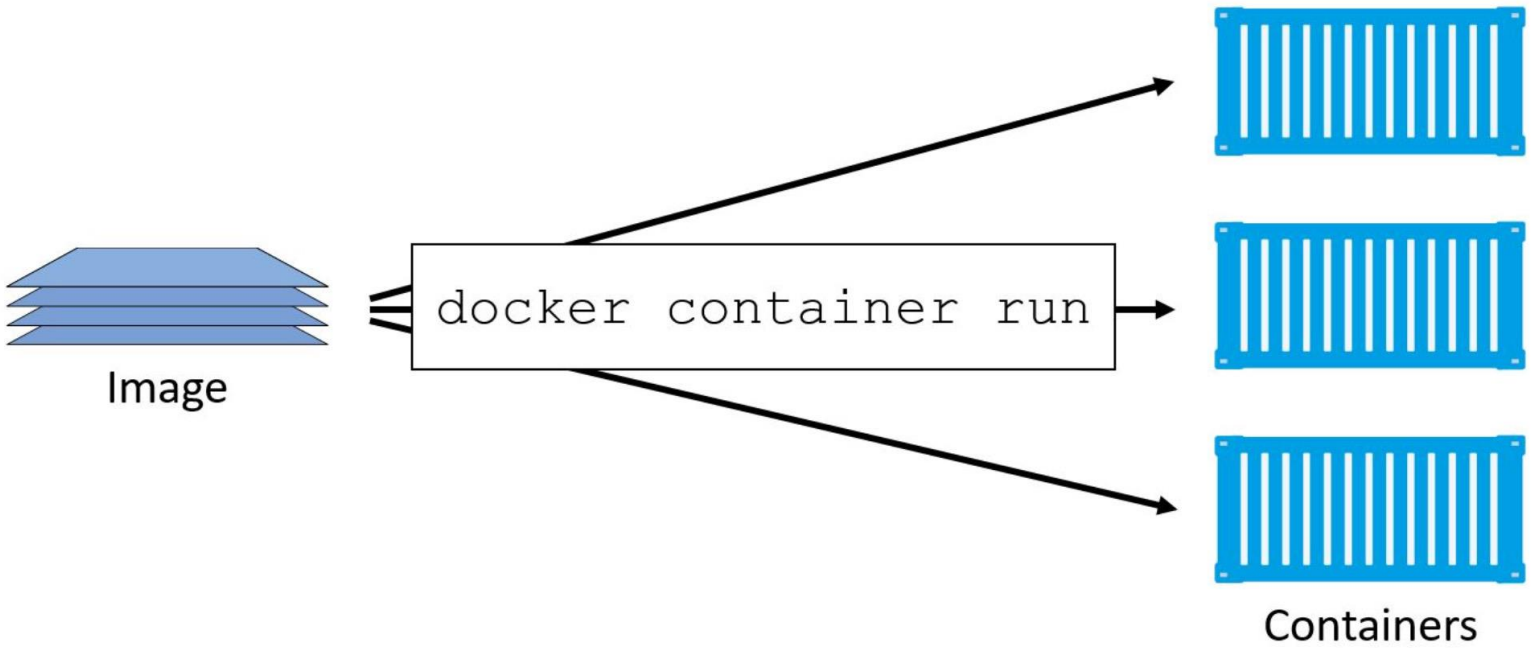


What is it?

docker container is running instance of image

docker run command

docker container run < image > < command>

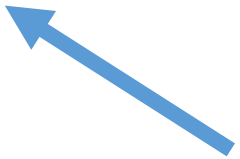


-it

-it flags connects current terminal window to the container's shell.

docker run –it ubuntu bash

```
root@b43d6f9a215e:/#
```



container id

Container process

- When we run container we specify command (process) to run
- This Process (with **pid 1**) is called main process
- killing the main process in the container will also kill the container
- **Ctrl-PQ** exits container without terminating it

Exec command

docker exec <container-id> –it <command> - creates new bash process and runs the command

If you exit the process, container will still be running

List containers

docker container ls – list running containers

docker container ls -a – list all (including stopped) containers

Hardware virtualization vs OS virtualization

- the physical server is powered on and the hypervisor boots
- hypervisor then divides hardware resources into virtual resources
- packages virtual resources into a VM

- OS boots
- we install a container engine
- container engine takes OS resources and divides them into into secure isolated constructs (containers)

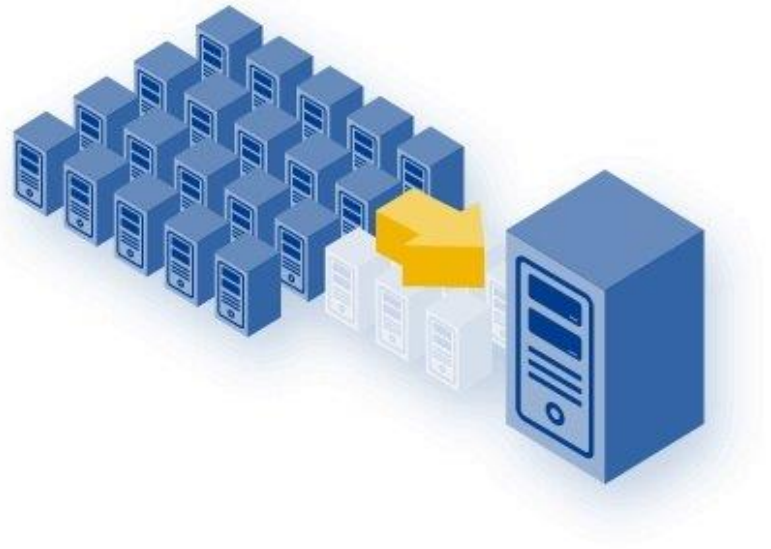
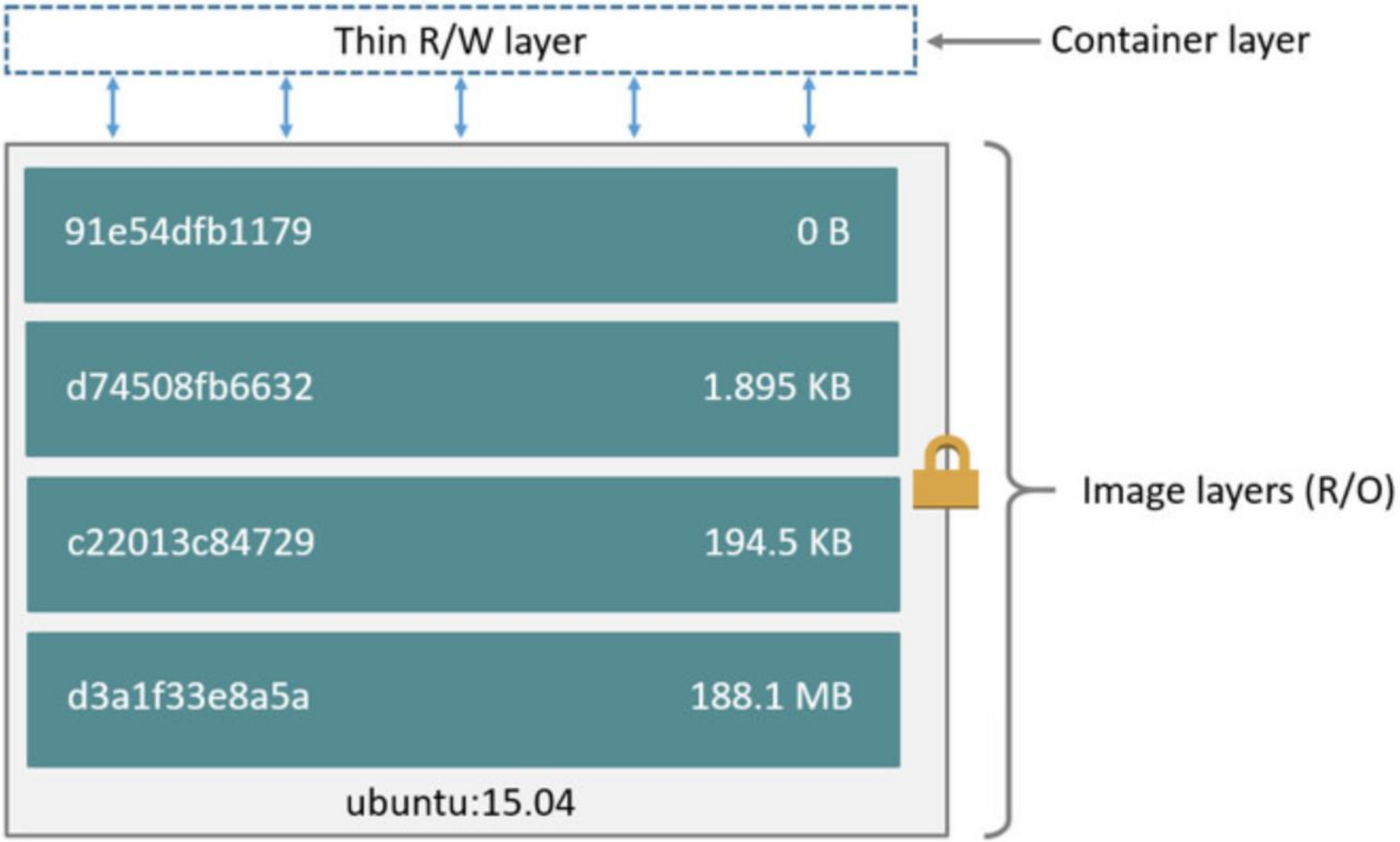
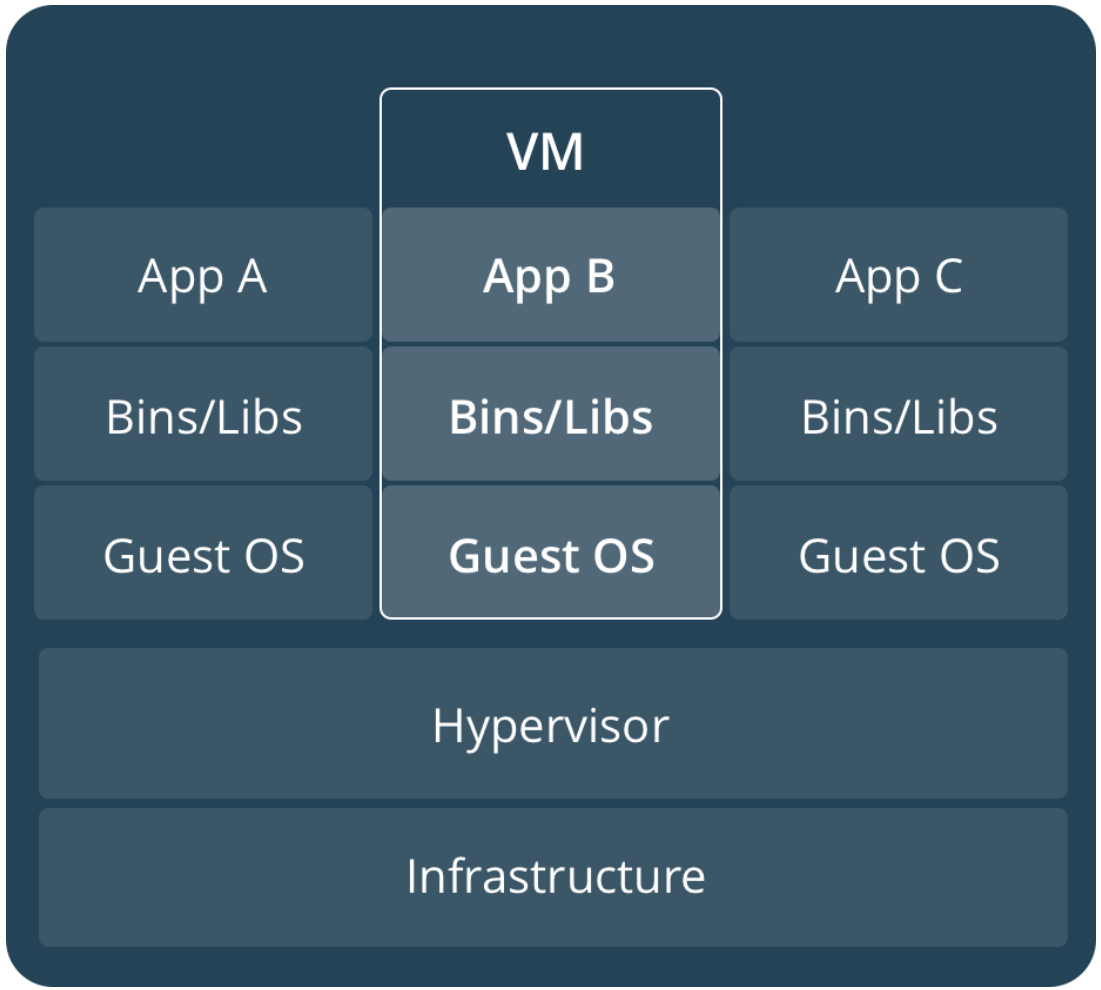
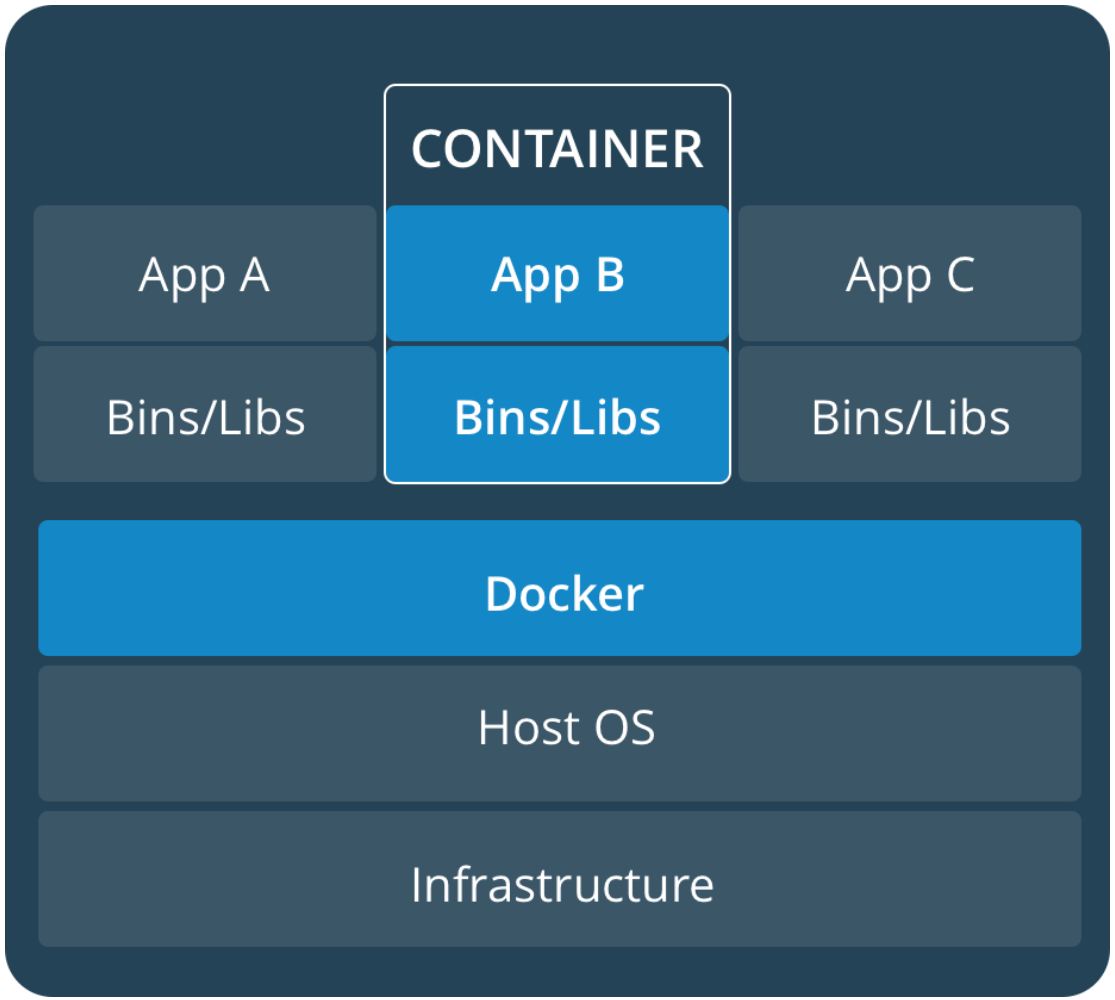


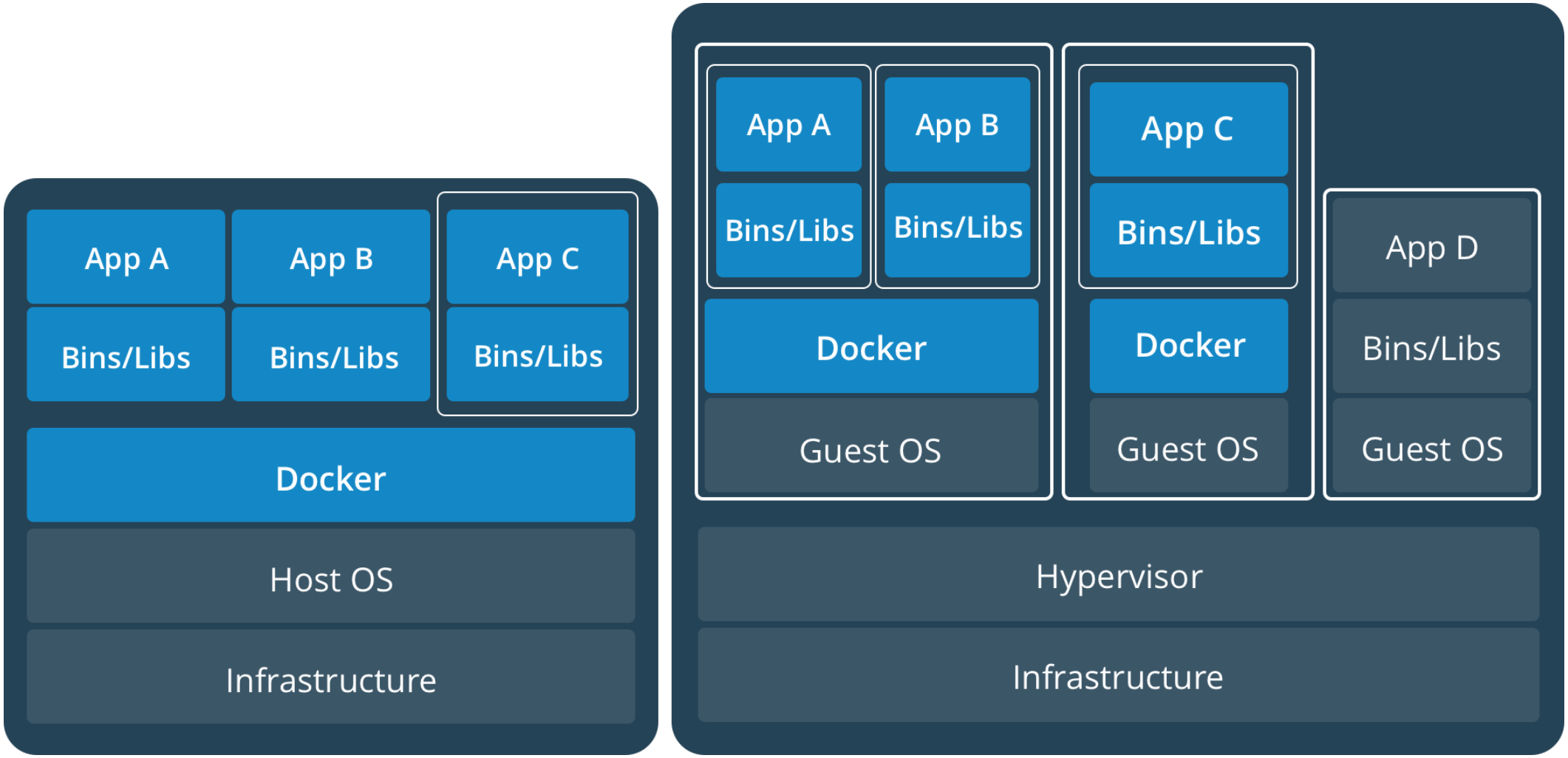
Image Layers and Container Layer



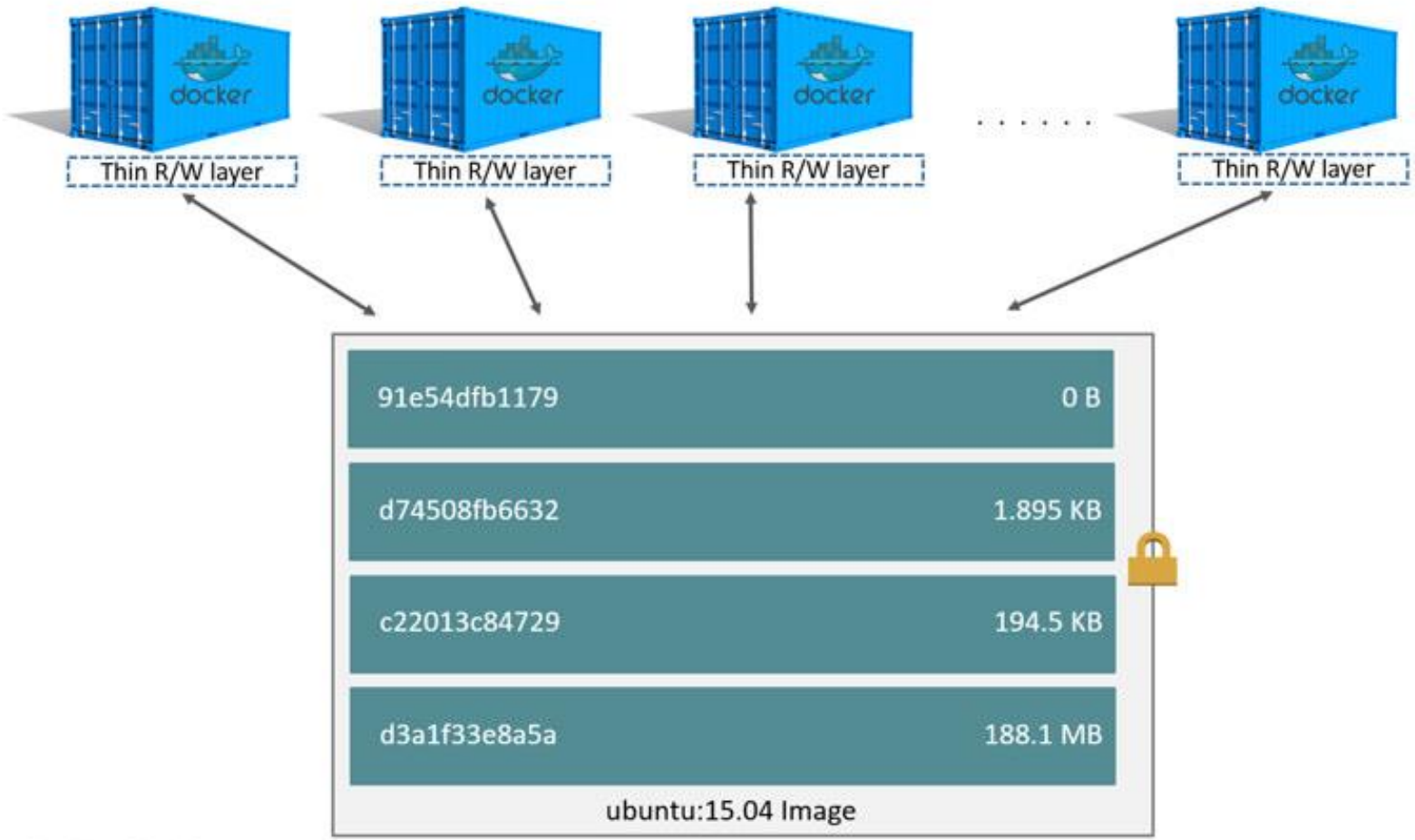

VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure


CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure

Containers and Virtual Machines Together



Sharing Image Layers



AUFS (AnotherUnionFS)

- AUFS storage driver merges all the files for each image layer together and presents them as one **single read-only directory** at the union mount point

- **Union Mount** is a way of combining numerous directories into one directory that looks like it contains the content from all the them

Where layers are stored

/var/lib/docker/aufs

- Image layers and their contents are stored in the **diff** directory.
- How image layers are stacked is in the **layers** directory.
- Running containers are mounted below the **mnt** directory (more explained below about mounts).

- https://docs.docker.com/storage/storagedriver/aufs-driver/#modifying-files-or-directories

Stopping containers gracefully

docker container stop - **SIGTERM** signal to the **PID 1** process inside of the container.

container has 10 seconds to stop, otherwise it will receive **SIGKILL**

Stopping containers immediately

docker container rm < container > -f - sends **SIGKILL** immediately

Restart policy

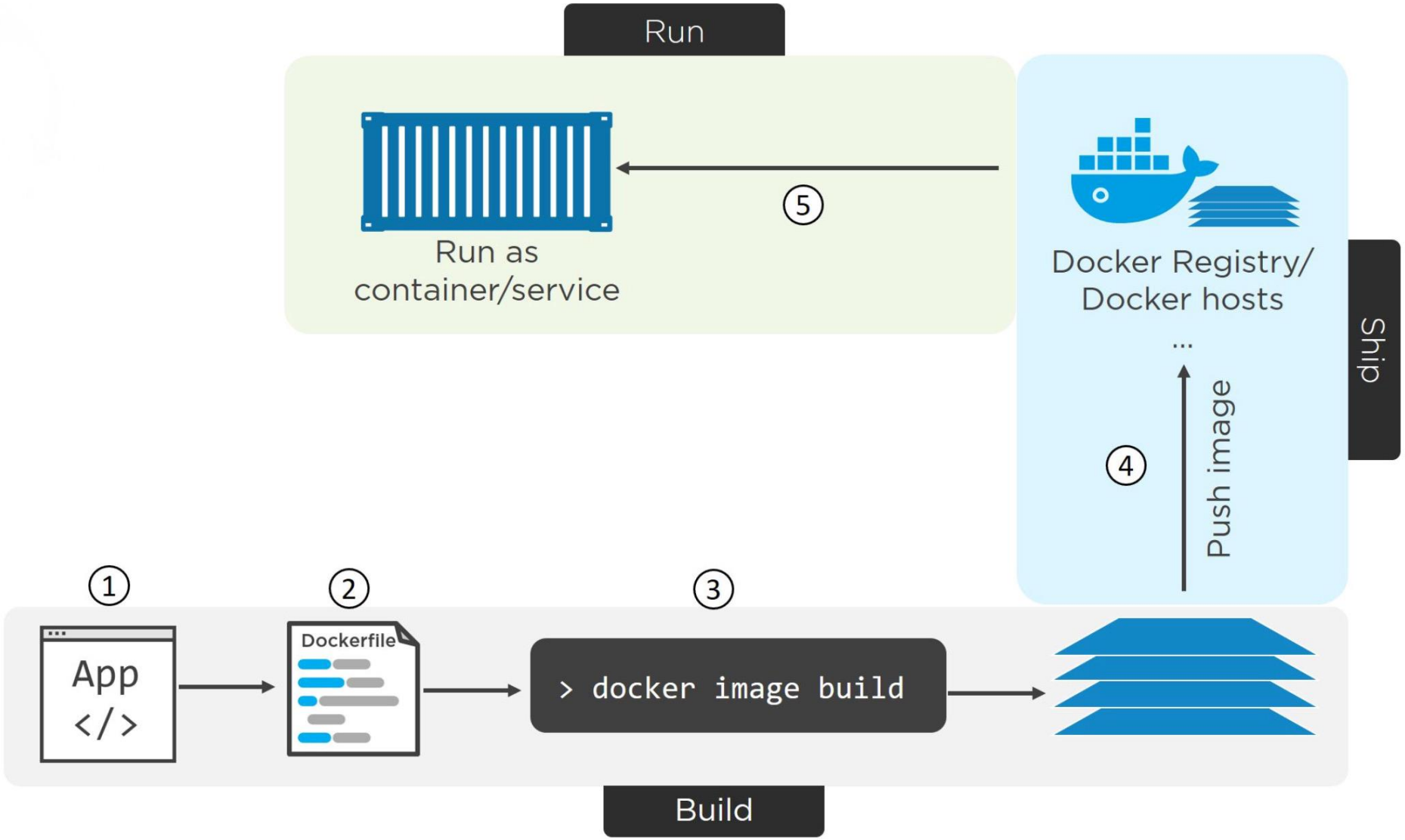
Restart policy - enables Docker to automatically restart them after certain events or failures have occurred

- always
- unless-stopped
- on-failed

docker run --restart <policy>

Containerize an app

Flow


Run
Run as
container/service
5
Docker Registry/
Docker hosts
...
4
Push image
Ship
1
App
</>
2
Dockerfile
3
> docker image build
Build

Dockerfile

- Describes how to pack the application into image
- Can be used as a documentation

Dockerfile example

```
FROM openjdk:8-jdk-alpine
MAINTAINER dkavtur@gmail.com
COPY build/libs/sample-webapp-0.0.1-SNAPSHOT.jar /app/app.jar
WORKDIR /app
RUN echo hello
EXPOSE 8080
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

```
FROM openjdk:8-jdk-alpine
```

- Describes base layer of the image
- All other layer will be added on top of that

FROM

MAINTAINER dkavtur@gmail.com

- Add a **Label** that describes image maintainer information
- Information is added to the image metadata

FROM

```
COPY sample-webapp-0.0.1-SNAPSHOT.jar /app/app.jar
```

- Copies jar file to the image as a new layer
- The file is taken from the **build context** (.)

COPY

FROM

`WORKDIR /app`

- Sets working directory for the rest of instructions
- Information is added to the image metadata

COPY

FROM

EXPOSE 8080

- informs Docker that the container listens on the specified network ports at runtime
- Information is added to the image metadata

COPY

FROM

```
ENTRYPOINT ["java","-Djava.security.egd=file:/de
```

- Sets main application that image container should run (as pid1)
- Information is added to the image metadata

COPY

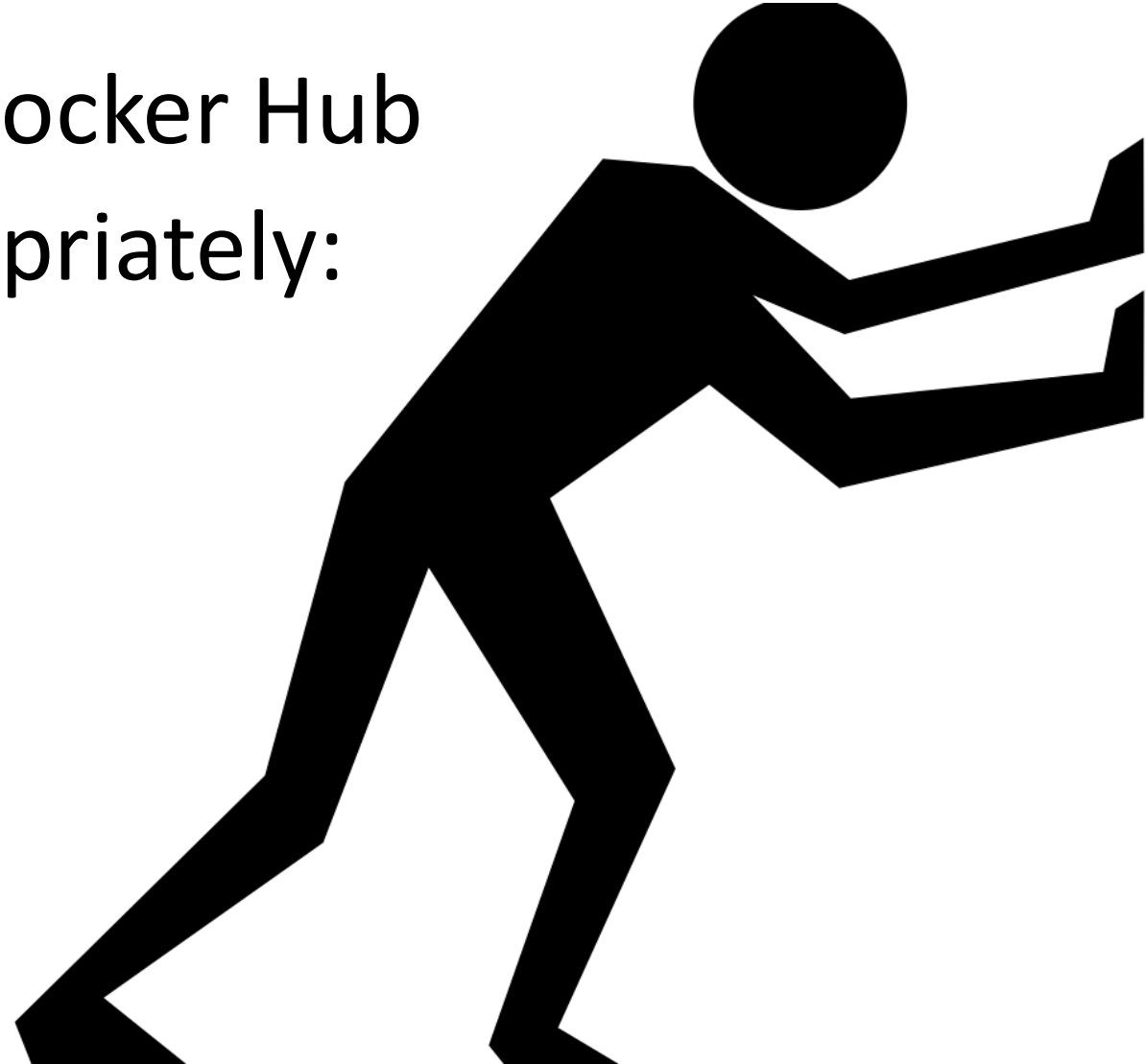
FROM

What is layer and what is not

- Instructions that add **any content** to image are layers
- **docker image history <image_id>** can be used to check

Push image

- Image can be pushed to Docker Hub
- Needs to be tagged appropriately: **account/repository:tag**



Docker compose

What is Docker Compose?

Docker Compose – tool to deploy and manage multi-container applications in **single-engine mode**.

- Written on Python
- Used to be **Fig** created by **Orchard** company that was acquired in 2014 by Docker Inc
- Lets to describe an entire application in single **YML**.

docker-compose.yml

[Compose file version 3 reference]

Deploying an app with Compose

docker-compose up -d

See logs

docker-compose logs <service_name>

Stop containers

docker-compose stop

Down

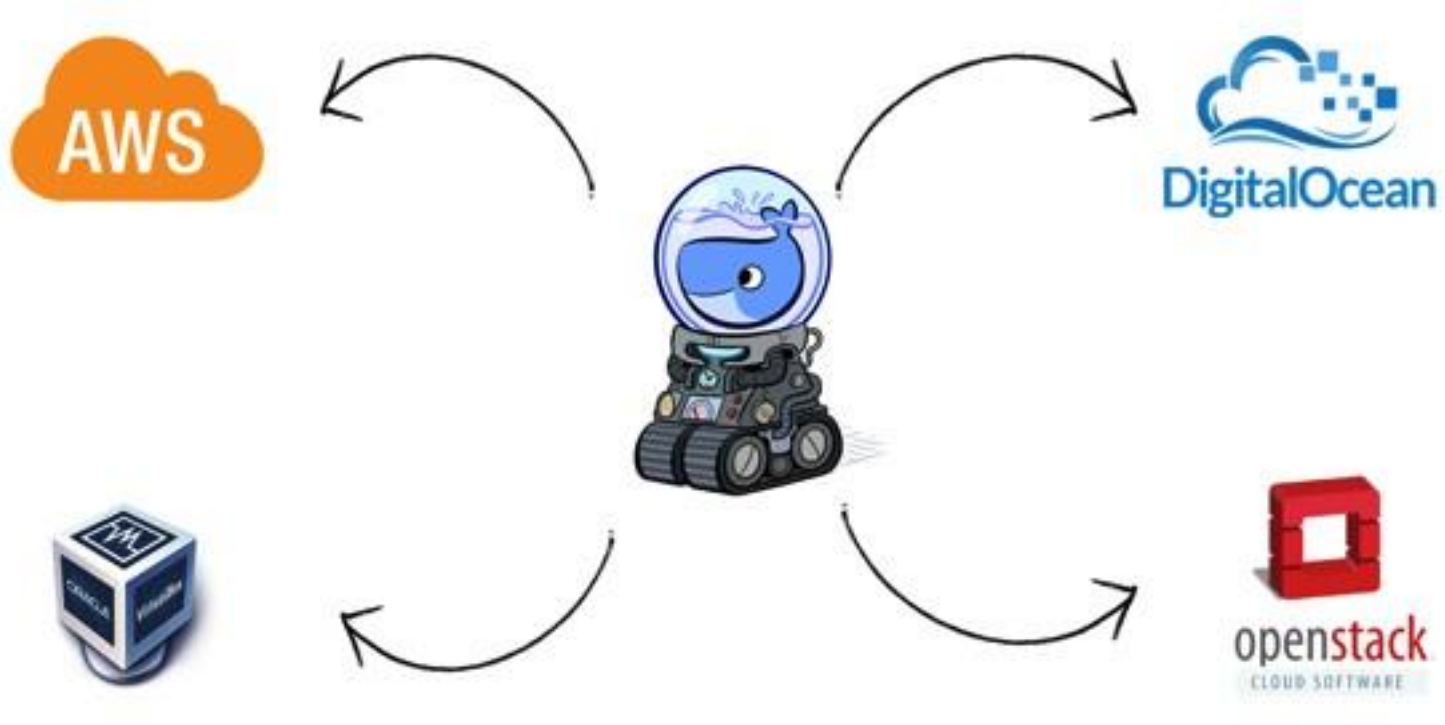
docker-compose down – stops and removes all the containers (not the images) as well as networks.

Does not remove volumes

Docker Machine

What is Docker Machine?

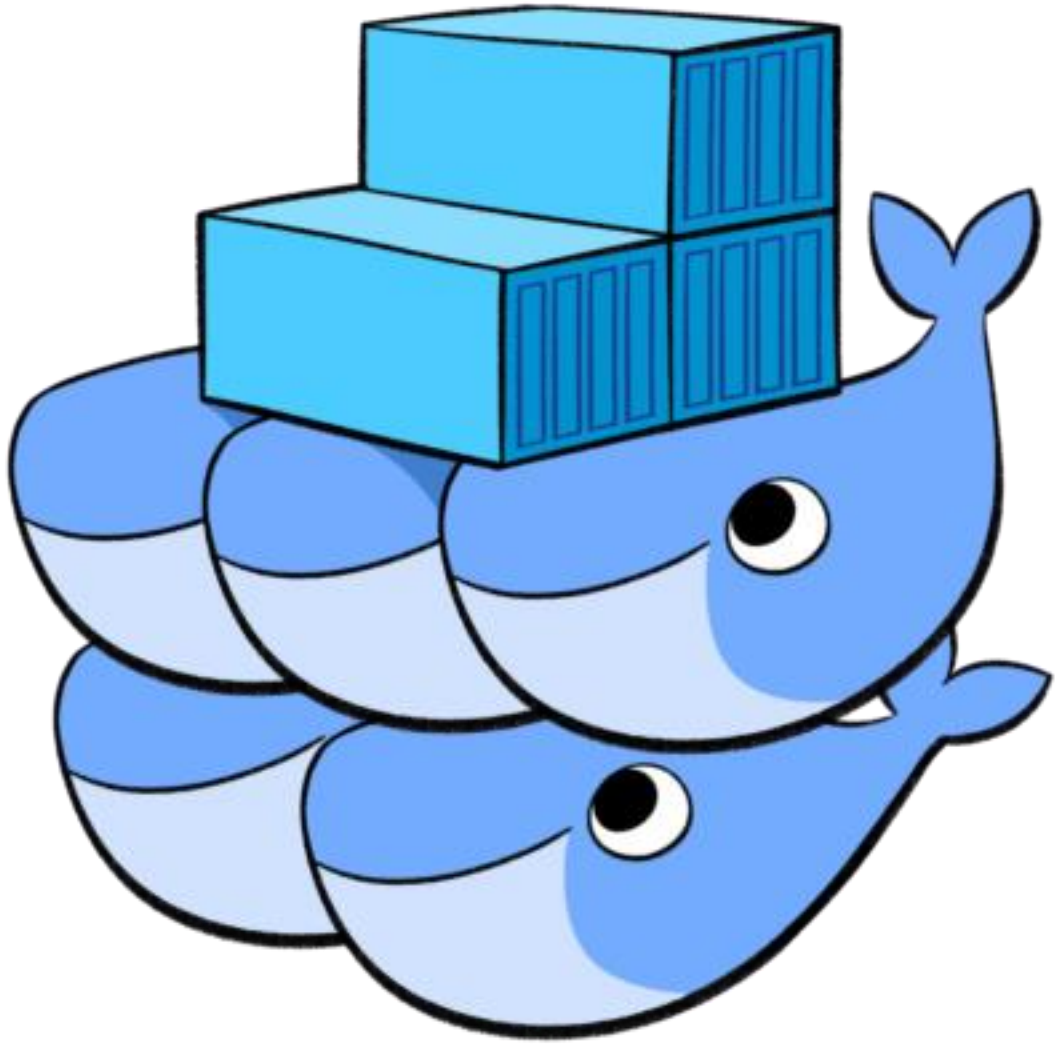
- It is a tool that lets to install **Docker Engine** on **virtual hosts**, and manage the hosts with **docker-machine** commands.



Docker Machine Basic command

- **docker-machine ls**
- **docker-machine create<name>**
- **docker-machine env <name>**
- **docker-machine kill <name>**

Docker Swarm

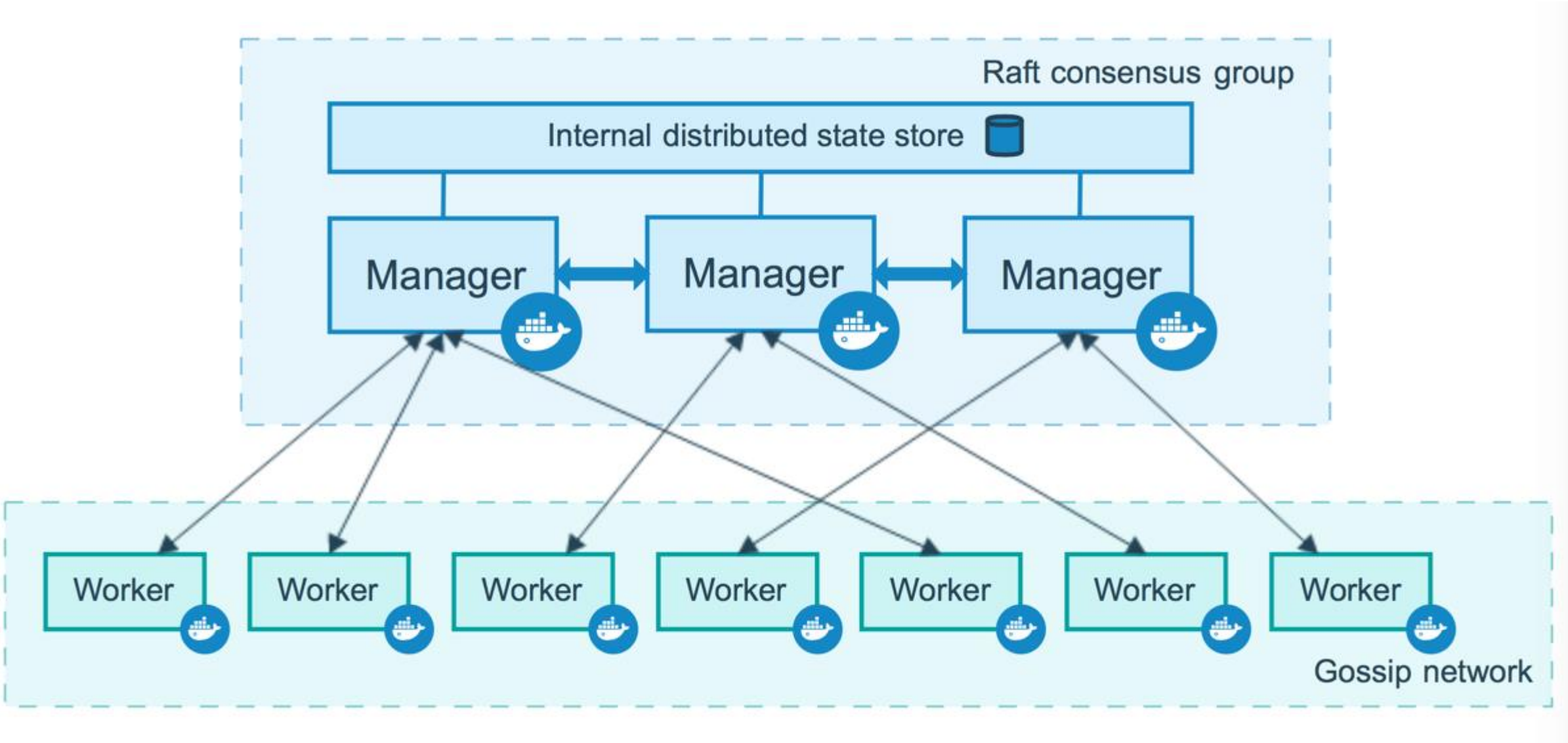


Docker Swarm

- A **swarm is a group of machines** that are running Docker and joined into a cluster.
- Swarm consists of one or more Docker nodes. These can be physical servers, VMs, or cloud instances.
- The only requirement is that all nodes can communicate over reliable networks.

Swarm Nodes

- Nodes are **managers** and **workers**
- Managers look after the control plane of the cluster: of the cluster and dispatching tasks to workers
- Workers accept tasks from managers and execute them
- Managers also operate as a workers



Configuration and state of the swarm is in a **distributed etcd database** located on all managers

Create a new swarm

docker swarm init

- switch node into swarm mode
- create a new swarm
- make the node the first manager of the swarm

advertise-addr

**docker swarm init **

--advertise-addr 192.168.99.100:2377

It is the address other nodes in the Docker swarm use to connect into your node.

The default port that swarm mode operates on is **2377**

List of nodes in Swarm

docker node ls

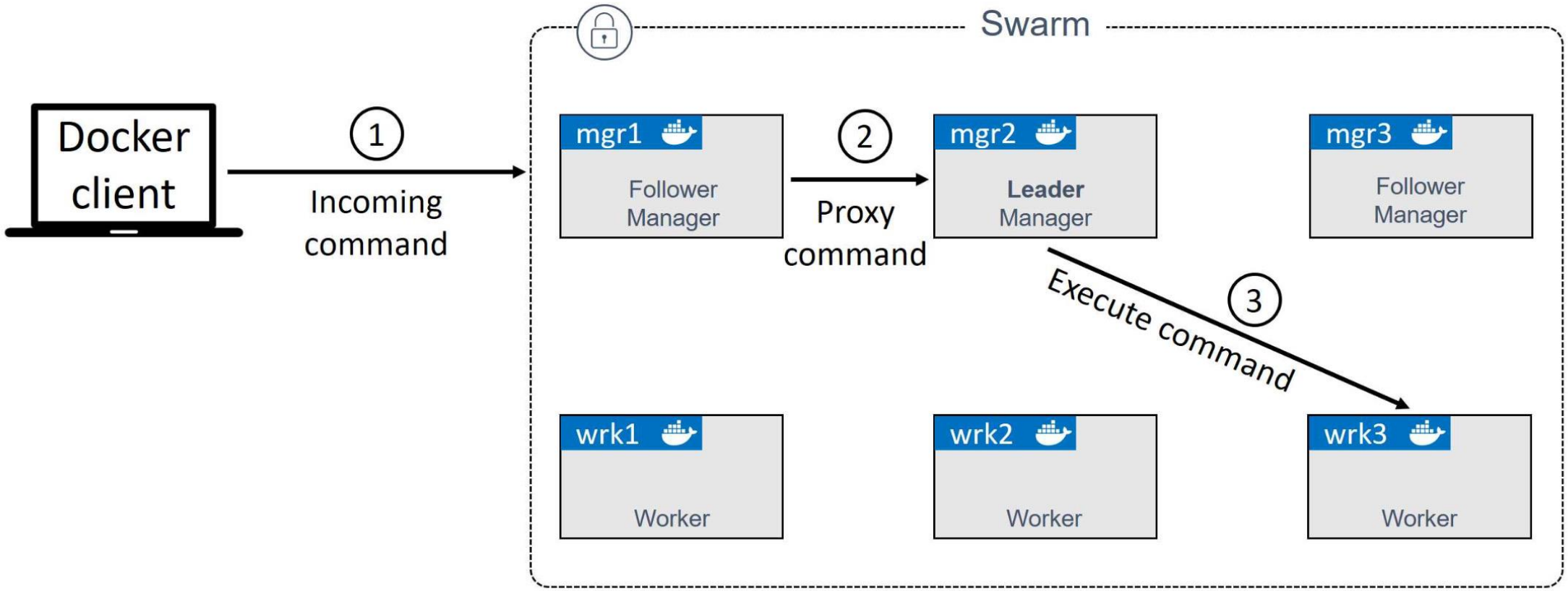
***Note**: can be run only from managers*

How to join swarm

- **docker swarm join-token manager**
- **docker swarm join-token worker**

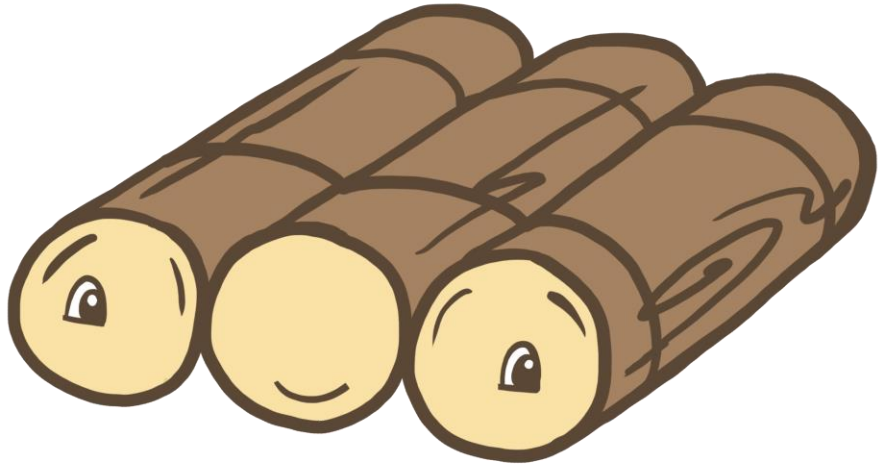
Swarm manager high availability (HA)

- Swarm implements a form of active-passive multi-manager HA
- One manager is active (leader)
- All other managers are followers
- There should be **odd number** of managers (3, 5 or 7)


Swarm
Docker client
1
Incoming command
mgr1
Follower Manager
2
Proxy command
mgr2
Leader Manager
mgr3
Follower Manager
3
Execute command
wrk1
Worker
wrk2
Worker
wrk3
Worker

Raft consensus algorithm

- https://raft.github.io
- http://thesecretlivesofdata.com/raft/



Swarm services

- Allow to specify run attributes of regular container
- But in addition allow to specify desired state
- Docker swarm takes care of managing and deploying it (orchestration)

Create new service

- **docker service create** command at manager node
- **--replicas <num_of replicas>**

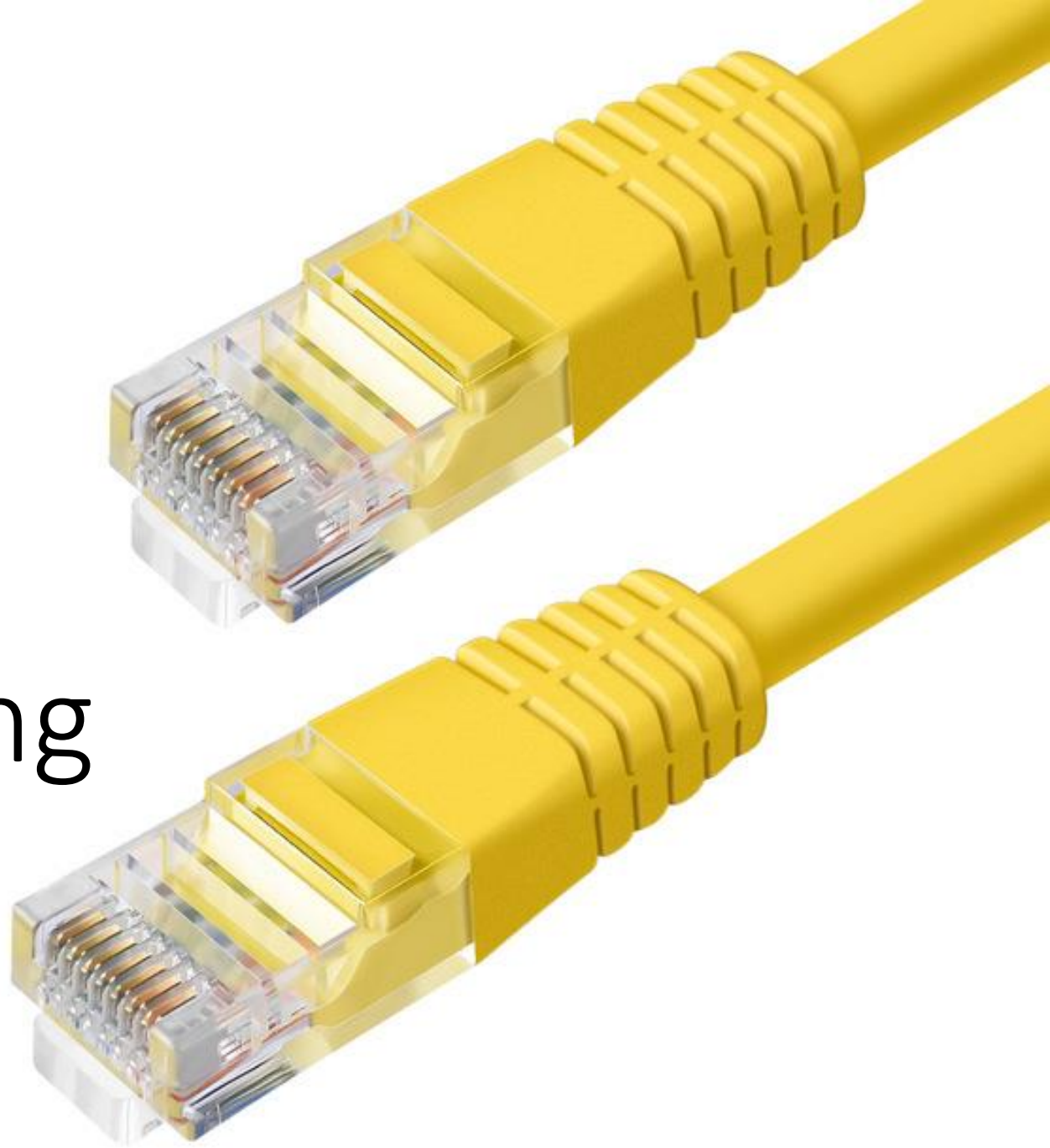
Scale service

docker service scale <service_name>=<num of replicas>

Rolling update

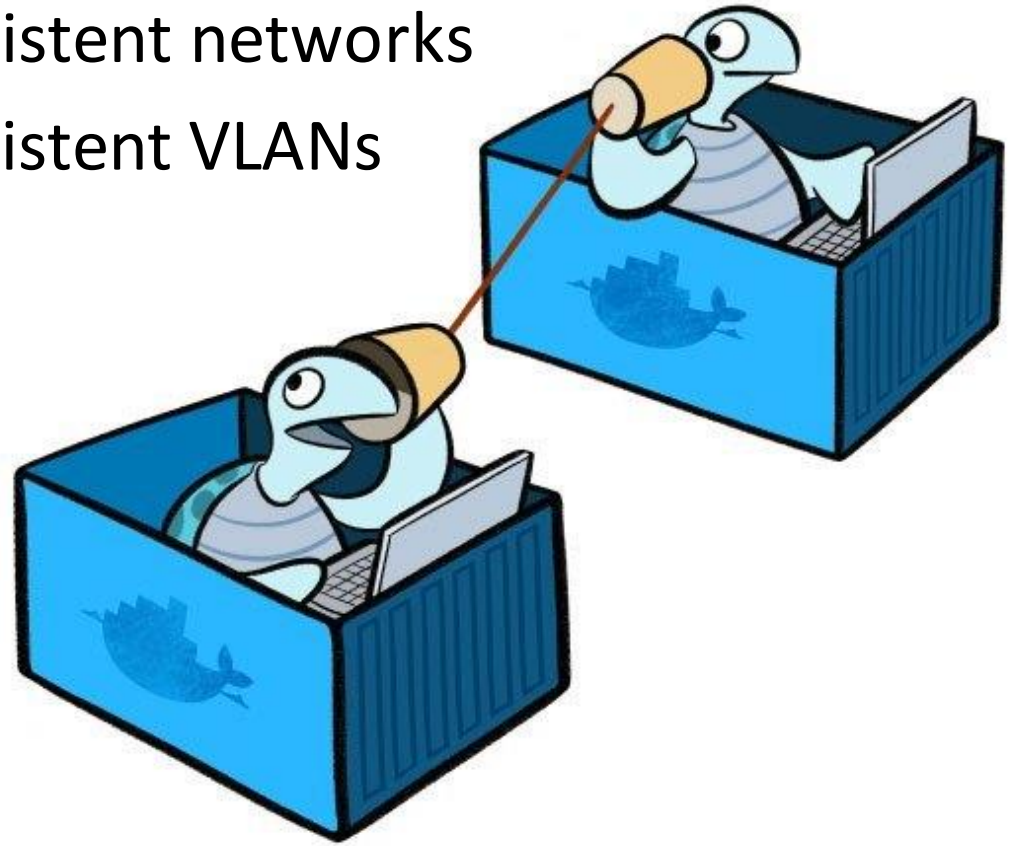
docker service update --update-delay=20s --update-parallelism=2 –image=<new image id> <service_name>

Docker Networking



Docker Networking

- Containers needs to communicate between each other
- Sometimes containers need to connect existent networks
- Sometimes containers need to connect existent VLANs



Networking Components

- The Container Network Model (CNM) – specification
- libnetwork – implementation of CNM
- **network drivers** – pluggable interfaces to provide special capabilities

Build-in drivers

- **Bridge**
- **Overlay**
- **Macvlan**

Bridge network (single host)

- Single-host
- Docker has default **bridge** network where attaches container by default

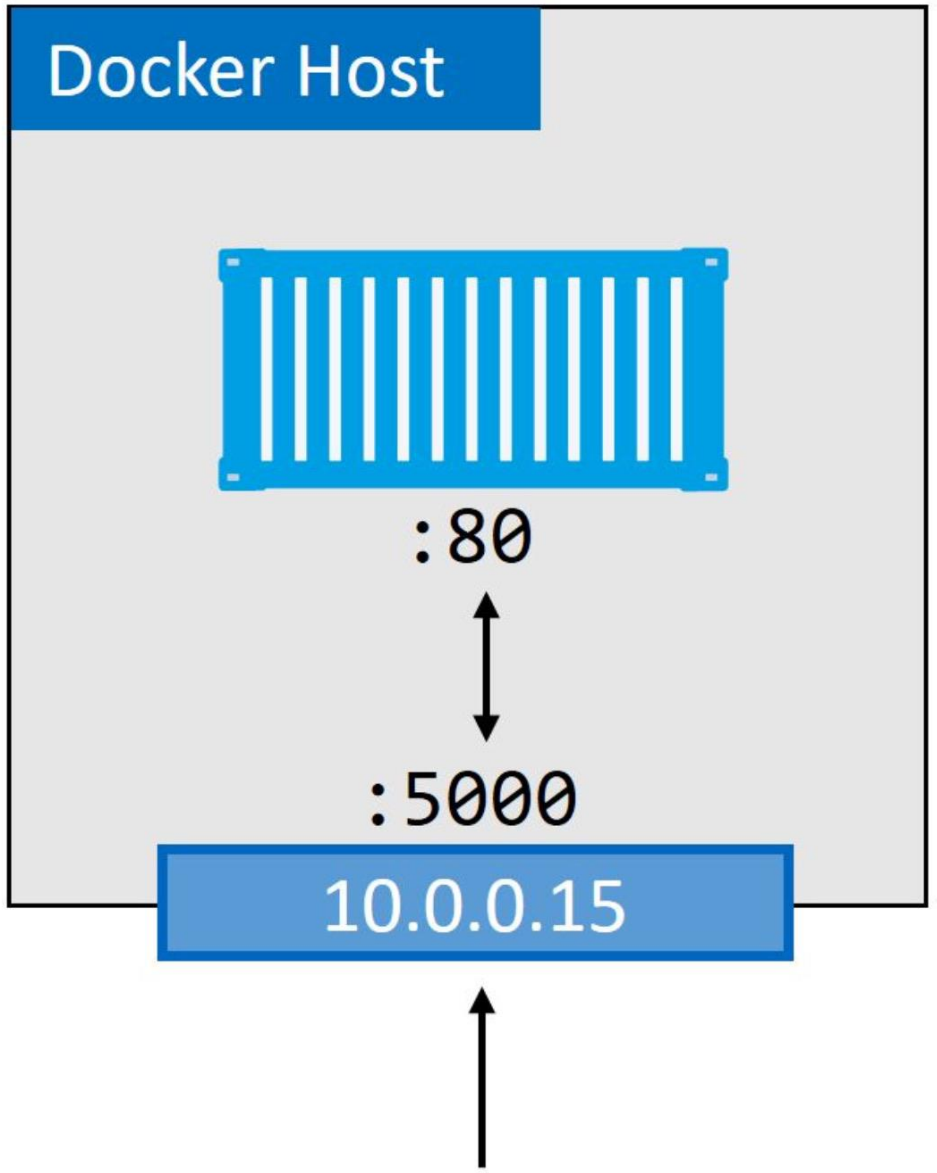
Custom bridge networks

- **docker network create <network_name>**
- Supports name resolution via Docker DNS service
- Containers can be attached and detached from user-defined networks on the fly
- **docker container run --network network_name**
- **docker network connect network_name container_name**

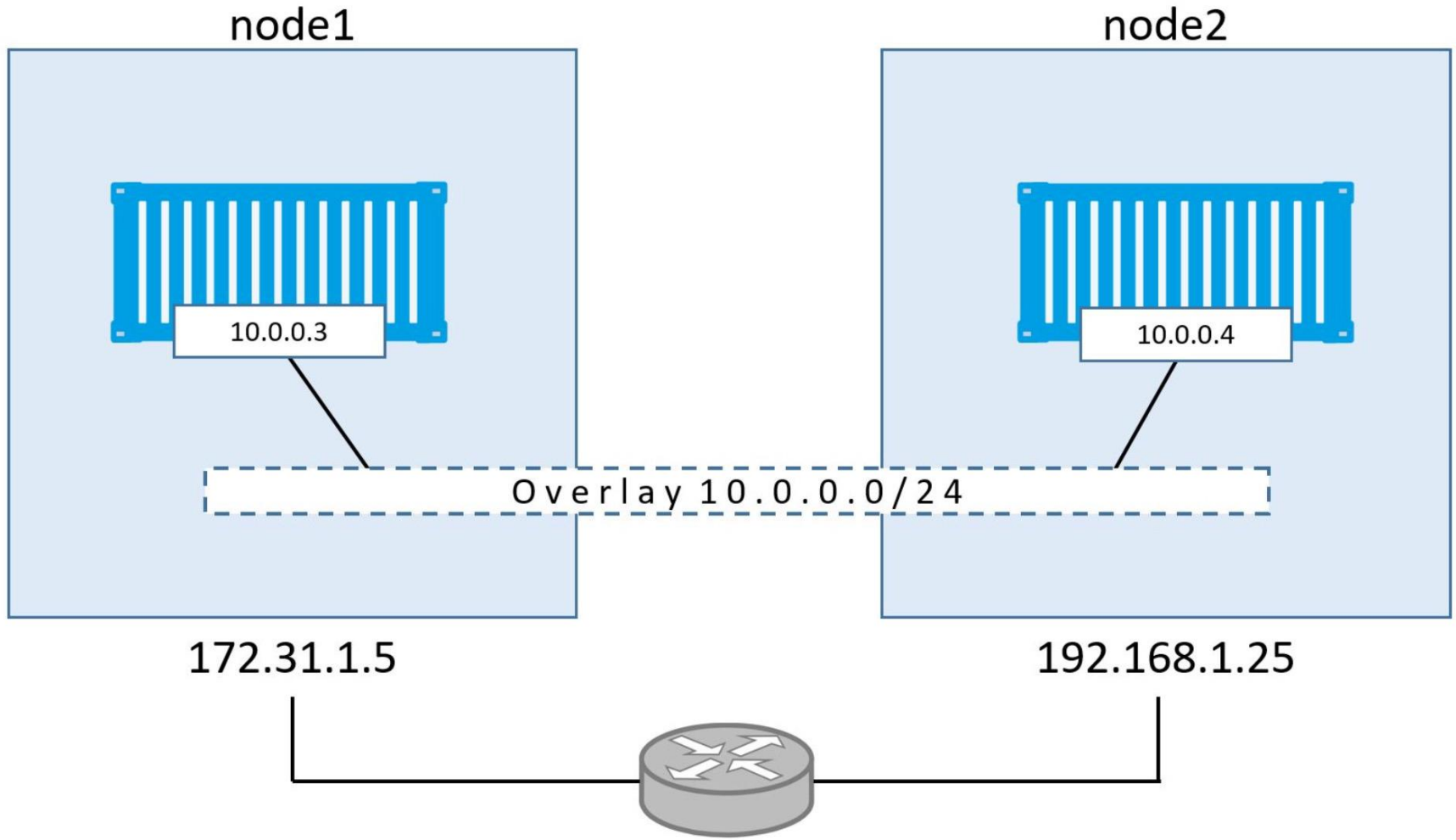
Overlay network

- allows you to create a flat, secure, layer-2 network, spanning multiple hosts
- containers connect to this and can communicate directly

Publishing port



Traffic to the 10.0.0.15:5000 will be redirected to the container :80


node1
node2
10.0.0.3
10.0.0.4
Overlay 10.0.0.0/24
172.31.1.5
192.168.1.25

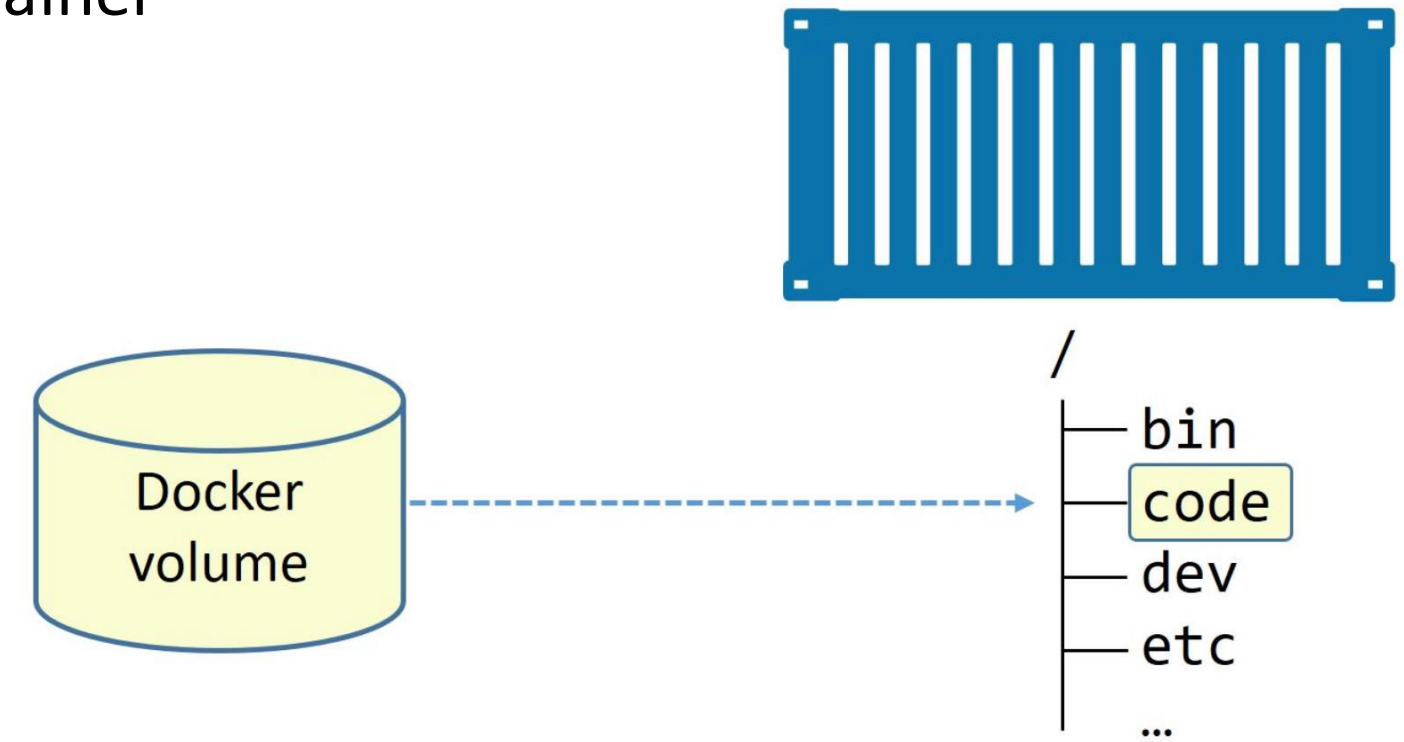
Volumes

Container writable layer

- Every container gets its own **non-persistent** storage (writable layer)
- It’s automatically created, alongside the container, and it’s tied to the lifecycle of the container
- It should not be used to persist data

How to persist data?

- Create volume
- Mount volume to the container



Create volume

docker volume create

- By default create volume with local driver (available only in the node where was created)

container and mount volume

docker container run -v vol1:/dir1 …

docker container run

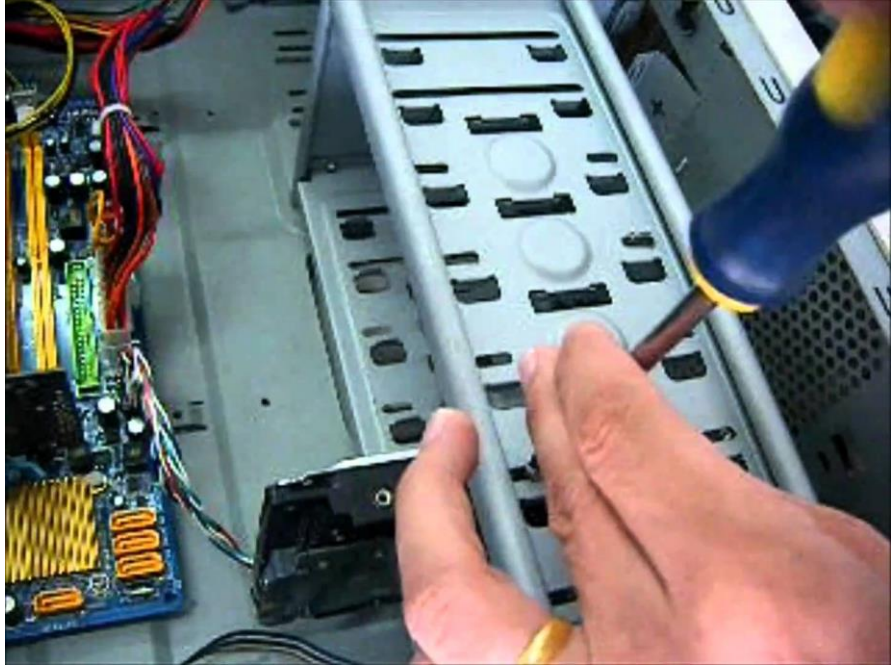
--mount source=vol1,target=/dir1 …

- If you specify an existing volume, Docker will use the existing volume
- If you specify a volume that does not exist, Docker will create it for you

container and mount volume

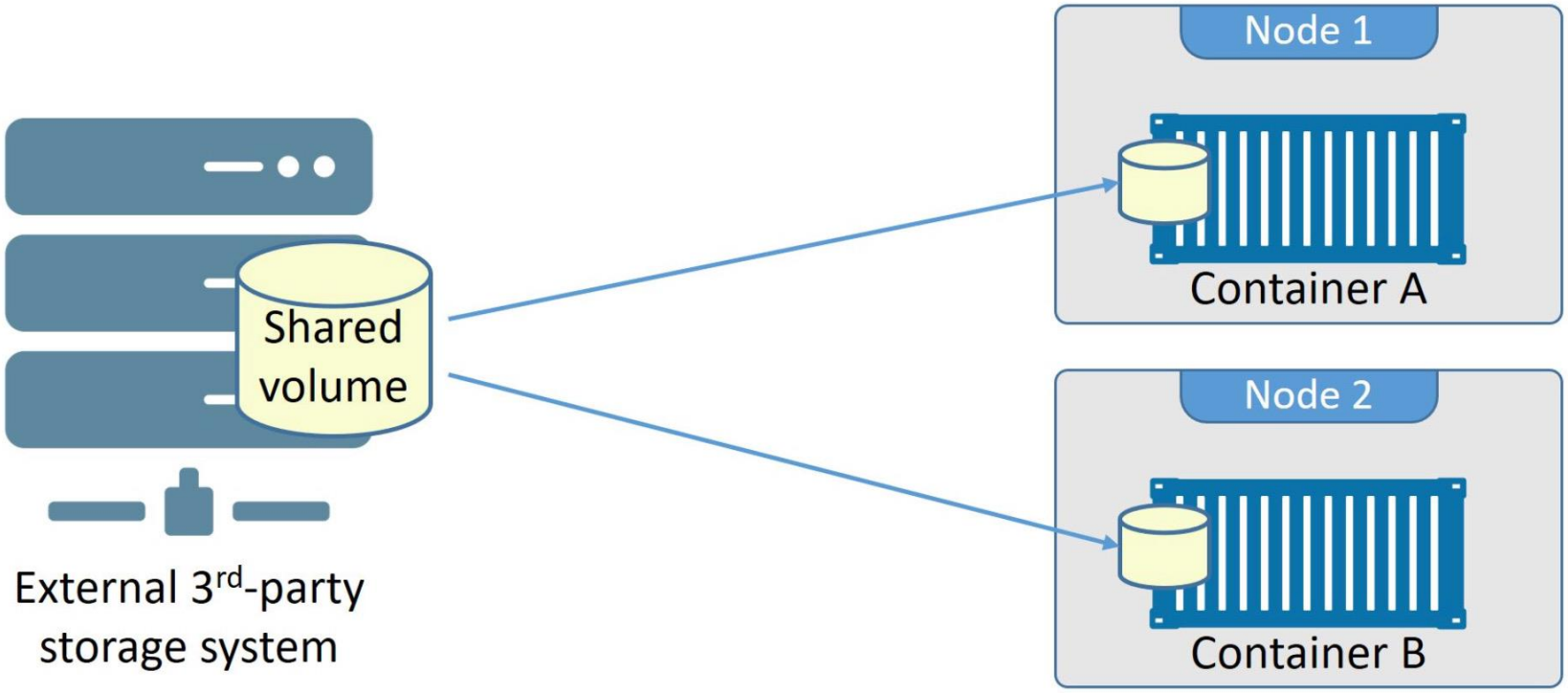
docker service run

--mount source=vol1,target=/dir1 …



- Worker where container is deployed creates volume

Cluster storage

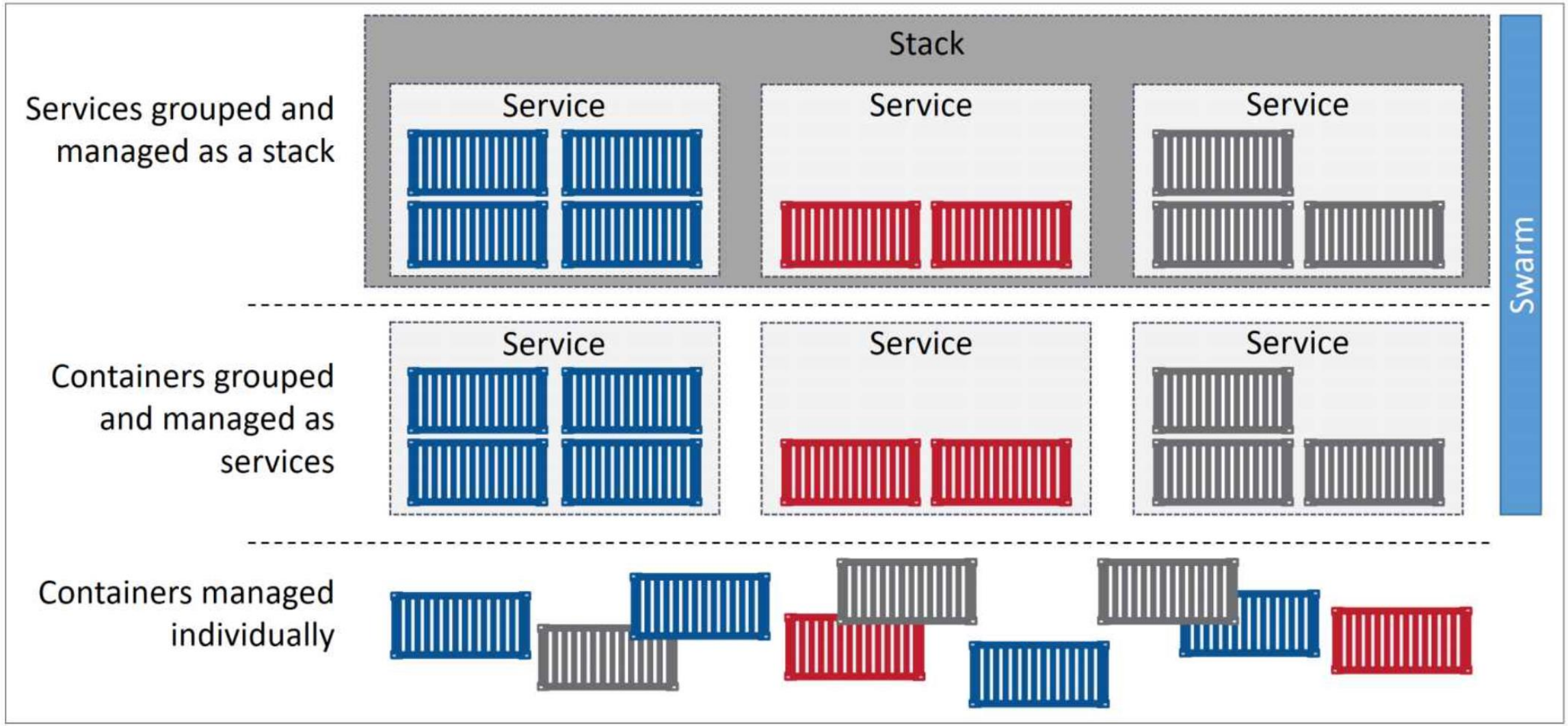


Docker Stacks

Docker stack

let to define complex **multi-service apps** in a single declarative file and manage its lifecycle

- deploy
- health checks
- scaling
- updates


Services grouped and managed as a stack
Stack
Service
Service
Service
Swarm
Containers grouped and managed as services
Service
Service
Service
Containers managed individually

Stack file is docker compose

- Ignores **build** section
- Supports **service.deploy** section
- Supports **secrets** section

Stack deploy section

```
deploy:
  replicas: 6
  update_config:
    parallelism: 2
    delay: 10s
  restart_policy:
    condition: on-failure
```

Docker stack constraints

- node.id == o2p4kw2uuw2a
- node.hostname == w1
- node.role != manager
- engine.labels.operatingsystem == “ubuntu 16.04” node.labels.zone == prod1
- node.labels.custom-label == label_value

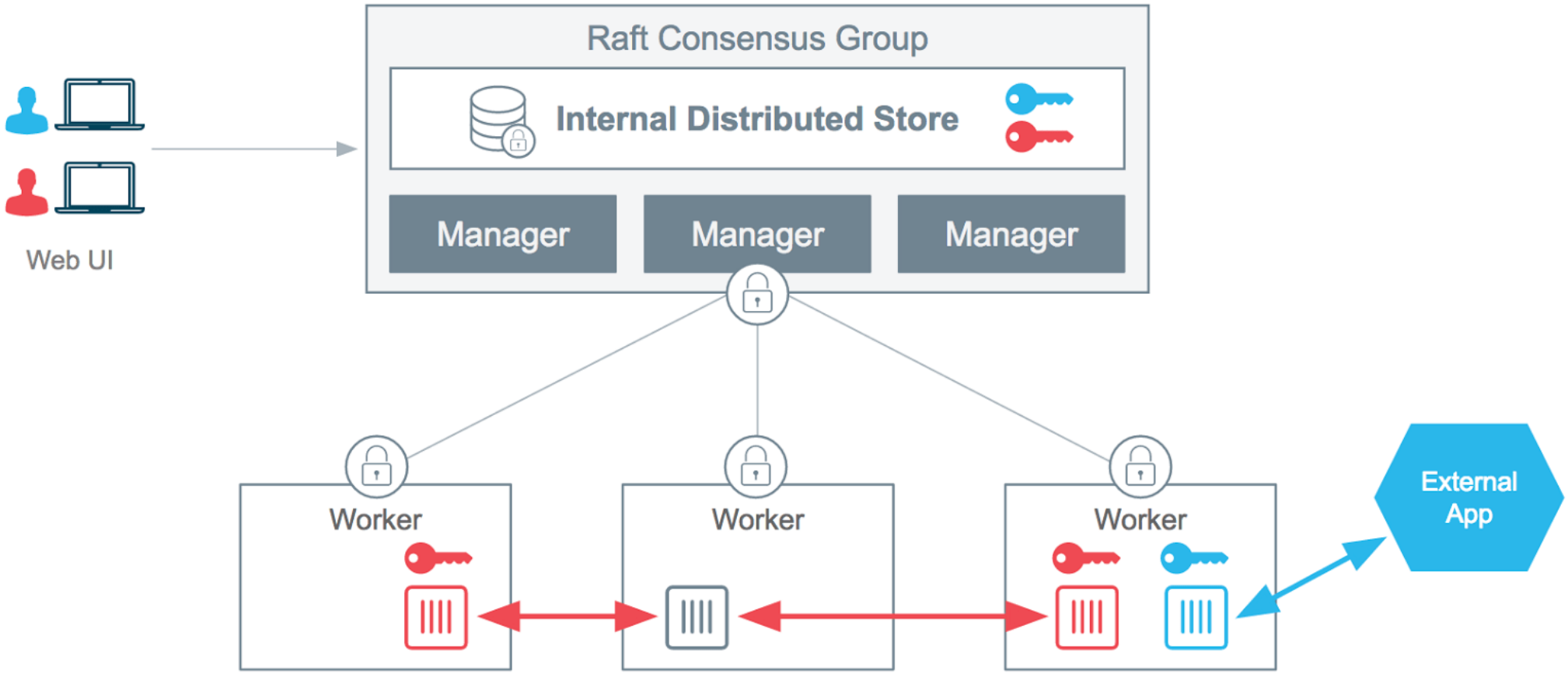
```
deploy:
  placement:
    constraints:
      - 'node.role == worker'
```

Labeling node

- **docker node update --label-add label1=valu1 w1**

Docker secrets

- Secrets get mounted into service replicas as a regular file.
- Linux mounts **/run/secrets** as an in-memory filesystem


Raft Consensus Group
Internal Distributed Store
Manager
Manager
Manager
Web UI
Worker
Worker
Worker
External App

Create secret

- **docker secret create secret_name file_with_secret**
- **echo “secret value” | docker secret create -**

Define secret in stack file

- **secrets:**
  pass2:
  external: true

- **service1.secrets:**
  - **source:** pass2
  target: /path

Docker in AWS (Cloud Formation)

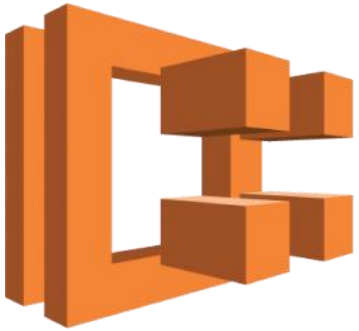
- https://docs.docker.com/docker-for-aws/why/

Docker “alternatives”

kubernetes

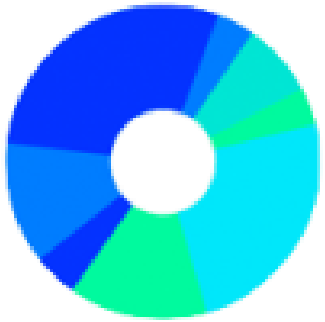
rkt

CoreOS



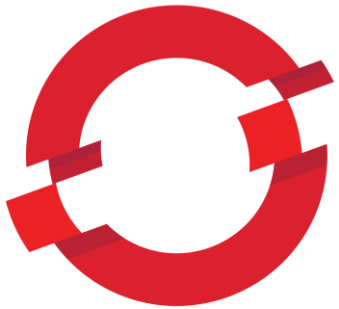
Amazon ECS

MESOS



MARATHON

DC/OS



OPENSHIFT

- Container orchestration framework
- Container agnostic
- Initially was developed by Google
- Was donated to [Cloud Native Comuting Foundation]