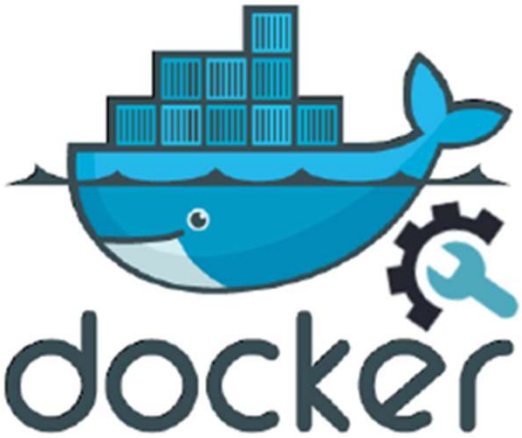
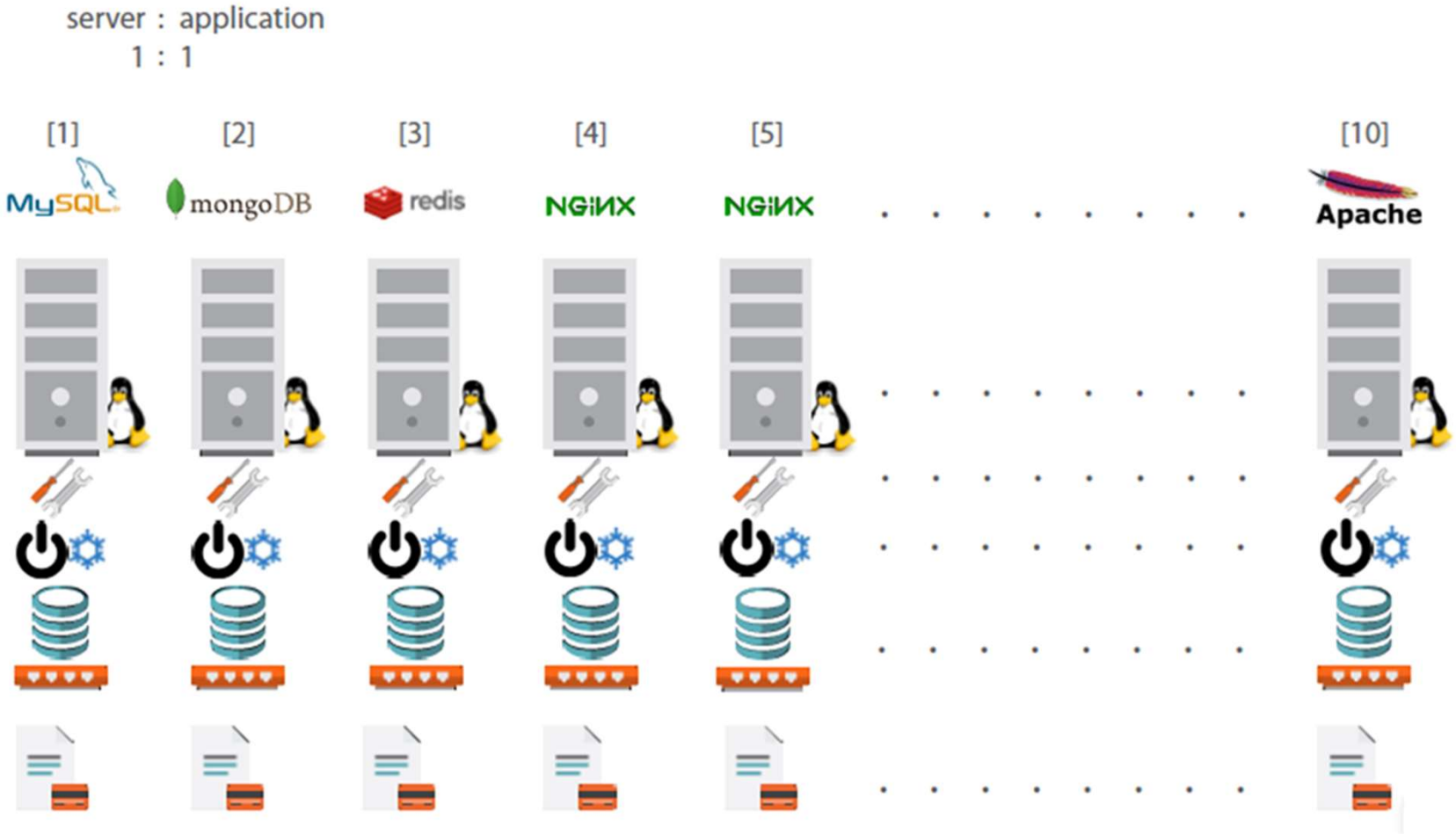
Docker – Day 1

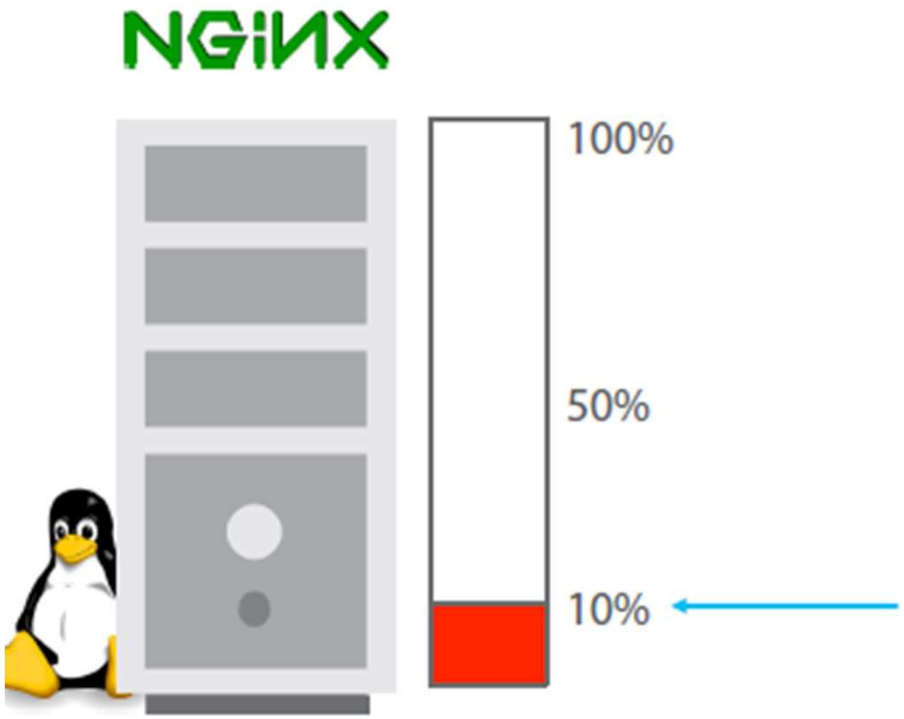
Docker



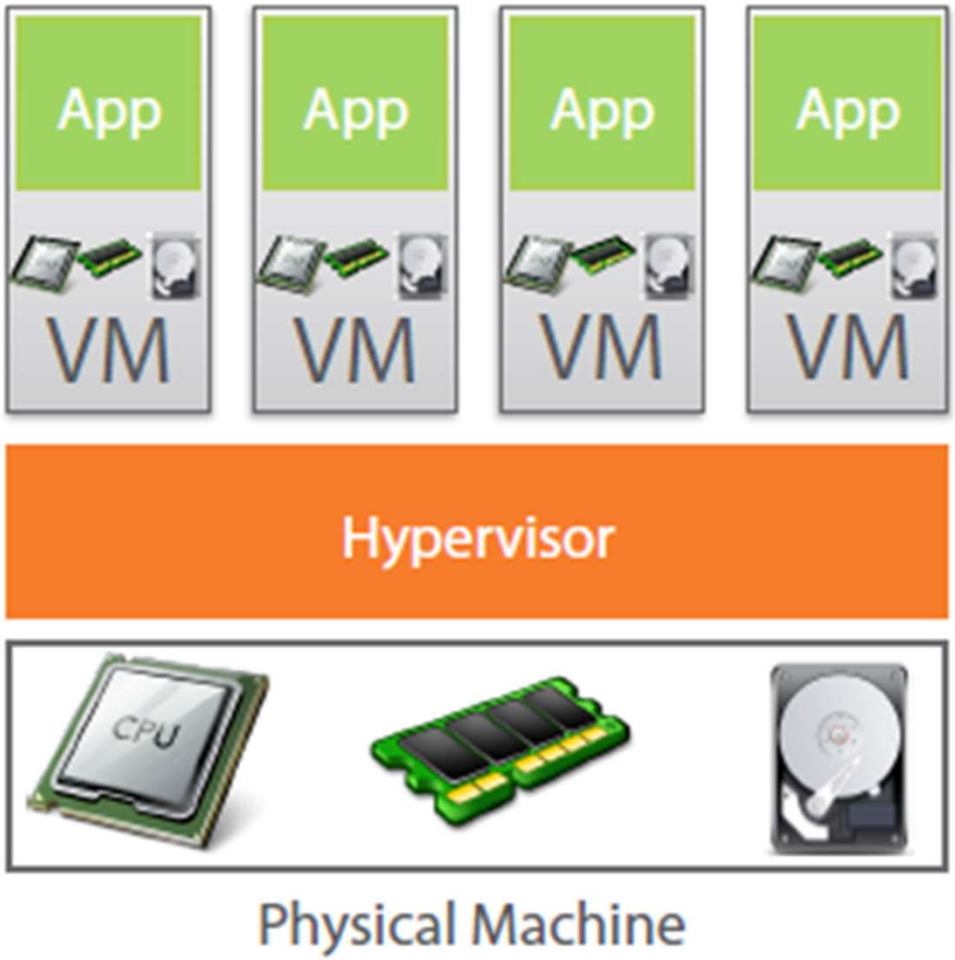
Traditional Deployment Architecture



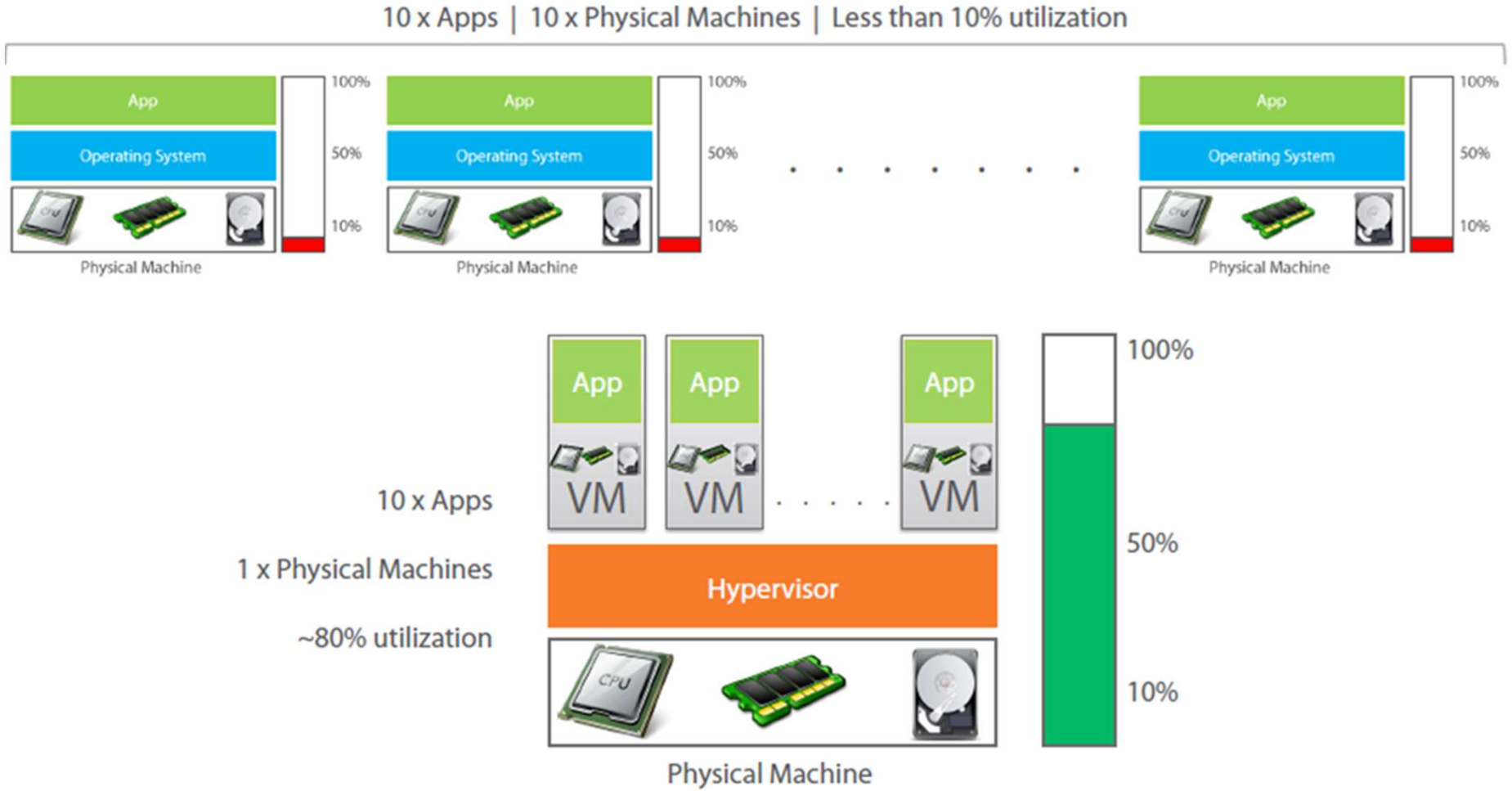
Less Utilization in Traditional Architecture



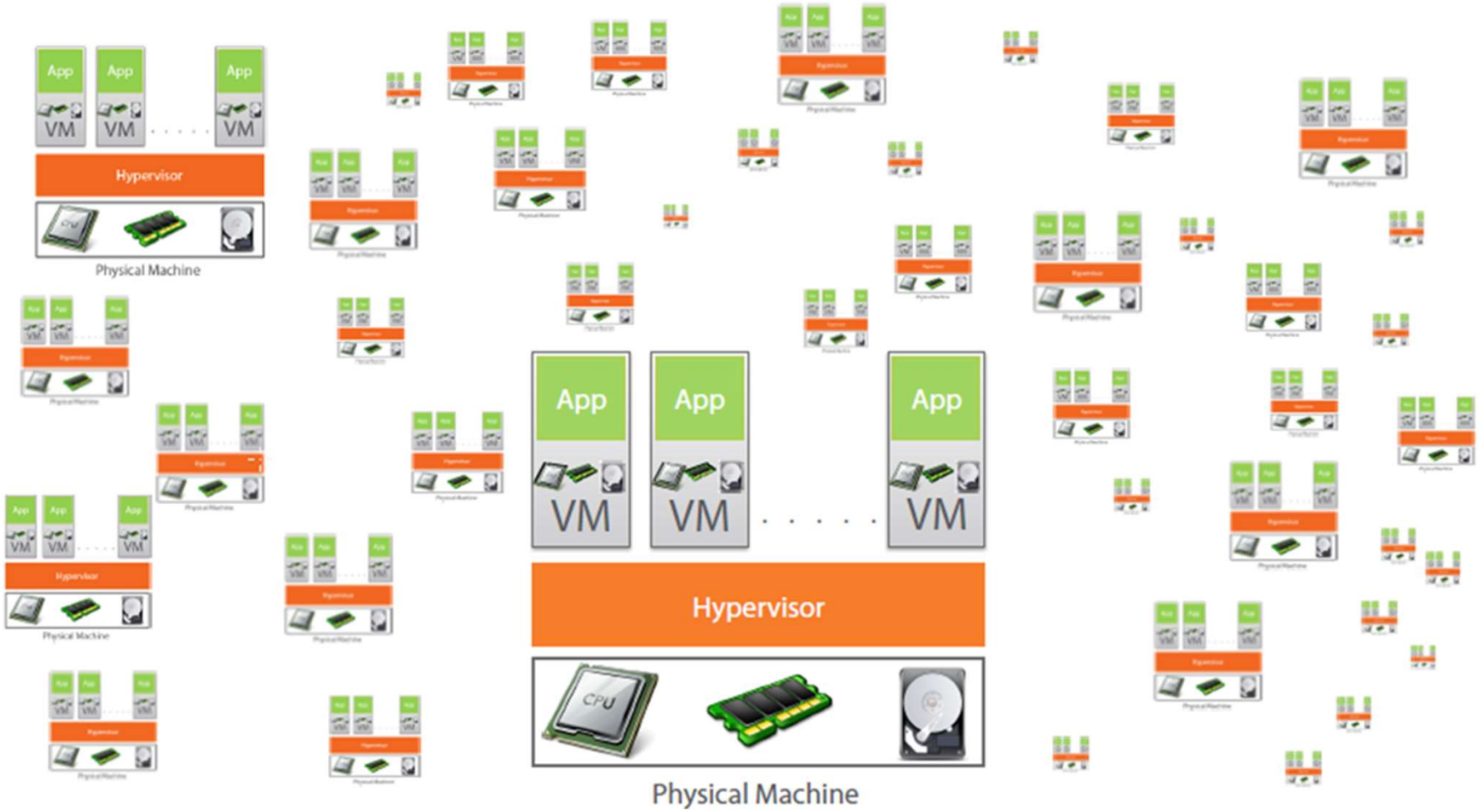
Virtual Machine to the Rescue



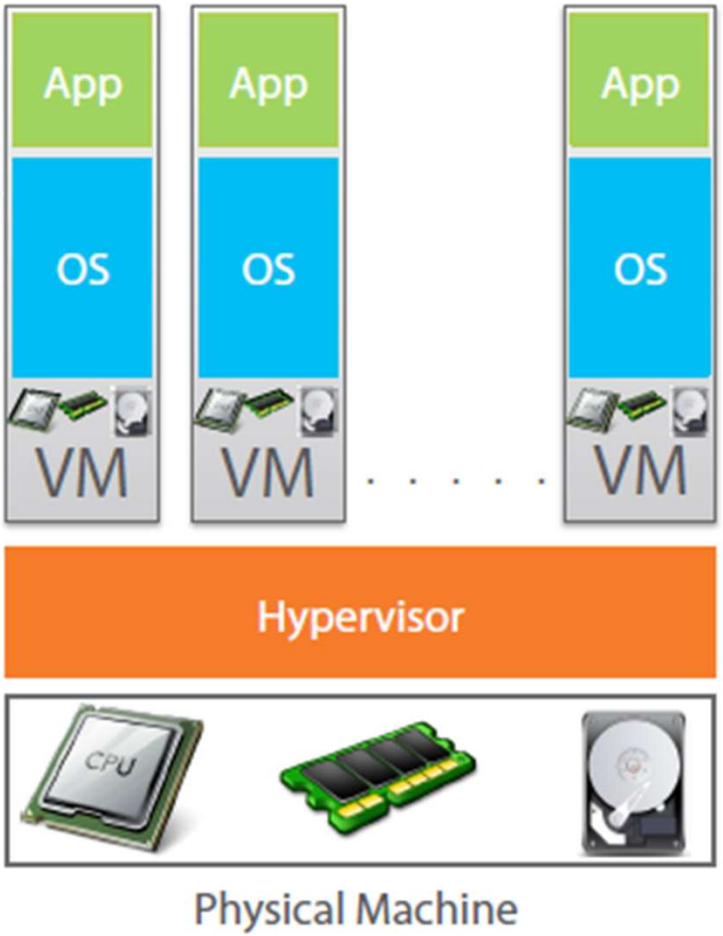
Virtual Machine provides better utilization



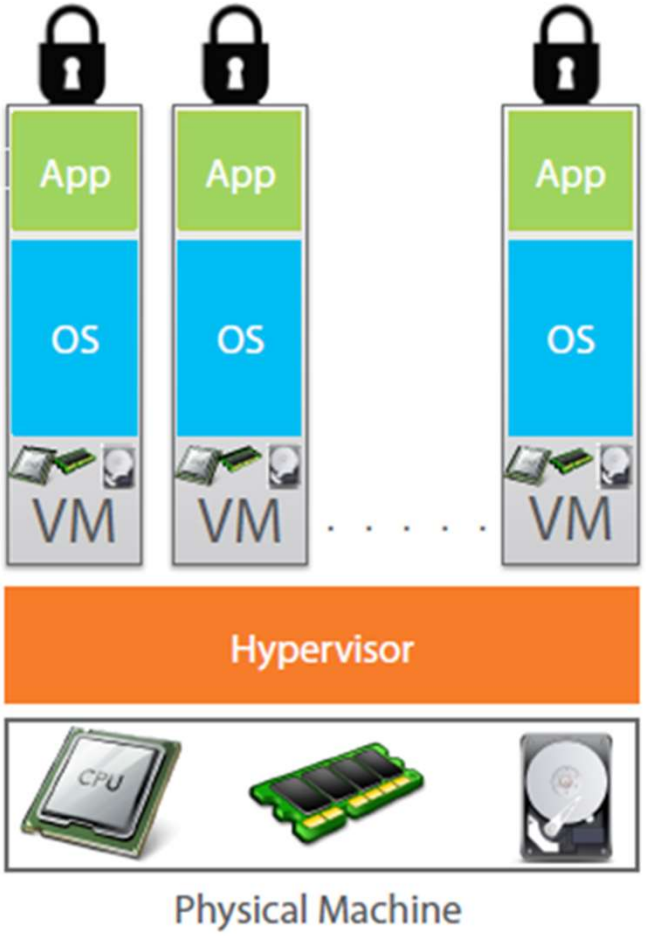
But Virtual Machine increases Licensing Cost



Each VM needs a separate OS

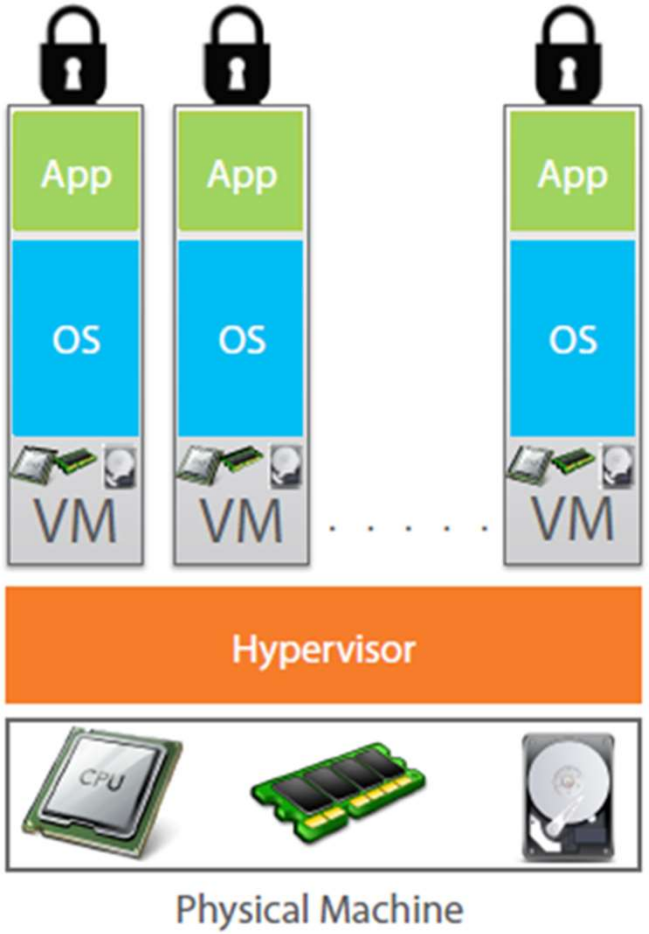


More OSes doesn’t increase Business Value



> OS != Business Value

OS takes most of the Resources

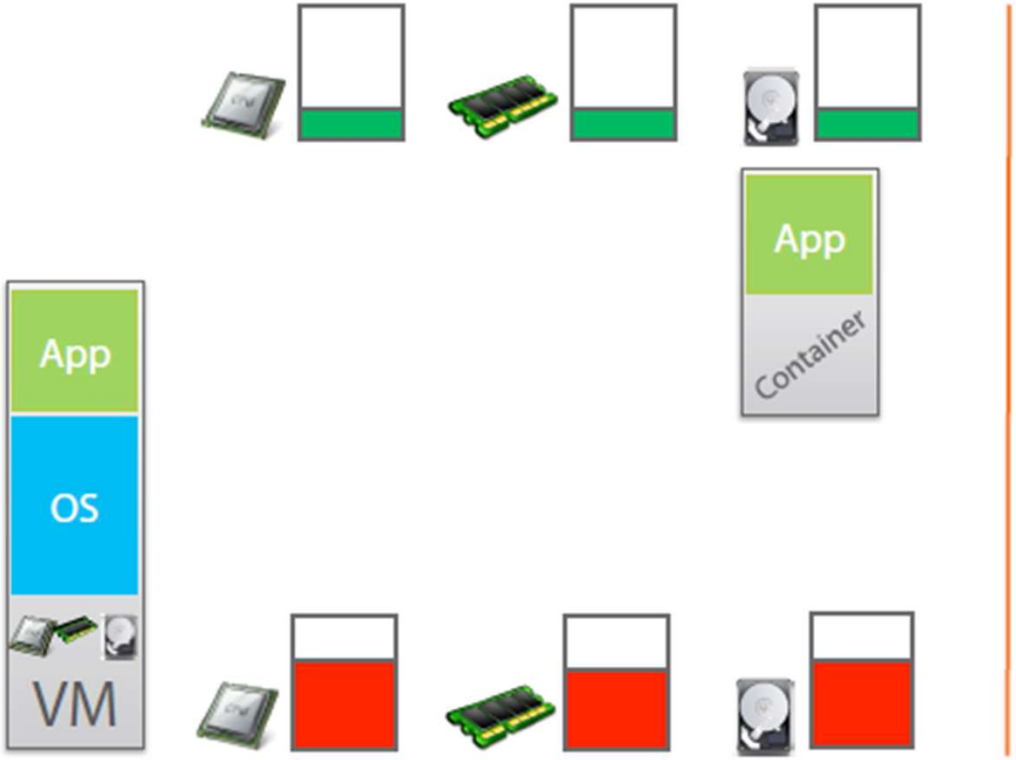


Why use separate OS for each App?

Containerization

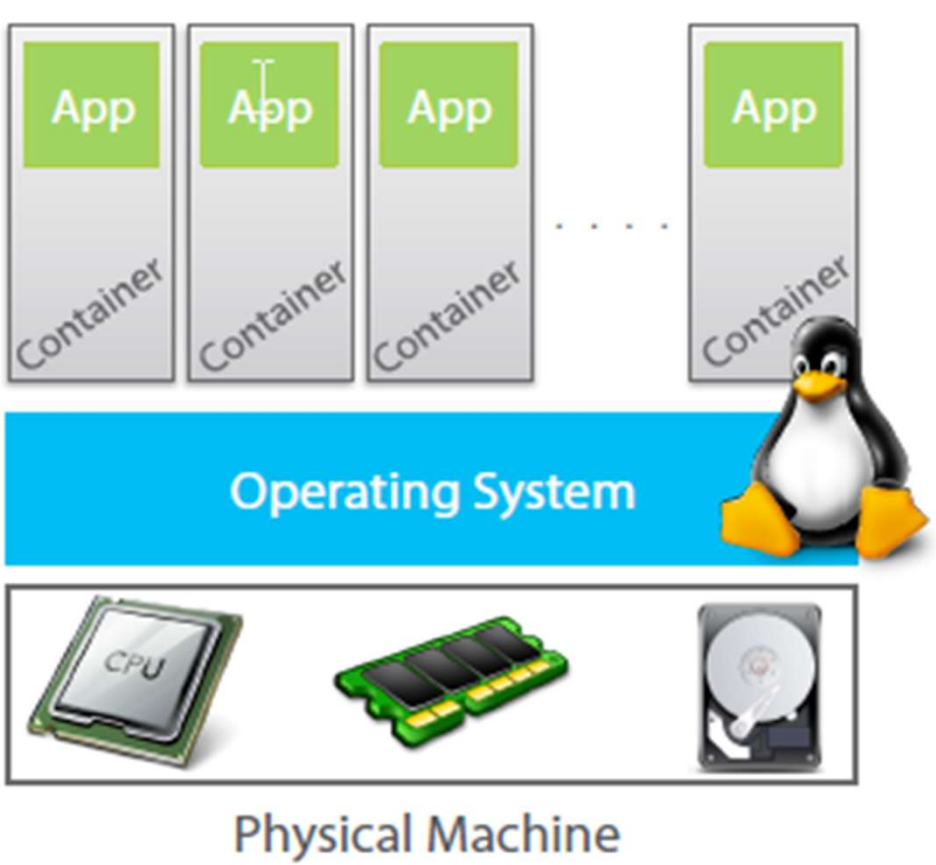
- Encapsulation of an application and its required environment.
- The process of packaging an application along with its required libraries, frameworks, and configuration files together so that it can be run in various computing environments efficiently.

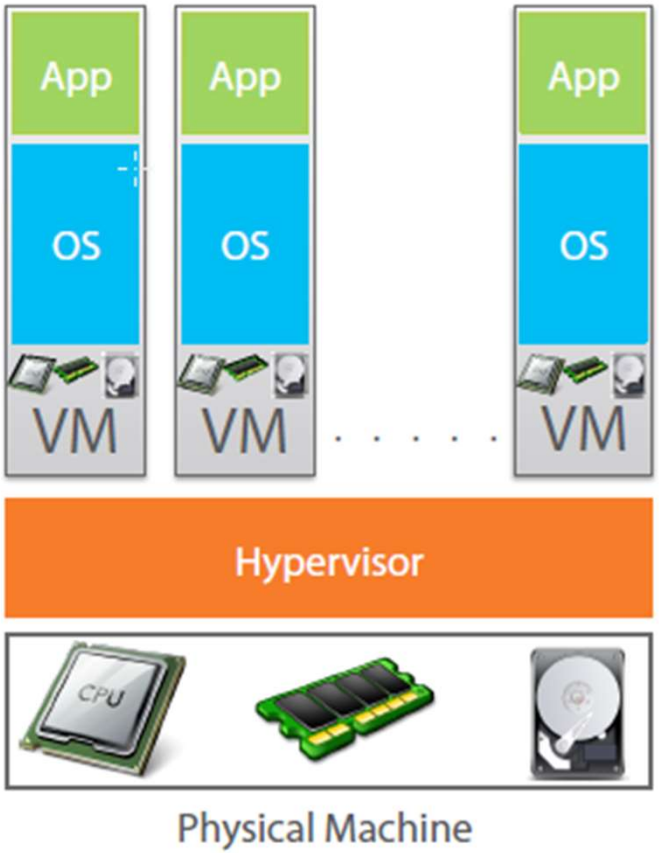
Containers to the Rescue



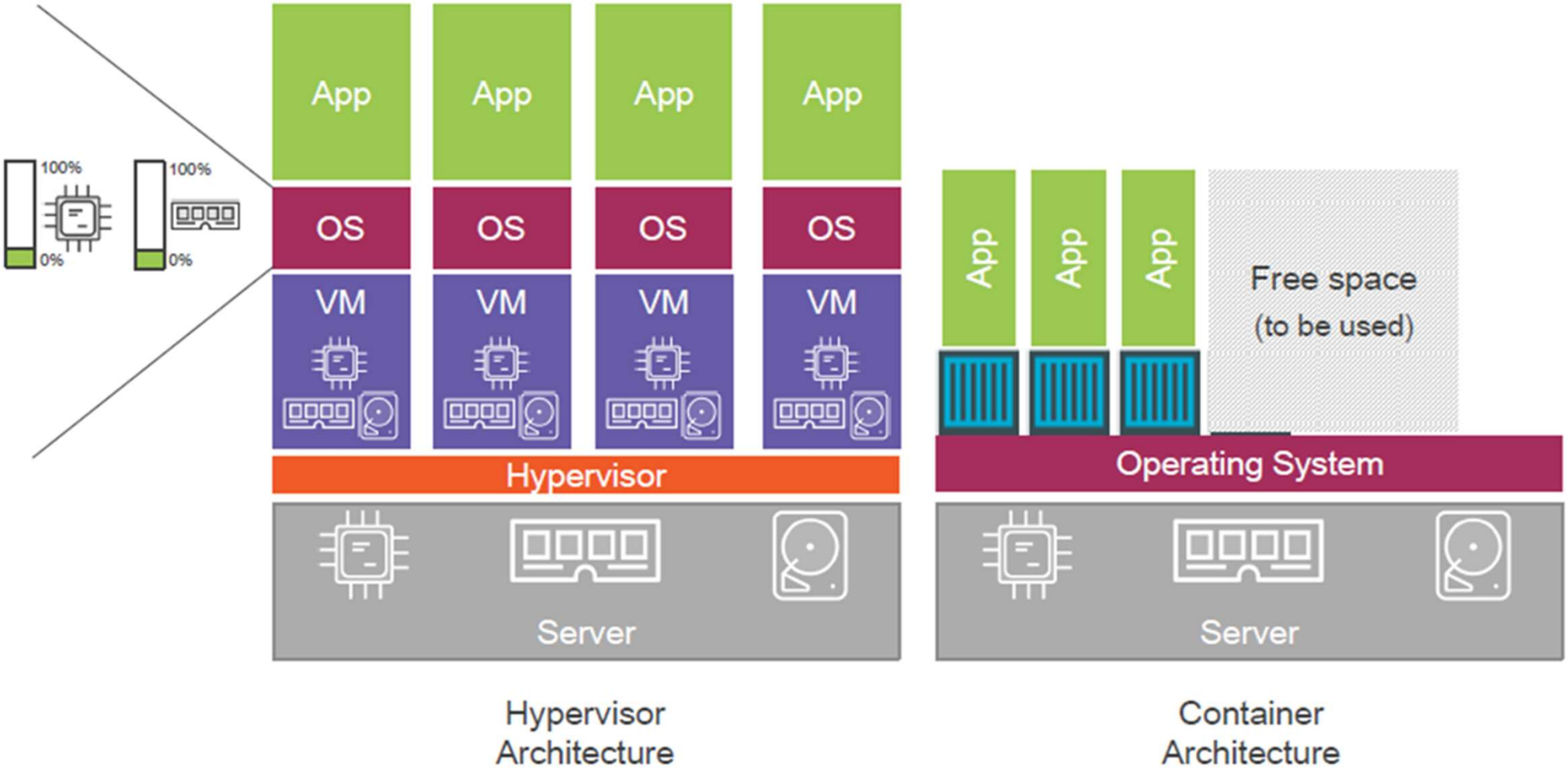
Containers are more lightweight than Virtual Machines

Containers vs VM

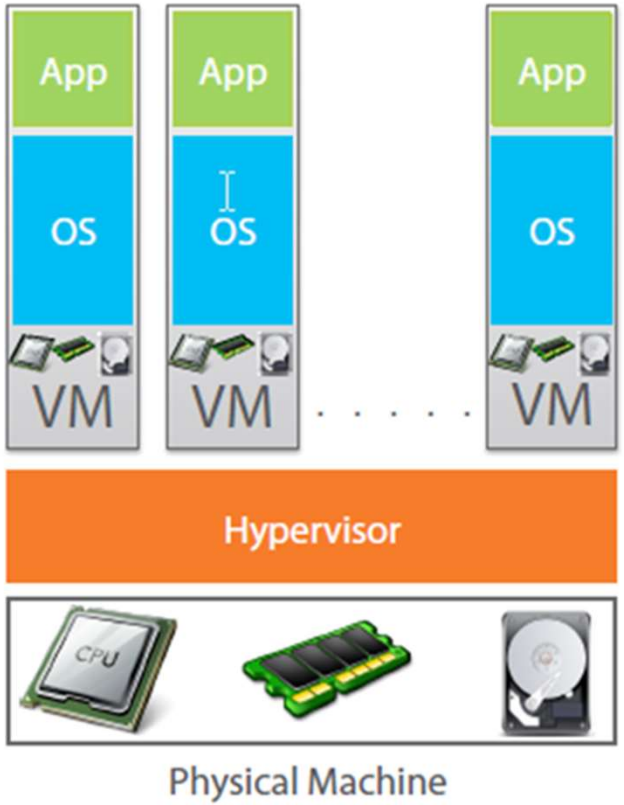


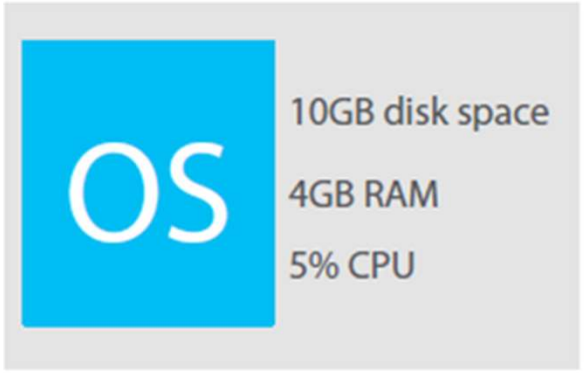


Containers vs VM



OS takes more resources and Licensing cost





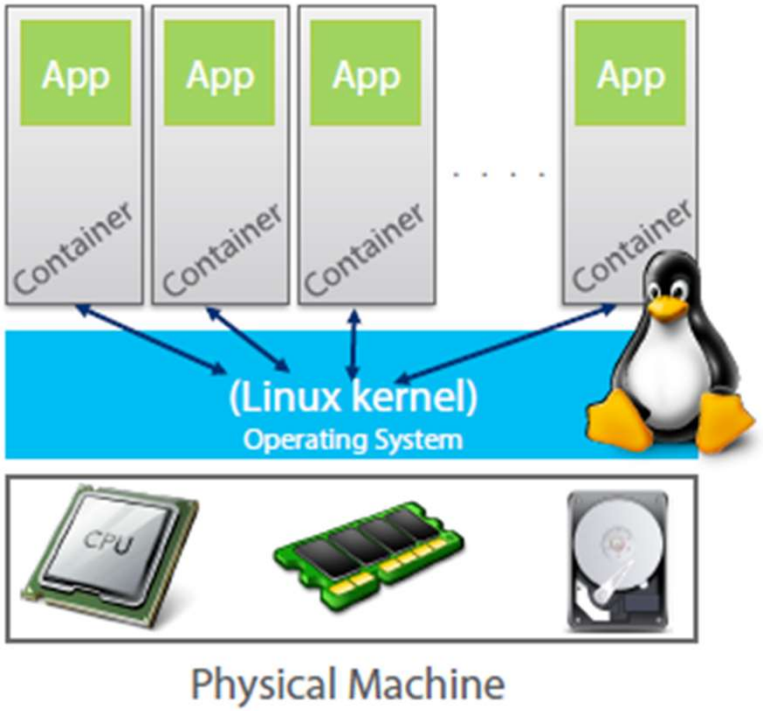
x 10 =

100GB disk space

40GB RAM

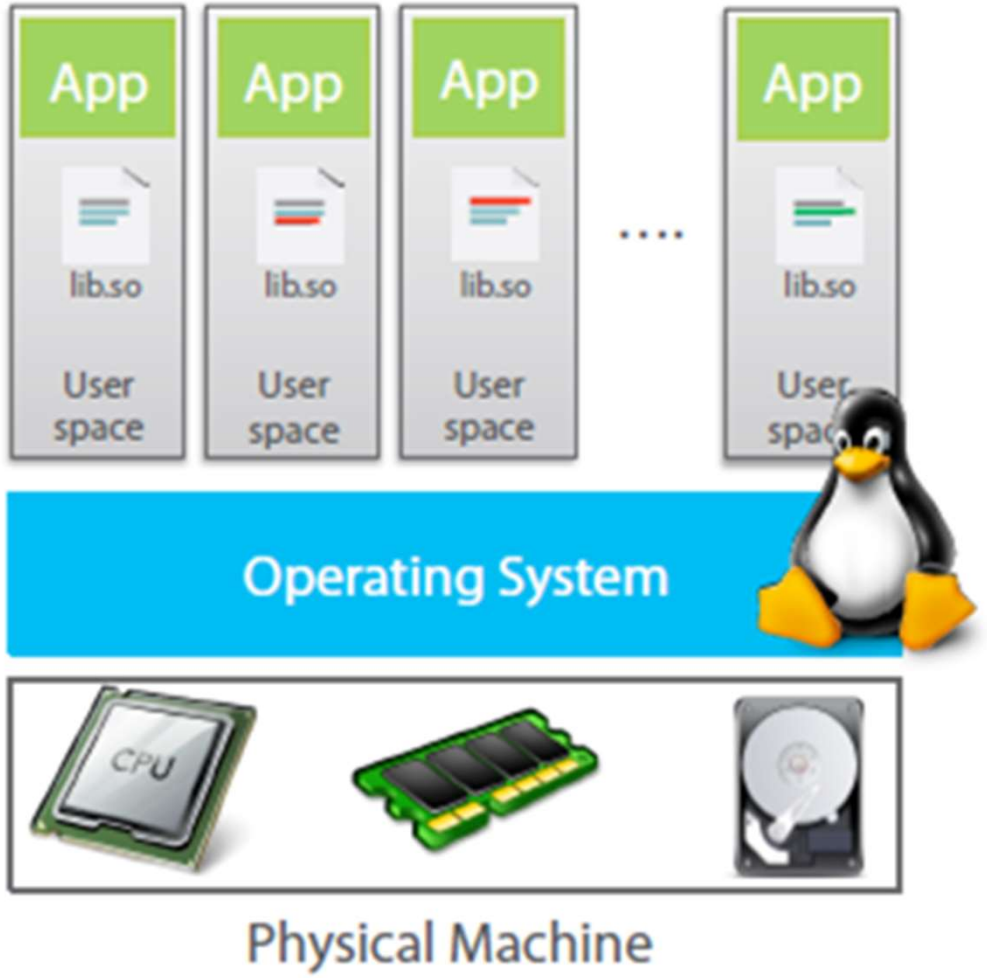
50% CPU

Containers takes less resources

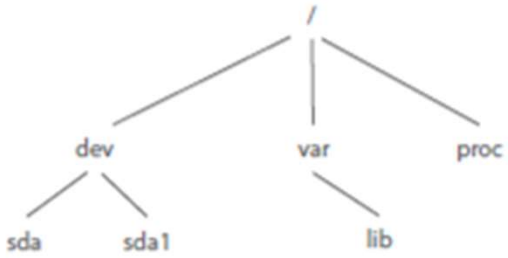


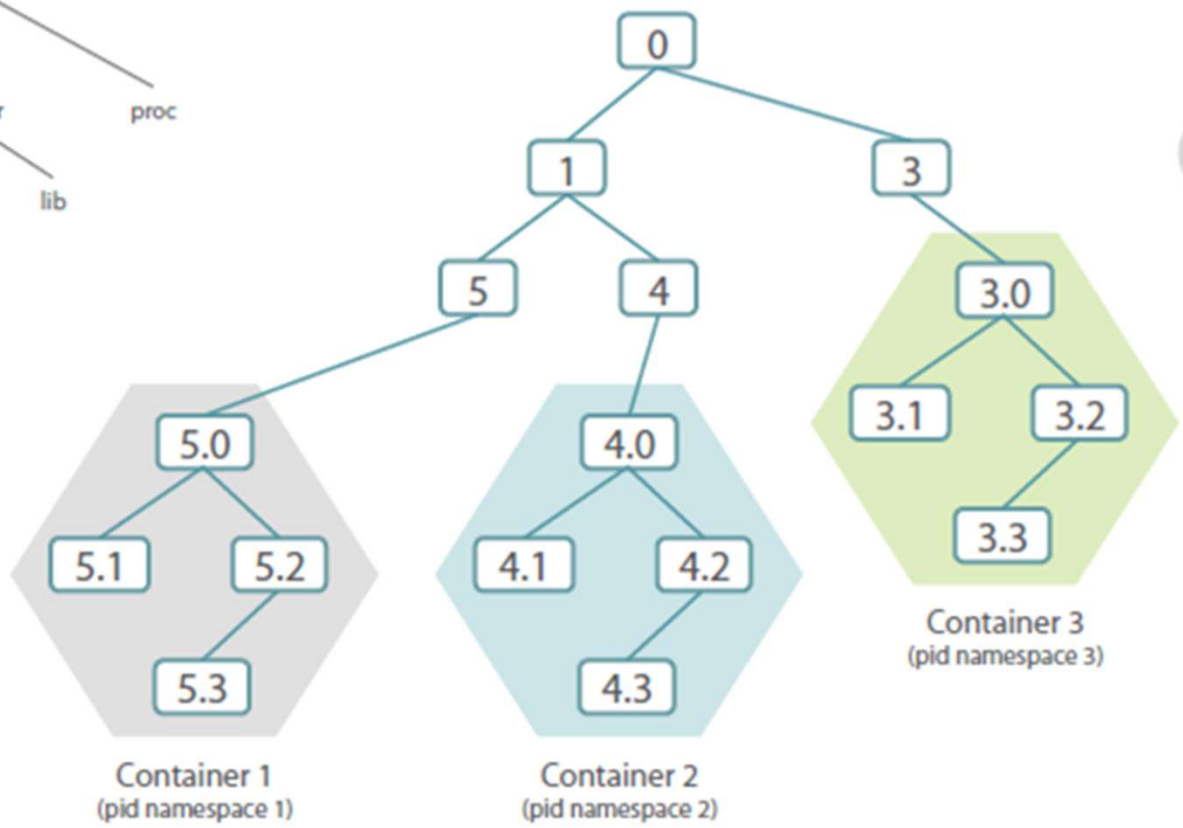
Containers consume less CPU, RAM and disk resource than Virtual Machines

How containers work?



How containers work?





What is Docker?

- Docker is an open-source project
  - that automates the deployment of applications inside software containers,
  - by providing an additional layer of abstraction and
  - automation of operating system–level virtualization on Linux.

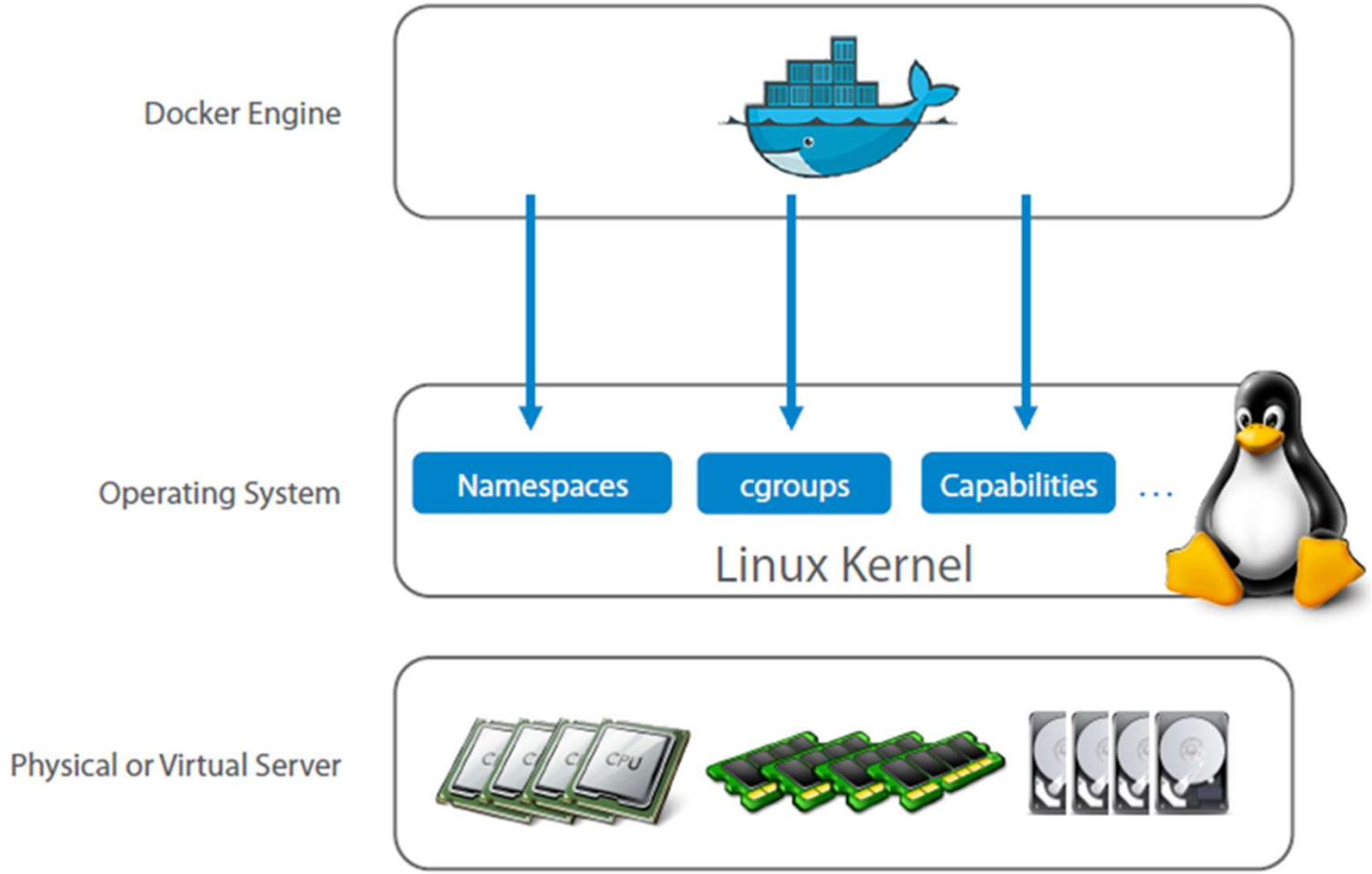
Practical

Practical Guide

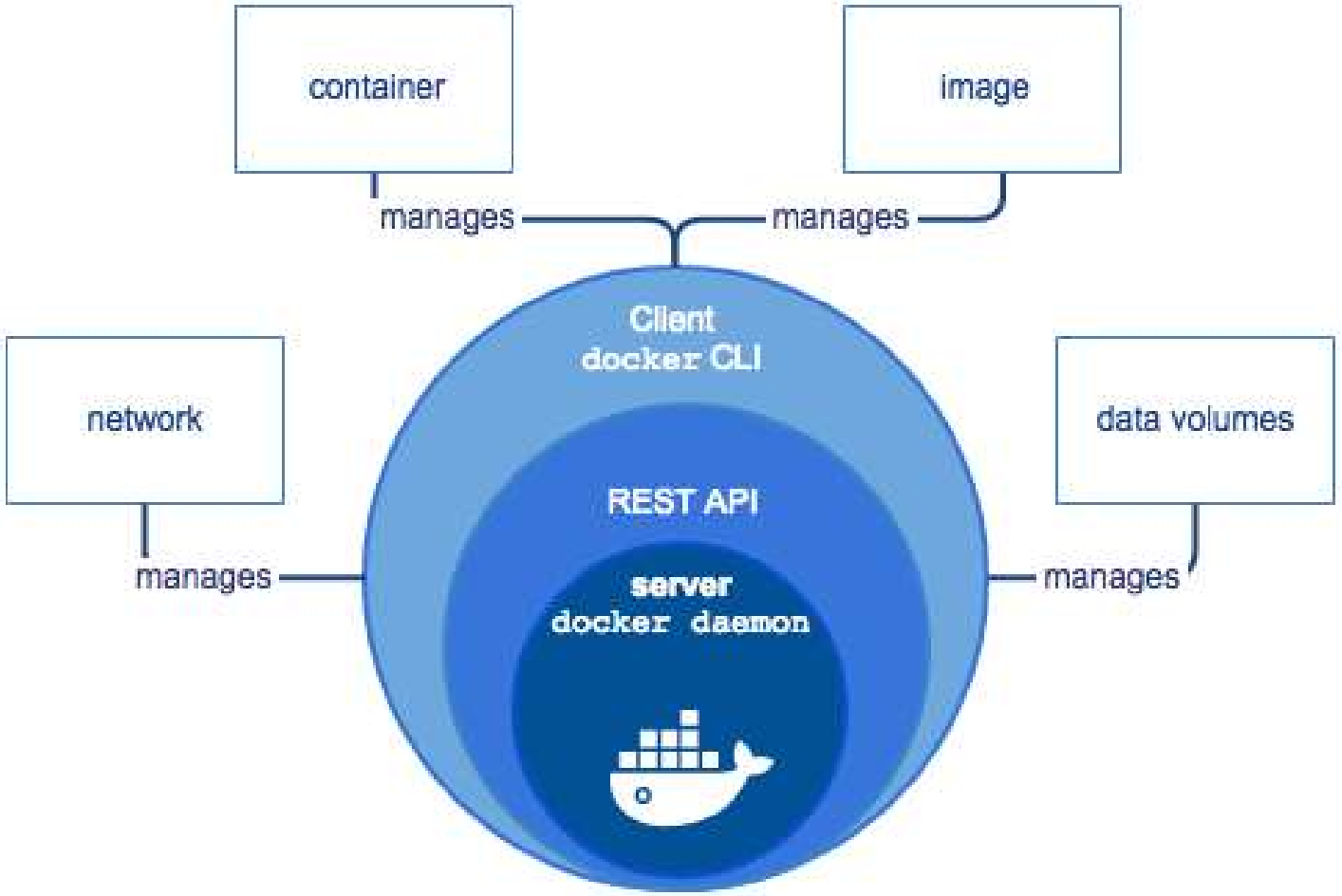
- Refer to the Practical Guide on below URL:
  - https://drive.google.com/open?id=1xcQk6enDDQvQls5V9X7MAtJbcGKl8Puk

- **Note:** We will get some hands-on for 30 minutes before moving on to next topic

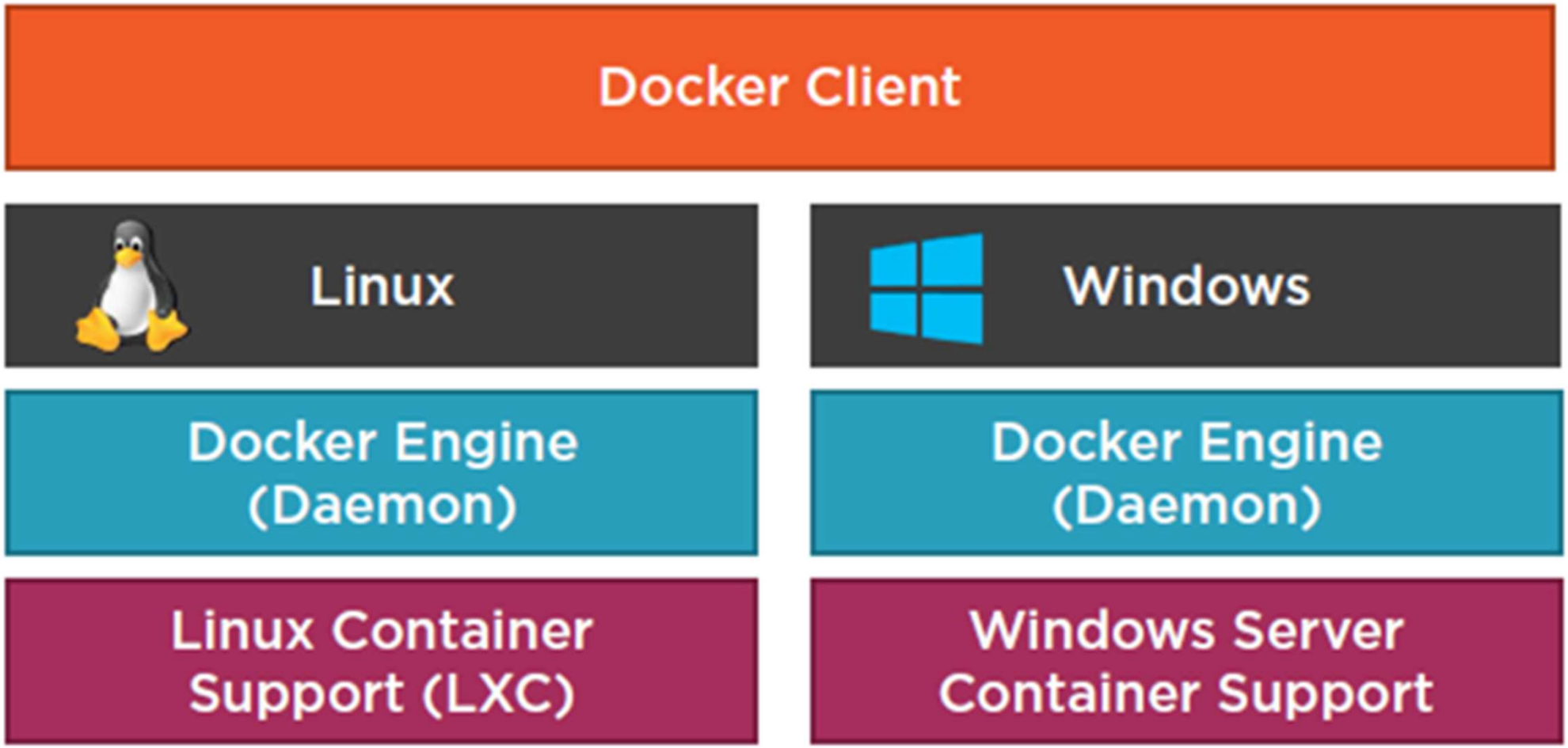
Docker Engine



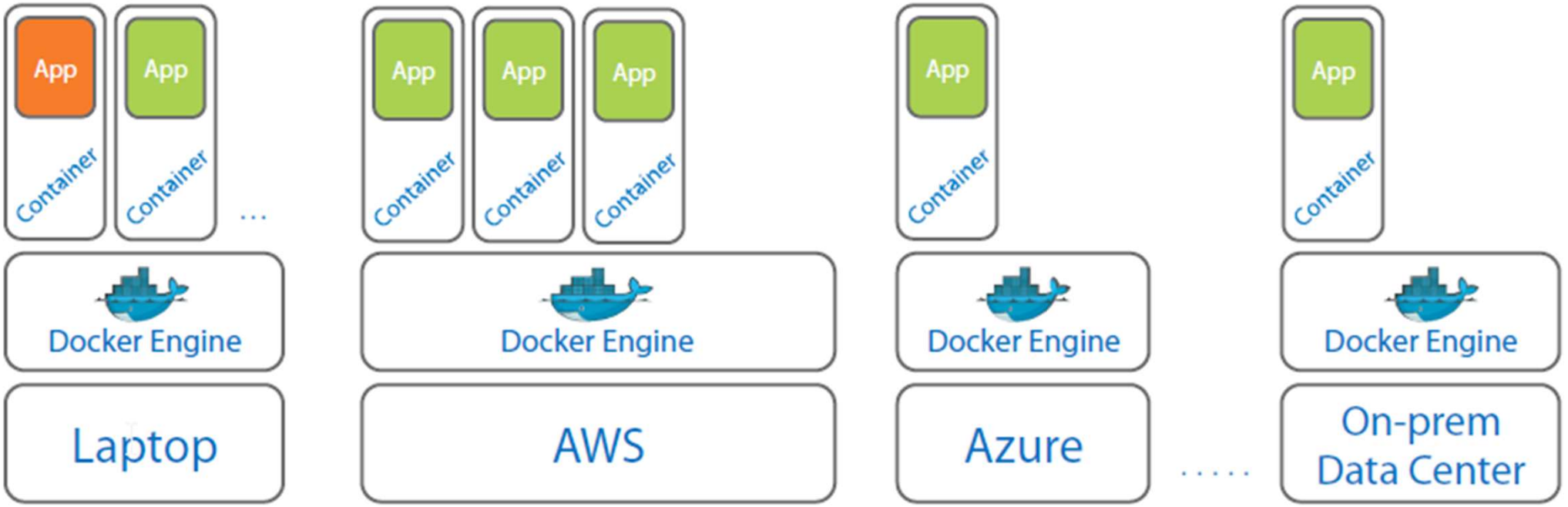
Docker Engine



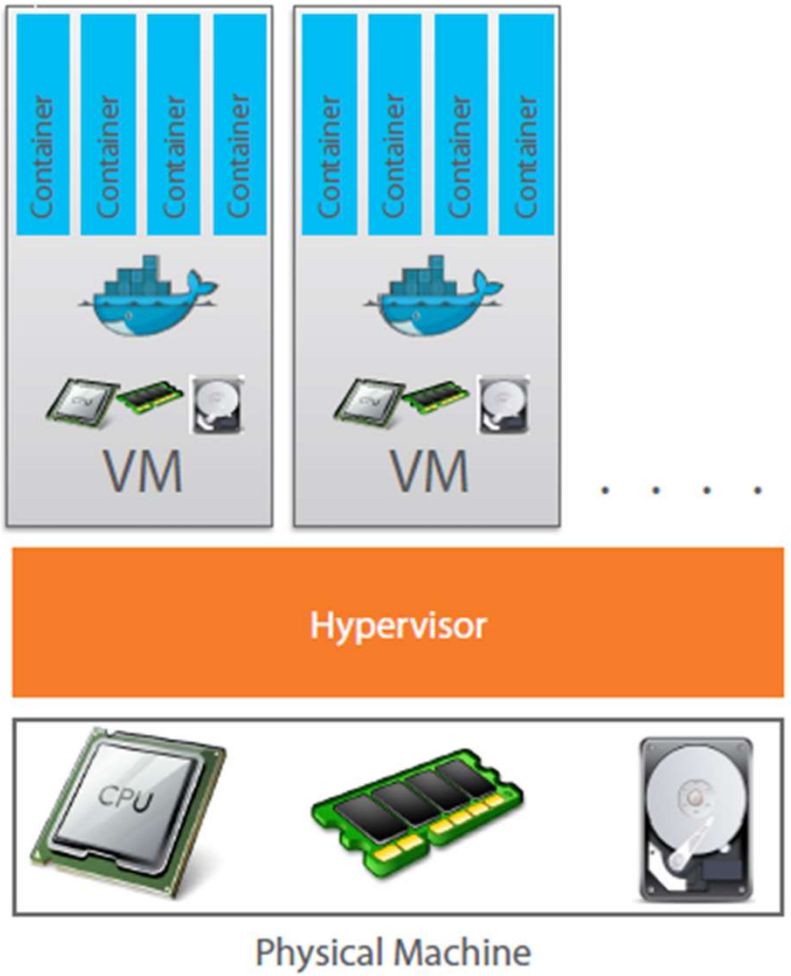
Where does Docker Run?

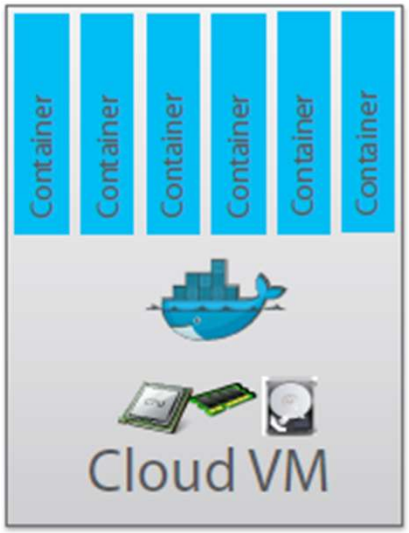


Docker can run anywhere



Docker on Physical Machine and also on Cloud

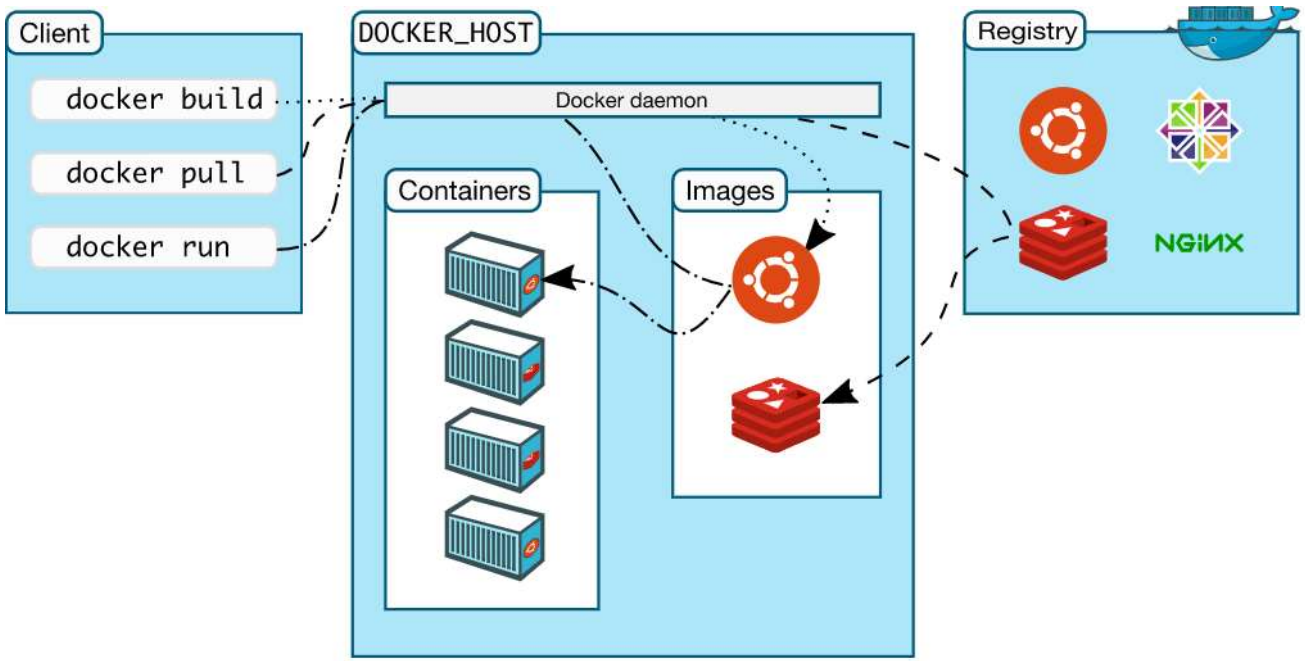




What can I use Docker for?

- Fast, consistent delivery of your applications
  - Continuous integration and continuous delivery (CI/CD) workflows.
  - Examples:
    - Developers write code and share their work with their colleagues using Docker containers.
    - Use Docker to push their applications into a test environment and execute tests.
    - When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
    - When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.
- Responsive deployment and scaling
  - Dynamically manage workloads, scaling up or tearing down applications and services
- Running more workloads on the same hardware

Docker Architecture



- Docker uses a client-server architecture.
- Docker client talks to the Docker daemon
- The Docker client and daemon can run on the same system, or can connect a client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API

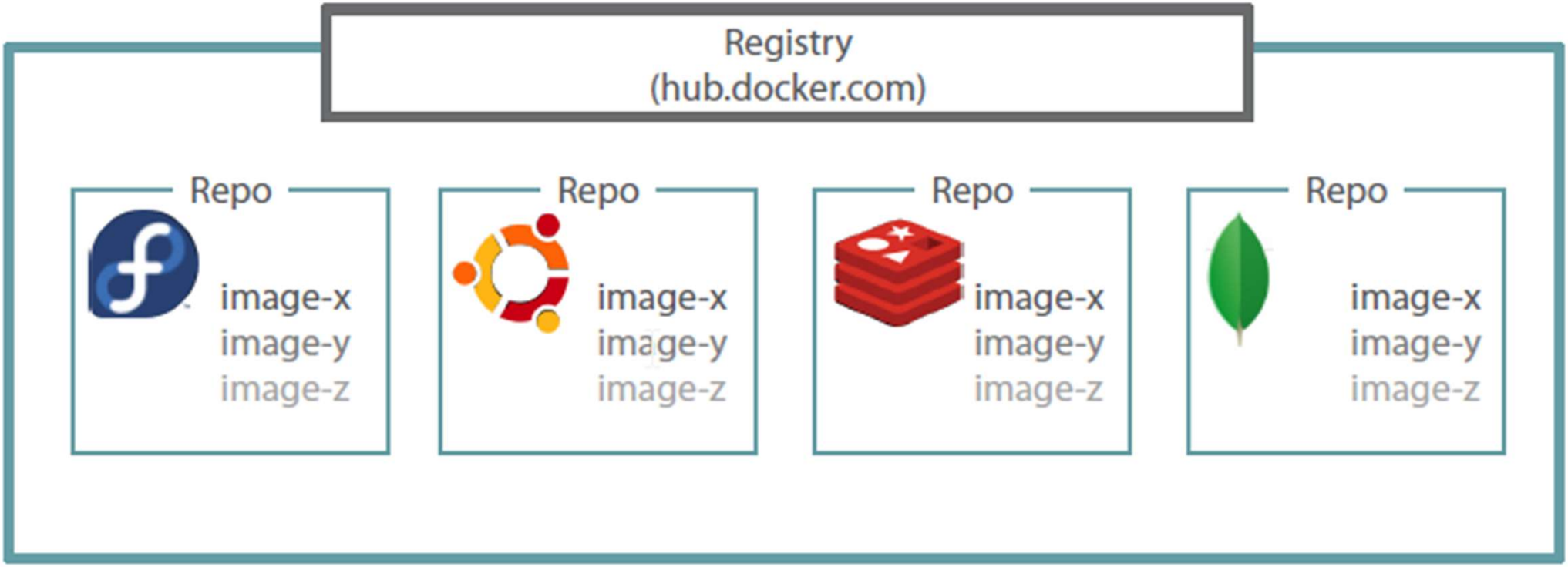
Image

- Persisted snapshot that can be run
- Common Docker Commands:
  - images: List all local images
  - run: Create a container from an image and execute a command in it
  - tag: Tag an image
  - pull: Download image from repository
  - rmi: Delete a local image

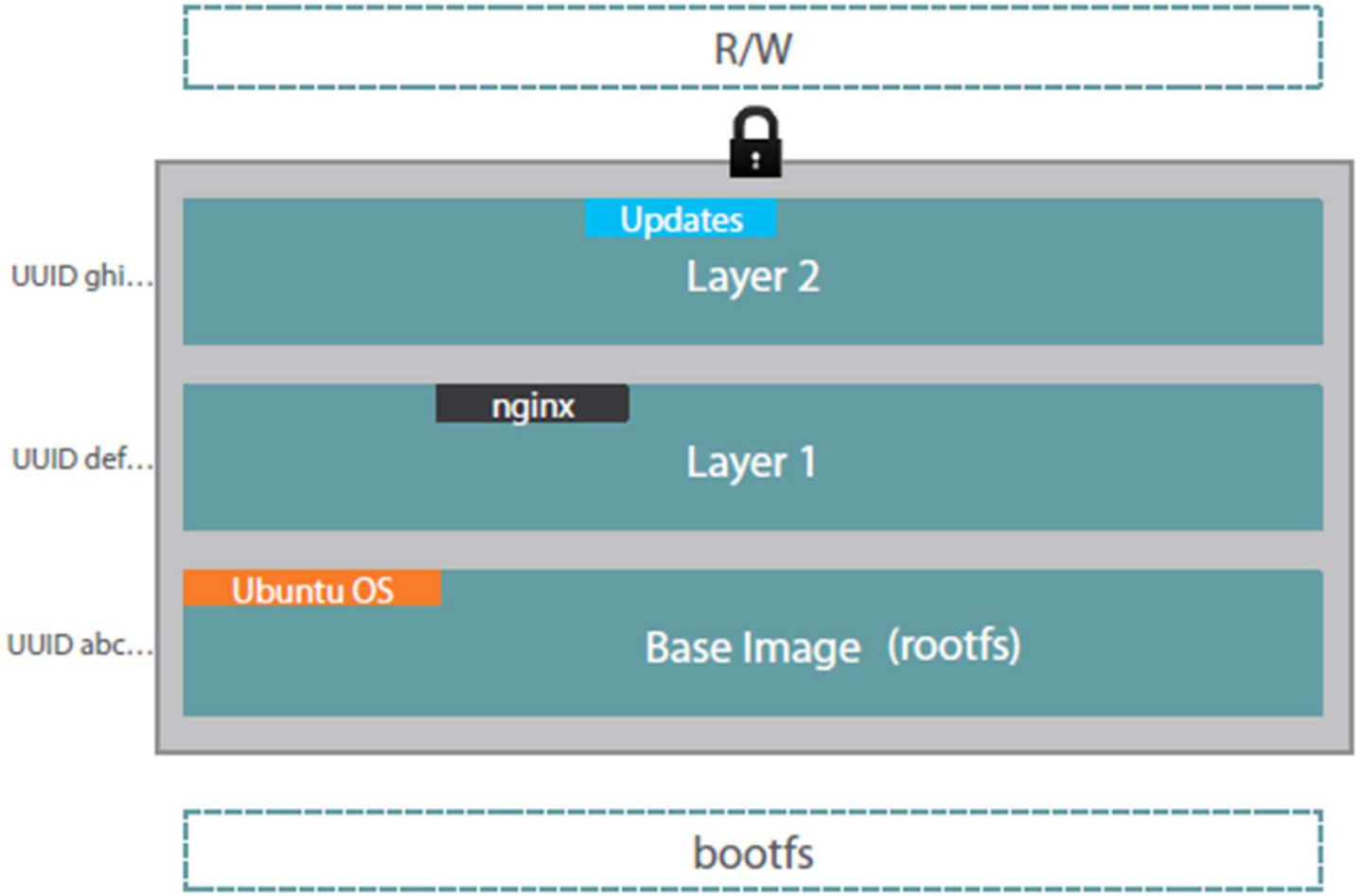
Container

- Runnable instance of an image
- Common Docker Commands
  - ps: List all running containers
  - ps –a: List all containers (incl. stopped)
  - top: Display processes of a container
  - start: Start a stopped container
  - stop: Stop a running container
  - pause: Pause all processes within a container
  - rm: Delete a container
  - commit: Create an image from a container

Docker Registry



Layers in Images



Docker Toolbox

- Docker Toolbox is for older Mac and Windows systems that do not meet the requirements of Docker Desktop.
- Docker Toolbox installs the
  - Oracle VirtualBox
  - Docker Engine
    - For running the docker commands
  - Docker Client,
    - Connects with the docker engine
  - Docker Machine,
    - For running docker-machine commands
  - Docker Compose
    - for running the docker-compose commands
  - Kitematic
    - Docker GUI

Install Docker Toolbox for Windows

- To run Docker, your machine must have a 64-bit operating system running Windows 7 or higher.
- Additionally, you must make sure that virtualization is enabled on your machine.
- Download from:
  - https://github.com/docker/toolbox/releases
- Install the downloaded setup and follow on screen instructions

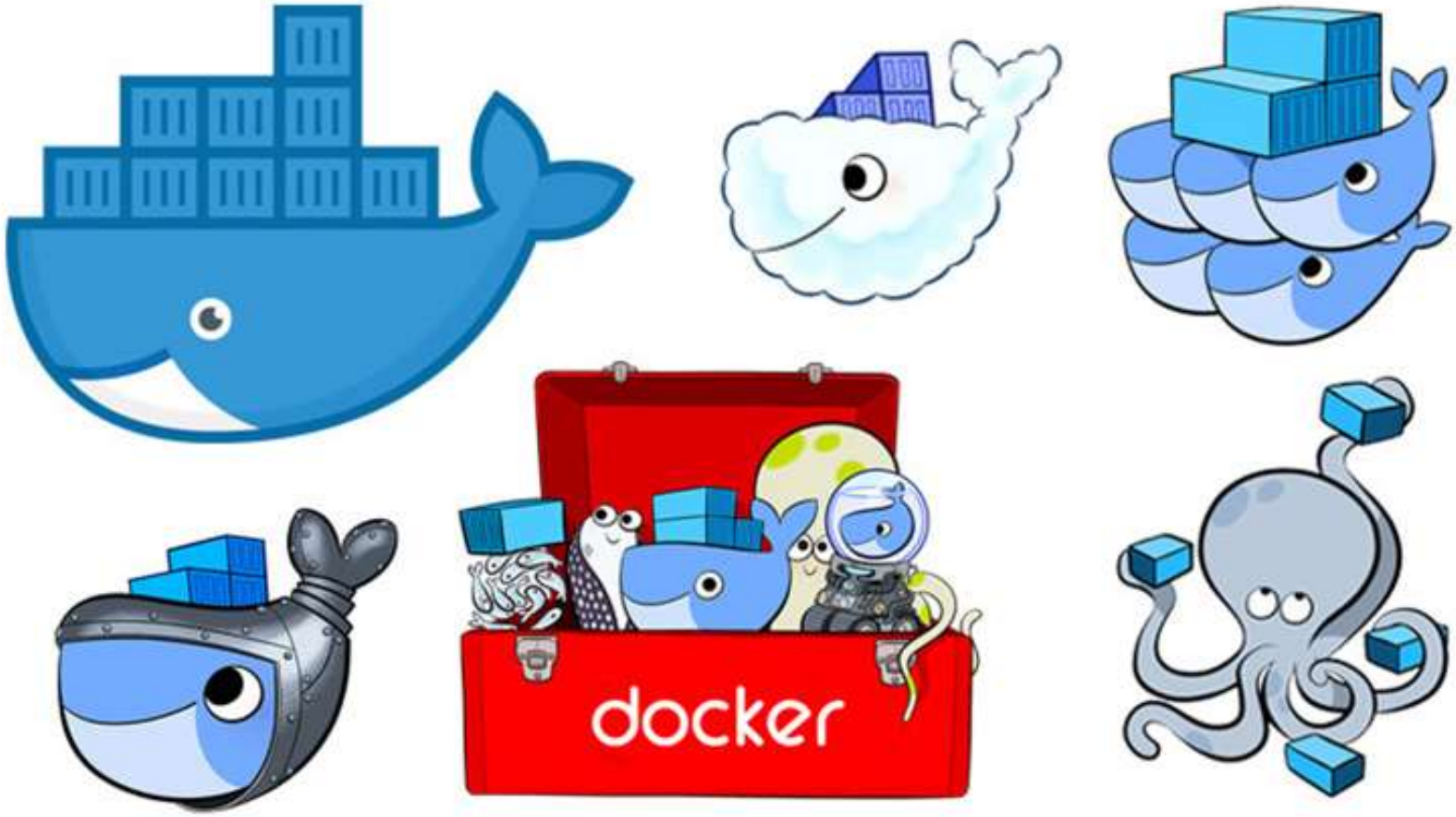
Install Docker Toolbox for Ubuntu

- Refer to our Docker Commands Guide:
  - https://drive.google.com/open?id=1xcQk6enDDQvQls5V9X7MAtJbcGKl8Puk

Hands-On

- We need to do the below hands-on:
  - Install and configure Docker
  - Deploy a Ubuntu 14.04 Server
  - ssh to Ubuntu server
  - Install Docker engine on Ubuntu 14.04
  - Validate docker engine is successfully installed
  - Launch a docker container
  - Login to container
  - Work in a container
  - List containers
  - Pause a container
  - Un-pause a container
  - Delete container
- Refer to the command guide for instructions

Docker Ecosystem

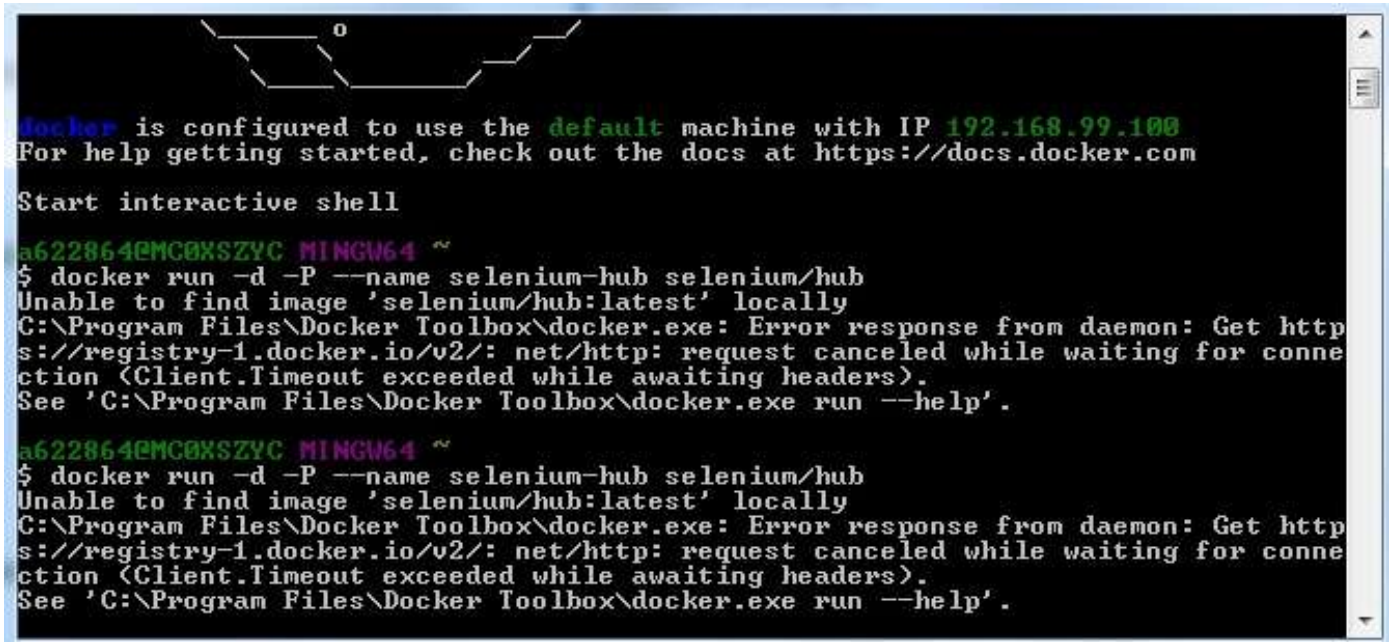


Docker Daemon

- Docker uses a client-server architecture.
- The Docker Daemon runs on your host operating system.
- This as a service that runs in the background.
- It is the brains of the operation when it comes to managing Docker containers.
- Docker client talks to the Docker daemon

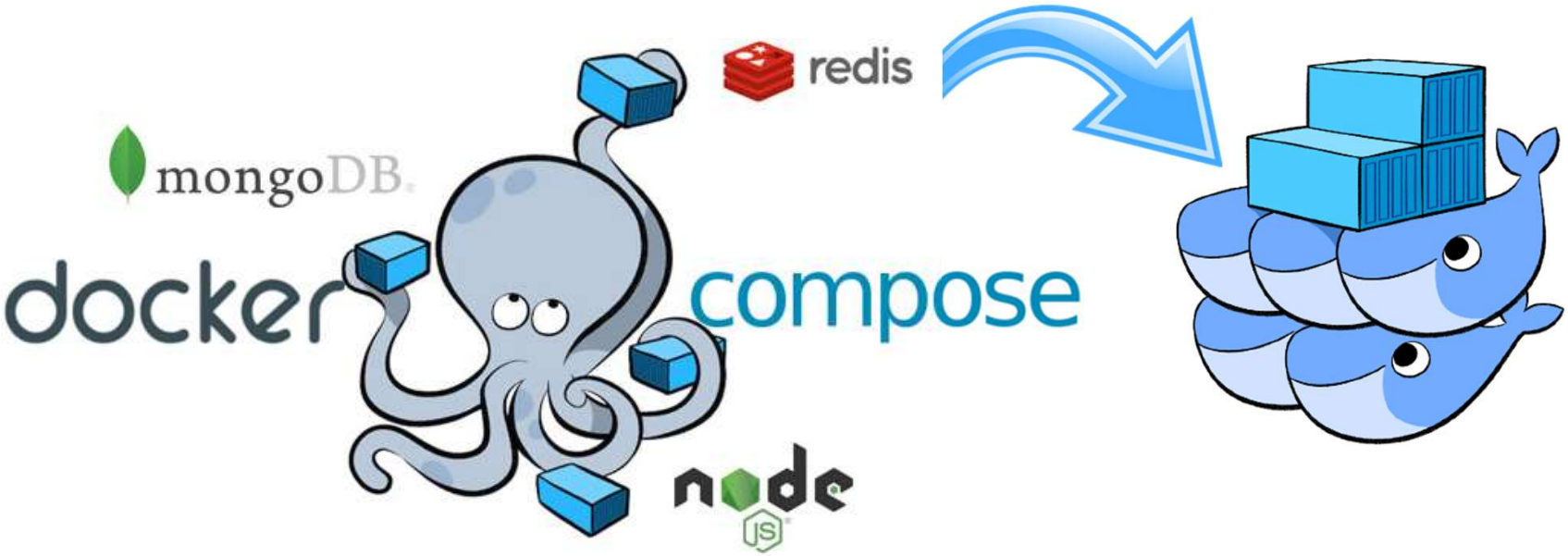
Docker CLI

- To interact with the Docker Daemon.
- The Docker Daemon exposes an API and the Docker CLI consumes that API.



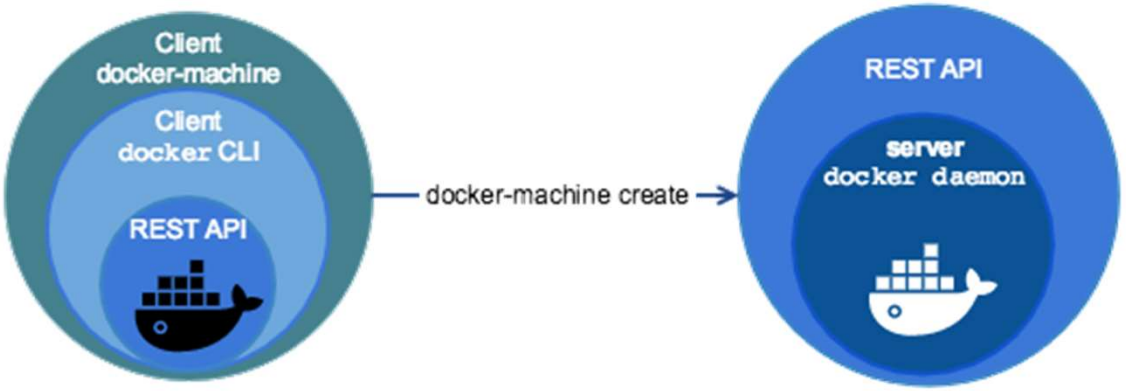
Docker Compose

- It lets you declare Docker CLI commands in the form of YAML



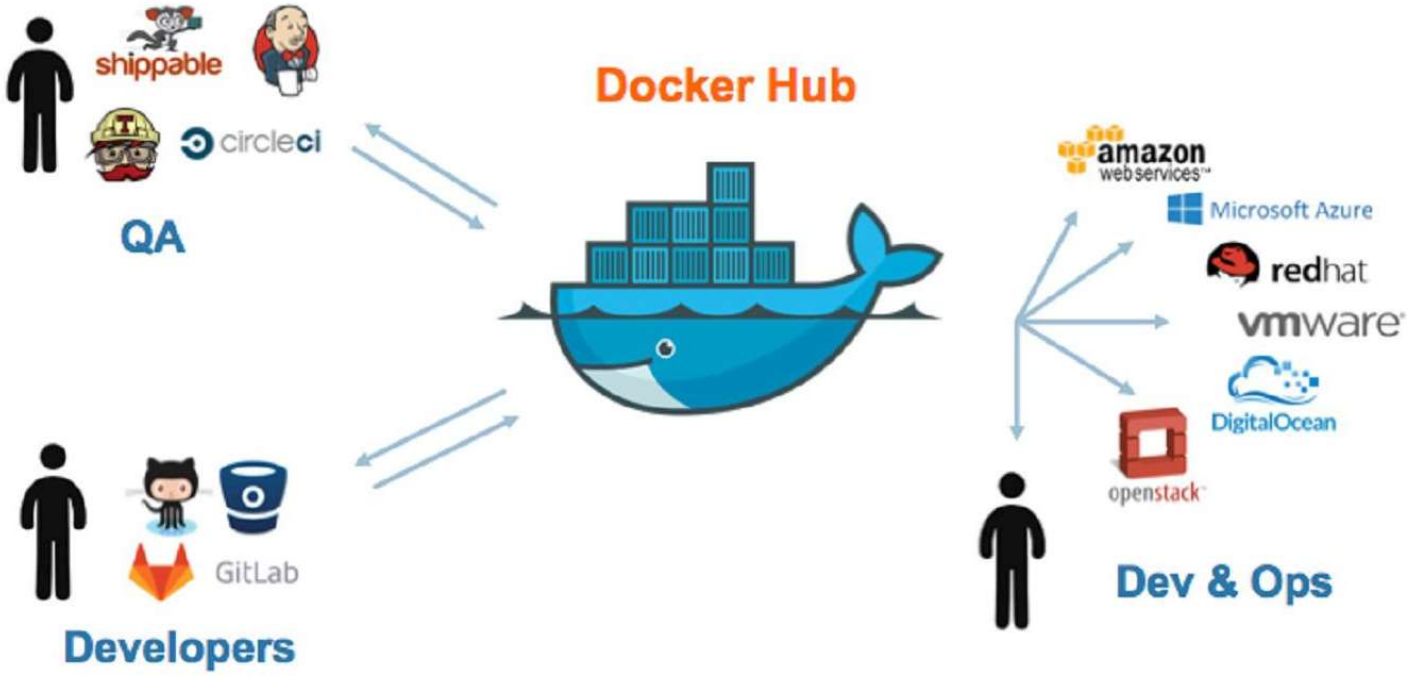
Docker Machine

- A command line tool that helps to create servers and then install Docker Engine on them.



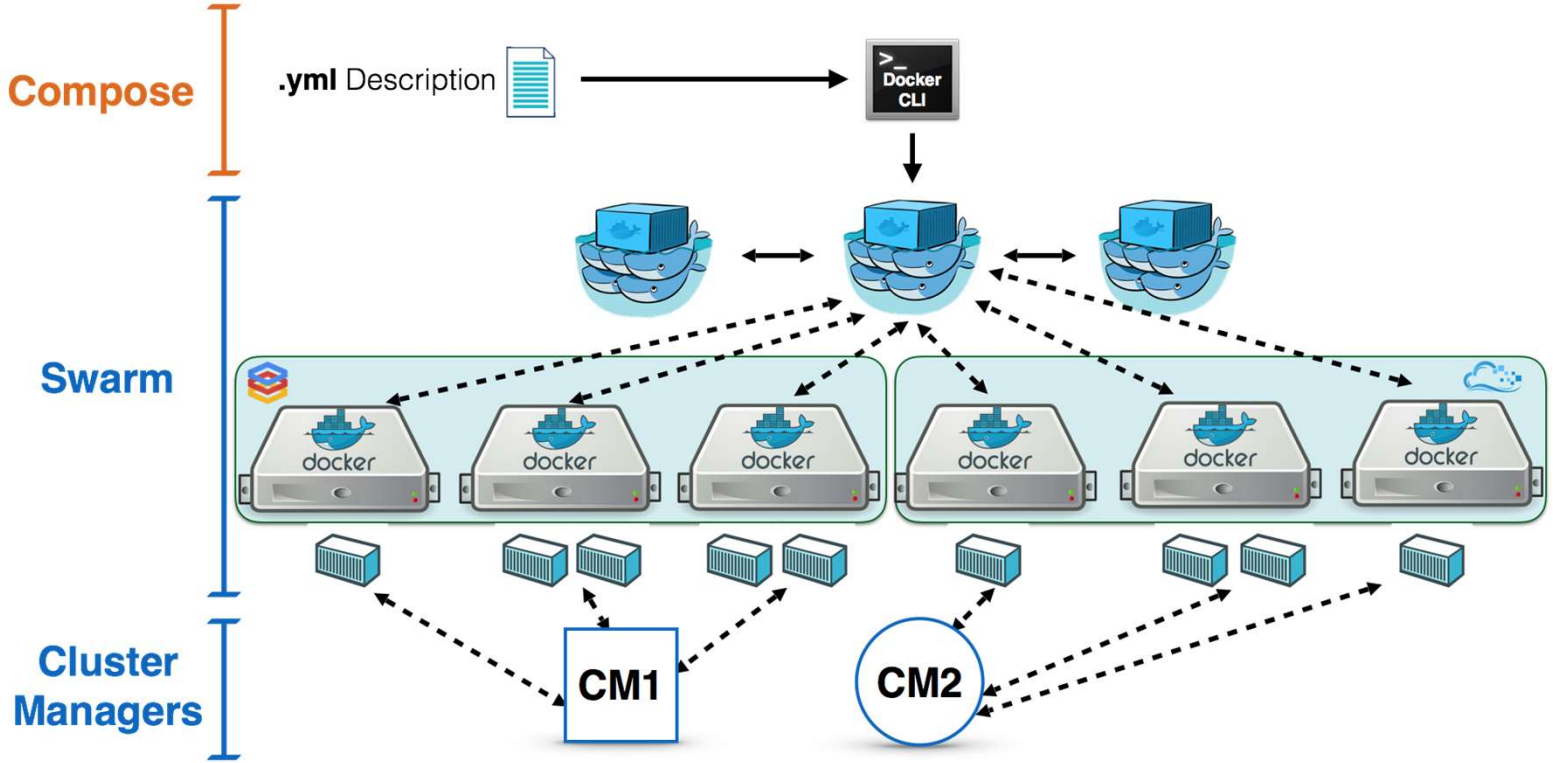
Docker Hub

- Are websites you can visit to find Docker images and even store your own (both publicly and privately).
- https://hub.docker.com/



Docker Swarm

- To manage a cluster of servers which can house 1 or more services.



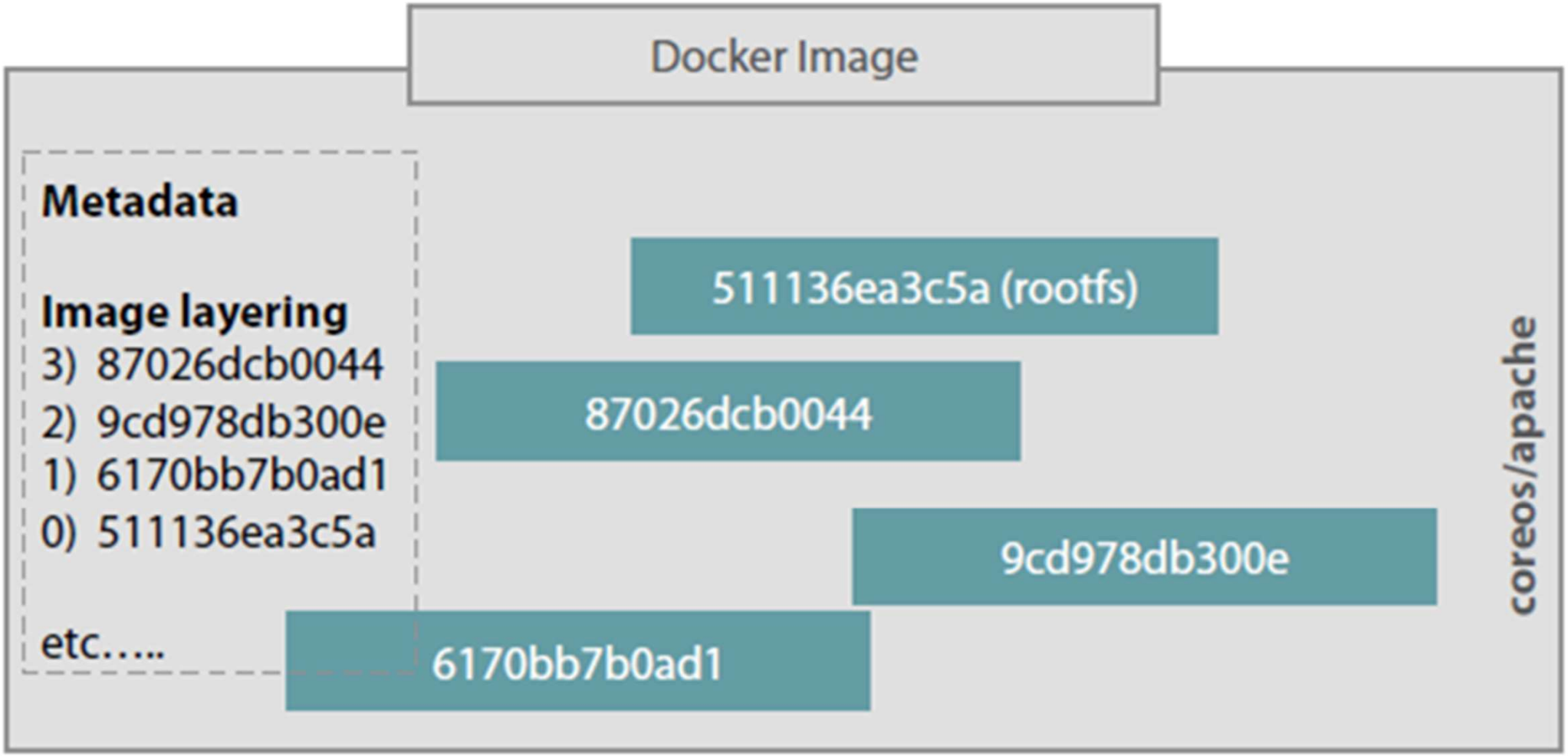
Docker Swarm

- A group of machines
- Have been configured to join together in a cluster.
- Activities are controlled by a swarm manager.
- Is a container orchestration tool
- Allows the user to manage multiple containers deployed across multiple host machines.
- Benefit: High Availability

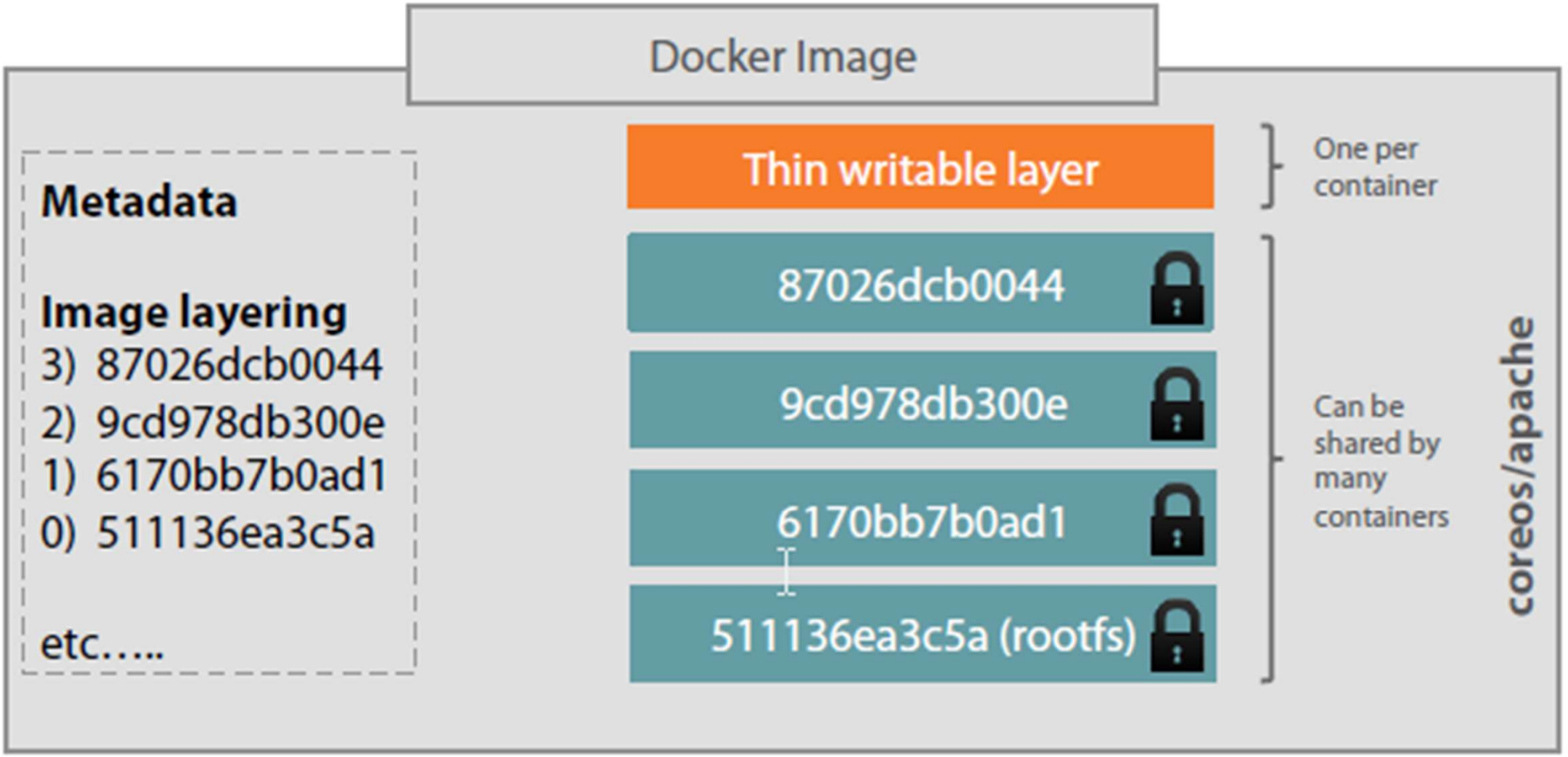

thanks

Day 2

Layers in Docker Image



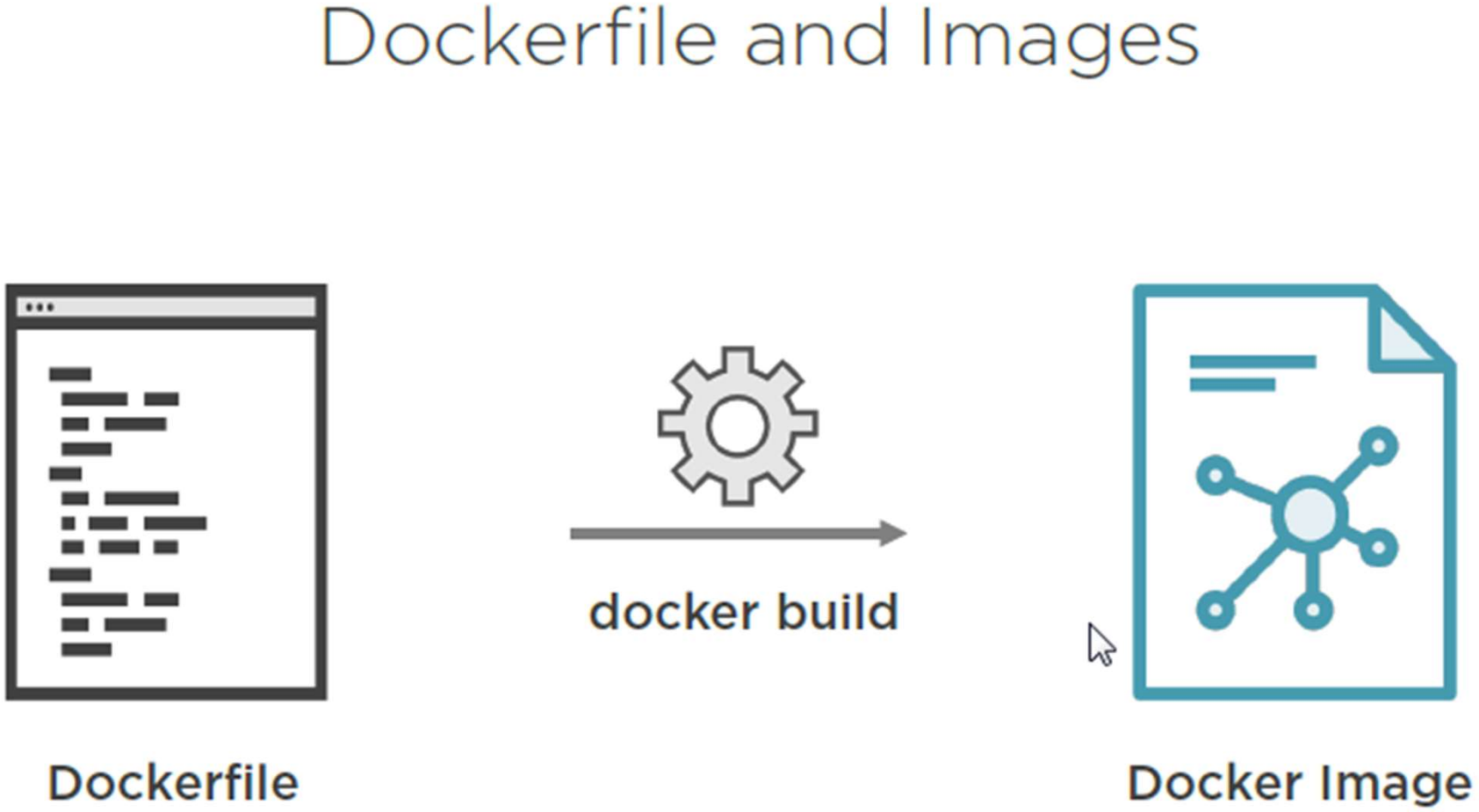
Layers in Docker Image



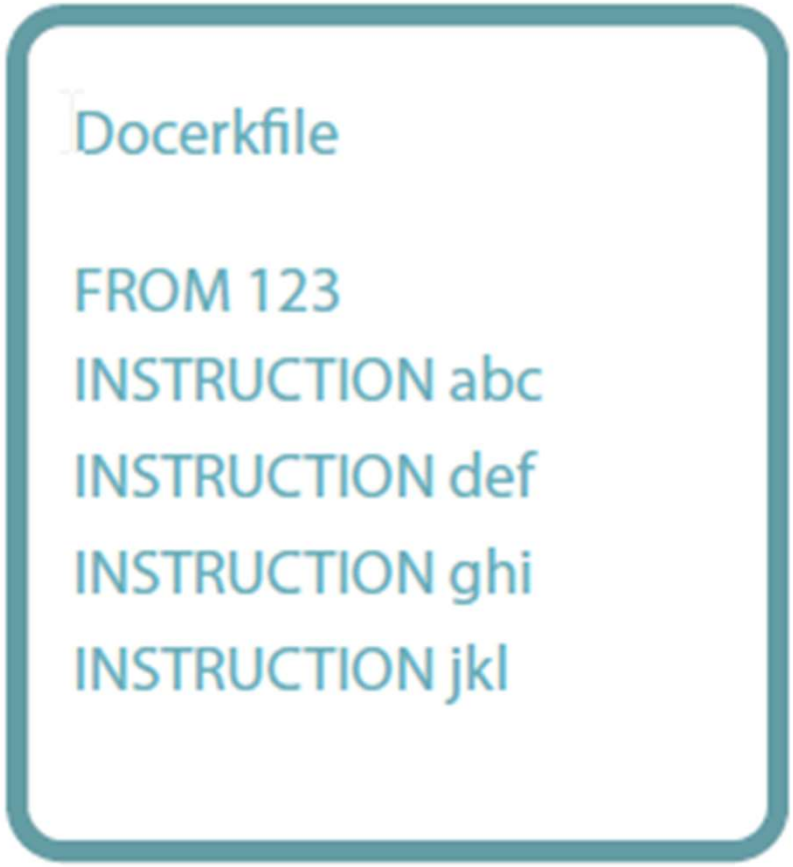
Dockerfile

- Create images automatically using a build script: «Dockerfile»
- Can be versioned in a version control system like Git
- Docker Hub can automatically build images based on dockerfiles on Github

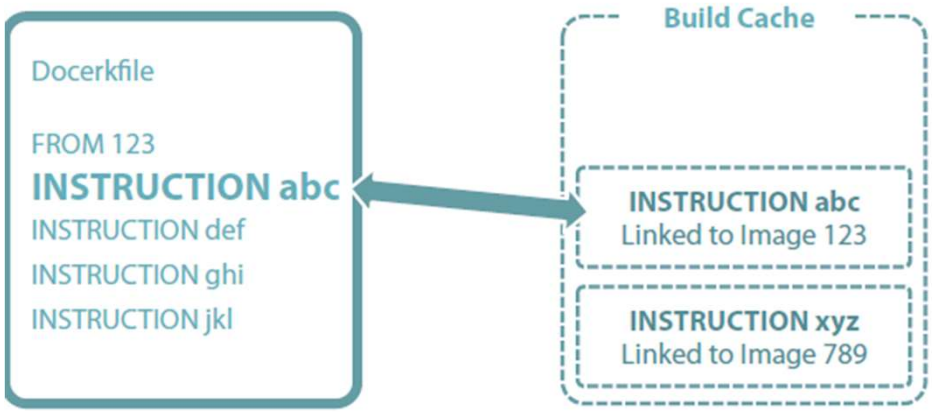
Dockerfile


Dockerfile and Images
docker build
Dockerfile
Docker Image

Dockerfile Template



Dockerfile Build Cache

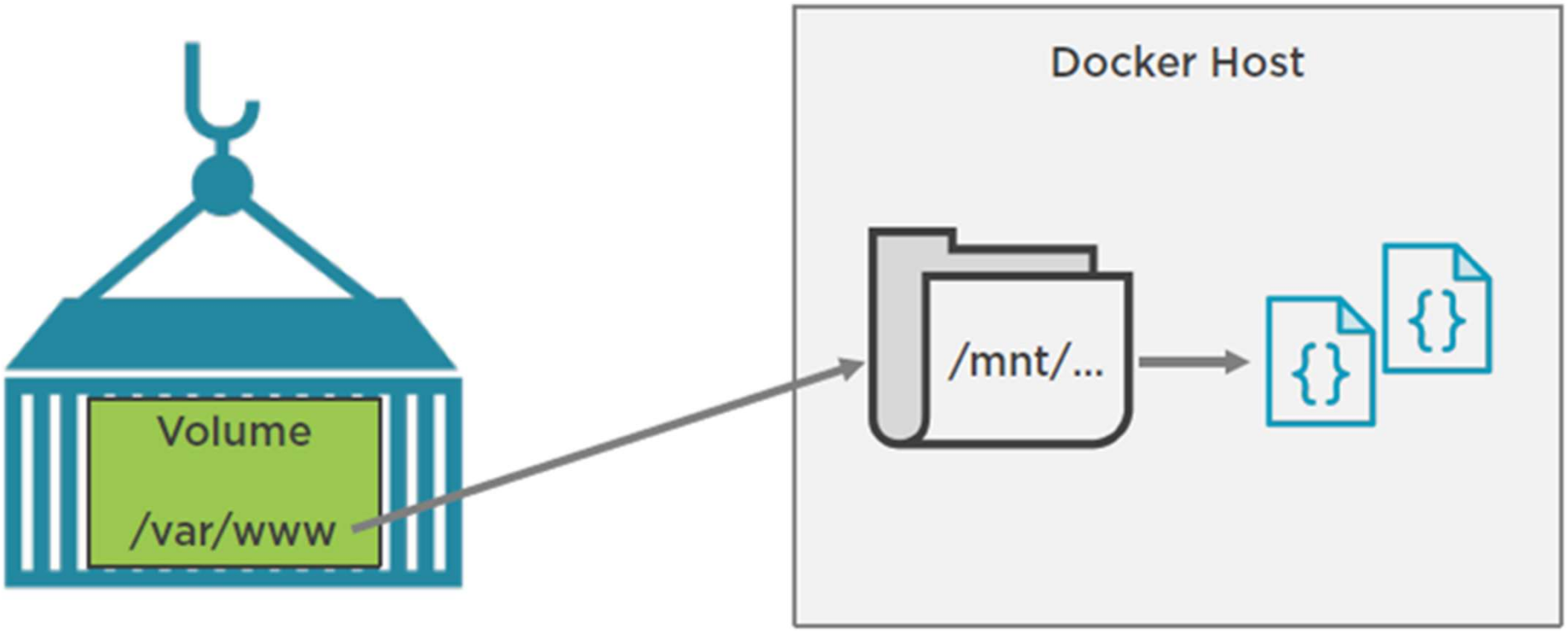


Dockerfile Example

- Dockerfile:
  - FROM ubuntu
  - ENV DOCK_MESSAGE Hello My World
  - ADD dir /files
  - CMD ["bash", "someScript"]

- docker build [DockerFileDir]
- docker inspect [imageId]

Docker Volume

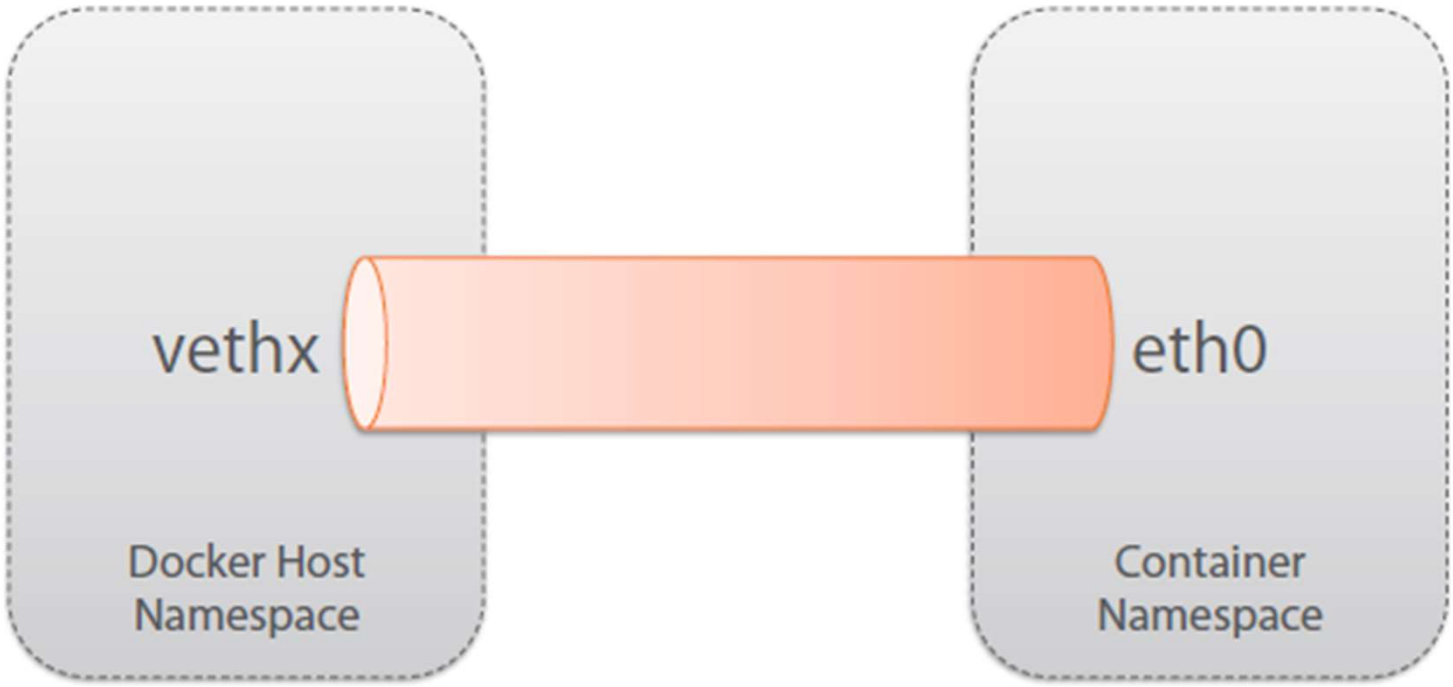


Mount Volumes

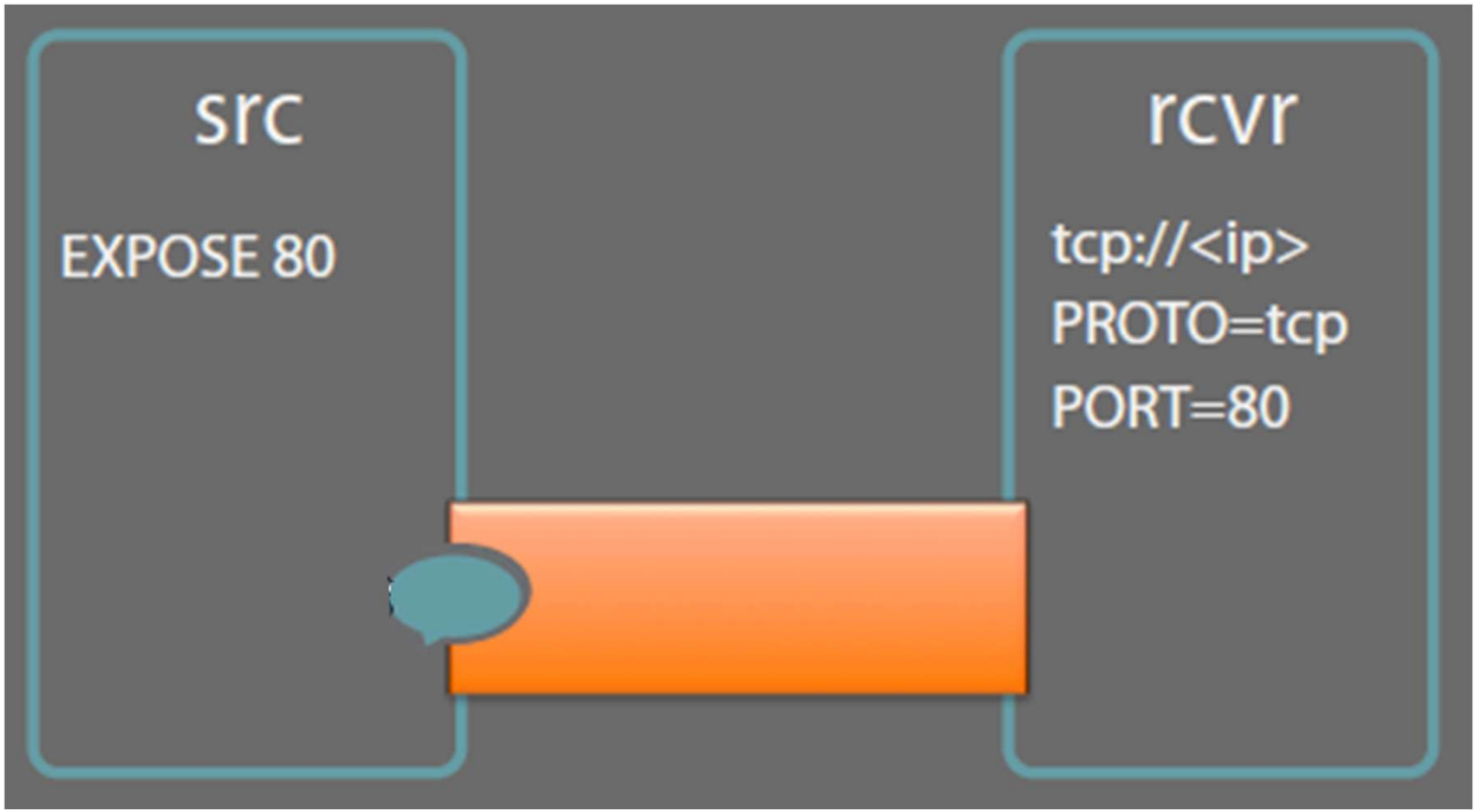
- docker run –ti –v /hostLog:/log ubuntu

- Run second container: Volume can be shared
  - docker run –ti –volumesfrom firstContainerName ubuntu

Docker Networking



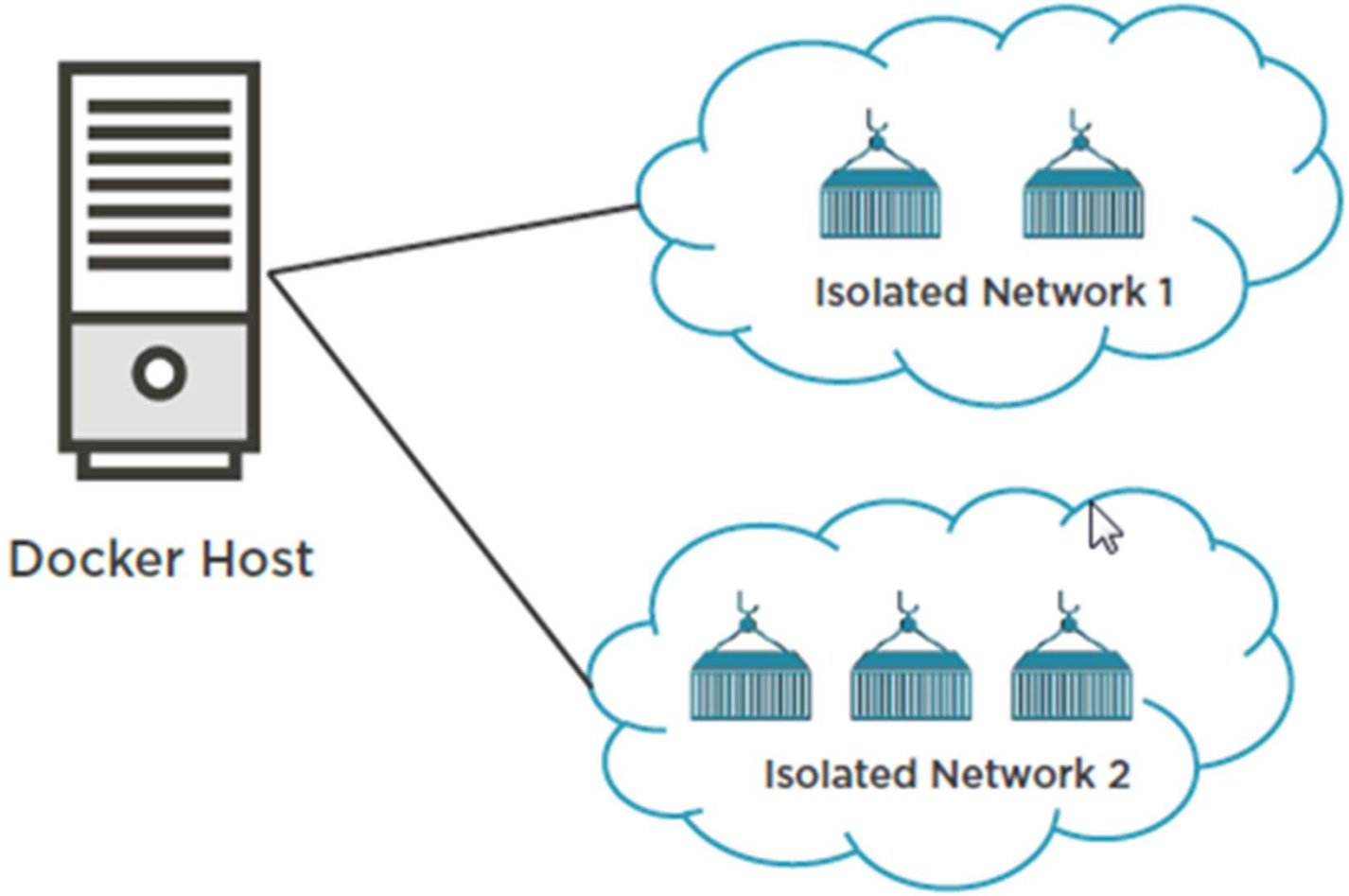
Connecting Containers



Publish Port

- docker run –t –p 8080:80 ubuntu
  - Map container port 80 to host port 8080
- Link with other docker container
  - docker run -ti –link containerName:alias ubuntu

Understanding Container Networks



Docker Hub

- Public repository of Docker images
  - https://hub.docker.com/
  - docker search [term]
- Automated: Has been automatically built from Dockerfile
  - Source for build is available on GitHub

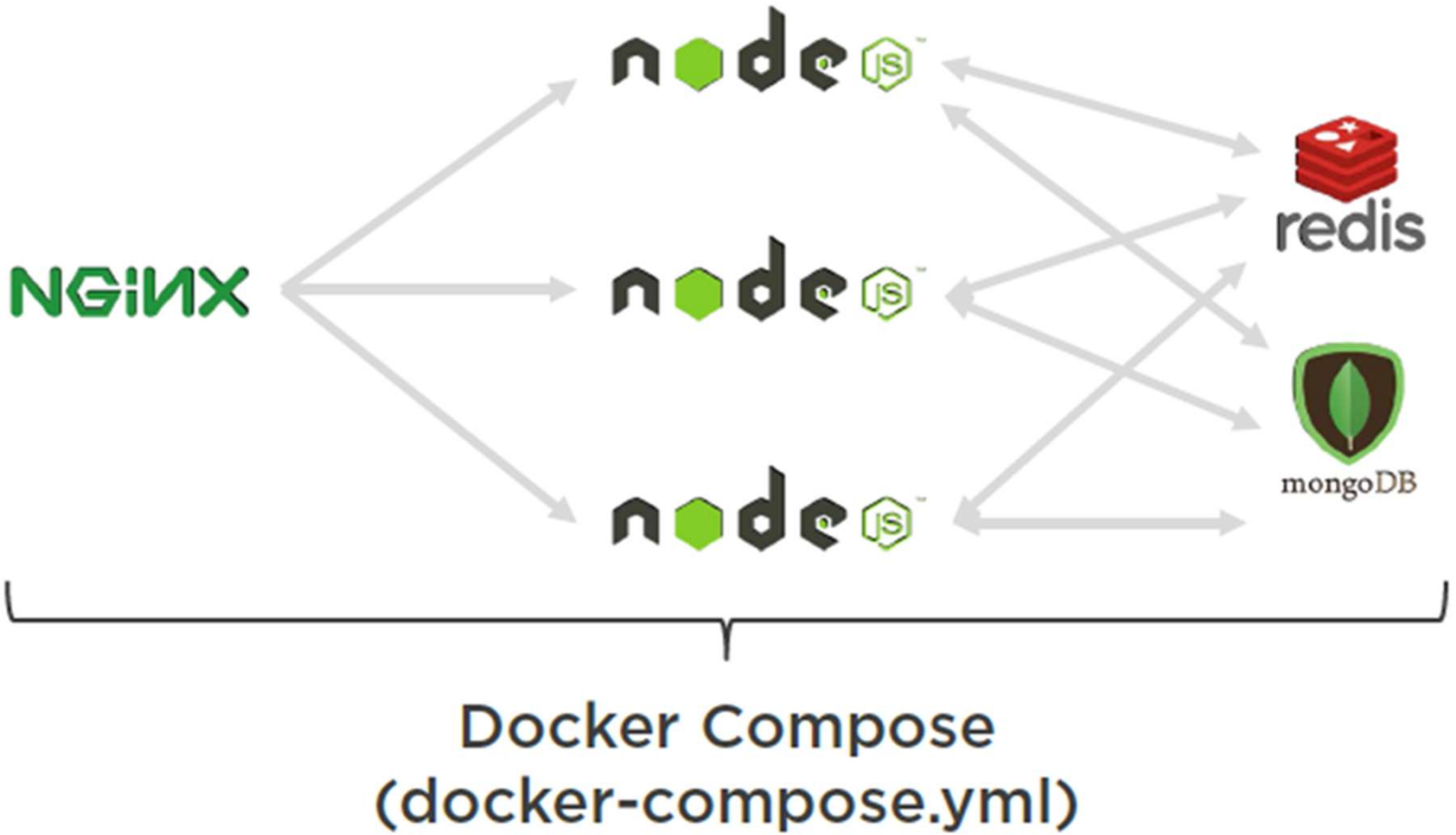
Resource Usage

- top / ps / free -m
- docker ps [containerId]

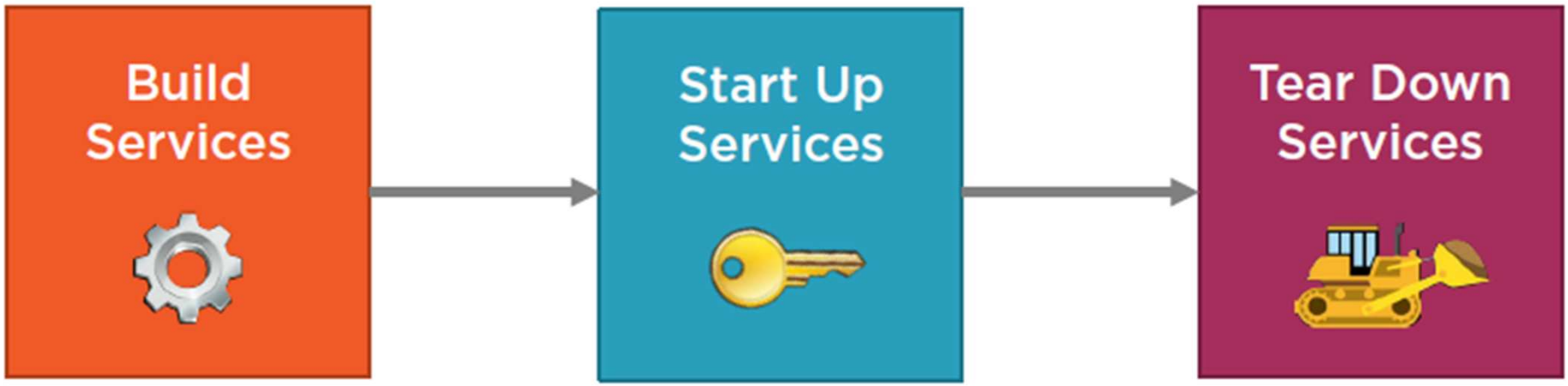
Docker Compose

- Manages the whole application lifecycle:
  - Start, stop and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service

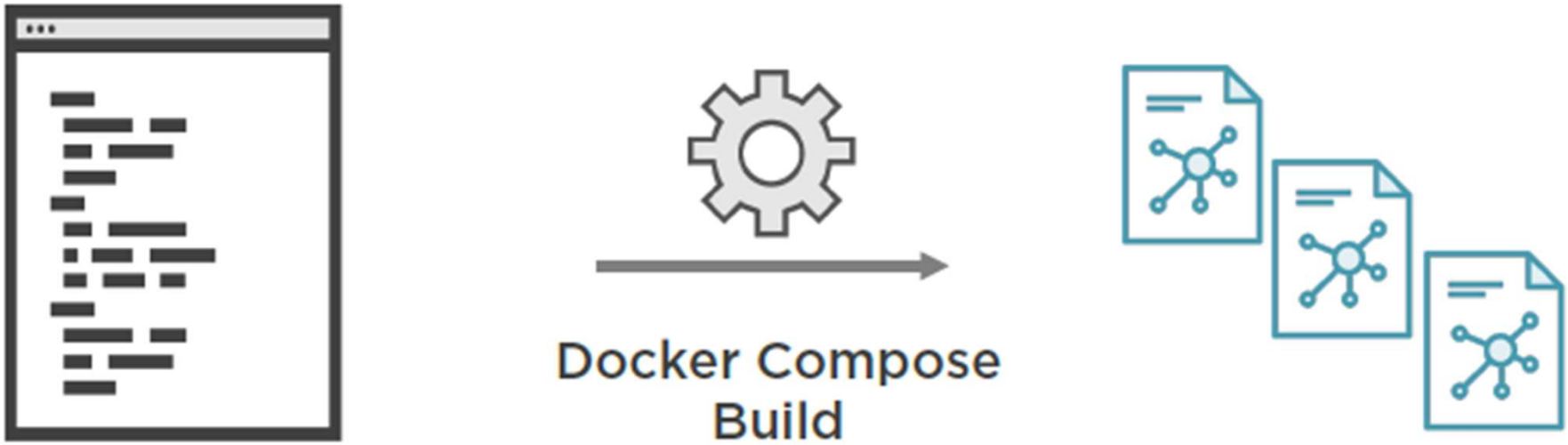
The need for Docker Compose



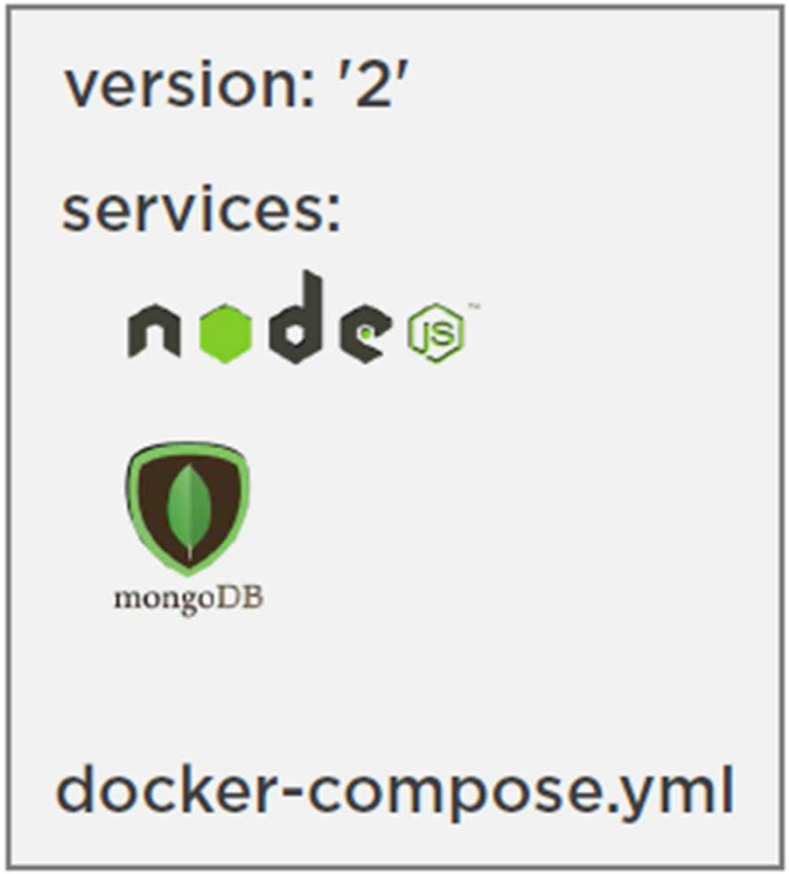
Docker Compose Workflow



The Role of the DockerCompose File



Docker Compose and Services



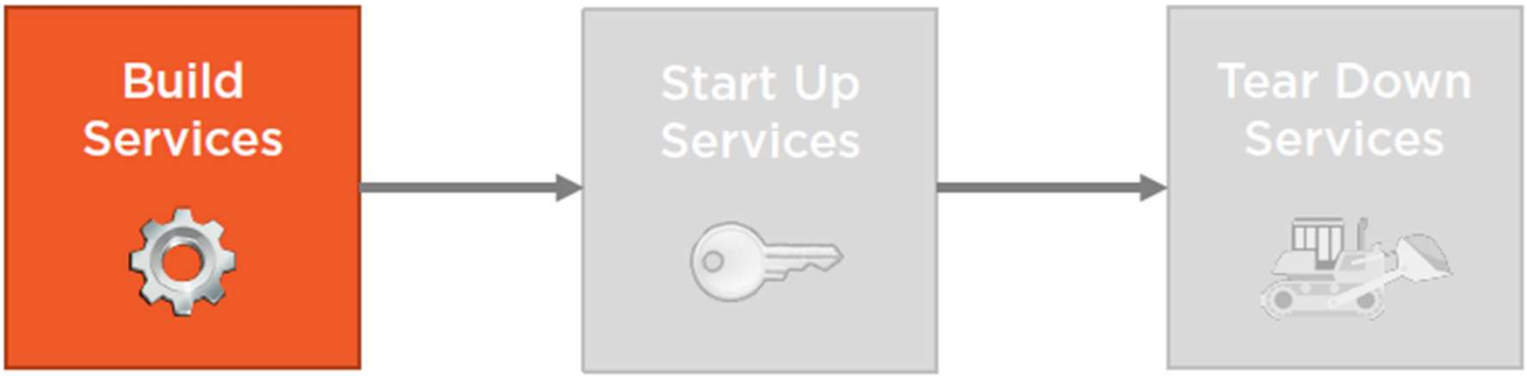
docker-compose.yml Example

- version: '2‘
- services:
  - node:
    - build:
      - context: .
      - dockerfile: node.dockerfile
    - networks:
      - -nodeapp-network
  - mongodb:
    - image: mongo
    - networks:
      - -nodeapp-network
- networks:
  - nodeapp-network
    - driver: bridge

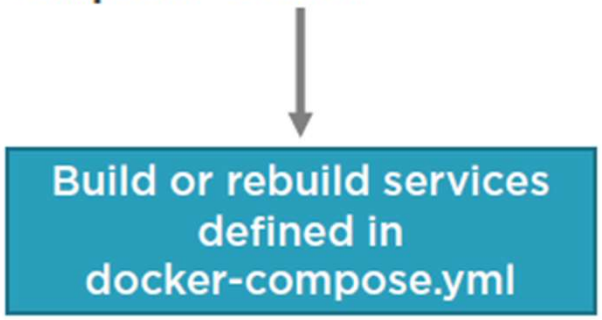
Key Docker Compose Commands

- docker-compose build
- docker-compose up
- docker-compose down
- docker-compose logs
- docker-compose ps
- docker-compose stop
- docker-compose start
- docker-compose rm

Building Services



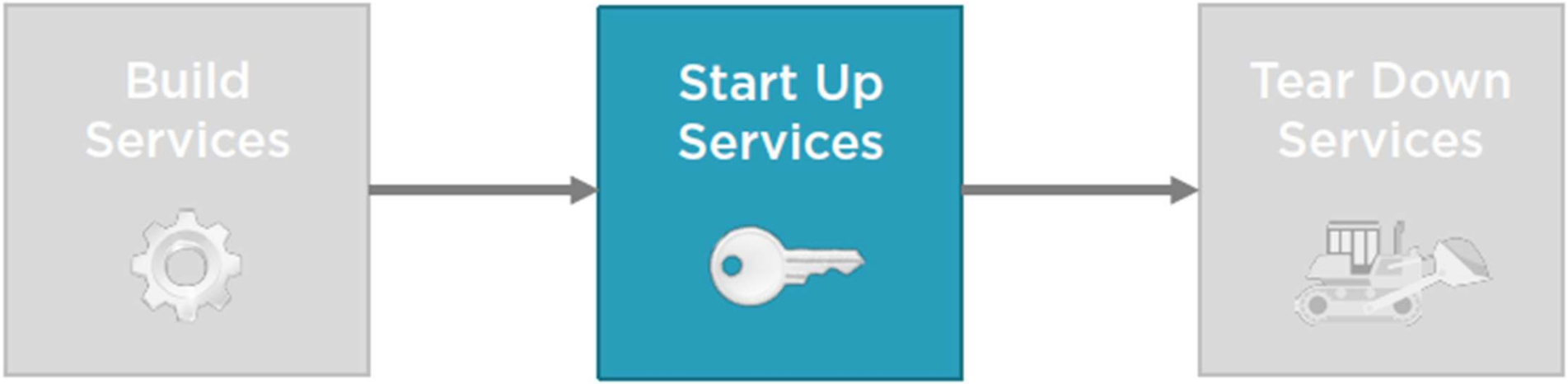

```
docker-compose build
```



Building Specific Services

```
docker-compose build mongo
```

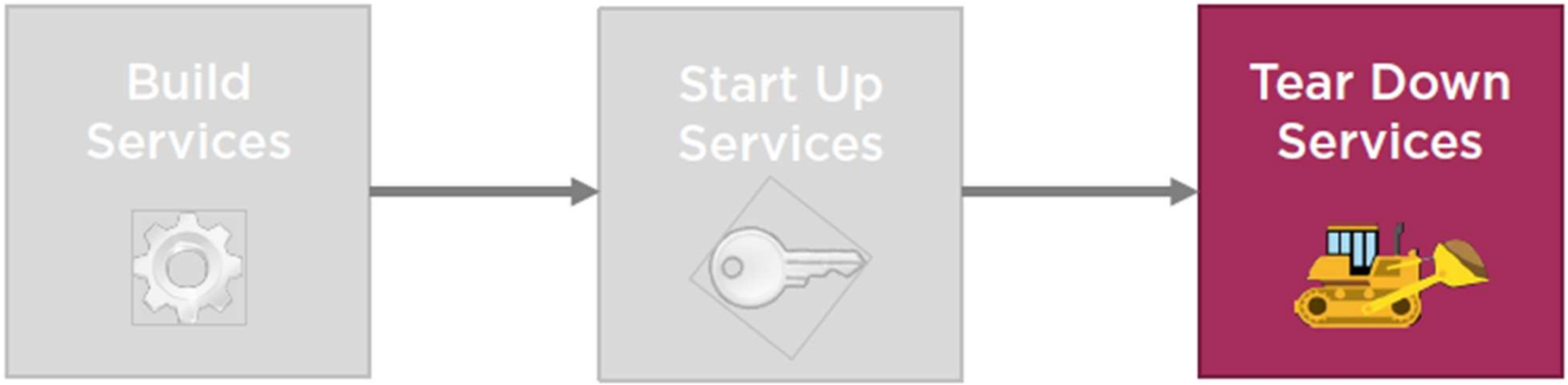
Starting Services Up




```
docker-compose up
```

Create and start the containers

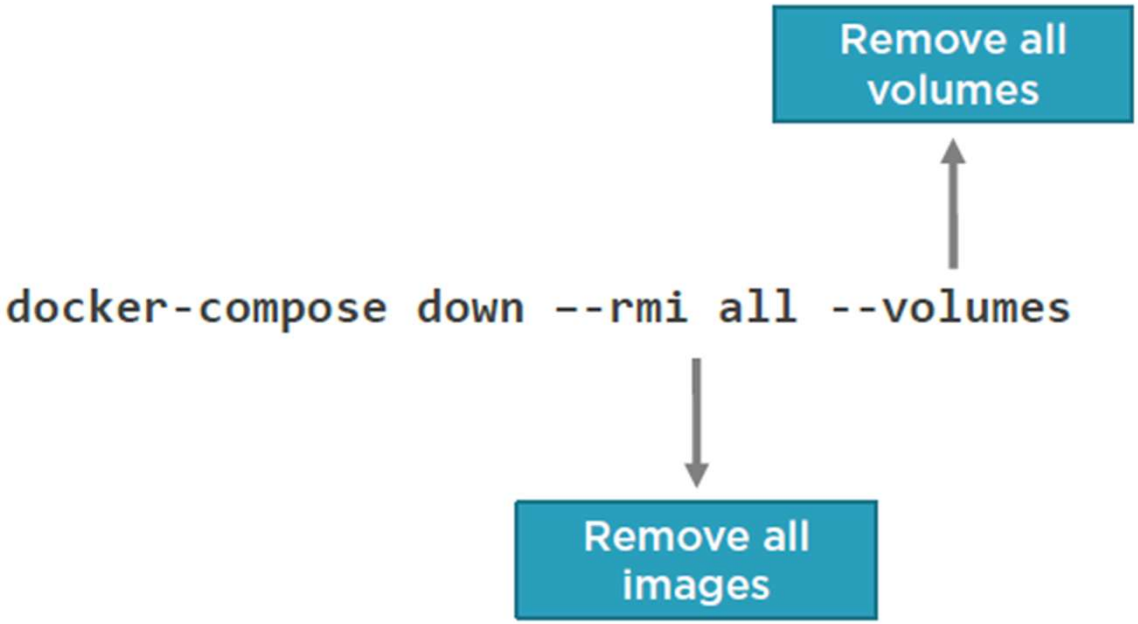
Tearing Down Services




```
docker-compose down
```

Take all of the containers down (stop and remove)

Stop and Remove Containers, Images, Volumes



Docker Use Cases

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment
  - dev -> test -> prod (local, VM, cloud, ...)