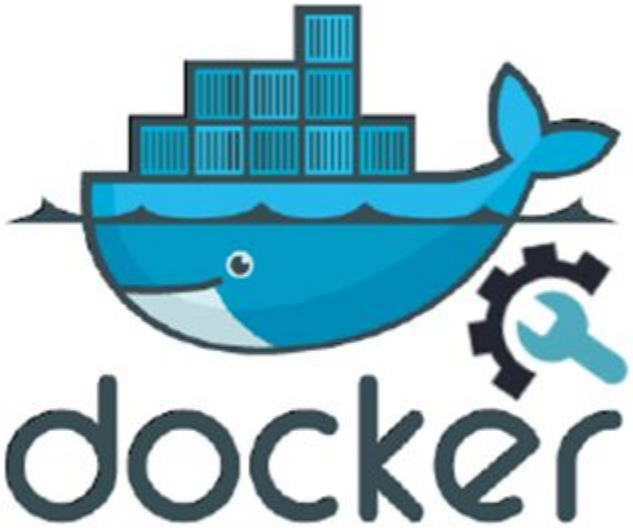
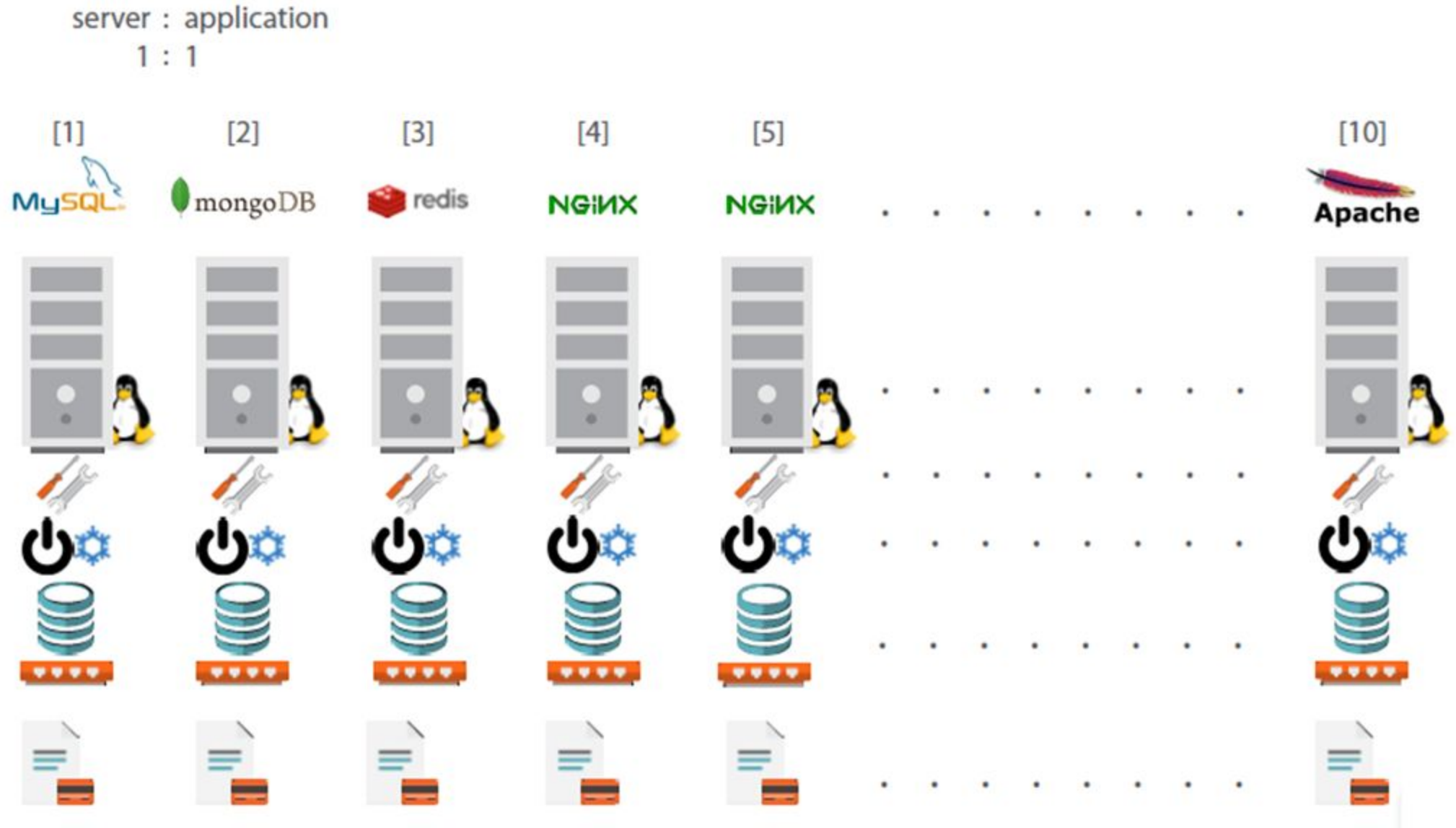
Docker – Day 1

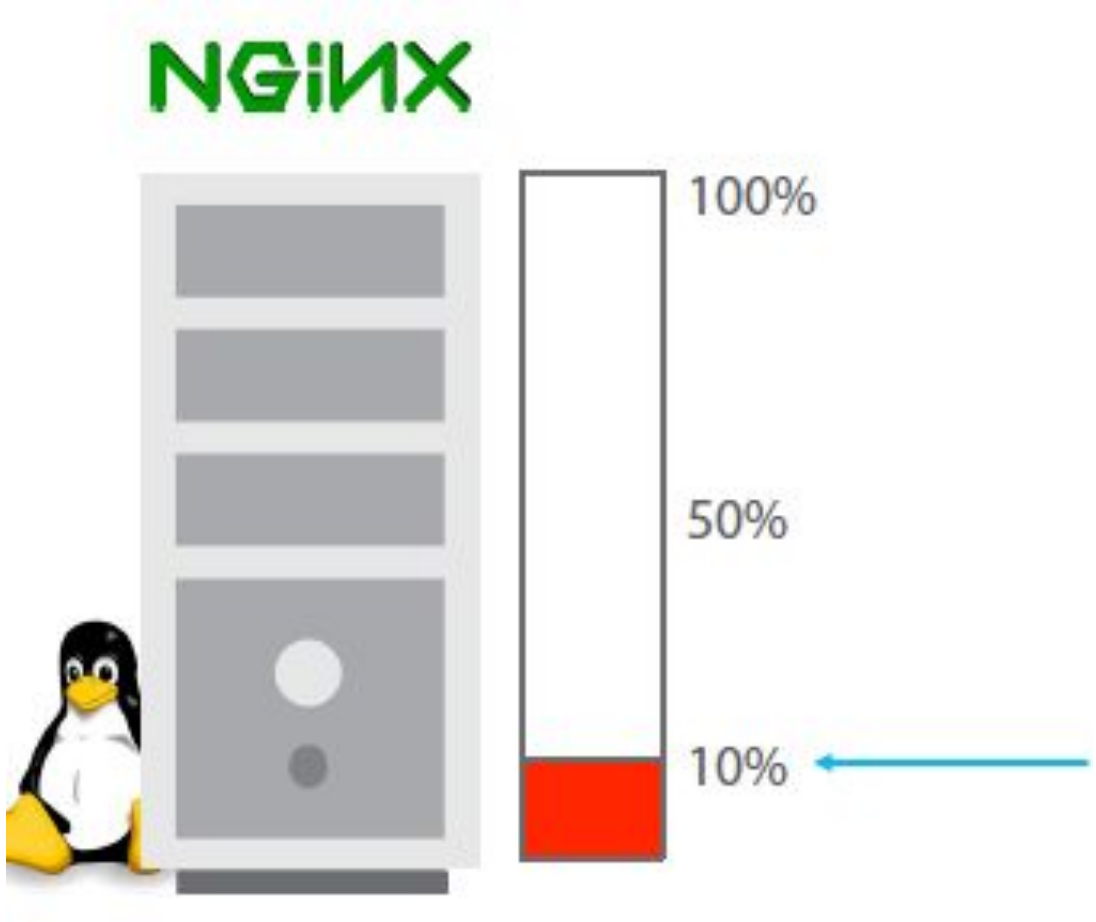
Docker



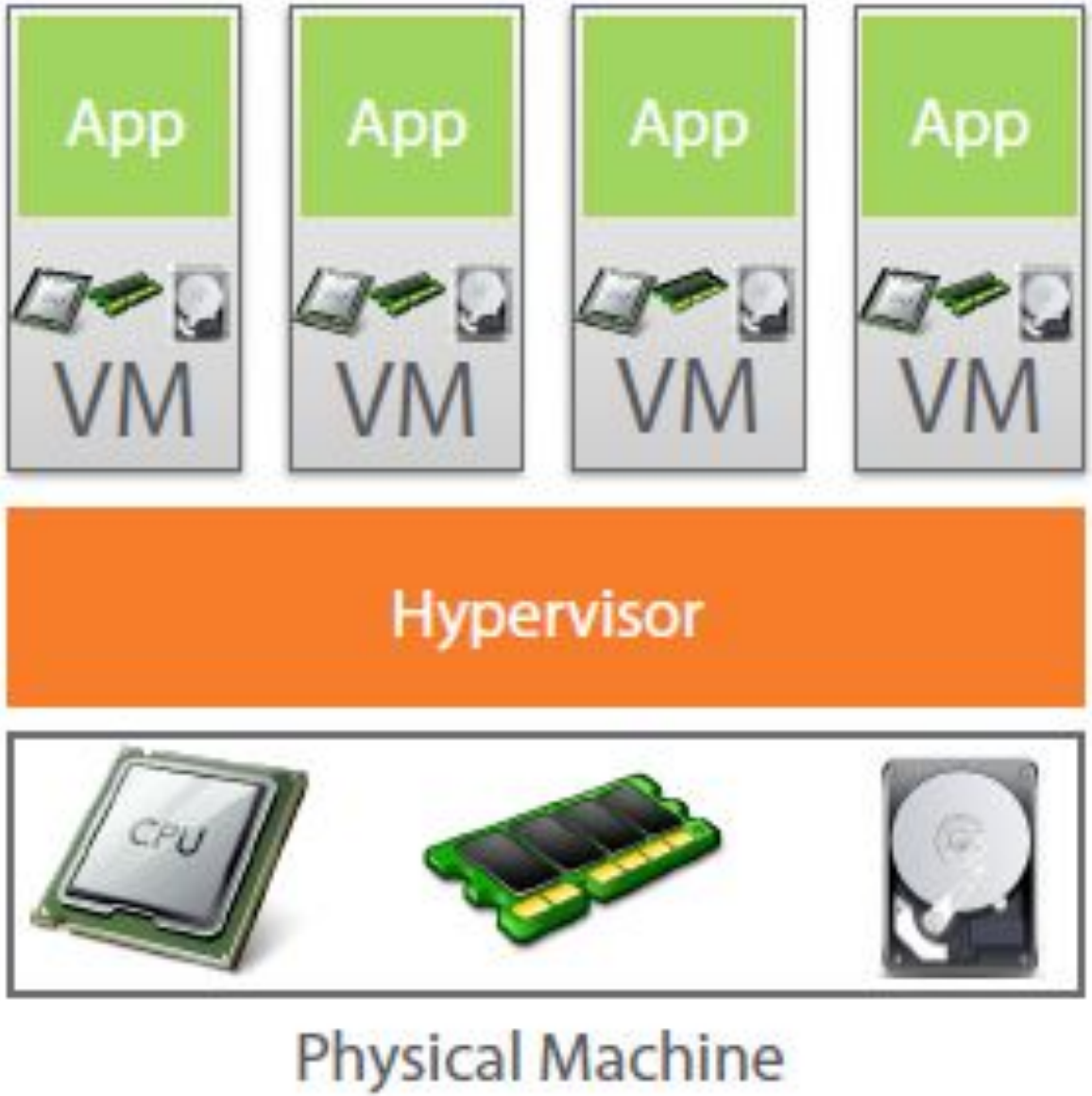
Traditional Deployment Architecture



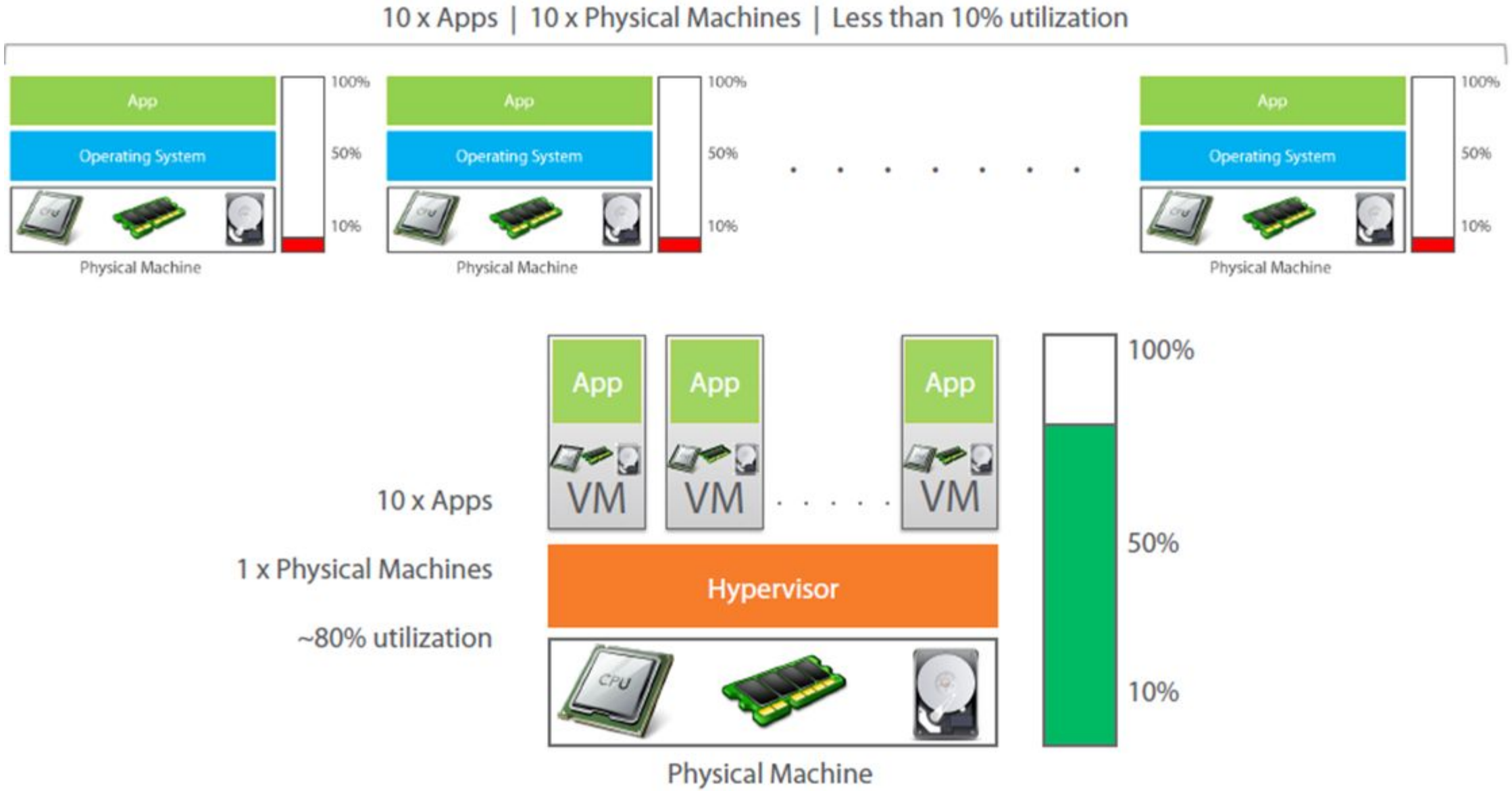
Less Utilization in Traditional Architecture



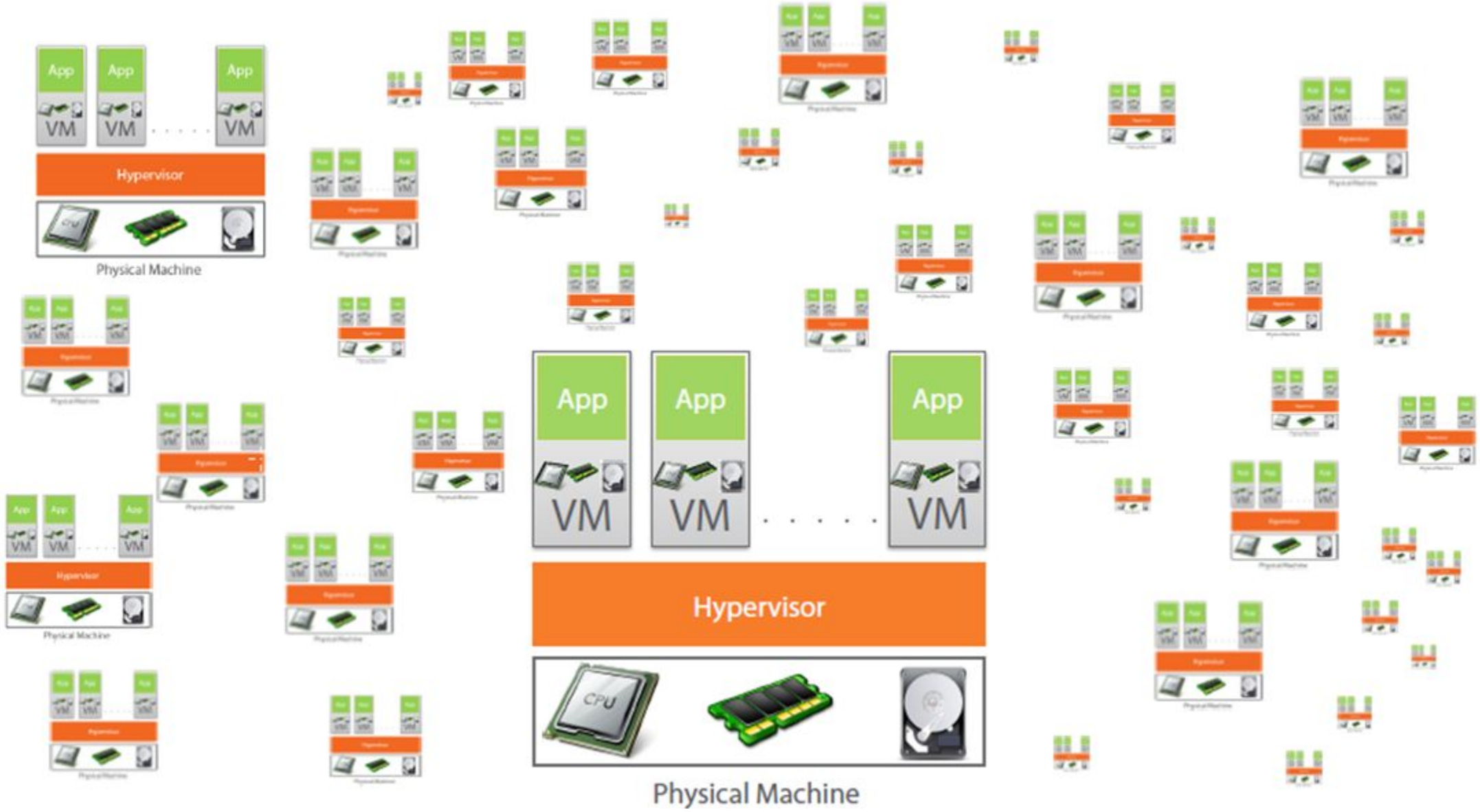
Virtual Machine to the Rescue



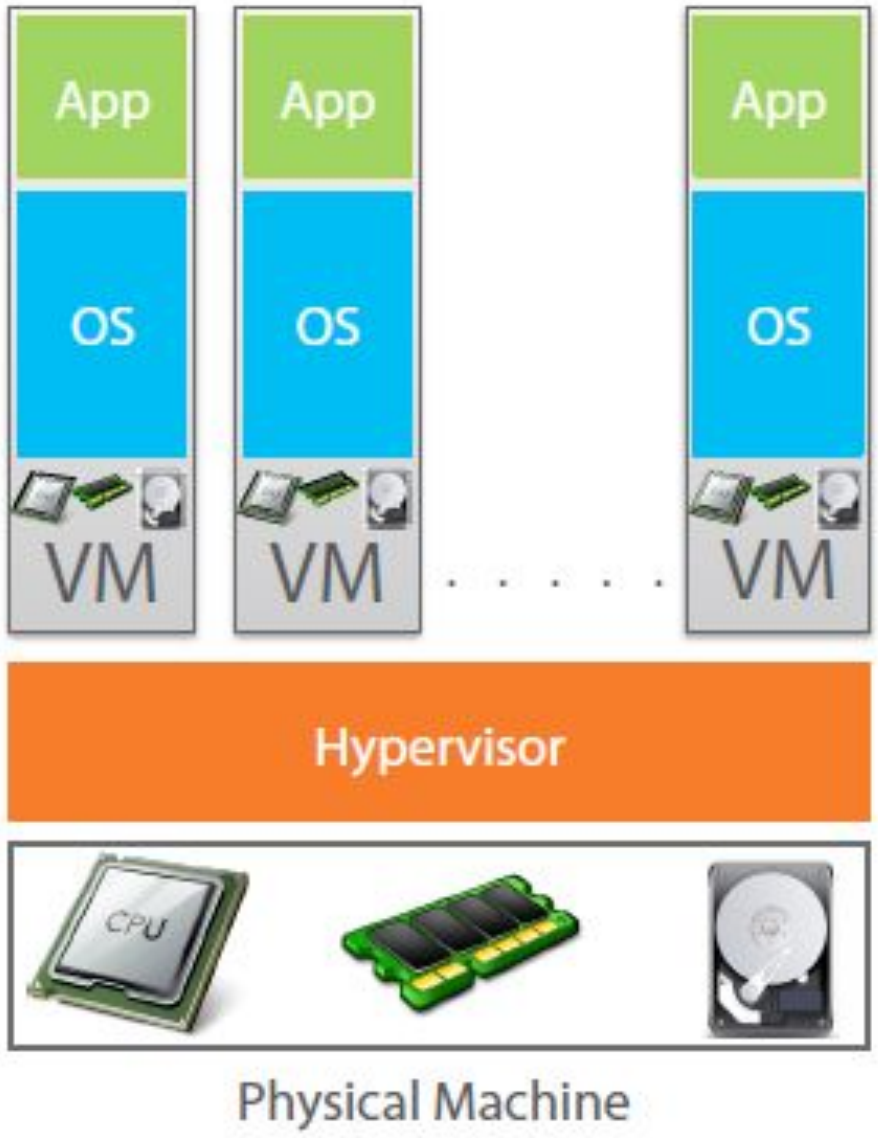
Virtual Machine provides better utilization



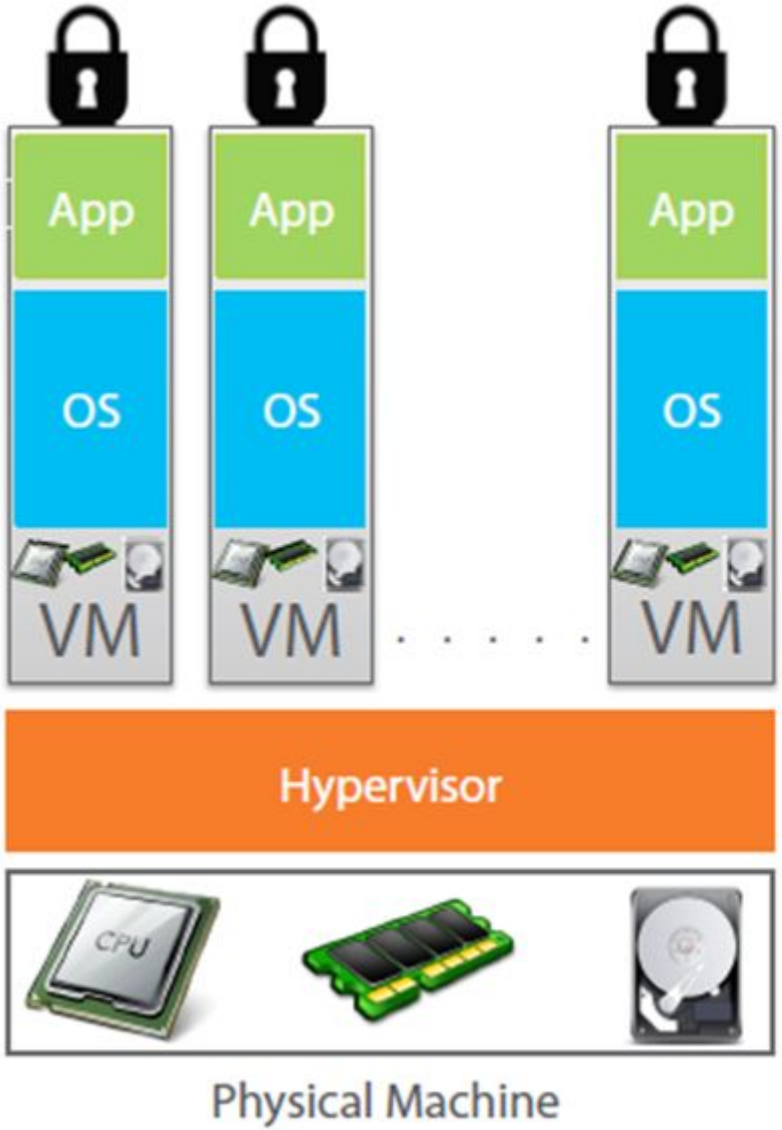
But Virtual Machine increases Licensing Cost



Each VM needs a separate OS

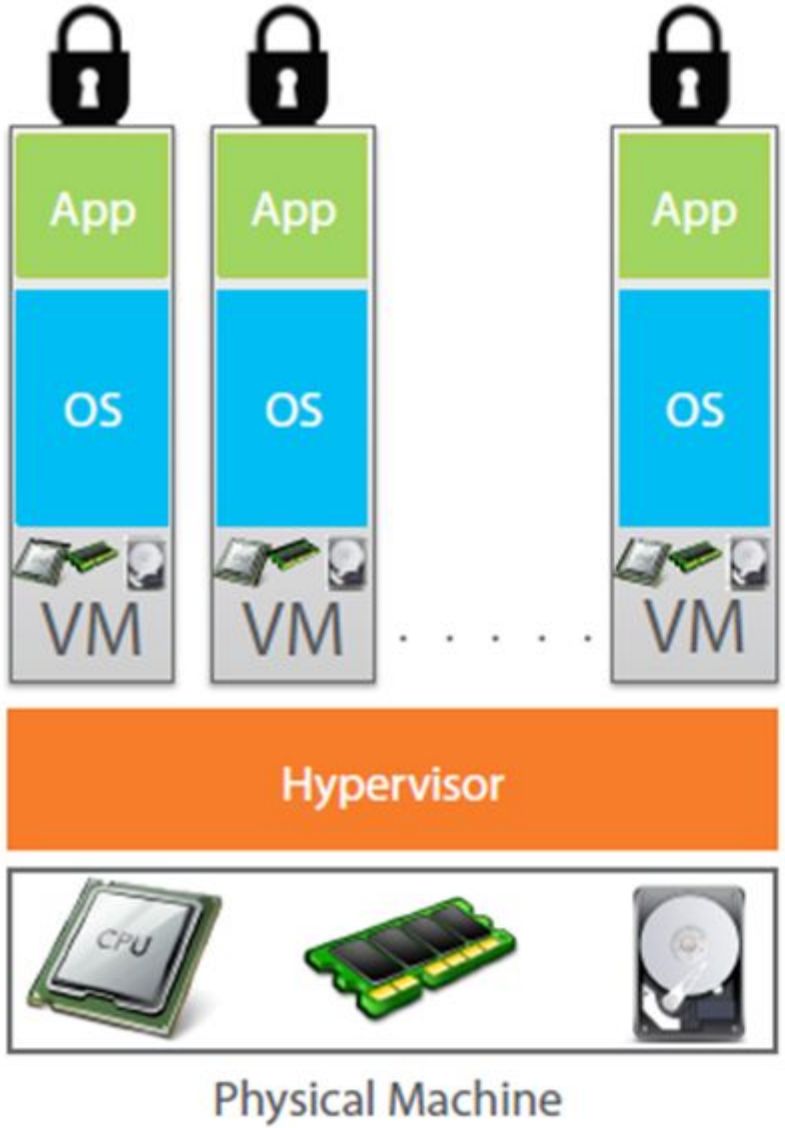


More OSes doesn’t increase Business Value



> OS != Business Value

OS takes most of the Resources

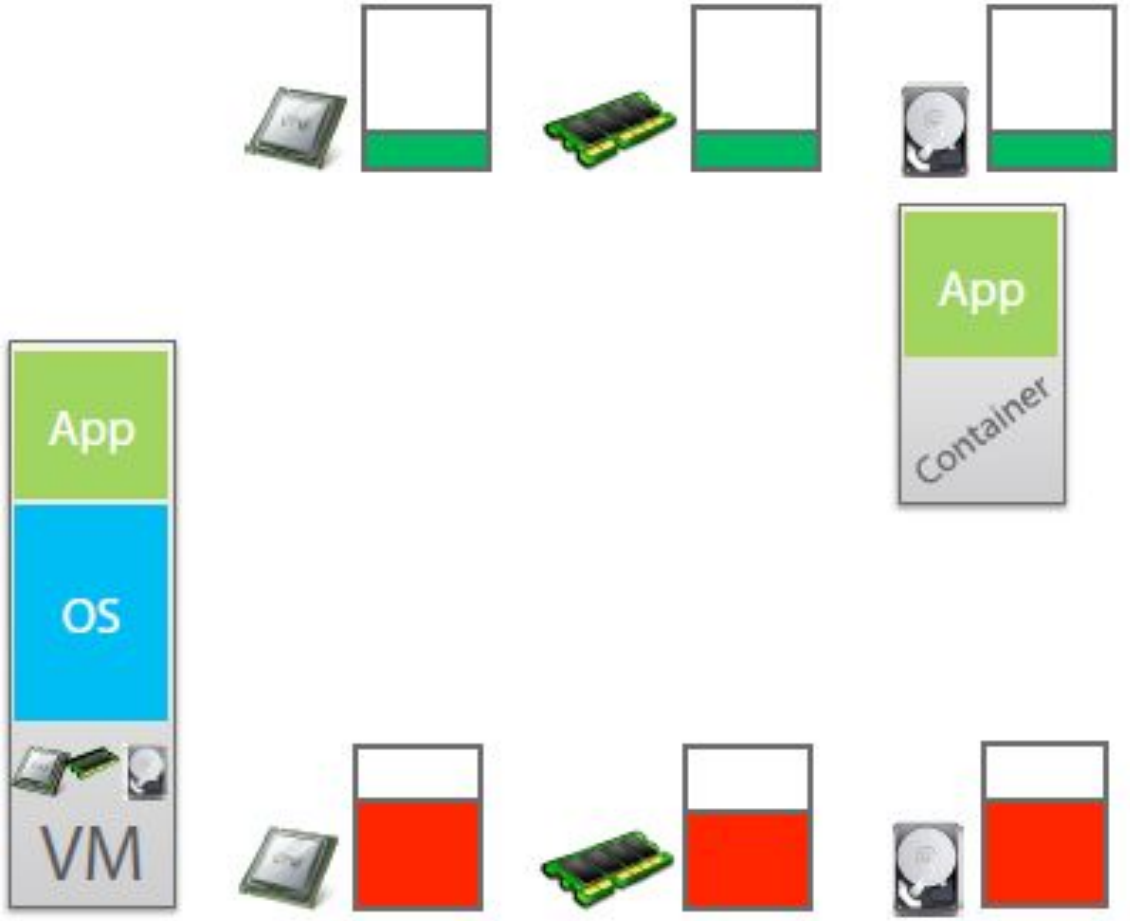


Why use separate OS for each App?

Containerization

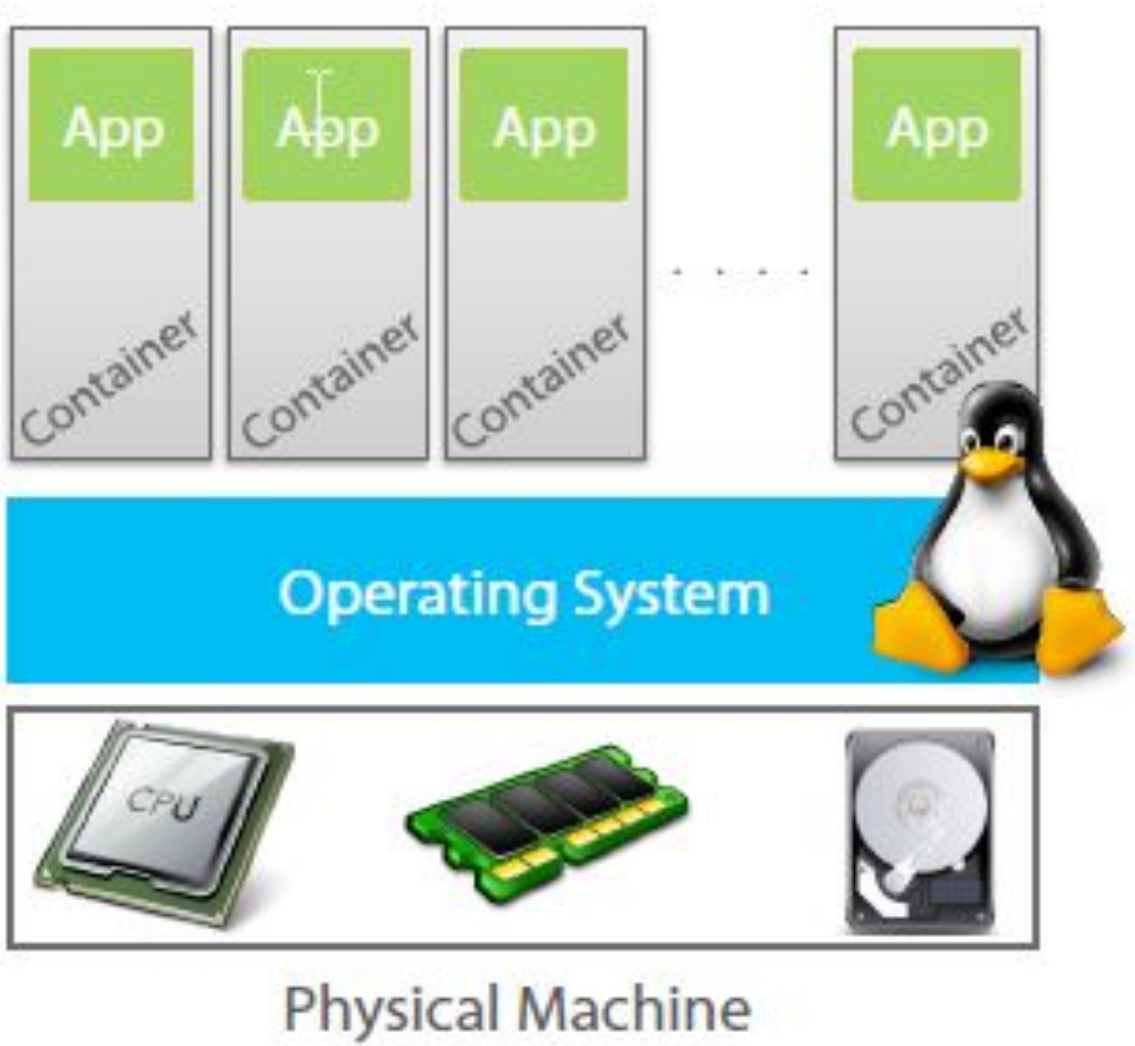
- Encapsulation of an application and its required environment.
- The process of packaging an application along with its required libraries, frameworks, and configuration files together so that it can be run in various computing environments efficiently.

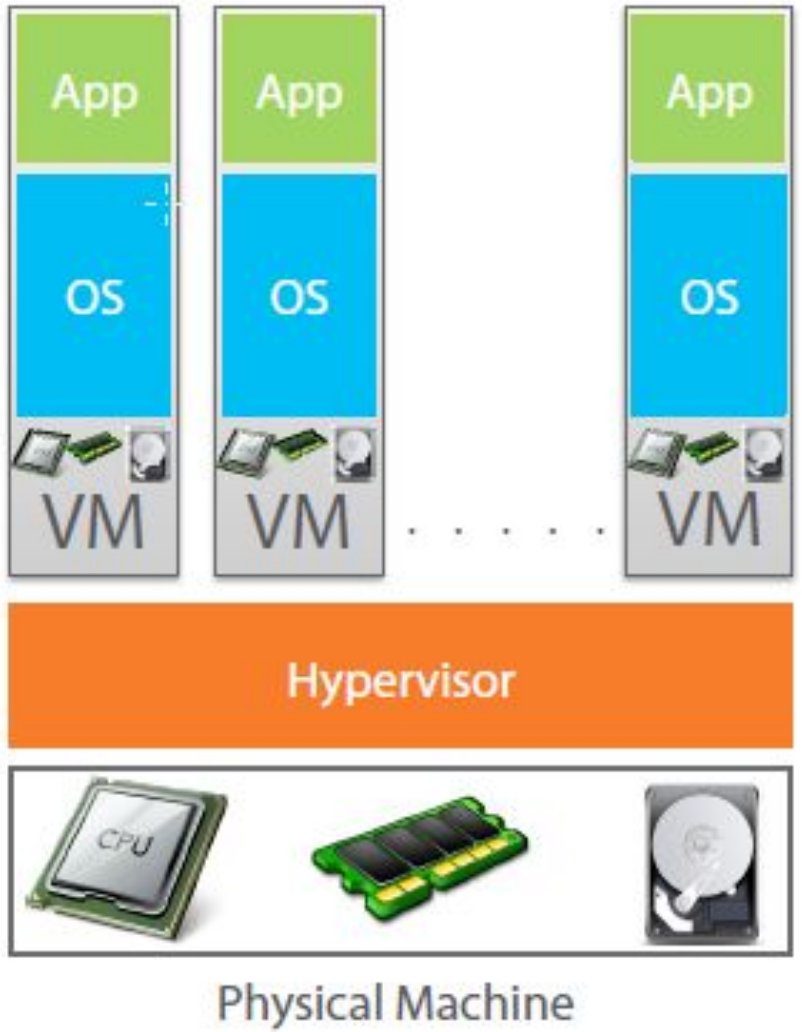
Containers to the Rescue



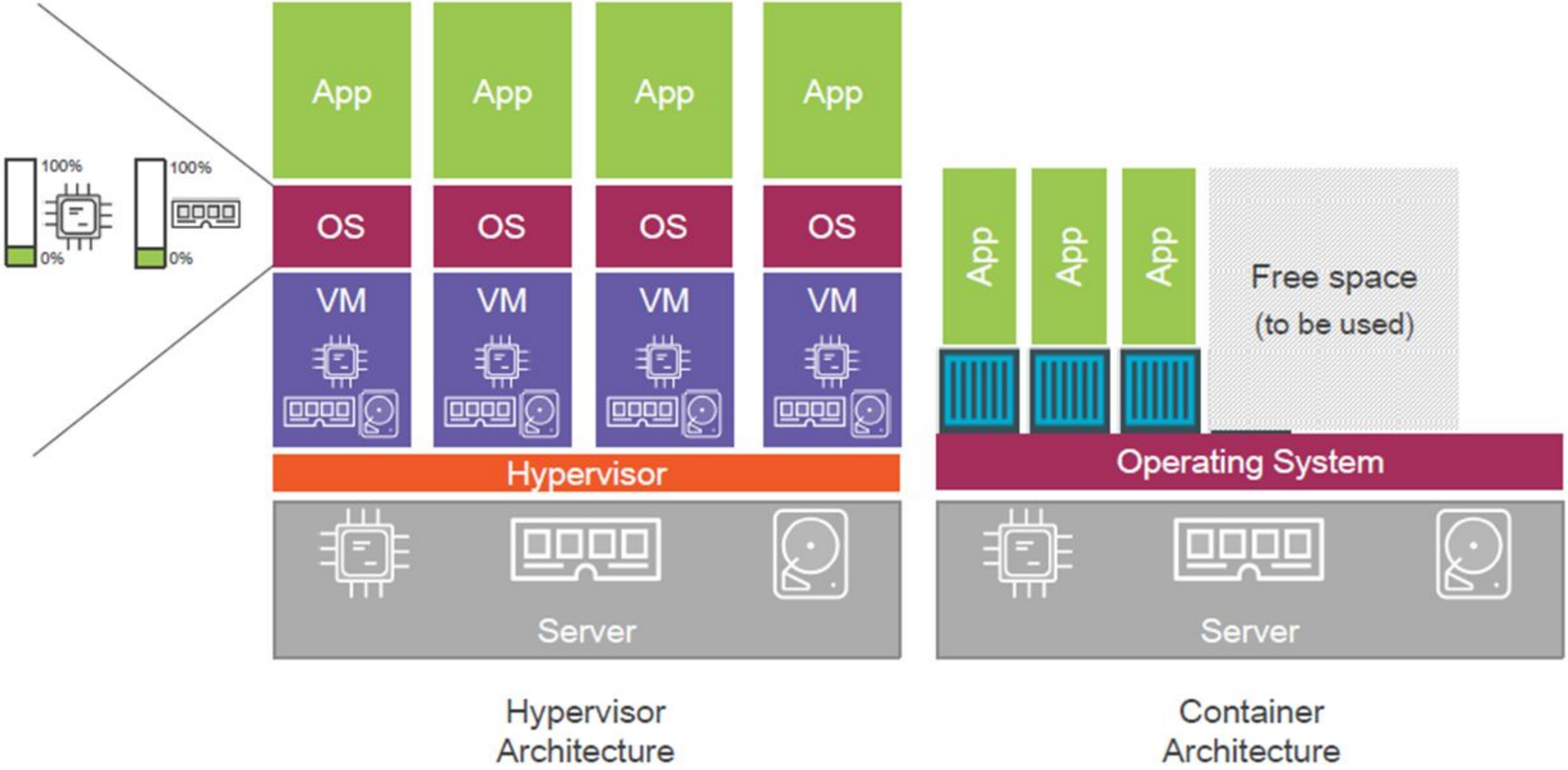
Containers are more lightweight than Virtual Machines

Containers vs VM

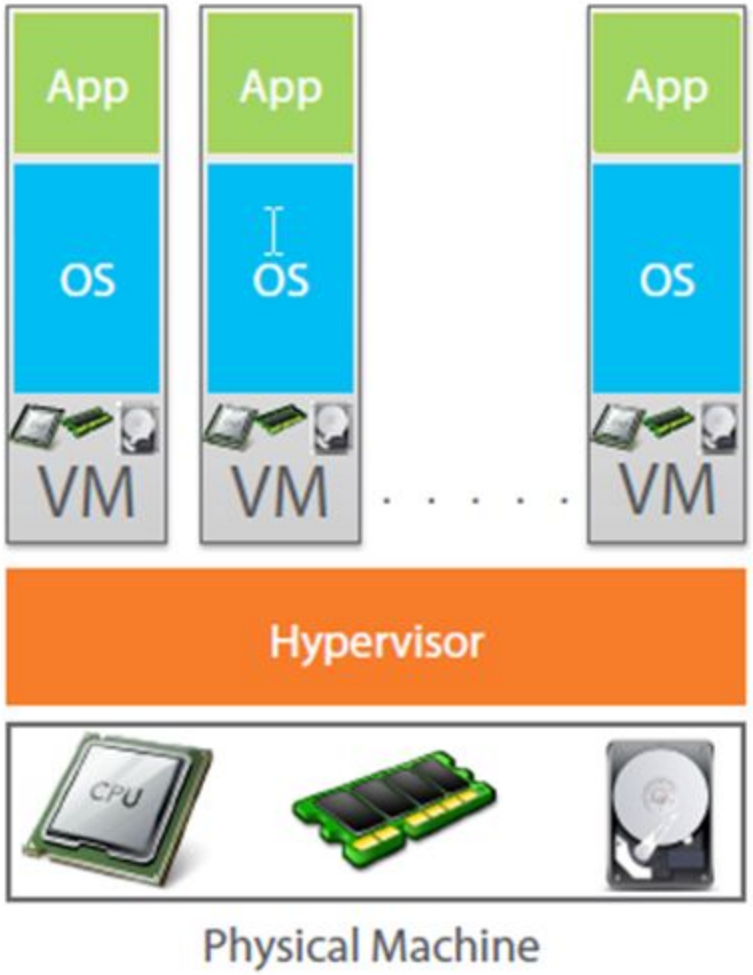


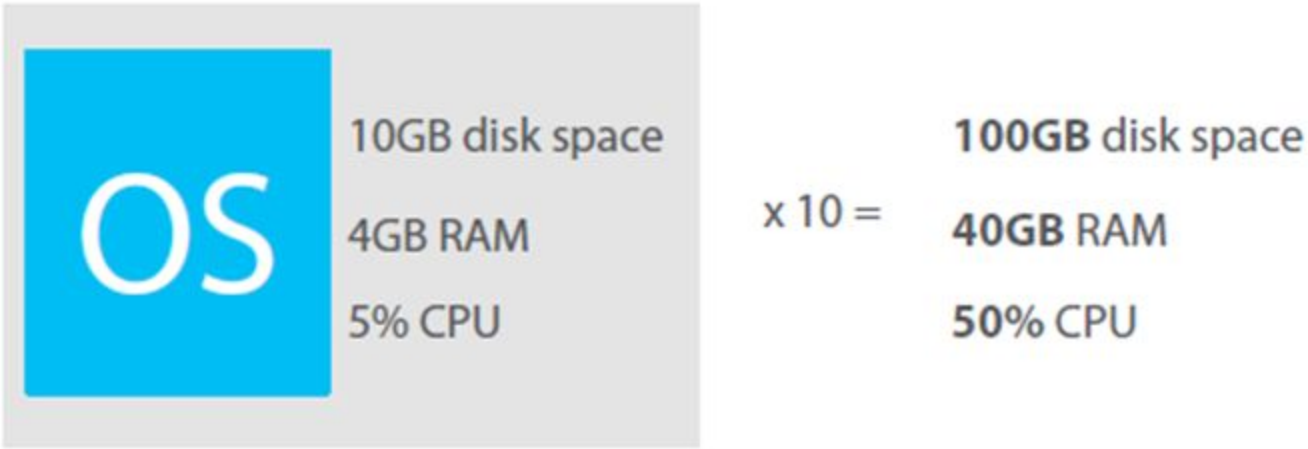


Containers vs VM

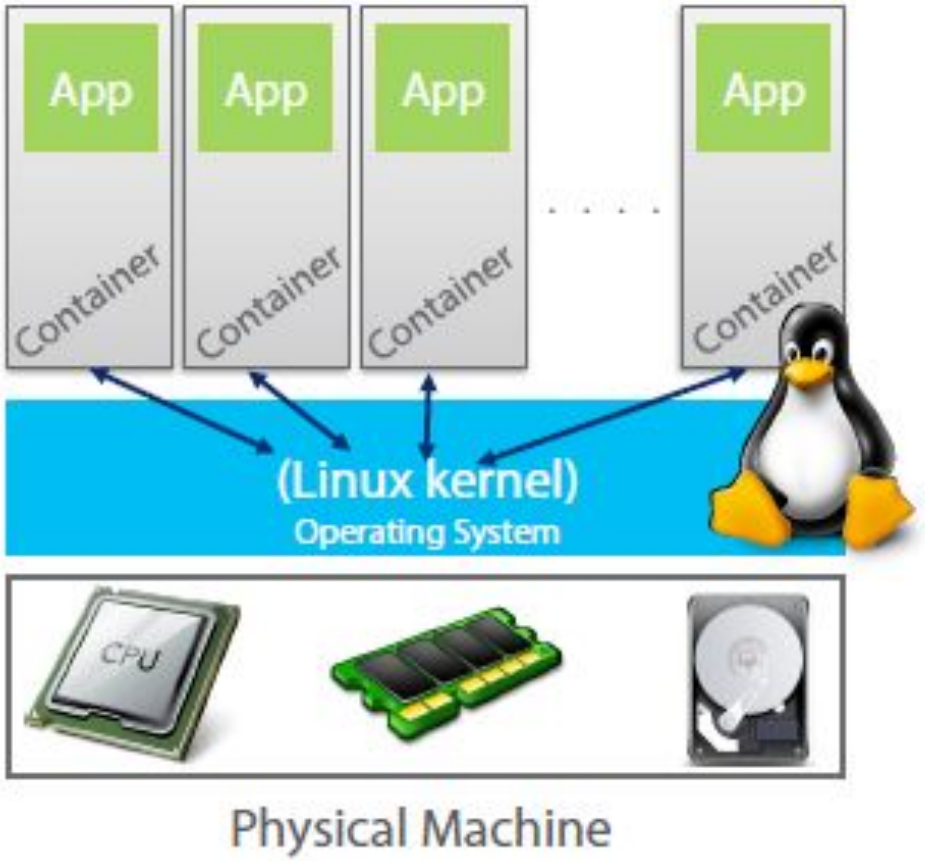


OS takes more resources and Licensing cost



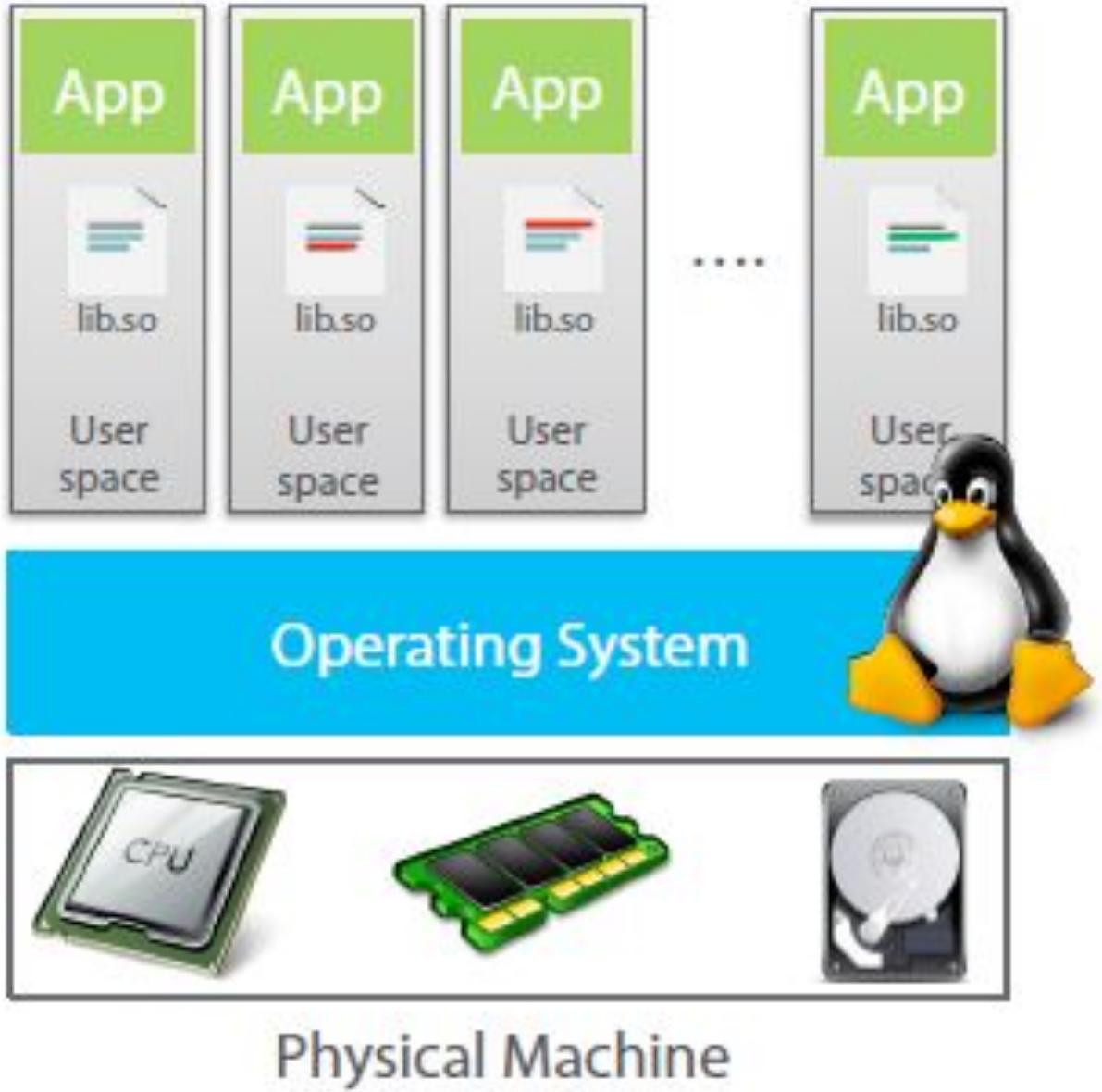


Containers takes less resources

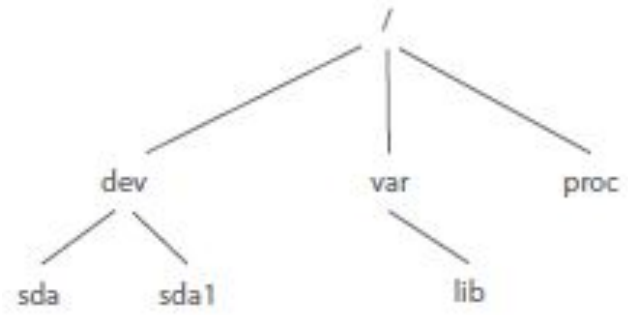


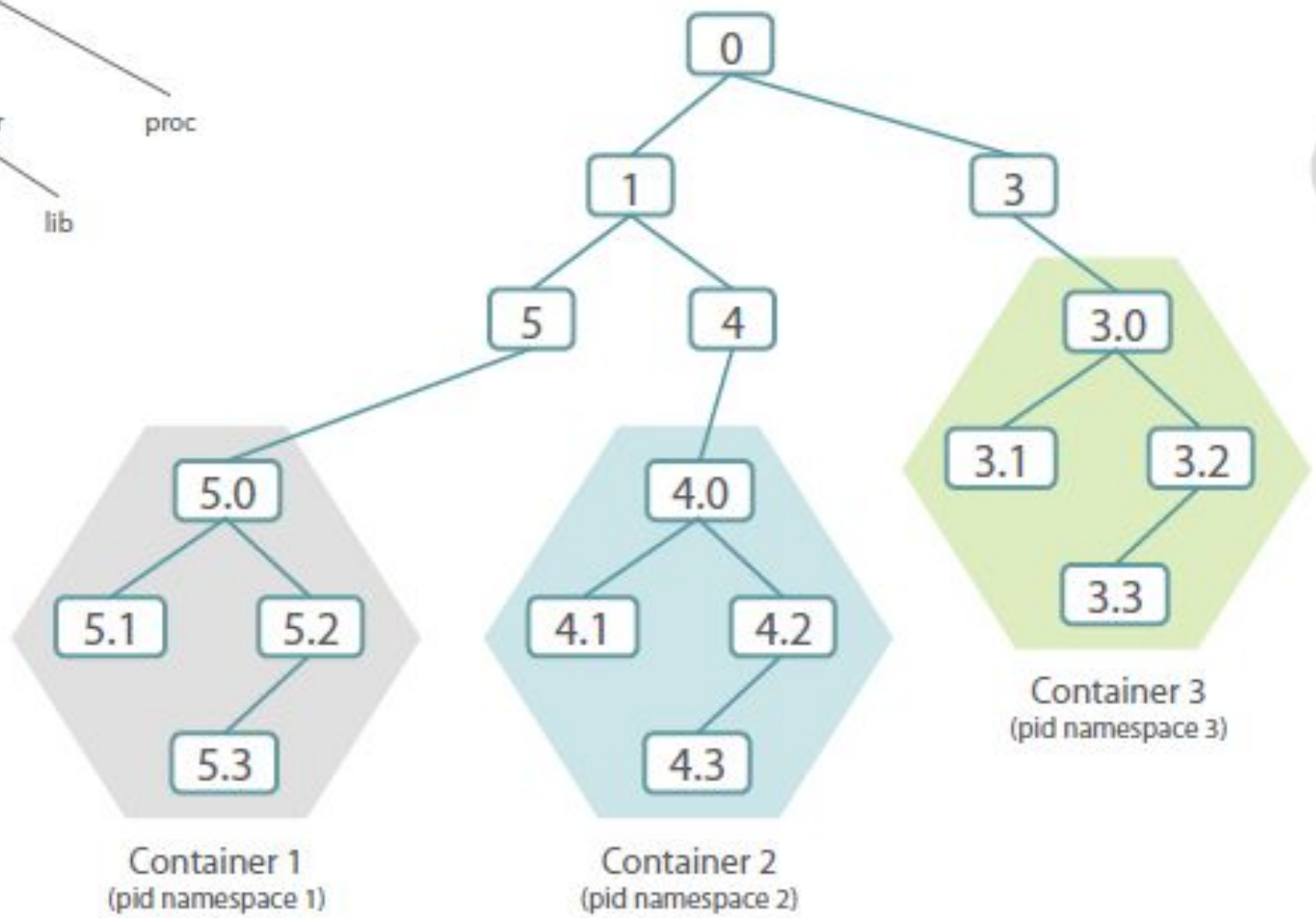
Containers consume less CPU, RAM and disk resource than Virtual Machines

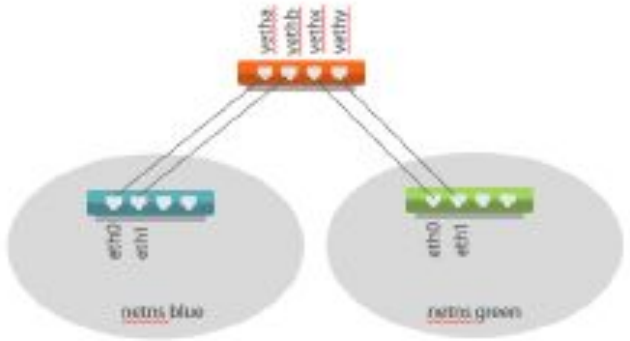
How containers work?



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.

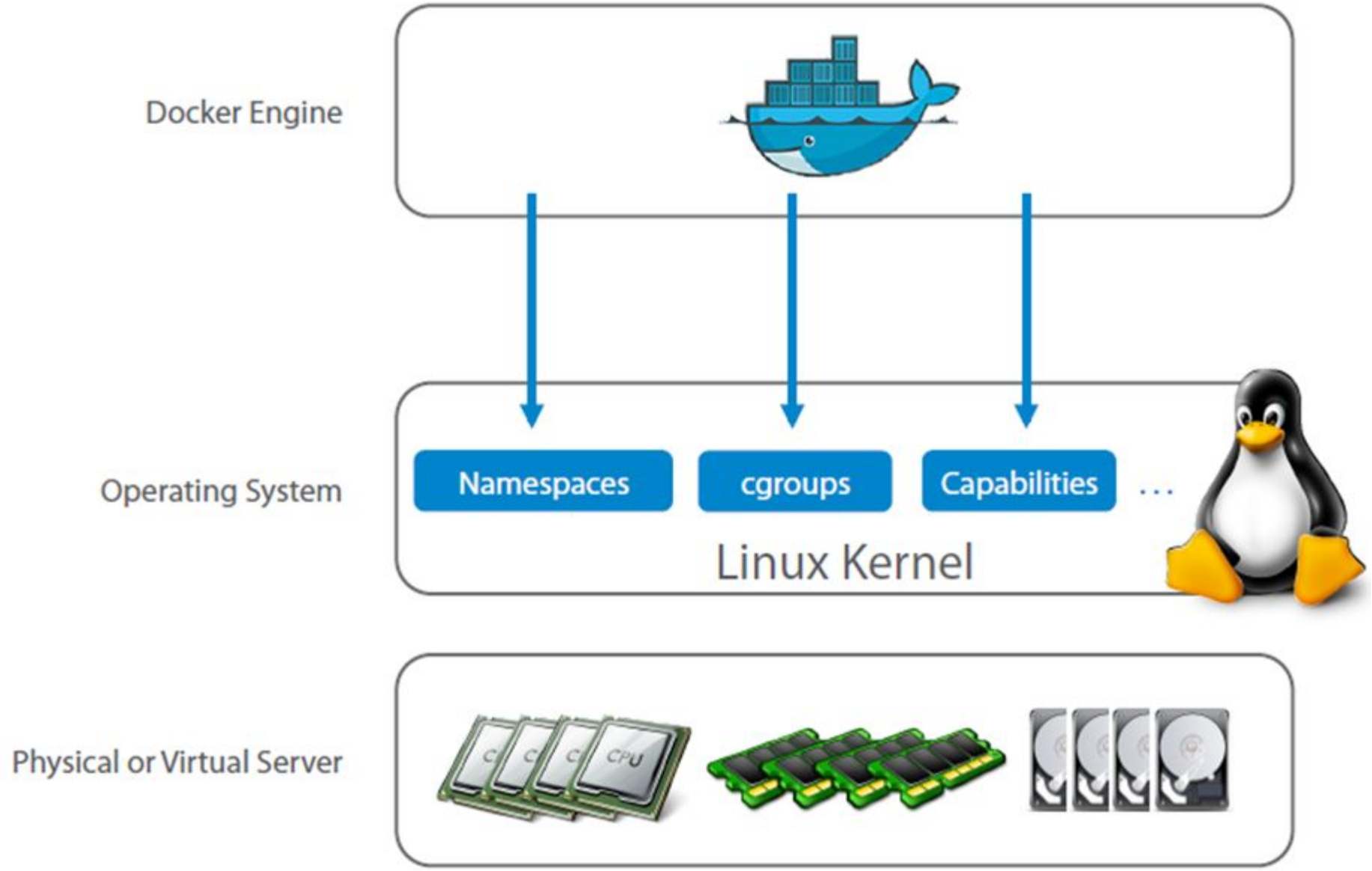
Practical

Practical Guide

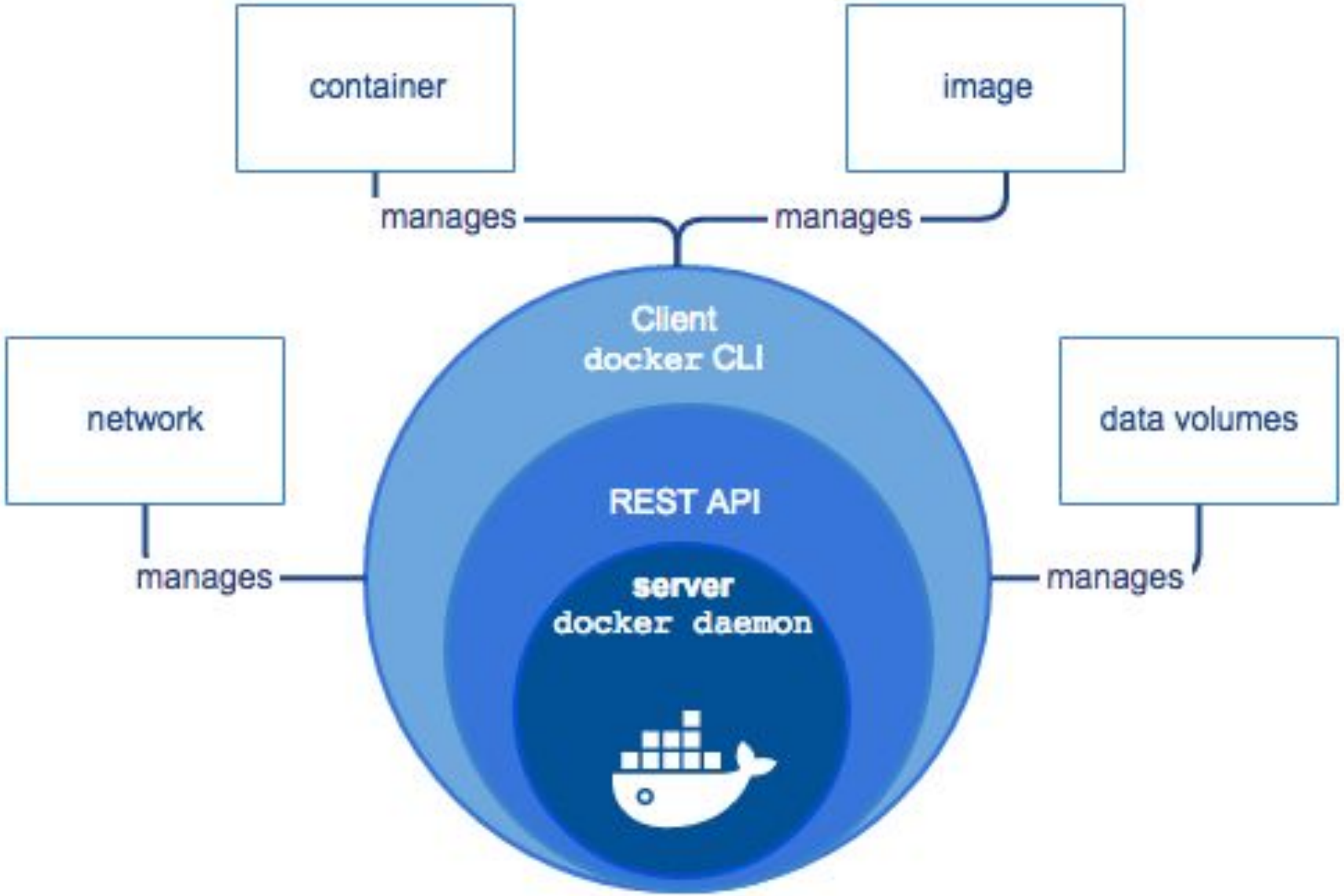
- Docker Installation on Ubuntu:
  - https://docs.google.com/document/d/1RH63K8F4b4Gyjgpkd4NihI1CYlViDM_GjG8tbe9wXs4/edit

- Refer to the Practical Guide on below URL:
  - https://drive.google.com/drive/folders/1WqhY1FoITI2j07N0z3DapaGF-tLVP6VA?usp=sharing

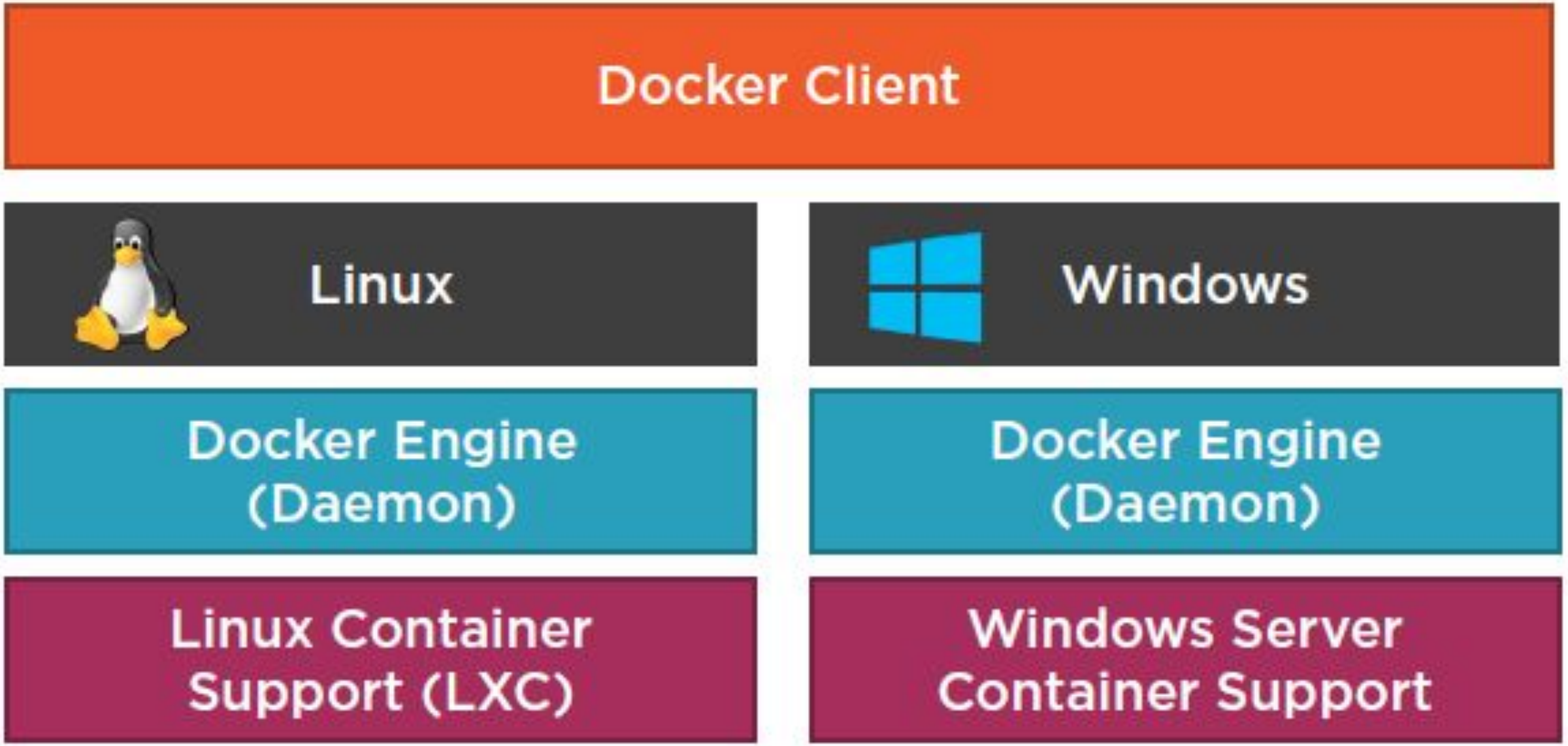
Docker Engine



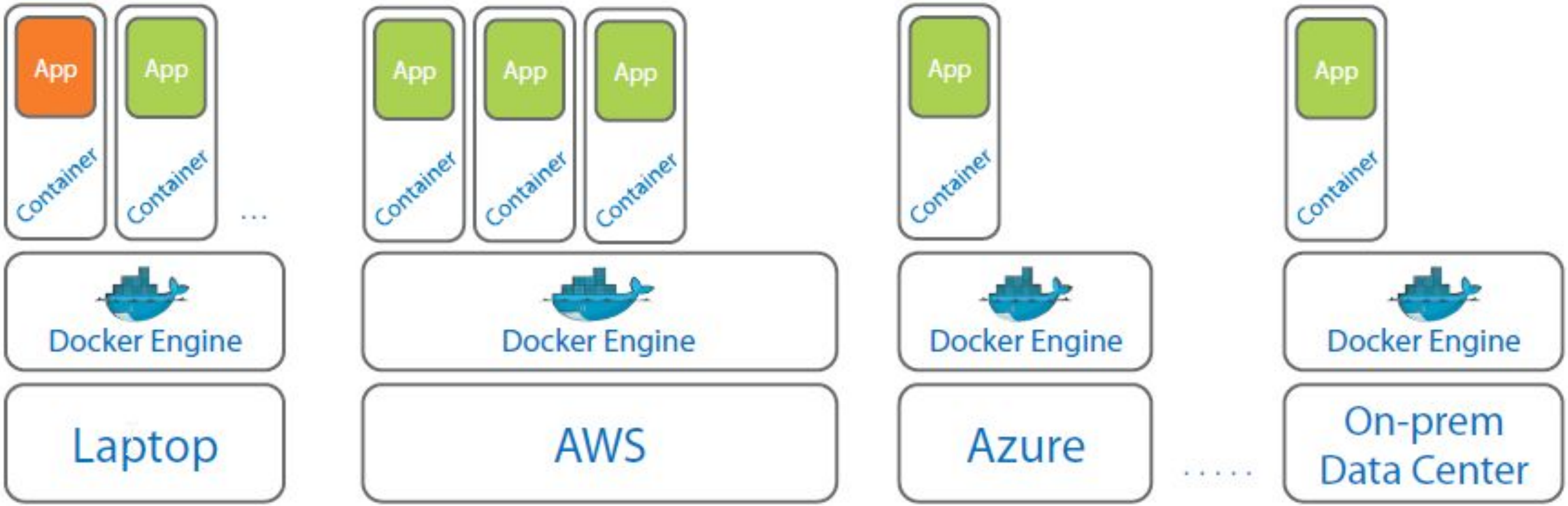
Docker Engine



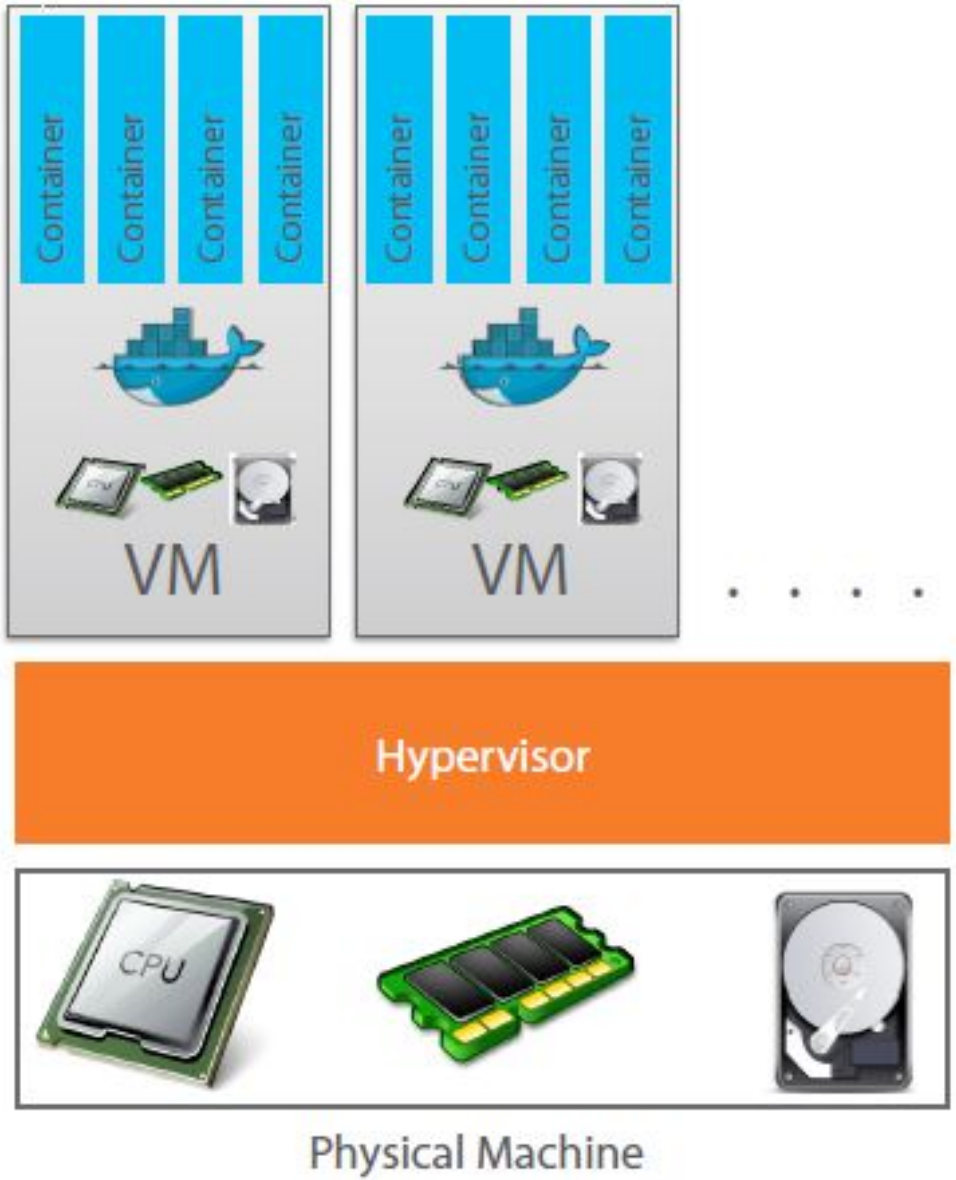
Where does Docker Run?

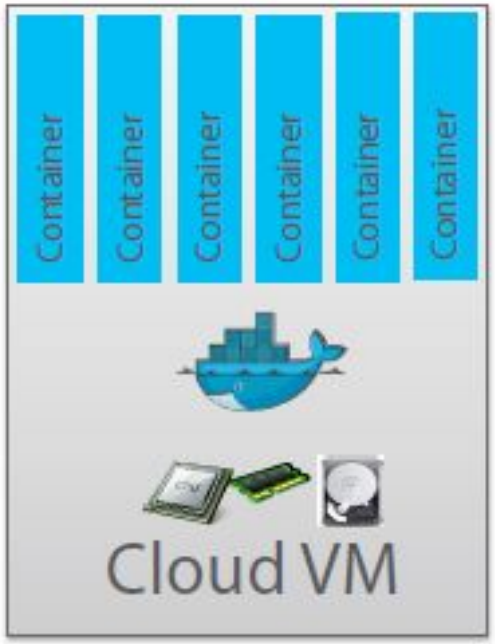


Docker can run anywhere



Docker on Physical Machine and also on Cloud

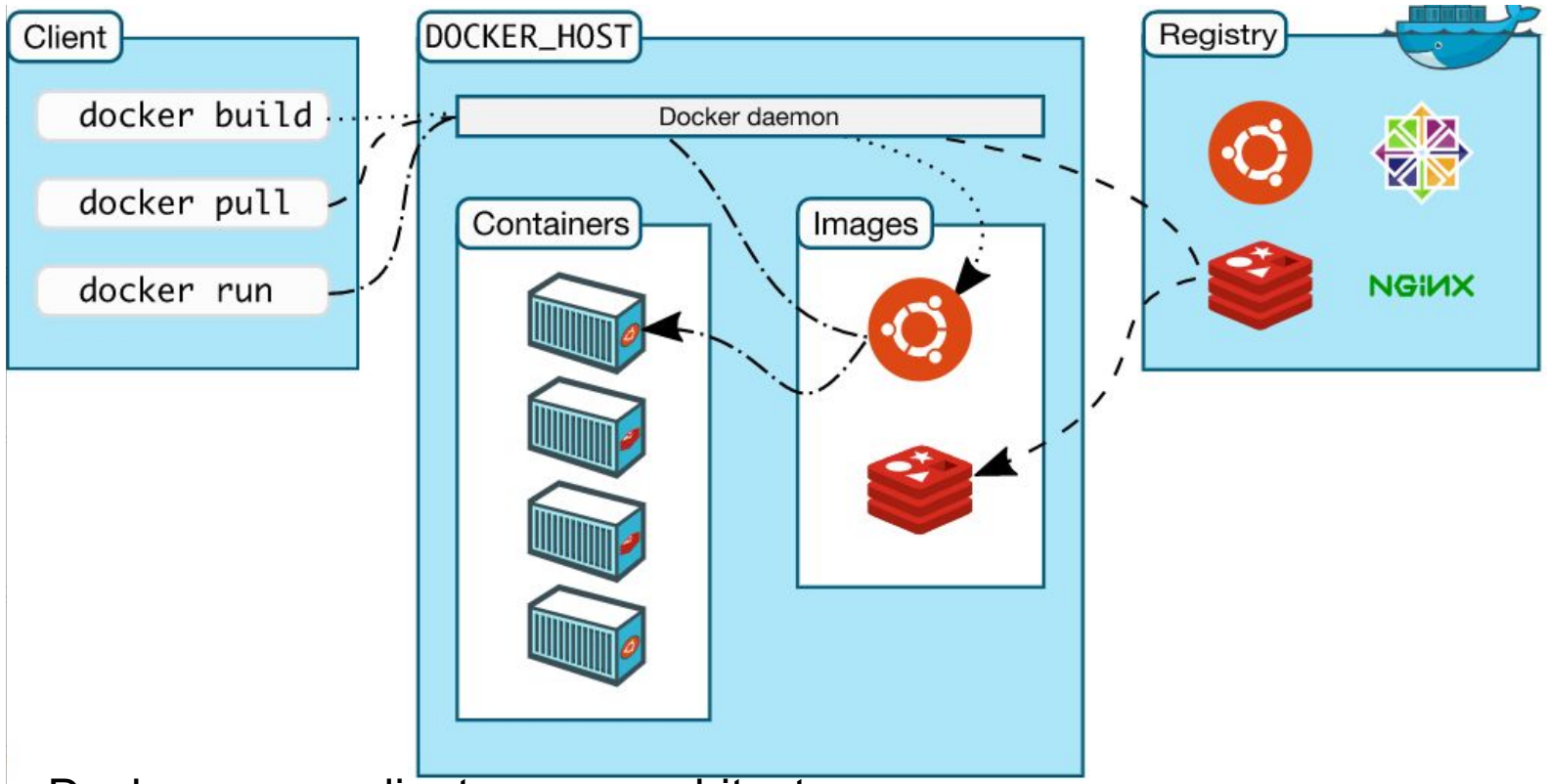




What can I use Docker for?

- 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

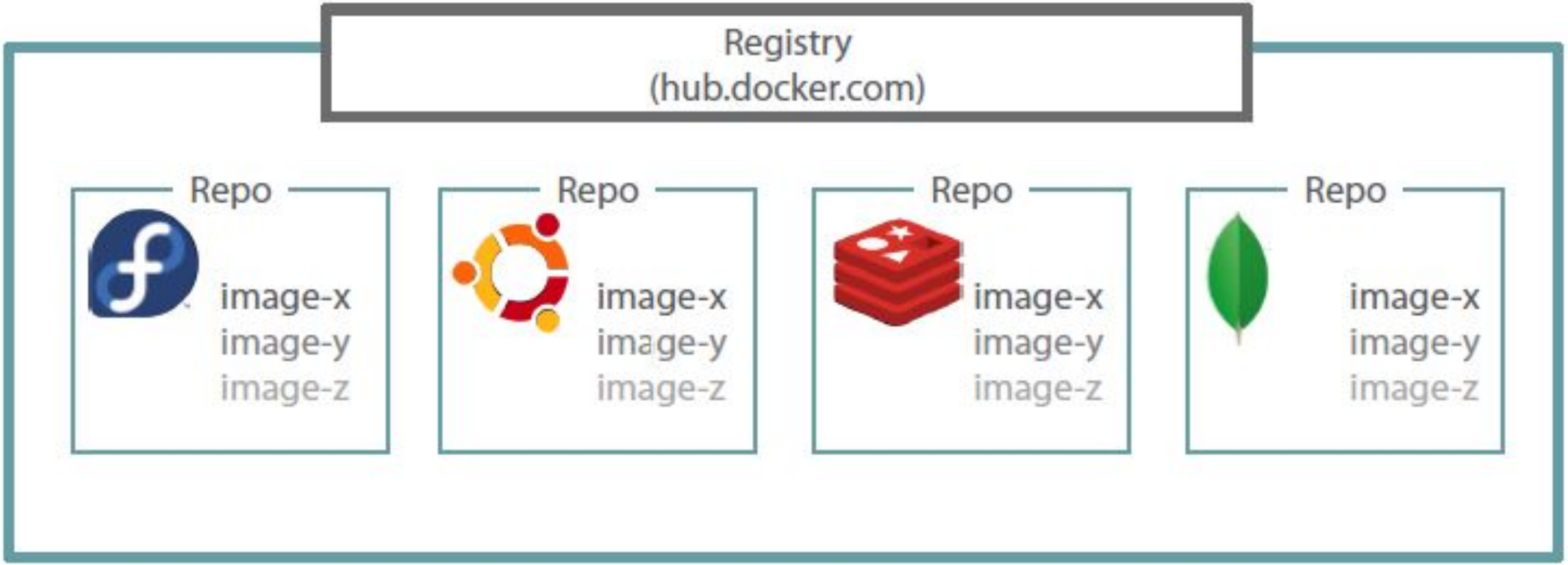
Image

- 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



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:
  - 001-Setup Docker.txt
    - https://docs.google.com/document/d/1RH63K8F4b4Gyjgpkd4NihI1CYlViDM_GjG8tbe9wXs4/edit

Hands-On

- We need to do the below hands-on:
  - ssh to Ubuntu server
  - Install Docker on Ubuntu 18.04
  - 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 “002-Creating and Using Containers.txt” in Commands guide for instructions

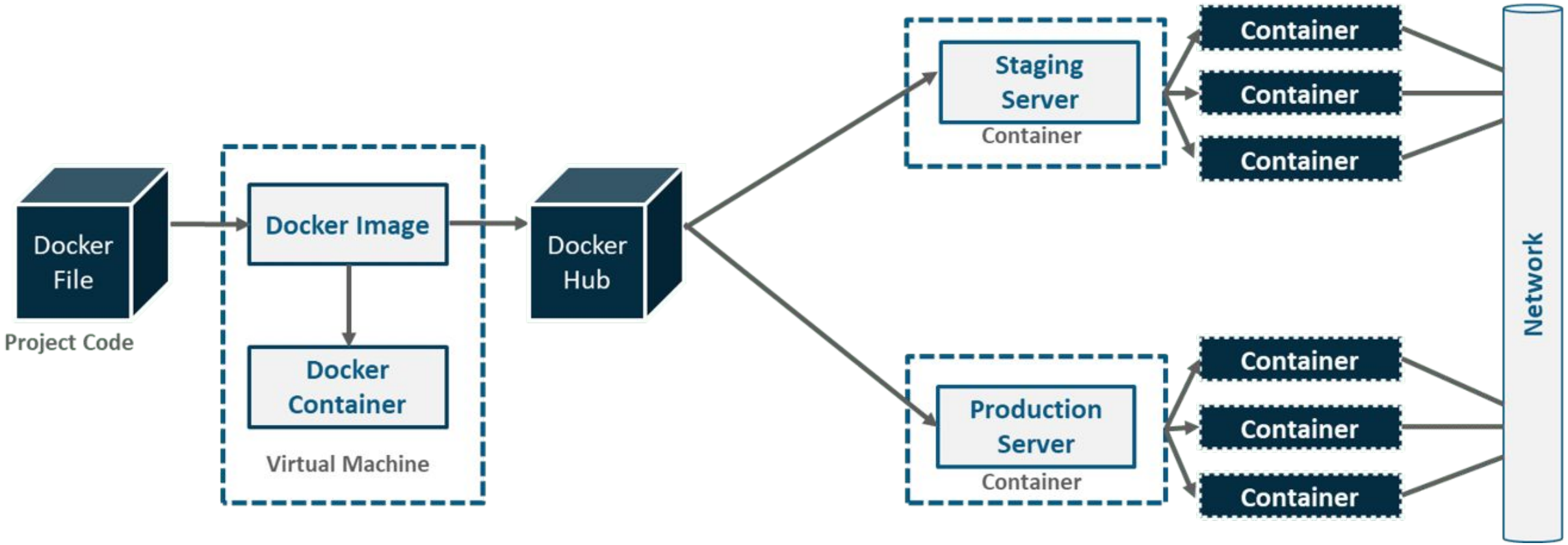
Assignment: Manage Multiple Containers

- Create 2 containers
  - httpd
    - Should receive traffic on port 8080 and redirect to the container on port 80
  - nginx
    - Should receive traffic on port 80 and redirect to the container on port 80
- Test if the containers are working using curl command.
-
- Take a screenshot of the execution and paste in google cloud document for review. Take the template from below URL:
  - https://docs.google.com/document/d/1RuxPegFiOotbDx_36Lb1PEJjbHX2xUdj3967ItmvYGk/edit
- Upload your assignments in google drive folder using below URL:
  - https://drive.google.com/drive/folders/14EB76N9pRkatLPWFzOVpxwlFsWLazPrQ?usp=sharing

Docker Networking

Docker Networking

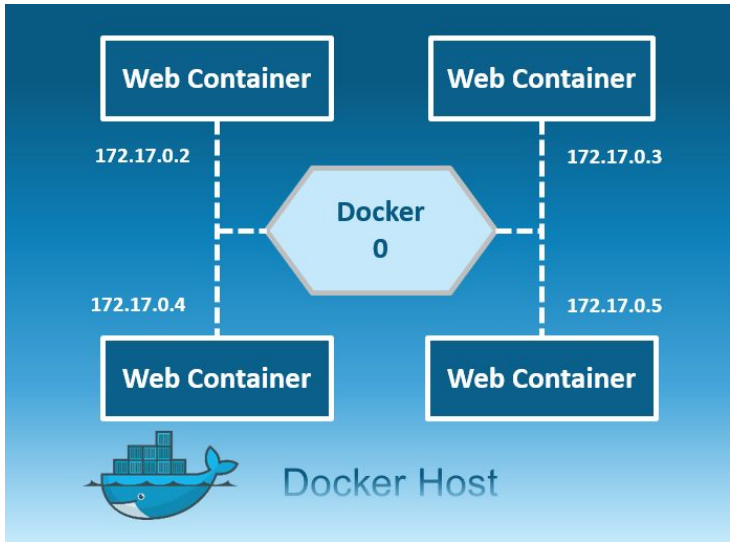
- Before I deep dive into Docker Networking let me show you the workflow of Docker.

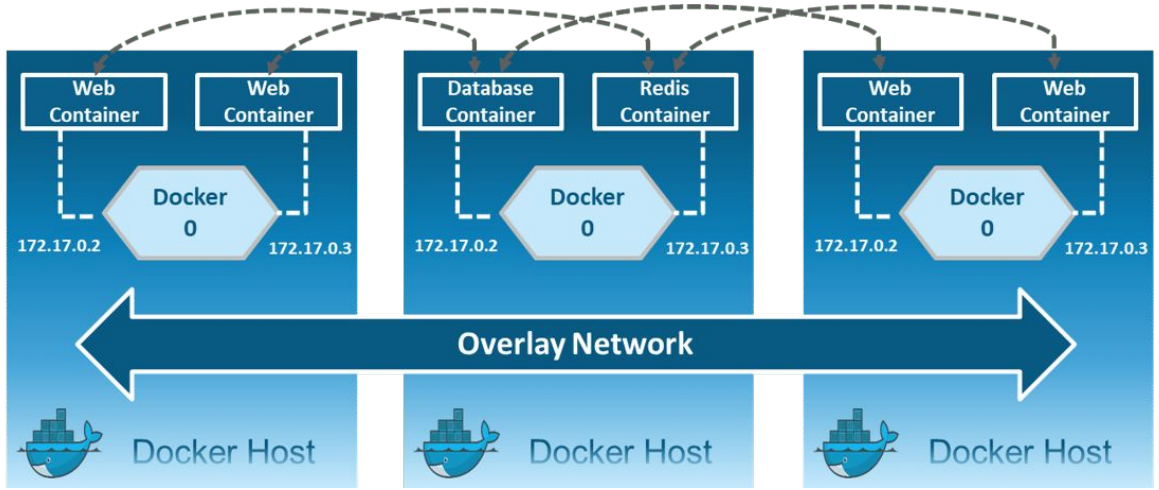


Goals of Docker Networking

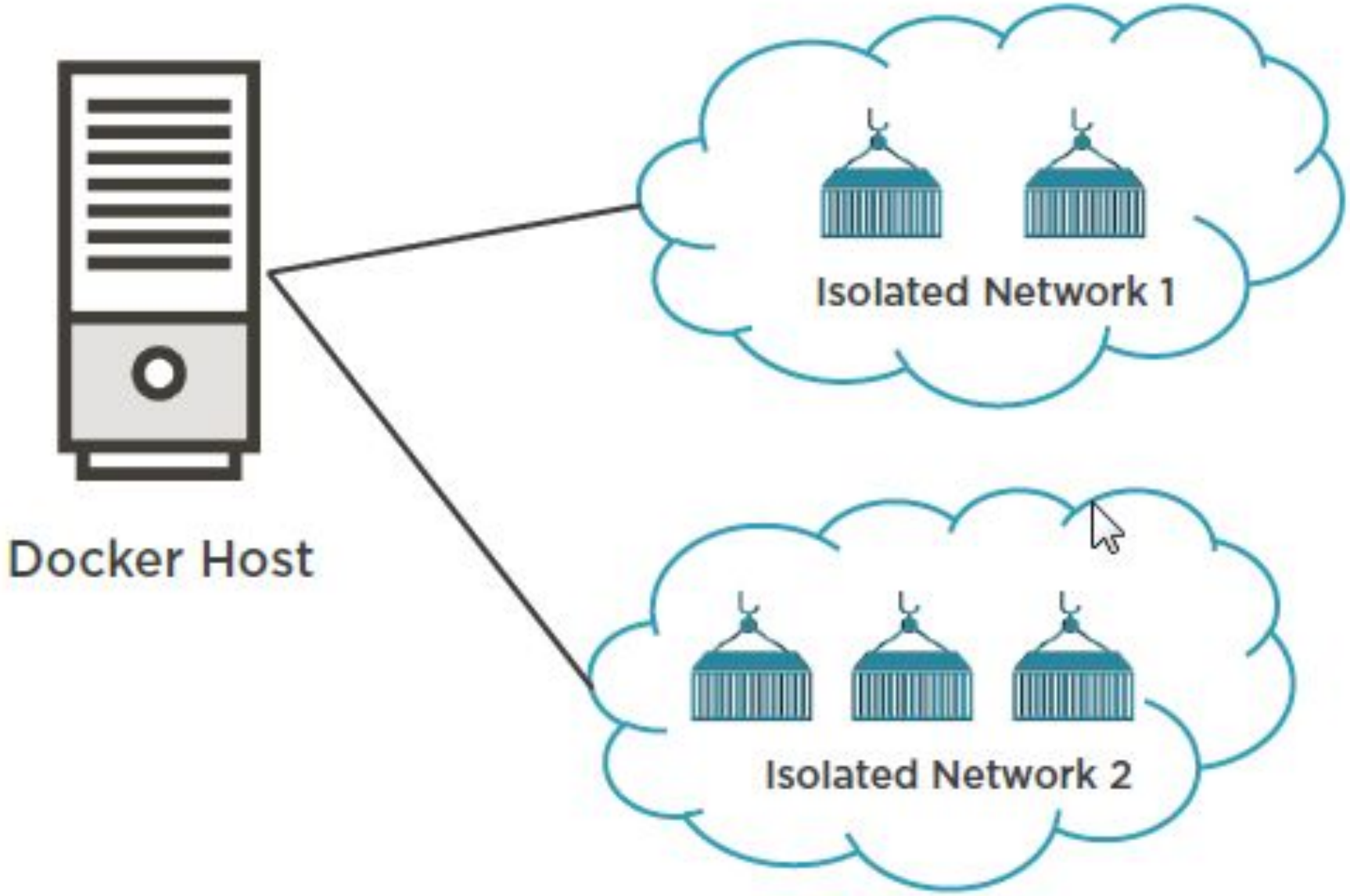
Network Drivers

- Bridge
  - Private default internal network created by docker on the host
  - All containers get an internal IP address
  - These containers can access each other, using this internal IP
- Host
  - This driver removes the network isolation between the docker host and the docker containers to use the host's networking directly
- None
- Overlay





Understanding Container Networks



Use the default bridge network

- docker network ls
  - NETWORK ID NAME DRIVER SCOPE
  - 17e324f45964 bridge bridge local
  - 6ed54d316334 host host local
  - 7092879f2cc8 none null local
- docker run -dit --name alpine1 alpine ash
- docker run -dit --name alpine2 alpine ash
  - Because you have not specified any --network flags, the containers connect to the default bridge network
- docker container ls / docker ps
  - Check that both containers are actually started
- docker network inspect bridge
  - Inspect the bridge network to see what containers are connected to it.

Use the default bridge network

- docker attach alpine1
  - Use the docker attach command to connect to alpine1.
- ip addr show
- ping google.com
  - Access google
- ping -c 2 172.17.0.3
  - ping the second container
- ping -c 2 alpine2
  - Ping by hostname

- Remove the containers you created
  - Docker stop alpine1; docker rm alpine1
  - Docker stop alpine2; docker rm alpine2

Use user-defined bridge networks

- docker network create --driver bridge alpine-net
  - Create the alpine-net network
- docker network ls
- docker network inspect alpine-net
- Now lets create 4 containers and attach those to the network
- docker run -dit --name alpine1 --network alpine-net alpine ash
- docker run -dit --name alpine2 --network alpine-net alpine ash
- docker run -dit --name alpine3 alpine ash # Will be connected to bridge
- docker run -dit --name alpine4 --network alpine-net alpine ash
- docker network connect bridge alpine4 # Connected to 2 networks
- docker container ls
- docker network inspect bridge
- docker network inspect alpine-net

Docker PS Filter

- docker ps --filter "status=exited”
- docker ps --filter "label=color"

Use user-defined bridge networks

- Now let’s ping the containers from each container to another

- Finally Stop and remove all containers and the alpine-net network.
  - $ docker container stop alpine1 alpine2 alpine3 alpine4
  - $ docker container rm alpine1 alpine2 alpine3 alpine4
  - $ docker network rm alpine-net

Networking using the host network

- docker run --rm -d --network host --name my_nginx nginx
- curl localhost:80
- sudo netstat -tulpn | grep :80
  - Verify which process is bound to port 80, using the netstat command
- docker container stop my_nginx

Hands-on

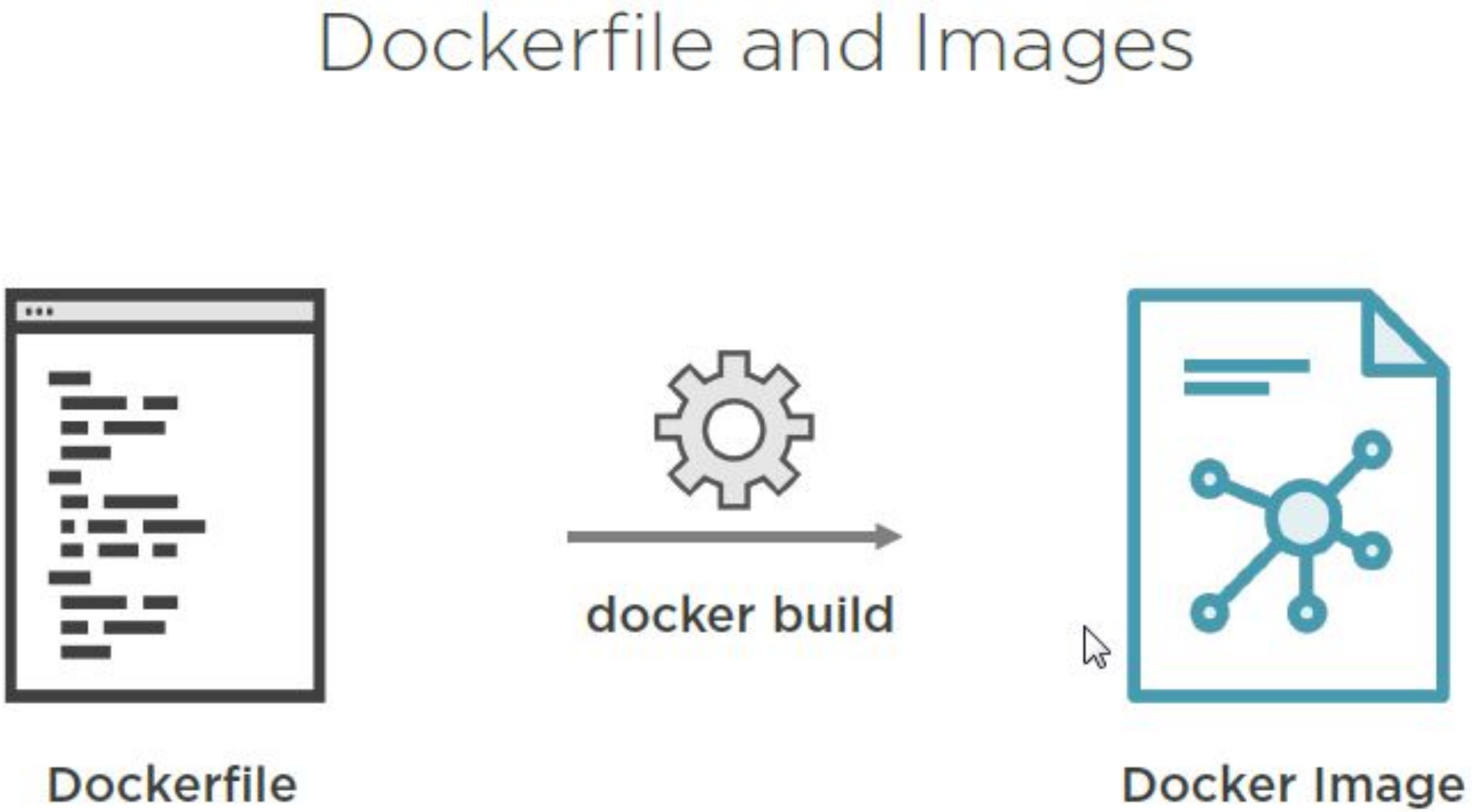
- Refer
  - 002-Creating and Using Containers.txt

Container Images and Dockerfile

Create Dockerized Application

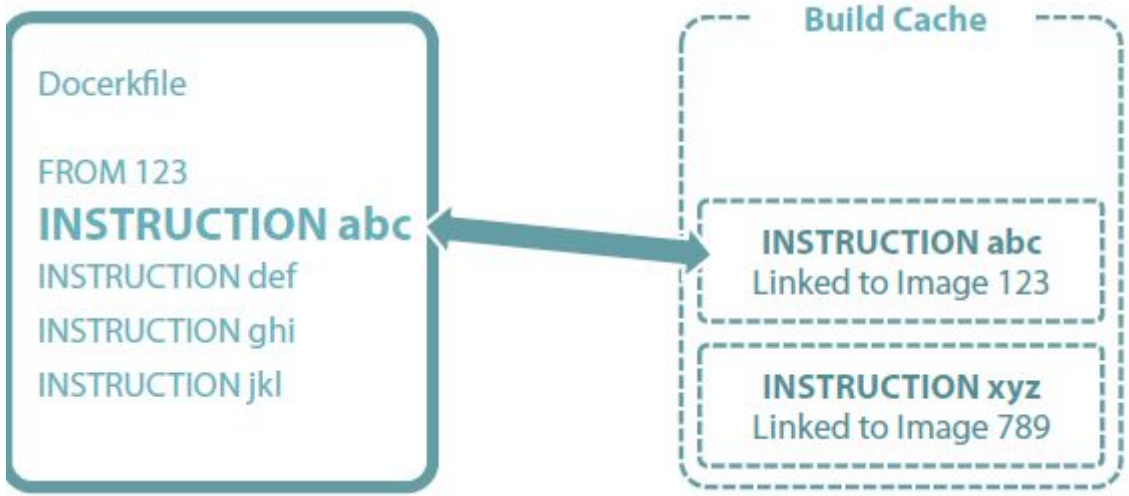
- We can dockerize our application using dockerfile
  - Dockerfile Create images automatically using a build script: «Dockerfile»
  - It Can be versioned in a version control system like Git
  - Docker Hub can automatically build images based on dockerfiles on Github
- This is a basic Dockerfile we need to dockerize a node application
  - FROM node:4-onbuild
  - RUN mkdir /app
  - COPY . /app/
  - WORKDIR /app
  - RUN npm install
  - EXPOSE 8234
  - CMD ["npm", "start"]
  -

Dockerfile


Dockerfile and Images
docker build
Dockerfile
Docker Image

Dockerfile Template

Dockerfile Build Cache



FROM node:4-onbuild

- Pulls/downloads a base image from docker hub which is a public hub for docker images.
- For running a node appication you need to install node in your system

RUN mkdir /app

- In this command we make create an empty directory which will be our working directory with the code files.

COPY . /app/

- Copies all files in current directory to the newly created app directory.
- Your Dockerfile should be in the parent directory of your project.

WORKDIR /app

- To switch from current directory to the app directory where we will run our application.

RUN npm install

- This npm command is related to node application.
- When we copied all dependencies, our main file - package.json would have been copied.
- So running above command installs all dependencies from the file and creates a node_modules folder with mentioned node packages.

EXPOSE 8234

- This command is to expose a port we want our docker image to run on.

CMD ["npm", "start"]

- This is a command line operation to run a node application.
- It may differ based on projects.

Build Image

- Now once we have our Dockerfile ready lets build an image out of it.
- Assuming you all have docker installed on your system lets follow some simple steps:-
  - Navigate to directory containing Dockerfile.
  - Run the following command on your terminal:-
    - docker build -t myimage .
- docker images
- docker run -p 8234:8234 'your image name'

Publish Port

- docker run –t –p 8080:80 ubuntu
  - Map container port 80 to host port 8080

Docker Hub

- Public repository of Docker images
  - https://hub.docker.com/
  - docker search [term]

- Use my own registry
  - To pull from your own registry, substitute the host and port to your own:
    - docker login localhost:8080
    - docker pull 164.52.197.86 :5000/test-image

Assignment: Build Your Own Dockerfile and Run Containers From It

- Upload the docker file in assignment folder.
- Make sure to change dockerfile with your name in format
  - Dockerfile<YourName>
-

Hands-on

- Refer
  - 003-Container Images.txt

Resource Usage

- docker top [container id]
- docker stats [container id]
- docker inspect [container id]

- docker stats –all

Clean Up

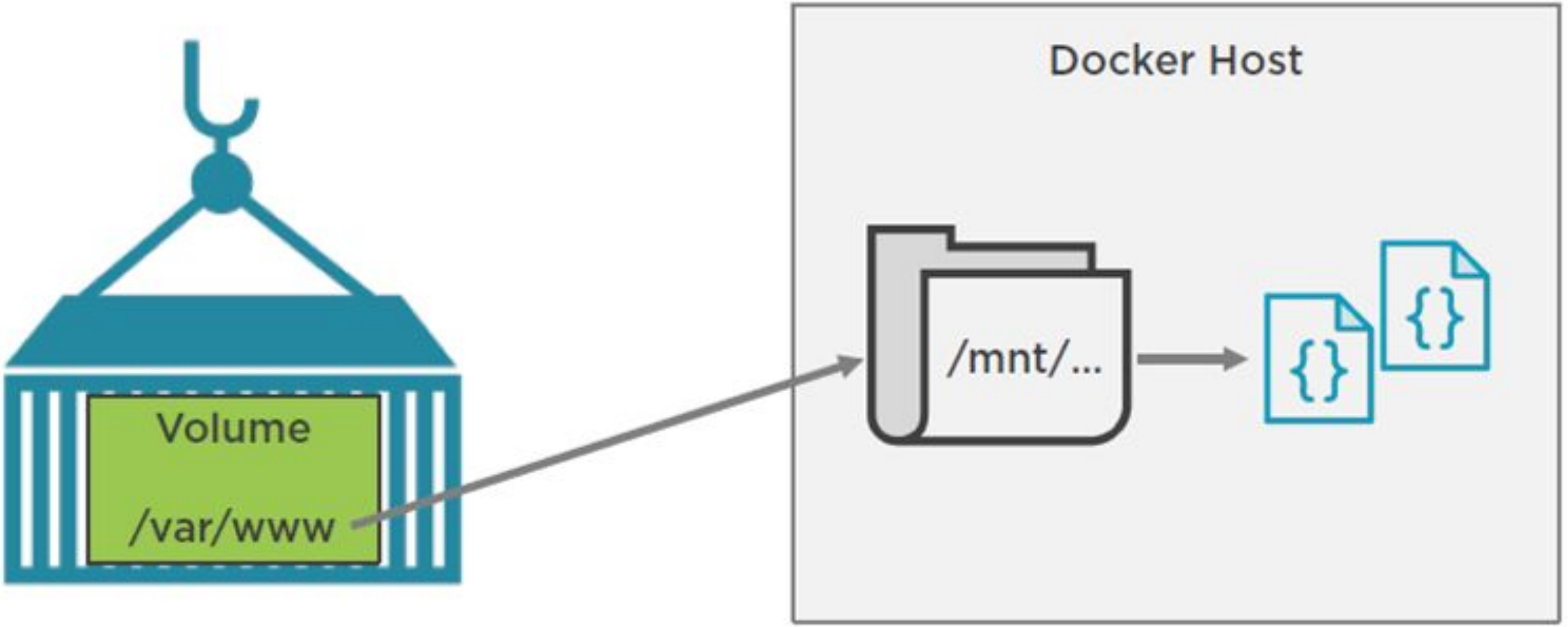
- docker stop $(docker ps -a -q) #stop ALL containers
- docker rm -f $(docker ps -a -q) # remove ALL containers

Persistence Storage

Persistence Storage

- By default all files created inside a container are stored on a writable container layer.
- This means:
  - The data doesn’t persist when that container is removed
- Docker uses volumes to persist data on host file system.

Docker Volume



Create and manage volumes

- docker volume create my-vol
- docker volume ls
- docker volume inspect my-vol
- docker volume rm my-vol

- docker volume create -o size=20GB my-named-volume

- Mount volume in running container
  - https://medium.com/kokster/mount-volumes-into-a-running-container-65a967bee3b5

Start a container with a volume

- docker run -d --name devtest -v myvol2:/app nginx:latest
  - mounts the volume myvol2 into /app/ in the container
- docker inspect devtest
  - To verify that the volume was created and mounted correctly.

- docker container stop devtest
- docker container rm devtest
- docker volume rm myvol2

Use a read-only volume

- docker run -d --name=nginxtest -v nginx-vol:/usr/share/nginx/html:ro nginx:latest
- docker inspect nginx

- docker container stop nginxtest
- docker container rm nginxtest
- docker volume rm nginx-vol

Mount Volumes

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

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

Hands-on

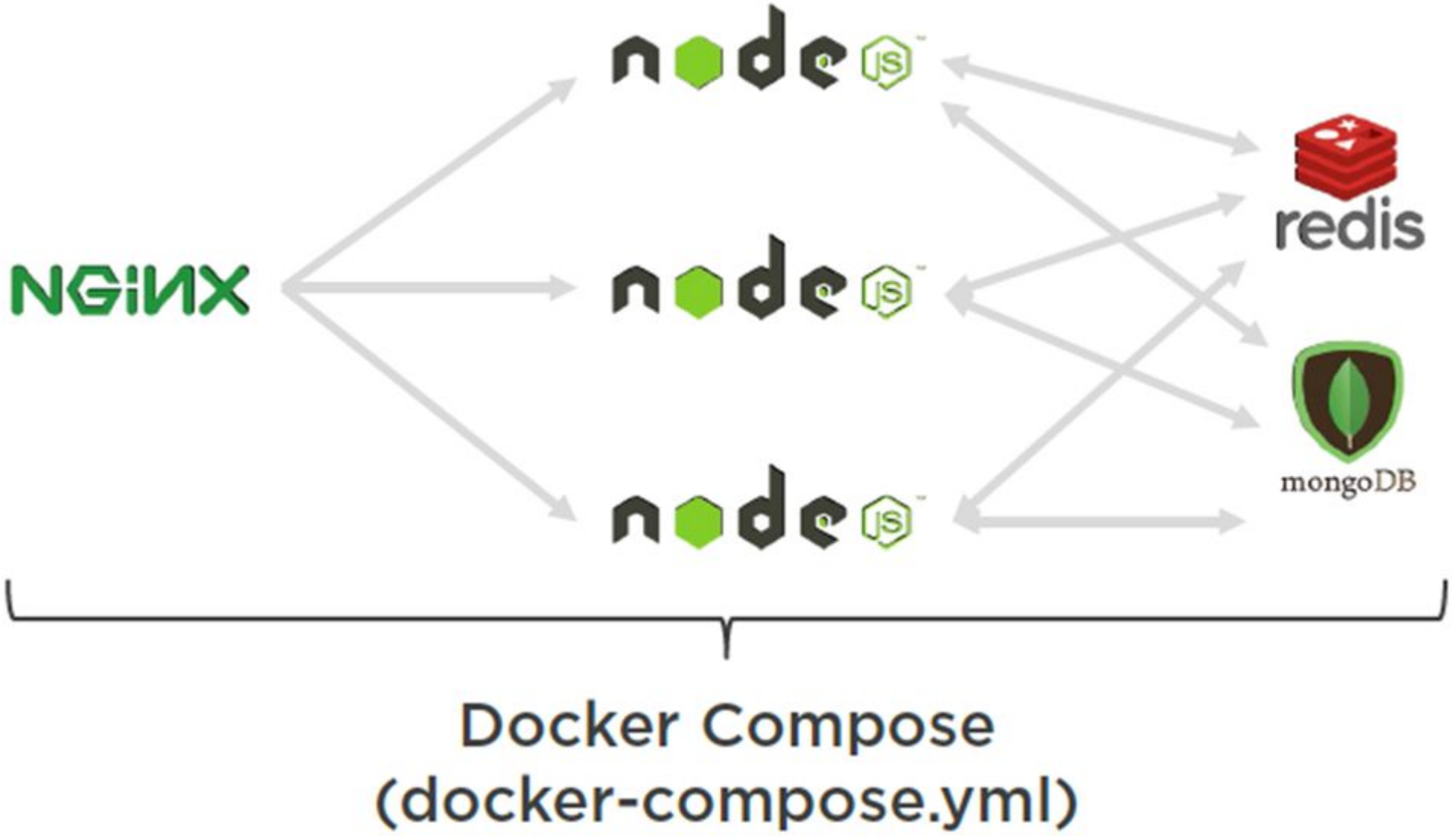
- Refer:
  - 004-Container Lifetime _ Persistent Data.txt

Docker Compose

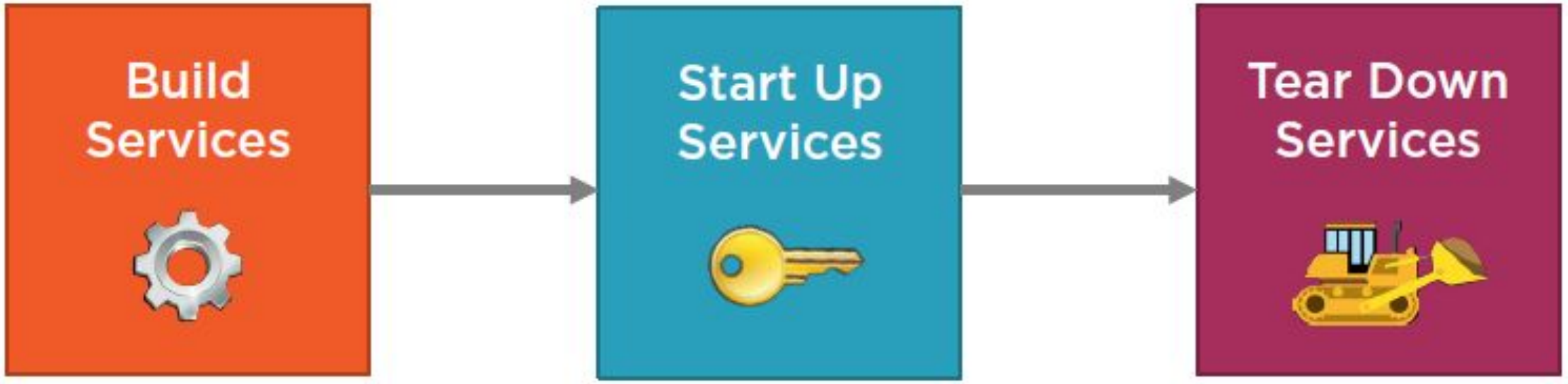
Docker Compose

- 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

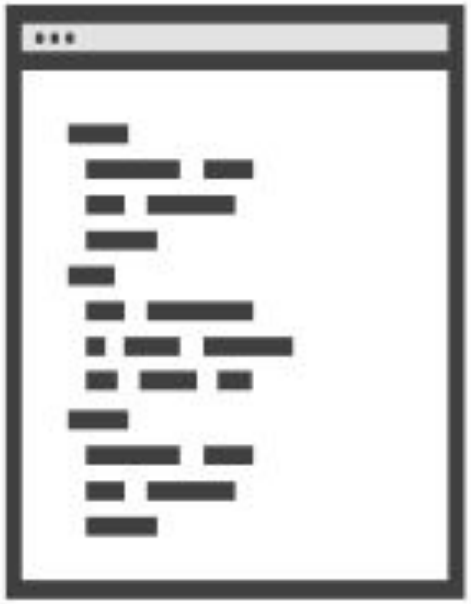
The need for Docker Compose



Docker Compose Workflow

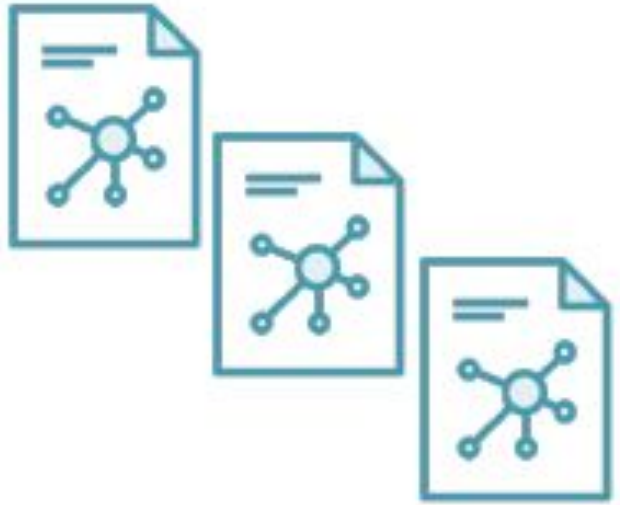


The Role of the DockerCompose File



docker-compose.yml
(service configuration)

Docker Compose
Build



Docker Images
(services)

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

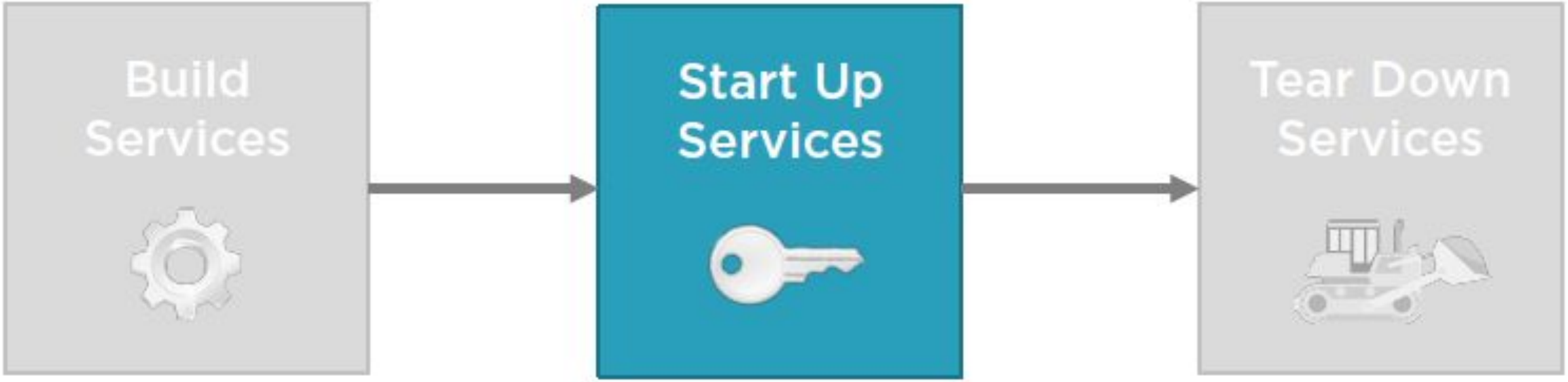

```
docker-compose build
```

Build or rebuild services defined in docker-compose.yml

Building Specific Services

```
docker-compose build mongo
```

Starting Services Up




```
docker-compose up
```

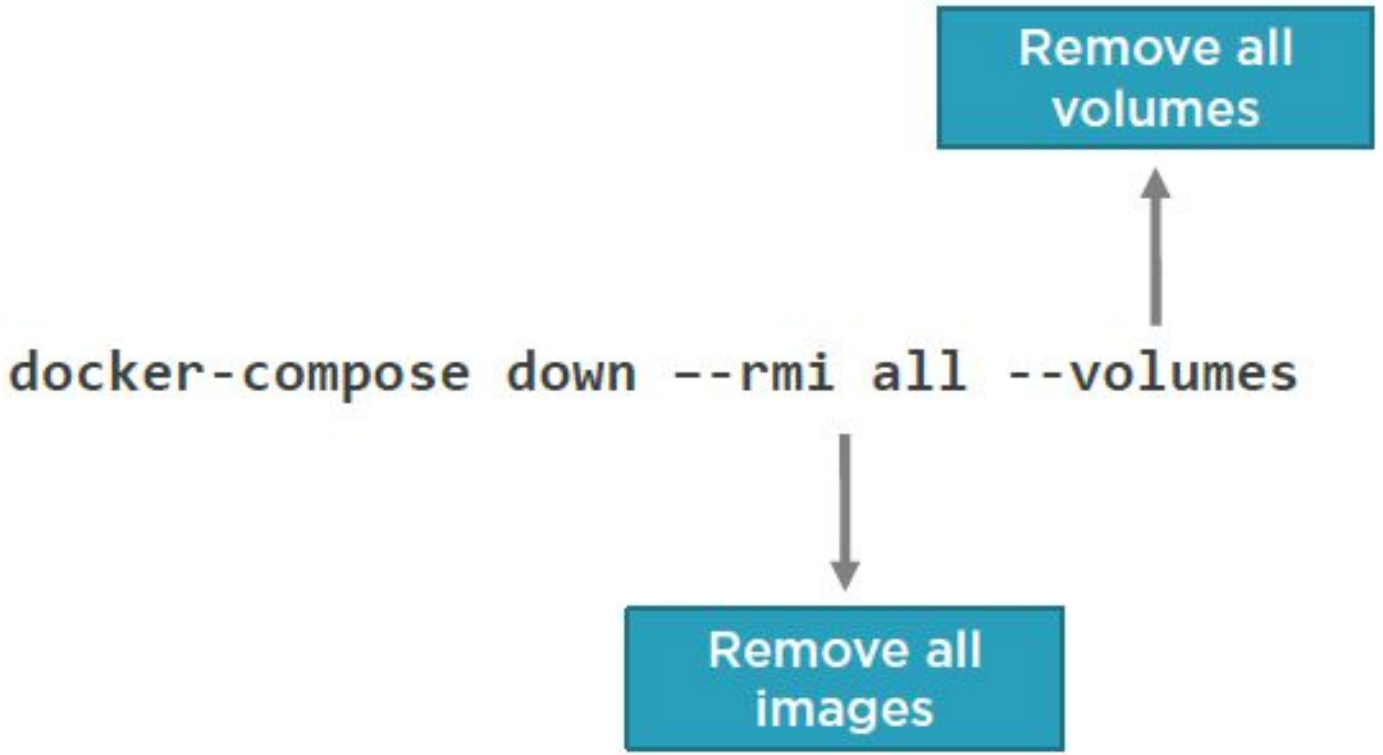
Create and start the containers

Tearing Down Services


```
docker-compose down
```

Take all of the containers down (stop and remove)

Stop and Remove Containers, Images, Volumes



Hands-on

- Refer
  - 005-Docker Compose.txt

Create DockerFile for Spring Boot application

- git clone https://github.com/atingupta2005/02-todo-web-application-h2
- cd 02-todo-web-application-h2
- vim Dockerfile #Review the file content to understand
- docker build -t atingupta2005/02-todo-web-application-h2 .

Take code from:

https://docs.google.com/document/d/1fkxH9LtVAgoYtH_Uyms1hgPyUjBweRKXXITU8OJDvH4/edit

Publish Docker image to public registry

- git clone https://github.com/atingupta2005/02-todo-web-application-h2
- cd 02-todo-web-application-h2
- docker build -t atingupta2005/02-todo-web-application-h2 .
- docker run -p 8080:8080 atingupta2005/02-todo-web-application-h2
- docker login
- docker push atingupta2005/02-todo-web-application-h2

For code refer:

https://docs.google.com/document/d/1TKtg6Zjgr-t9ezGWdeME6eWoSDiztT38L_fg6jUZwq0/edit


thanks