/kahootali

DevOps-Week 2-Docker

M. Ali Kahoot

DevOps Course By M. Ali Kahoot - Dice Analytics

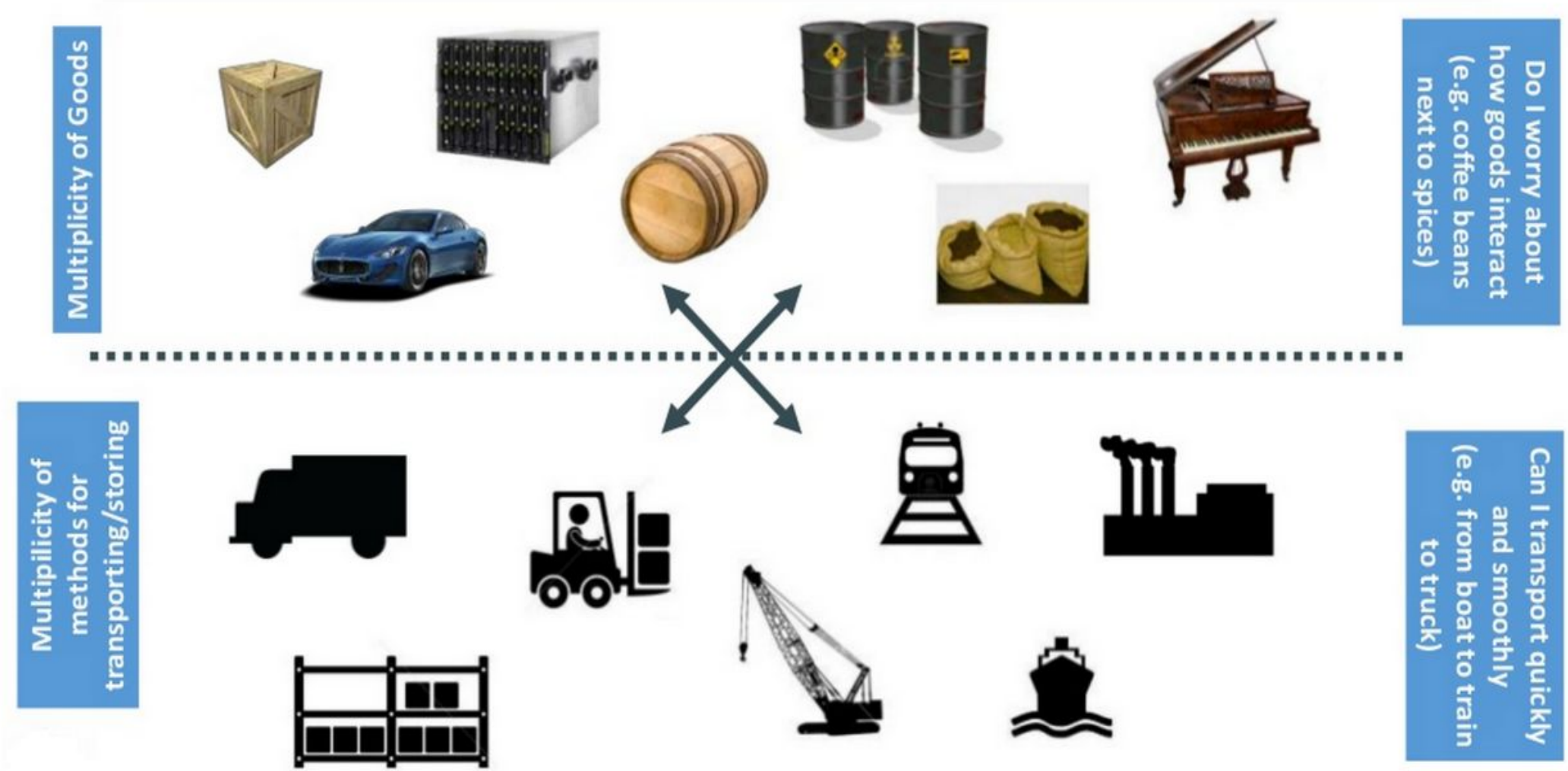
Disclaimer

These slides are made with a lot of effort, so it is a humble request not to share it with any one or reproduce in any way.

/kahootali

Introduction to Containers

DevOps Course By M. Ali Kahoot - Dice Analytics



DevOps Course By M. Ali Kahoot - Dice Analytics

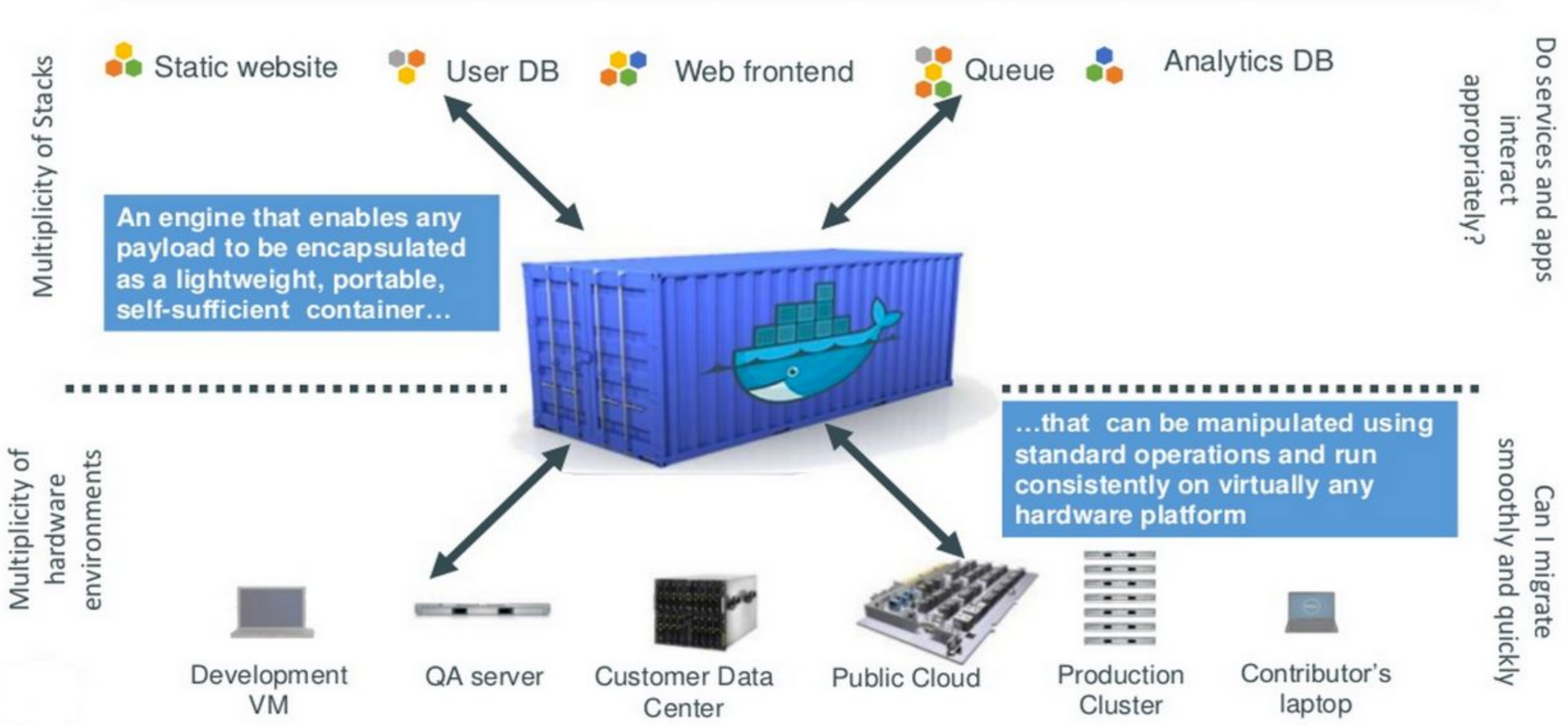
FROM: https://www.bryanbraun.com/2014/07/15/the-frontend-developers-guide-to-docker/

/kahootali

DevOps Course By M. Ali Kahoot - Dice Analytics

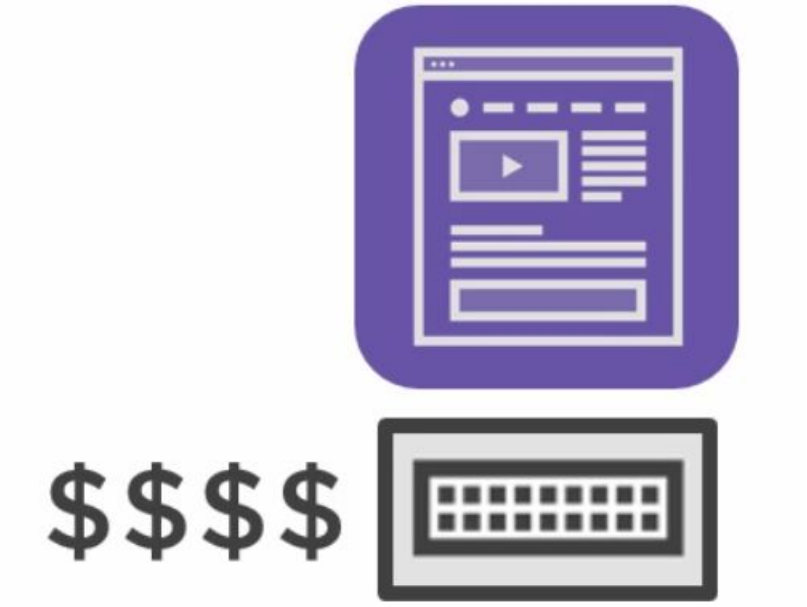
FROM: http://scm.zoomquiet.io/data/20131004215734/docker_container.jpg

/kahootali



History

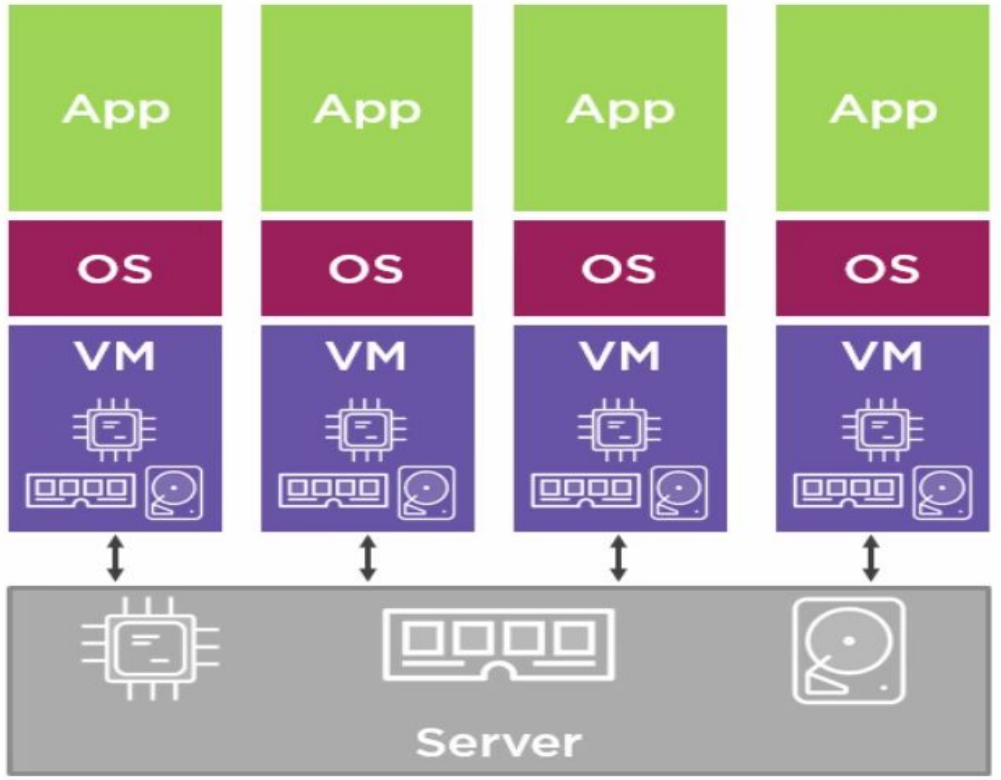
Before cloud even before virtualization a web site would be hosted on physical server



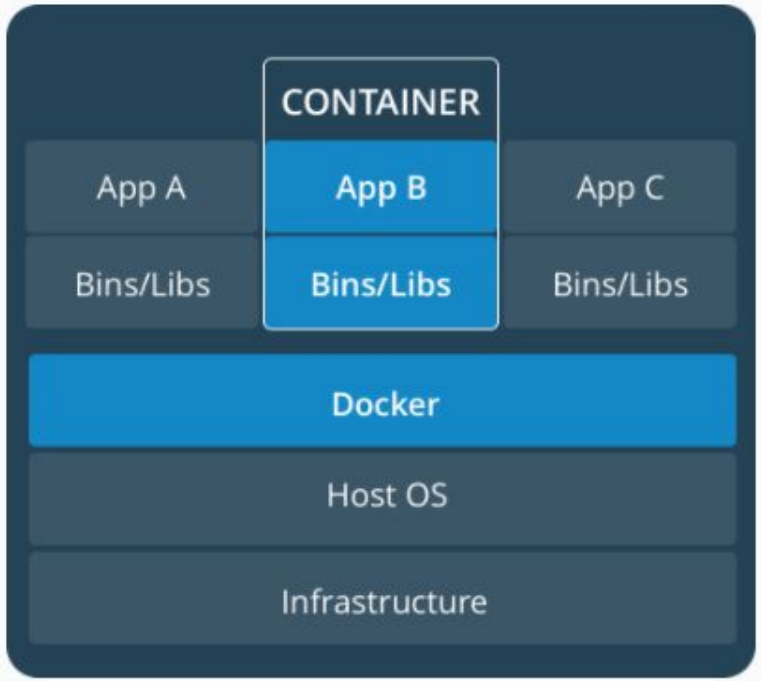
History

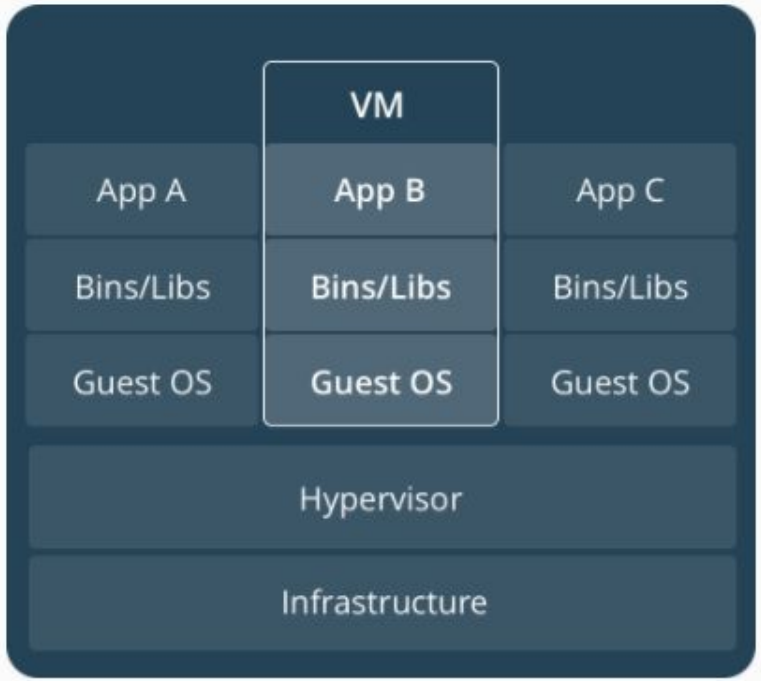
Hypervisors/VM

- Needs multiple OS installations. OS may have license cost
- Uses CPU, RAM, Disk
- Requires admin time



Containers vs VM





CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

Source: https://www.docker.com/what-container#/package_software

Containers vs VM

VMs

- Hardware level virtualization i.e. abstraction of physical hardware
- Share Hardware but has own OS
- Each VM has a full copy of an operating system + application + binaries + libraries
- can take up to tens of GBs.
- VMs are isolated, apps are not
- Complete OS, Static Compute, Static Memory, High Resource Usage

Containers

- OS level virtualization i.e. abstraction at the app layer (code + dependencies)
- Share hardware, host OS kernel but can have own OS
- take up less space (typically tens to hundreds of MBs in size)
- containers are isolated, so are the apps
- Container Isolation, Shared Kernel, Burstable Compute, Burstable Memory, Low Resource Usage

Containers vs VM

VMs

- Ops were responsible for creating VMs, installing Software Dependencies, then installing Software which might not work due to some compatibility issues
- Dev responsible for Software Development and running on local machine vs Ops running the Software on VM with newly installed Libraries
- Works on my machine issue

Containers

- Ops are responsible for VM creation and installing Docker only
- Dev writes code and tests in local container based on the same image
- Same image is deployed in Stage, Prod
- Ideally no “WORKS ON MY MACHINE” issue
- Process level isolation but relatively less secure

Containers vs VM

Source: http://www.lukewilson.net/2017/02/22/Docker-Thinking-inside-the-box/

Virtual Machines
(Houses)

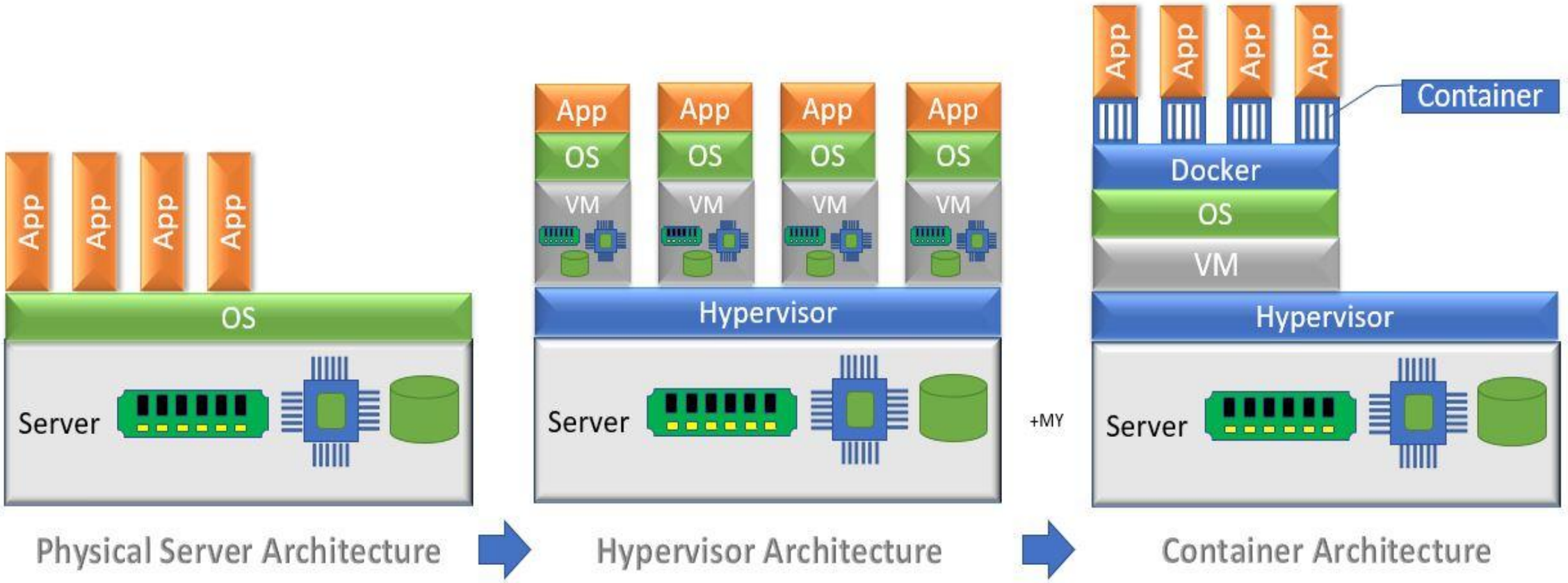
- **Has its own infrastructure**
- Has more necessary things that make it a house, e.g:
  - Roof
  - At least one bedroom
  - Bathroom
  - Kitchen
  - Living area
  - Garage
  - Yard

vs

Containers
(Apartments)

- **Shares existing infrastructure**
- Comes in a variety of different setups:
  - Studio / 2 br / penthouse
  - Kitchen vs kitchenette
  - Living area?
  - Parking space?
  - Balcony?

Summary


App
App
App
App
OS
Server
Physical Server Architecture
App
App
App
App
OS
OS
OS
OS
VM
VM
VM
VM
Hypervisor
Server
Hypervisor Architecture
+MY
App
App
App
App
Container
Docker
OS
VM
Hypervisor
Server
Container Architecture

OS Terms

Host OS

- For Linux and non-Hyper-V containers, the Host OS shares its kernel with running Docker containers.
- For Hyper-V each container has its own Hyper-V kernel.

Container OS

- windows containers require a Base OS, while for Linux containers, its optional.

Operating System Kernel

- The Kernel manages lower level functions such as memory management, file system, network and process scheduling.

What is Container?

- Container provides operating system level virtualization
  - Shares the same kernel of the host system
  - Container thinks it has its own copy of OS
- Container decouples applications from operating system
- OS is abstracted away from containers
- User can have clean and minimal operating system and run everything else in one or more isolated containers on top of host OS
- Image is format that describes an applications and its dependencies
  - Ship container(image) instead of application

What is Container?

- Provides a standard way to package Application Code, Configuration & Dependencies
- Run as isolated process
- Run anywhere
- Improve resource utilization
- Scale quickly
- Lightweight
- Use Cases:
  - Microservices, Batch processing, Machine Learning, Application migration to cloud

Containers

- Container Host
  - Physical or virtual system running a complete OS and with CRI like Docker installed. It will run containers.
- Container Image
  - Container is a running instance of an image
  - It works in layers
  - Container OS image is the first layer
  - Multiple containers can share the same image

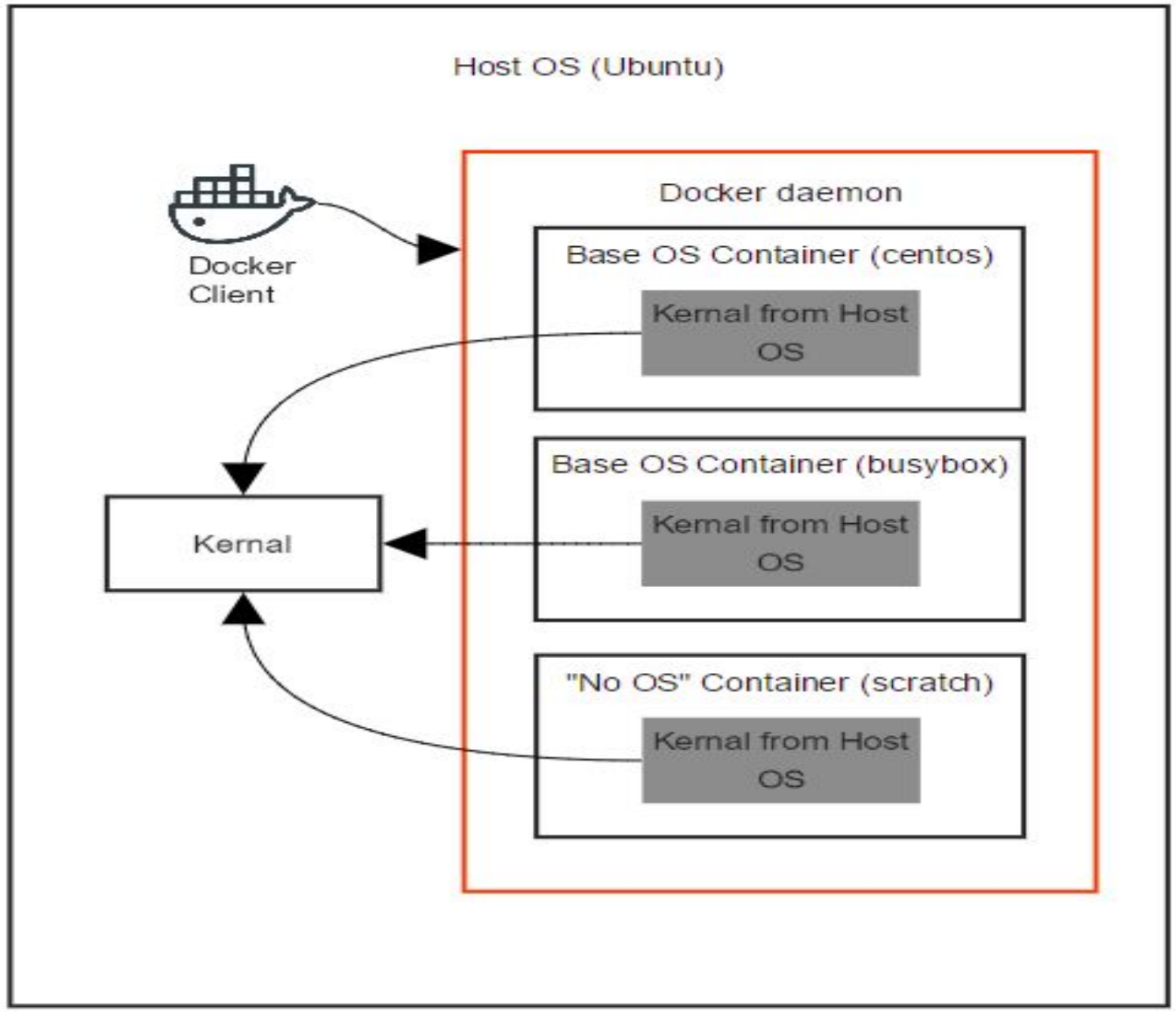
Containers Shortcomings

- Type of container must be the same as of host system as it shares the kernel
- Can not run windows container on Linux host or vice-versa
- Isolation between the host and containers is not as strong as hypervisors based virtualization

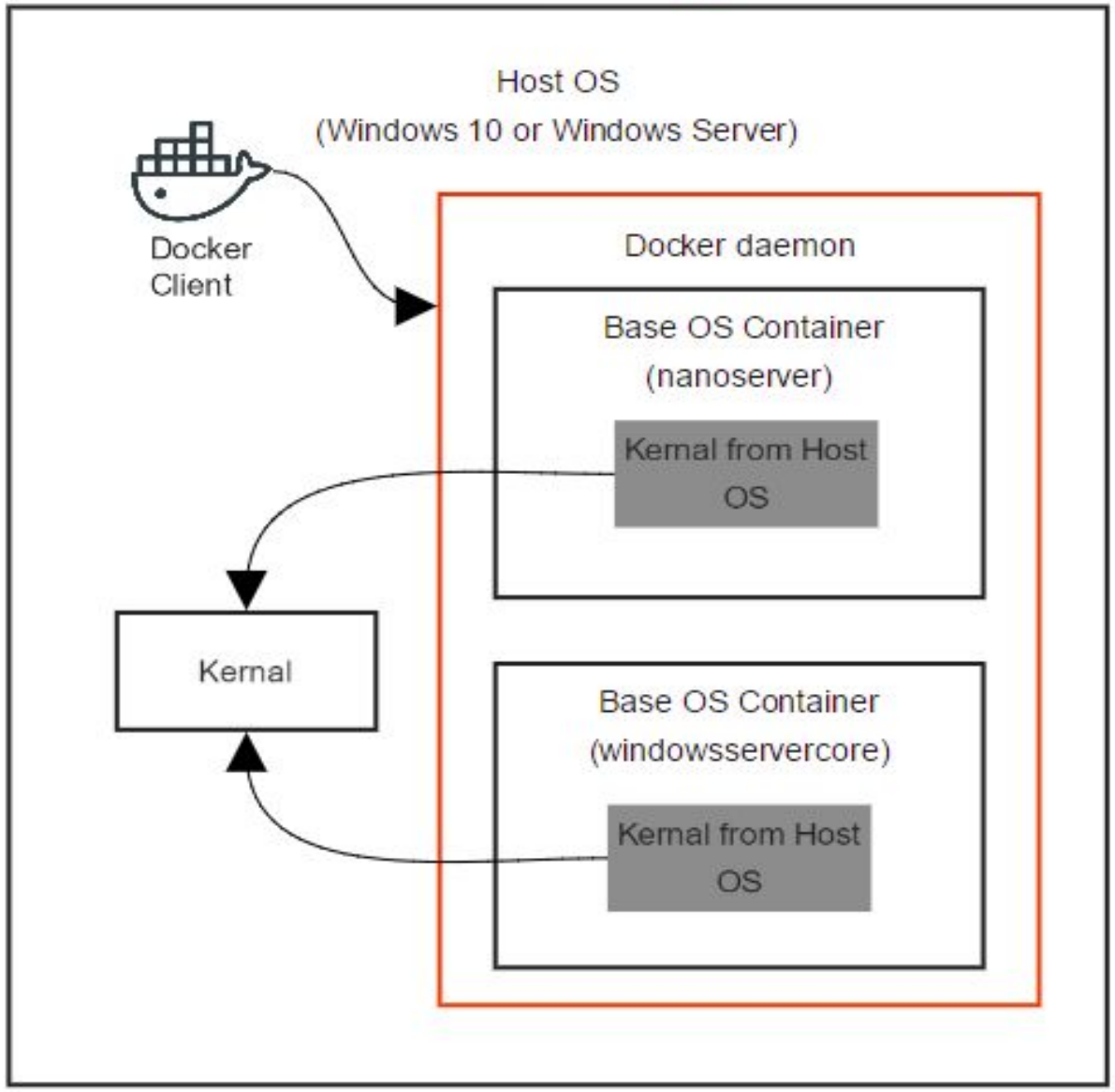
Docker

- Platform that introduced sharing of images
- Docker is a utility that can create, ship and run containers
- Docker is to containers what VMware is to hypervisors
- Other container technologies are Docker, Apache Mesos, rkt (pronounced “rocket”)
  - These technologies allow to create containers that can run as isolated processes
- Docker restricts container to run as single process
  - Enabling micro service architecture
  - Means database running in one container and app in other
  - Docker allows to link these containers and constructs an application

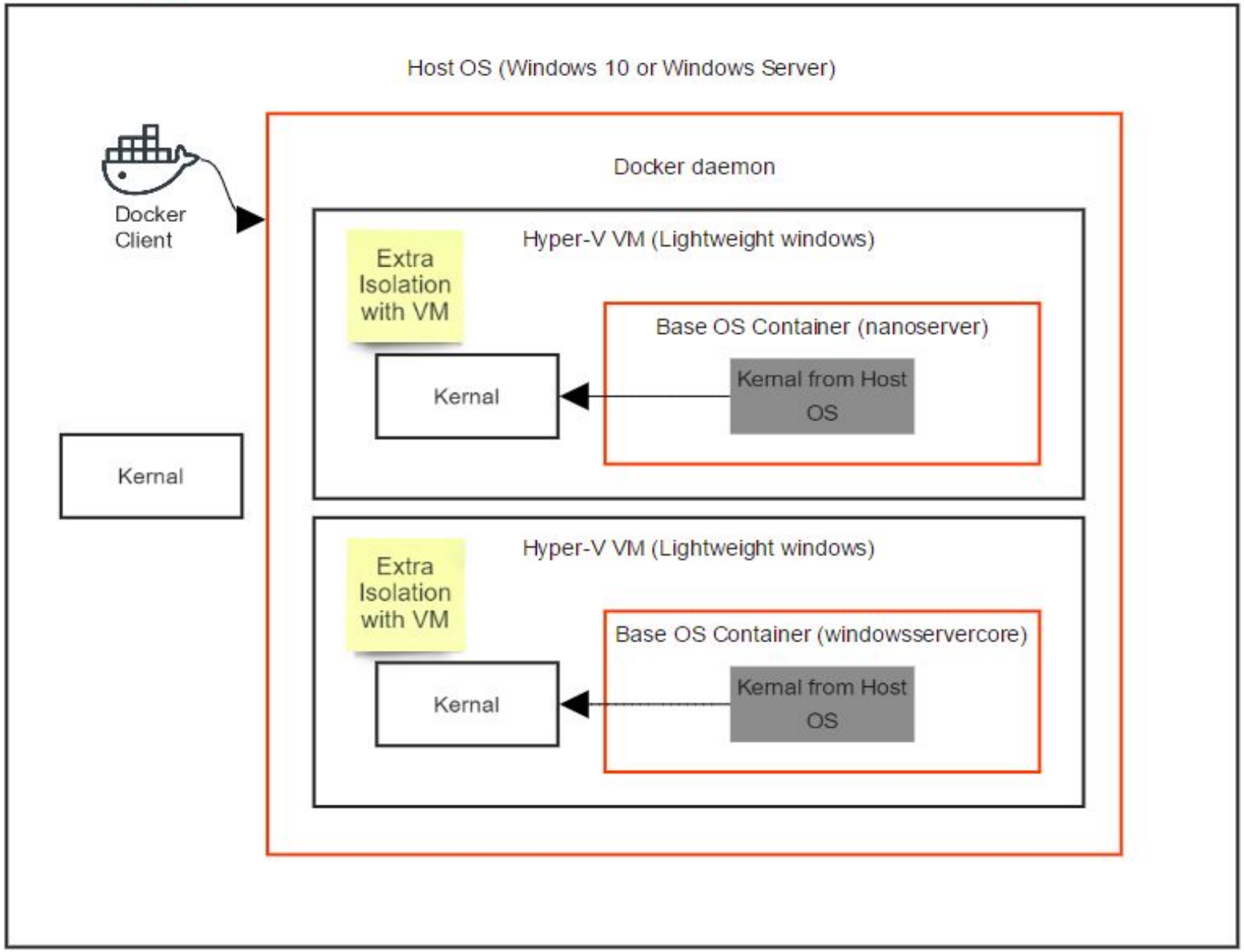
Linux Containers



Windows Server Containers - Non Hyper-V



Windows Hyper-V Containers



Basic Container Terms

Container Host

Physical or virtual system configured with the OS container feature. It will run containers.

Docker Image

Ordered Collection of layers of Root Filesystem Changes. Multiple containers can share the same image

Dockerfile

A file containing the Instructions to build a Docker Image

Basic Container Terms

Container

Runtime instance of an Image

OS level virtualization

Consists of a Docker image, an execution environment and set of instructions

Docker Registry

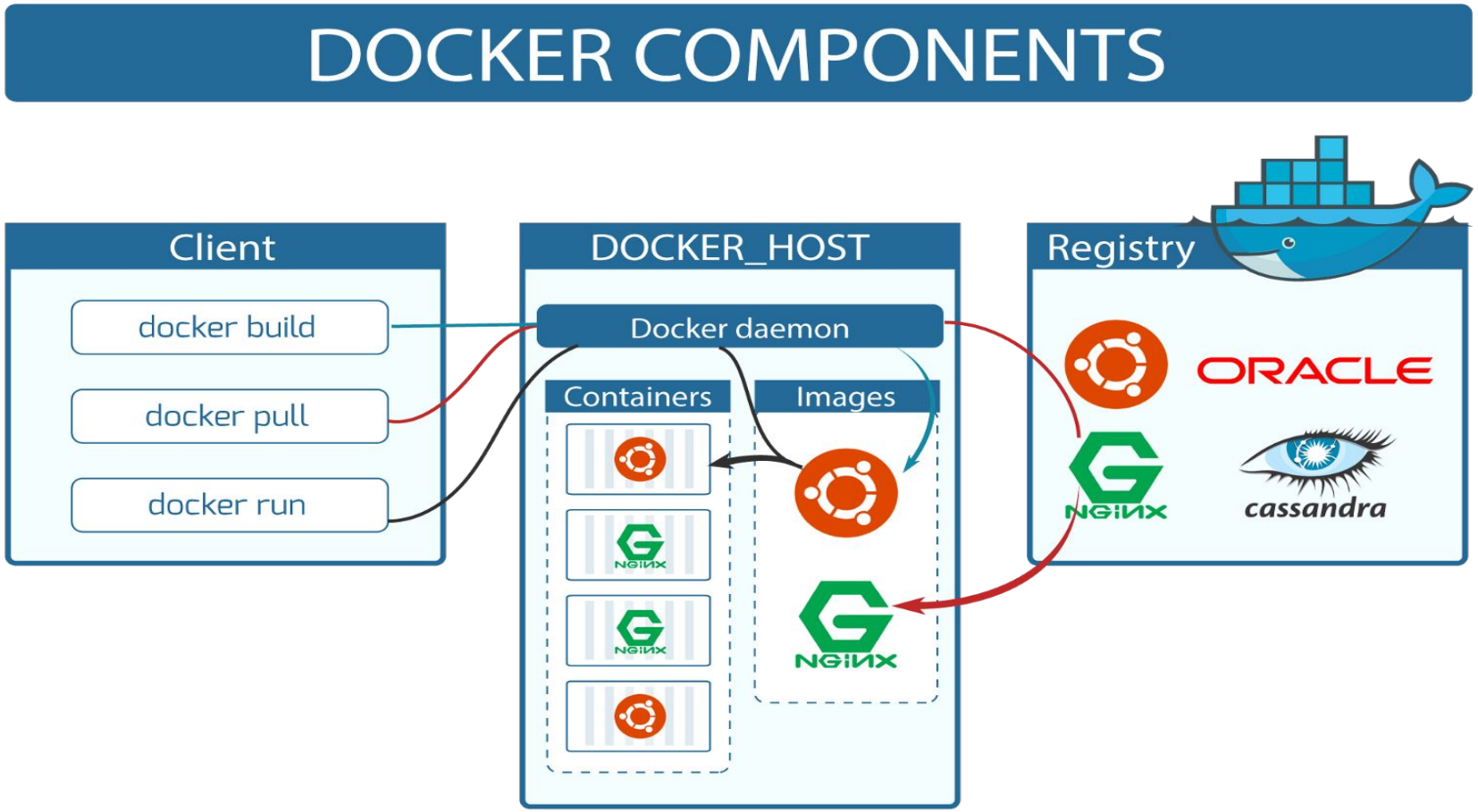
A central place to store Docker images for use by others

Can be public e.g. Docker Hub / private e.g. Azure Container Registry

By default, Docker looks for images on Docker Hub

A private registry can also be set up

Docker Architecture



FROM: https://medium.freecodecamp.org/comprehensive-introductory-guide-to-docker-vms-and-containers-4e42a13ee103

Docker Architecture

Client

- A simple CLI to interact with Daemon
- Primary way to interact with Docker Daemon, which carries them out

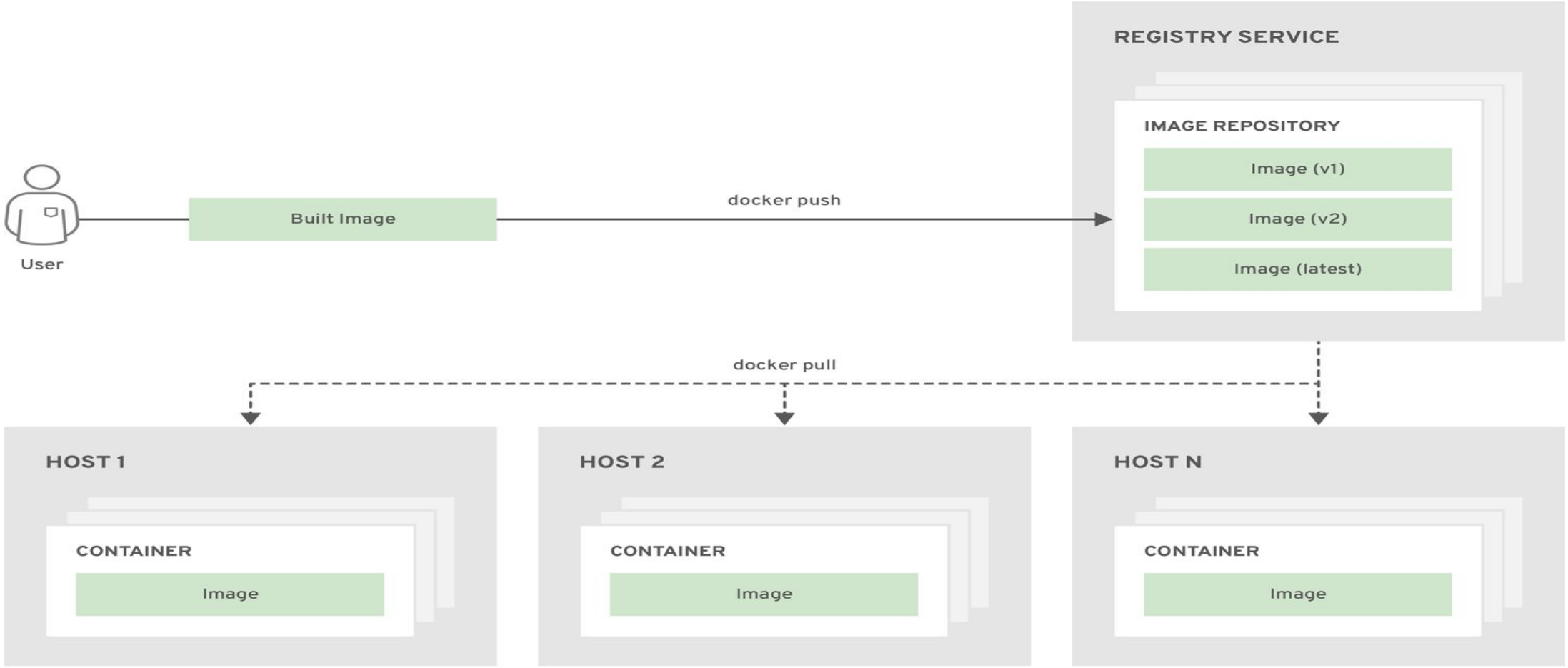
Daemon

- Main component running on OS, communicating with OS
- Build and store Images
- Create, run and monitor Containers

Registry

- Store and distribute images
- DockerHub, private registries

Development Cycle with Docker



RUN YOUR FIRST CONTAINER

```
docker container run --publish 8080:80 nginx
```

Looks for nginx image locally, if it does not find

Looks in remote image repository, defaults to Docker Hub Downloads the latest version, nginx:latest

Gives it a virtual IP on a private network inside docker - engine

Opens up port 8080 on host and forwards to port 80 on - container

Starts container

Docker Commands

Verify version: `docker version`

Display configuration: `docker info`

List running Containers: `docker container ls`

List running and stopped Containers: `docker container ls -a`

Run Container: `docker container run <image-name>`

Stop Container: `docker container stop <container-name/id>`

Docker Commands

Delete Container: `docker container rm <container-name/id>`

Delete Running Container: `docker container rm -f  <container-name/id>`

Pull Image: `docker image pull`

List all local images: `docker image ls`

Remove local image: `docker image rm <image-name>`

Container Info

List Process in Container: `docker container top <container-id/name>`

Details of Container cong: `docker container inspect <container-id/name>`

Stats for all containers: `docker container stats`

Log of Container: `docker container logs <container-id/name>`

Get shell inside running container:

```
docker container exec -it <container-id/name> <command>
```

LAB - FIRST CONTAINER

- Run Container

```
docker container run --publish 8080:80 --name nginx-container nginx
```

--publish specifies to forward traffic on 8080 on host to port 80 of container --name specifies name of container. Access nginx server by specifying localhost:8080 in your browser

- List running containers:

```
docker container ls
```

- List running and stopped containers:

```
docker container stop nginx-container

docker container ls -a
```

LAB - FIRST CONTAINER

- Run Container in Background

```
docker container run -d --publish 8081:80 --name nginx-background nginx
```

-d species to run the container in detached mode. Access nginx server by specifying localhost:8081 in your browser

- List running containers:

```
docker container ls
```

- List running and stopped containers:

```
docker container ls -a
```

LAB - FIRST CONTAINER

- Pull Image

```
docker image pull centos
```

This will pull the latest image of centos, if you need a specific version you can specify that after the image

```
docker image pull centos:7
```

LAB - FIRST CONTAINER

- What's Going On In Container

List process of a container

```
docker container top nginx-background
```

List details of container cong:

```
docker container inspect nginx-background
```

Display logs of container:

```
docker container logs nginx-background
```

Display performance stats of all container:

```
docker container stats
```

LAB - FIRST CONTAINER

- Get Shell Inside Container

Start a new container interactively

```
docker container run -it --name centos-container centos
```

Exit the shell

```
exit
```

Once you exit the shell container will stop, verify that by running list command

```
docker container ls
```

```
docker container ls -a
```

LAB - FIRST CONTAINER

Get shell inside running container

```
docker container exec -it nginx-background /bin/bash
exit
```

- Container Logs

Get container logs of a running container:

```
docker container logs nginx-background
```

LAB - FIRST CONTAINER

- Stop Containers

Stop all running containers:

```
docker container stop nginx-container
docker container stop nginx-background
```

- Delete Containers

Delete stopped containers:

```
$ docker container rm nginx-container
$ docker container rm nginx-background
$ docker container rm centos-container
```

LAB - FIRST CONTAINER

- Manage Multiple Containers

We will run an nginx and httpd container. We will map nginx container on host port 8081 and httpd container on port 8082, access both in your browser, then stop and delete both containers

Run nginx container:

```
docker container run -d --publish 8081:80 --name nginx-cont nginx
```

Access nginx server by specifying localhost:8081 in your browser

LAB - FIRST CONTAINER

Run httpd container:

```
docker container run -d -p 8082:80 --name apache-cont httpd
```

Access apache server by specifying localhost:8082 in your browser

Stop and delete containers:

```
docker container stop apache-cont
docker container stop nginx-cont
docker container rm apache-cont
docker container rm nginx-cont
```

DOCKER NETWORKS

Each container connected to a private virtual network - "bridge"

Each virtual network routes through NAT firewall on host - IP

All containers on a virtual network can talk to each other

Best practice is to create a new virtual network for each application

Can attach containers to more than one virtual network - (or none)
Skip virtual networks and use host IP (--net=host)

DOCKER NETWORKS

Types of Network Drivers

- Bridge
- Overlay
- Host
- None

DOCKER NETWORKS

Intercommunication never leaves host

All externally exposed ports closed by default

You must manually expose via -p, which is better default - security

Containers shouldn't rely on IP's for inter-communication

DNS is the key to easy inter-container communication

DNS for friendly names is built-in if you use custom - networks

DOCKER NETWORKS COMMANDS

Show networks: `docker network ls`

Inspect a network: `docker network inspect`

Create a network: `docker network create --driver`

Attach a network to container: `docker network connect` or `docker container run --net <network-name> <container-name/id>`

Detach a network from container: `docker network disconnect`

LAB - NETWORKS

- Create Network
- List Network
- Intercommunication of containers
- Delete Containers
- Delete Network

LAB - NETWORKS

- Create Network

```
docker network create new-network
```

By default bridge driver type is used

- List Network

List all virtual networks:

```
docker network ls
```

LAB - NETWORKS

- Intercommunication of containers

Run 2 containers and try to ping them

```
docker container run --net new-network --name centos1 -it centos
```

Open another terminal on your system and run second container:

```
docker container run --net new-network --name centos2 -it centos
```

LAB - NETWORKS

- Intercommunication of containers

Access centos1 from shell of centos2 using ping:

```
ping centos1
```

Go back to shell of centos1 and run the command:

```
ping centos2
```

LAB - NETWORKS

- Delete Containers

Delete centos1 and centos2 containers:

```
docker container rm centos1 centos2
```

- Delete Network

```
docker network rm new-network
```

Things to do before next class

- Create your first contribution to this repo

https://github.com/firstcontributions/first-contributions

- Docker should be installed on Ubuntu environment
- Complete all labs for git & docker till now