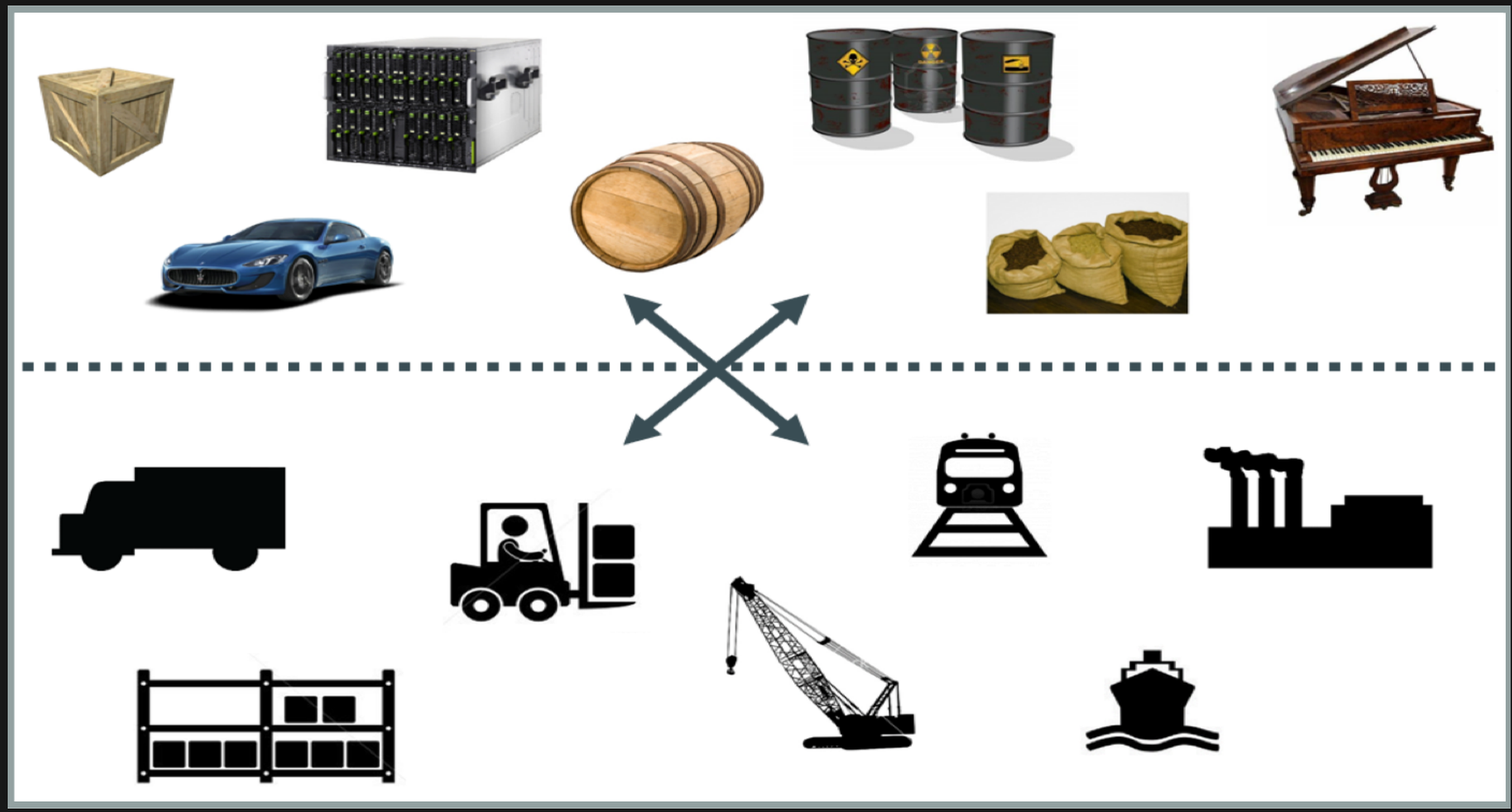
DOCKER

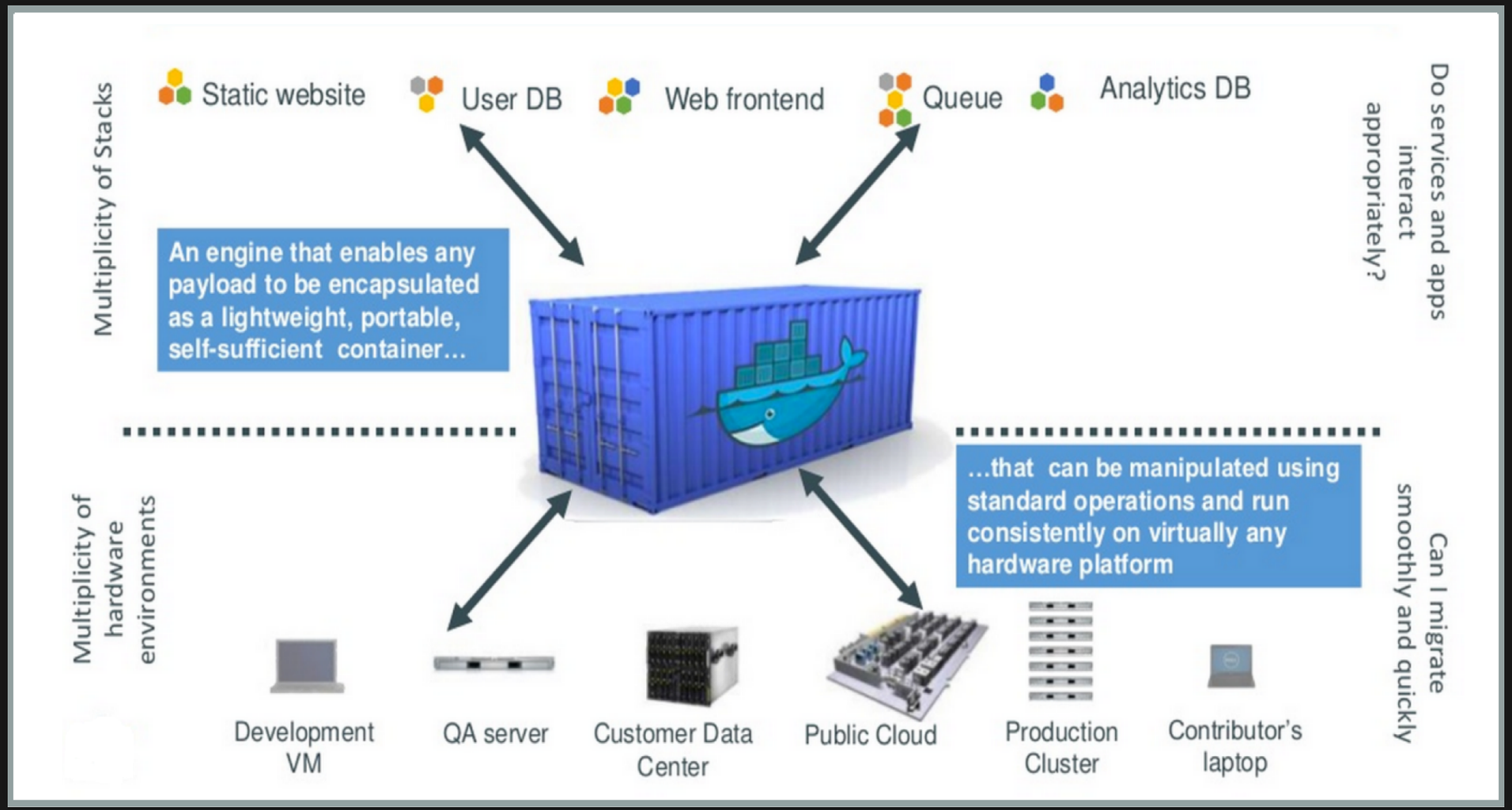
DevOps Course by Ali Kahoot

USEFUL ANALOGY

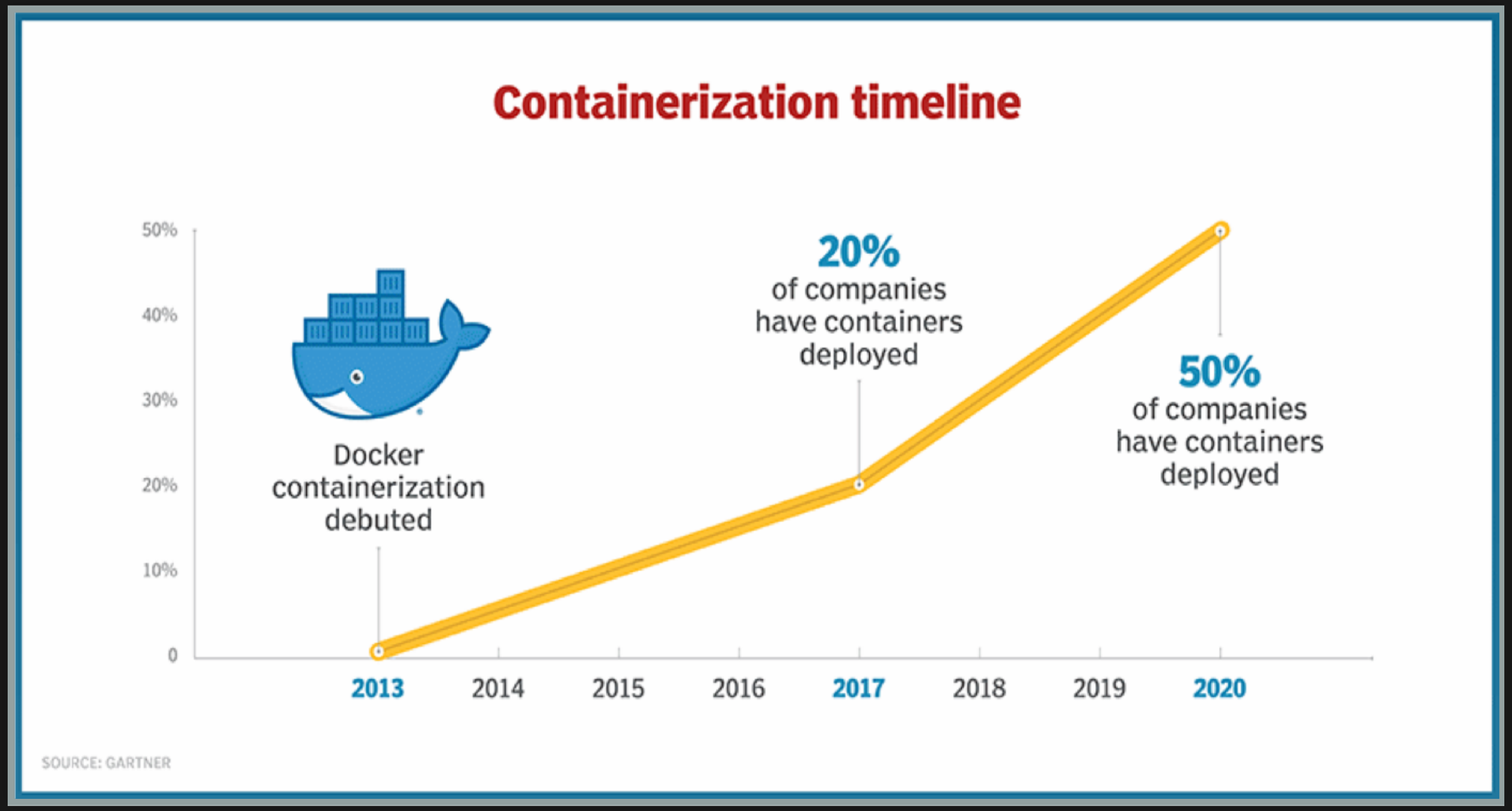


SOLUTION

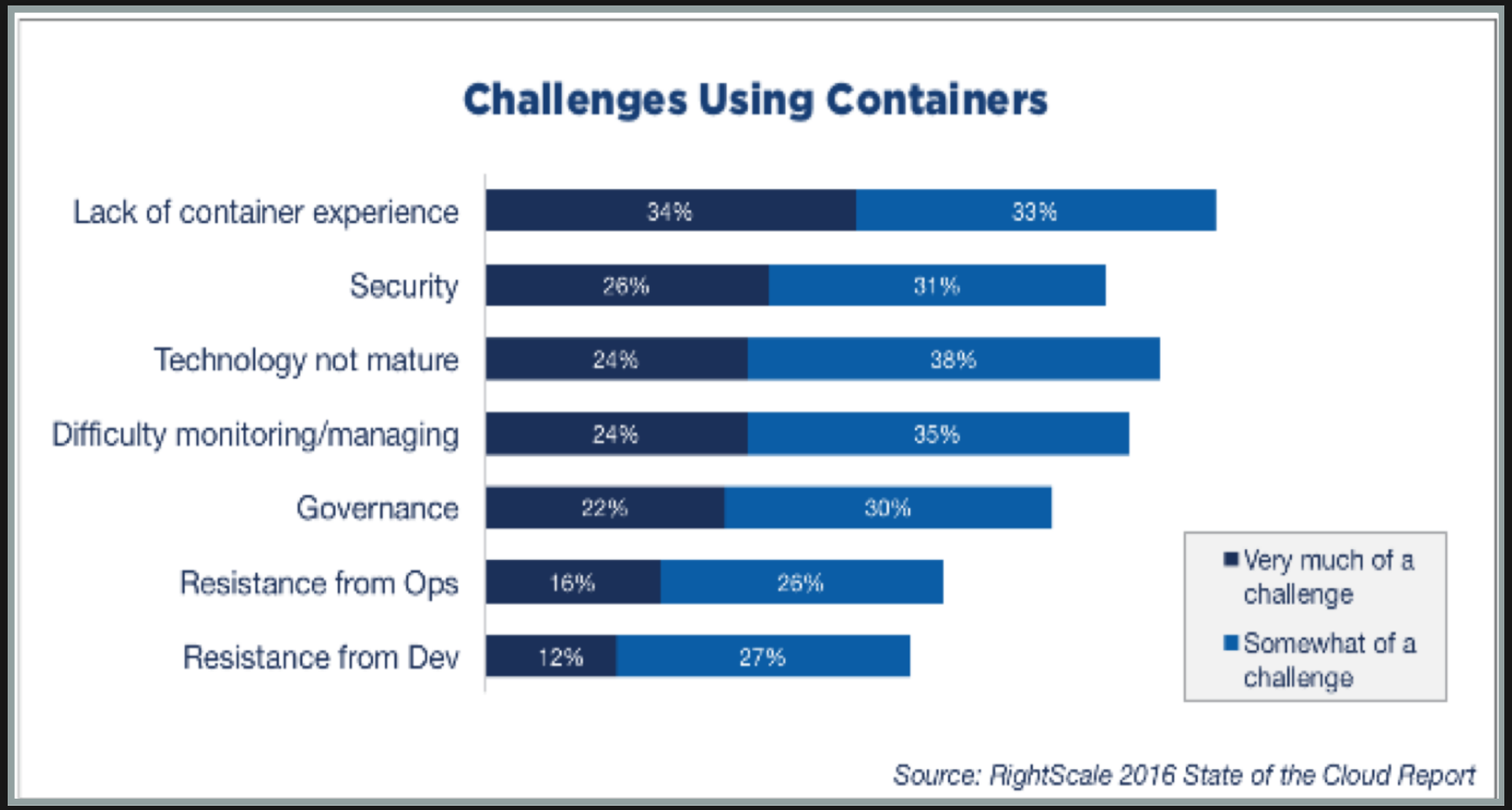

Solution: Intermodal Shipping Container
Multiplicity of Goods
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.
Do I worry about how goods interact (e.g. coffee beans next to spices)
...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another
Can I transport quickly and smoothly (e.g. from boat to train to truck)
Multiplicity of methods for transporting/storing
docker


Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly

TRENDS


Containerization timeline
50%
40%
30%
20%
10%
0
Docker containerization debuted
20% of companies have containers deployed
50% of companies have containers deployed
2013
2014
2015
2016
2017
2018
2019
2020
SOURCE: GARTNER

TRENDS


Challenges Using Containers
Lack of container experience
34%
33%
Security
26%
31%
Technology not mature
24%
38%
Difficulty monitoring/managing
24%
35%
Governance
22%
30%
Resistance from Ops
16%
26%
Resistance from Dev
12%
27%
Very much of a challenge
Somewhat of a challenge
Source: RightScale 2016 State of the Cloud Report

HISTORY

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

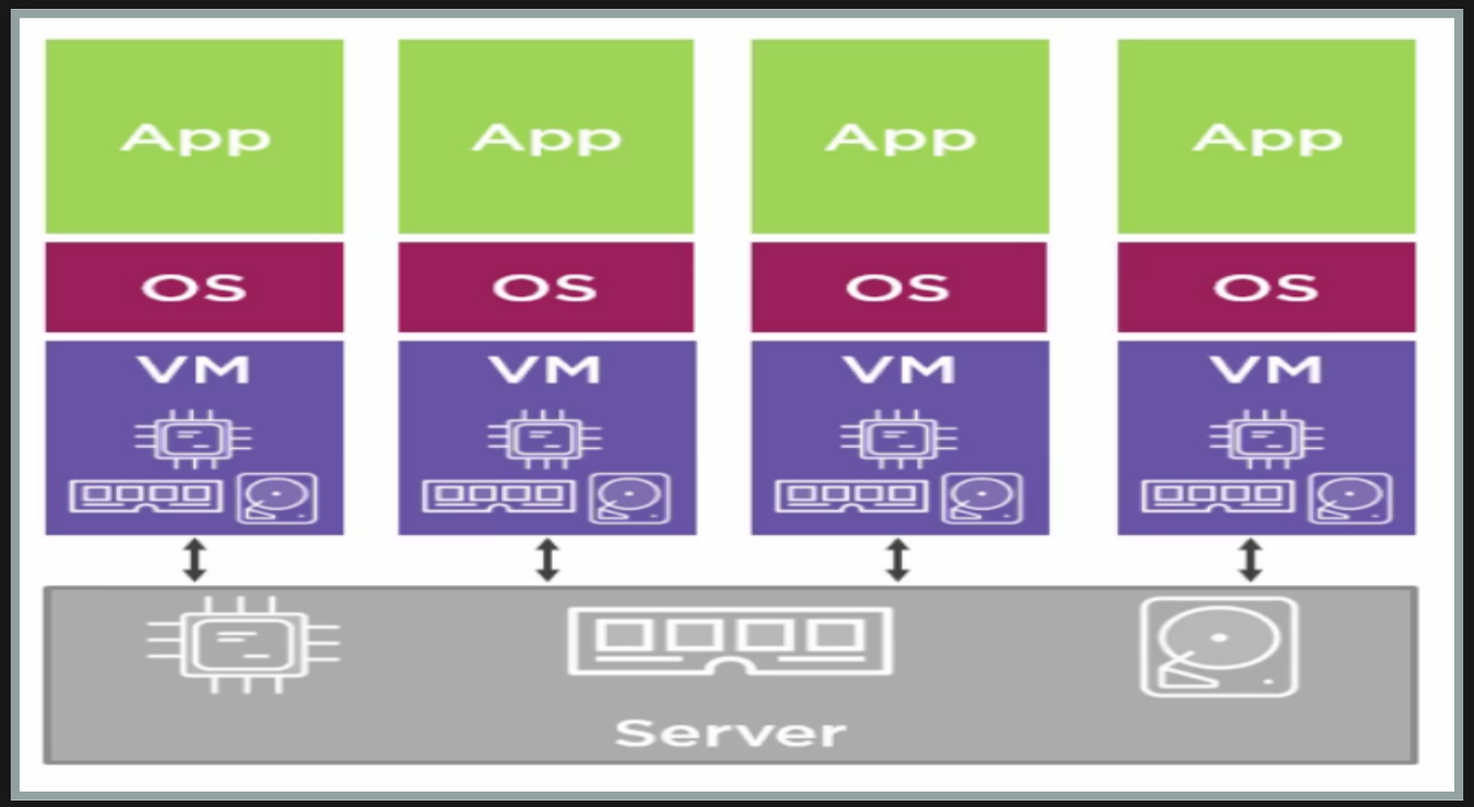
HISTORY

OS TERMS

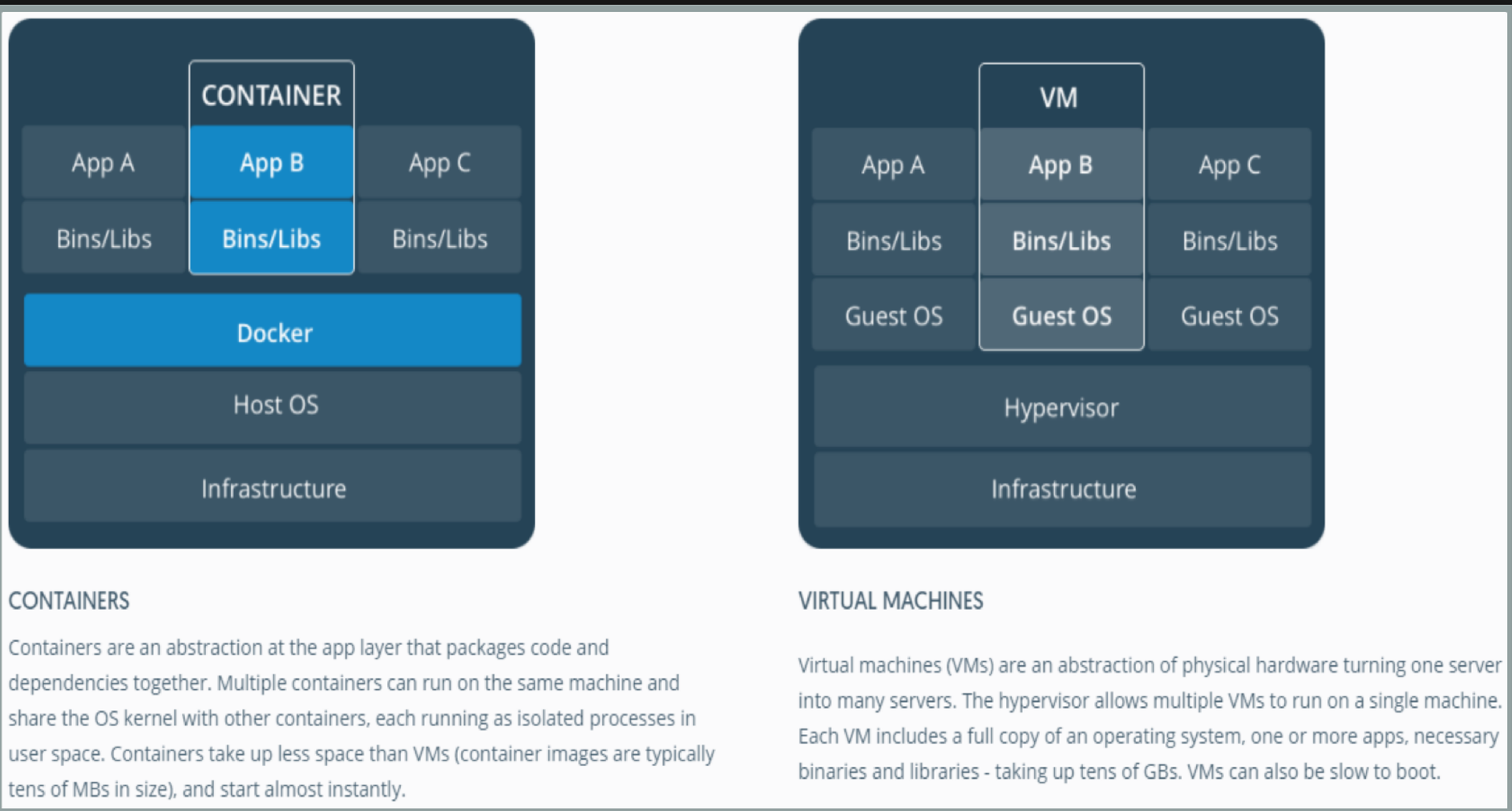
- Host OS
  - For Linux and non-Hyper-V containers, the Host OS shares - its kernel with running Docker containers.
  - For Hyper-V each container has its own Hyper-V kernel.
- Container OS
  - windows containers require a Base OS, while for Linux - containers, its optional.
- Operating System Kernel
  - The Kernel manages lower level functions such as memory - management, file system, network and process scheduling.

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.

CONTAINERS VS VM

- VMs
  - Hardware level virtualization i.e. abstraction of - physical hardware
  - Has own Hardware & 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

CONTAINERS VS VM

- VMs
  - Complete OS, Static Compute, Static Memory, High Resource - Usage
  - 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 VS VM

- 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

- 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

Virtual Machines
(Houses)

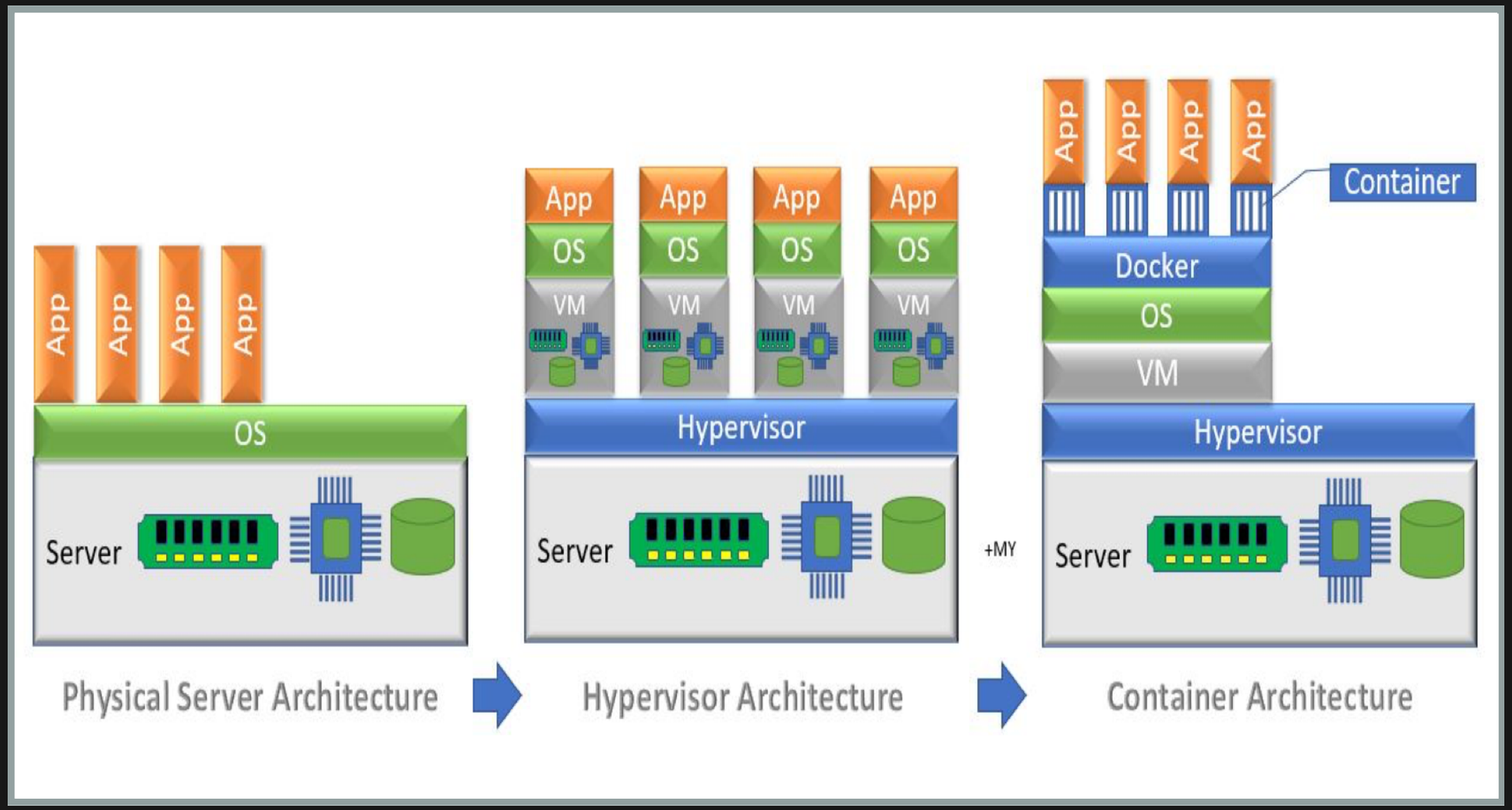
- **Has its own infrastructure**
- Has more necessary things that make it a house, e.g:
  - Roof
  - At least one bedroom
  - Bathroom
  - Kitchen
  - Living area
  - Garage
  - Yard

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

CONTAINERS

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

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

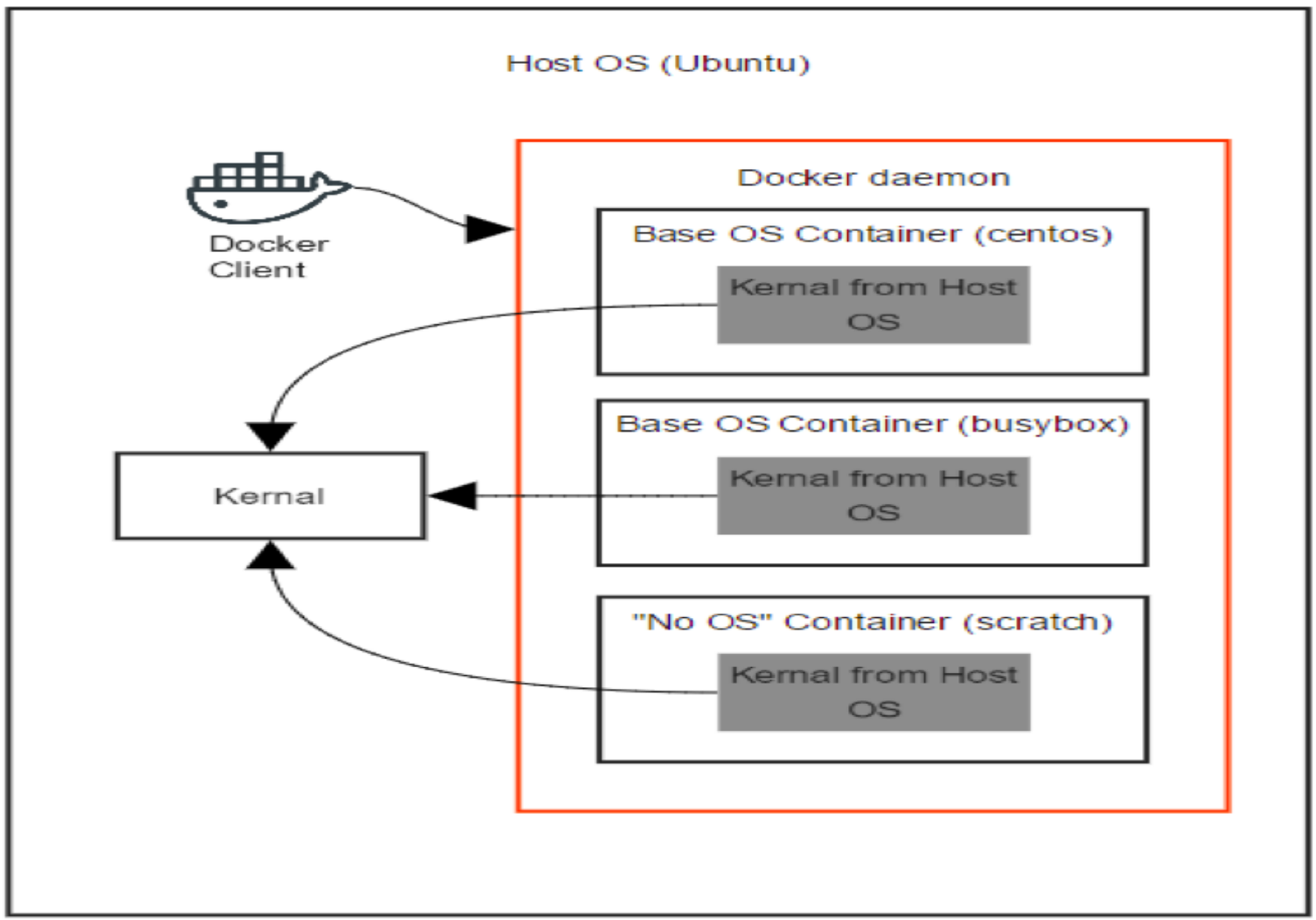
DOCKER

- Open source platform to create containers
- Built for Linux, now runs on Windows and MacOS as well
- Docker Editions
  - Community Edition: Free
  - Enterprise Edition: Paid
- Benefits
  - Isolation
  - Portability
  - Rapid Deployment
  - Security
  - Compatibility and Maintainability
  - Simpler and Faster

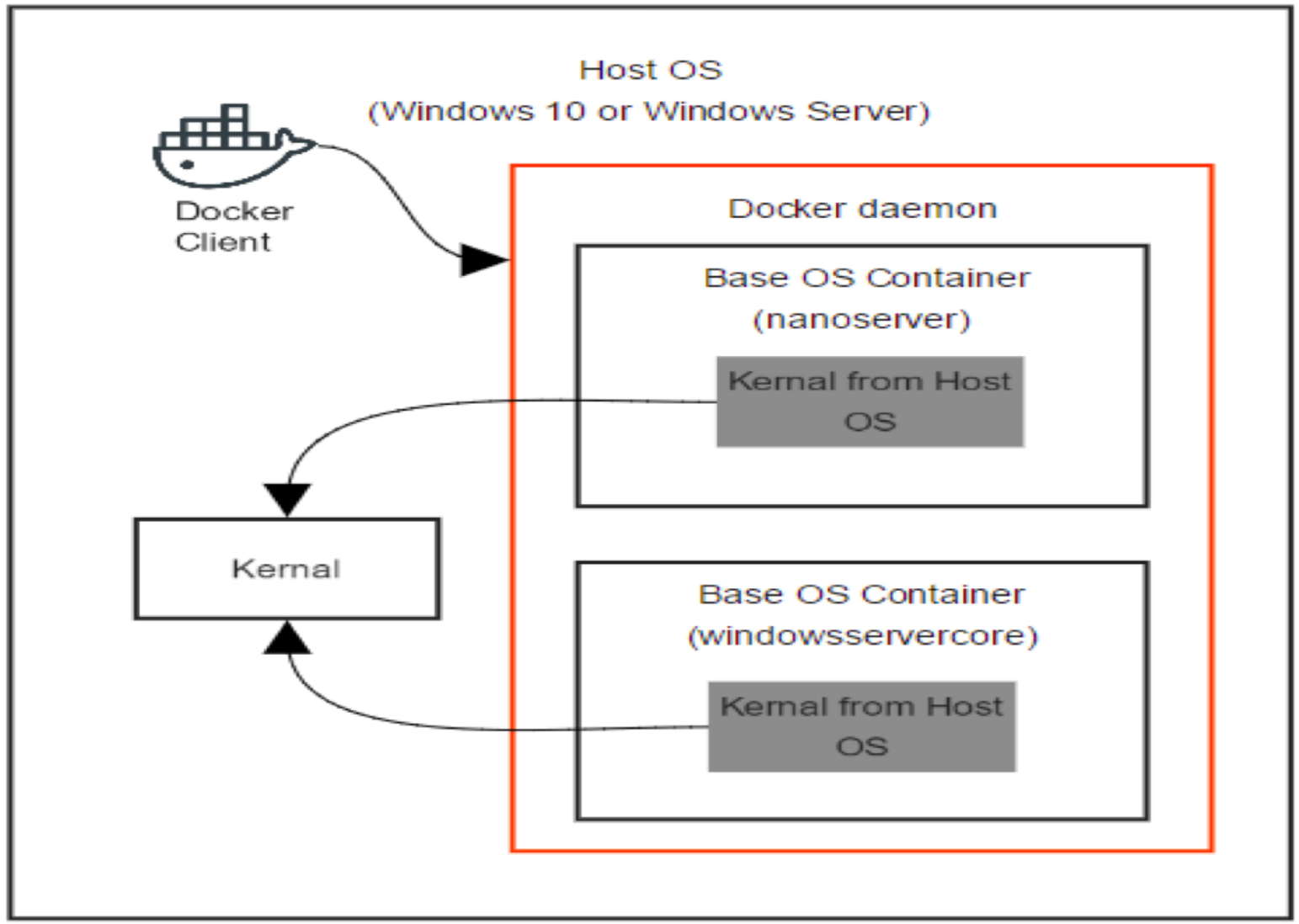
DOCKER

- Docker is a utility that can create, ship and run - containers
- 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



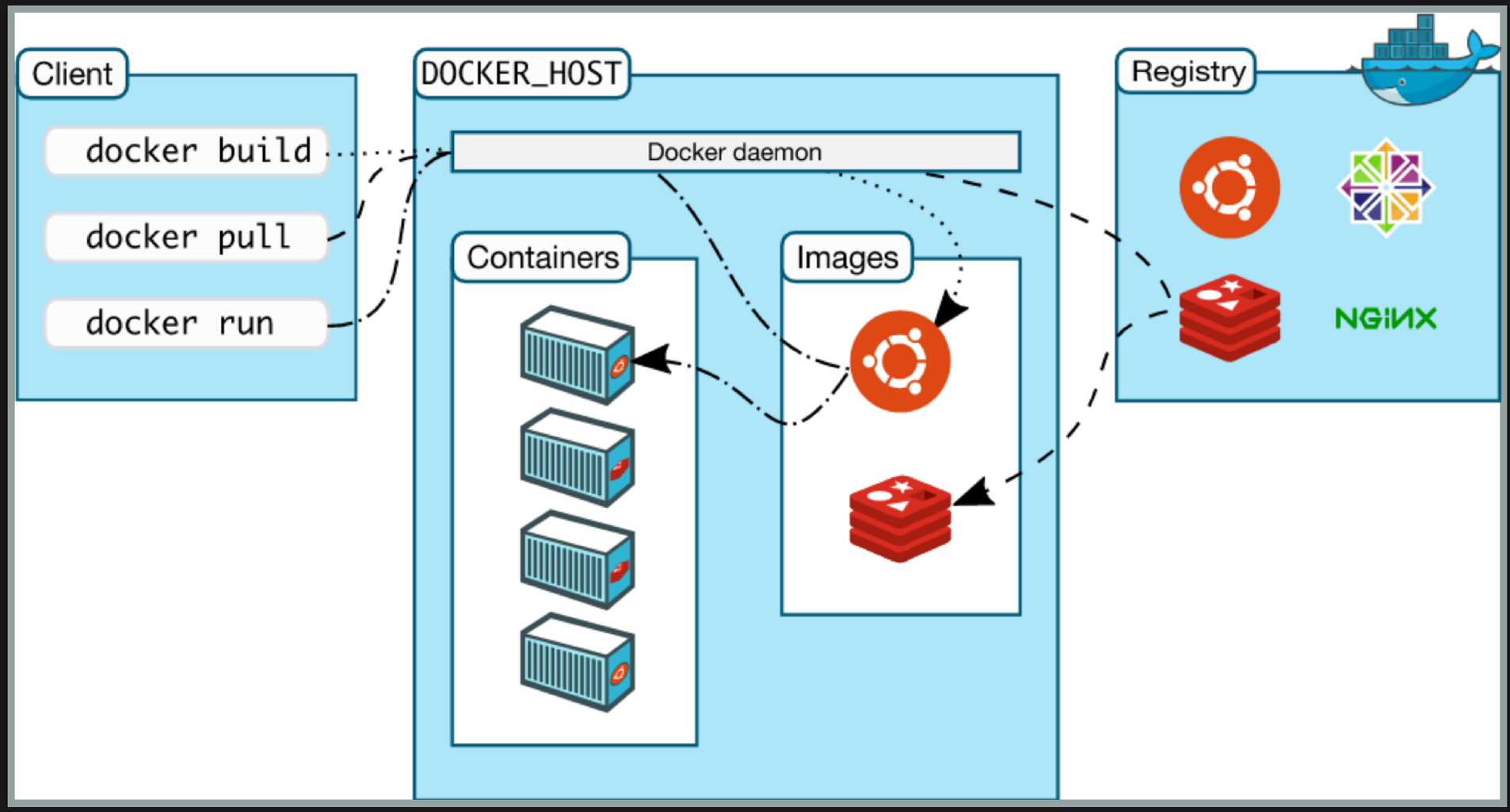
BASIC CONTAINER TERMS

- 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
  - Thinks it has a separate OS
  - 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

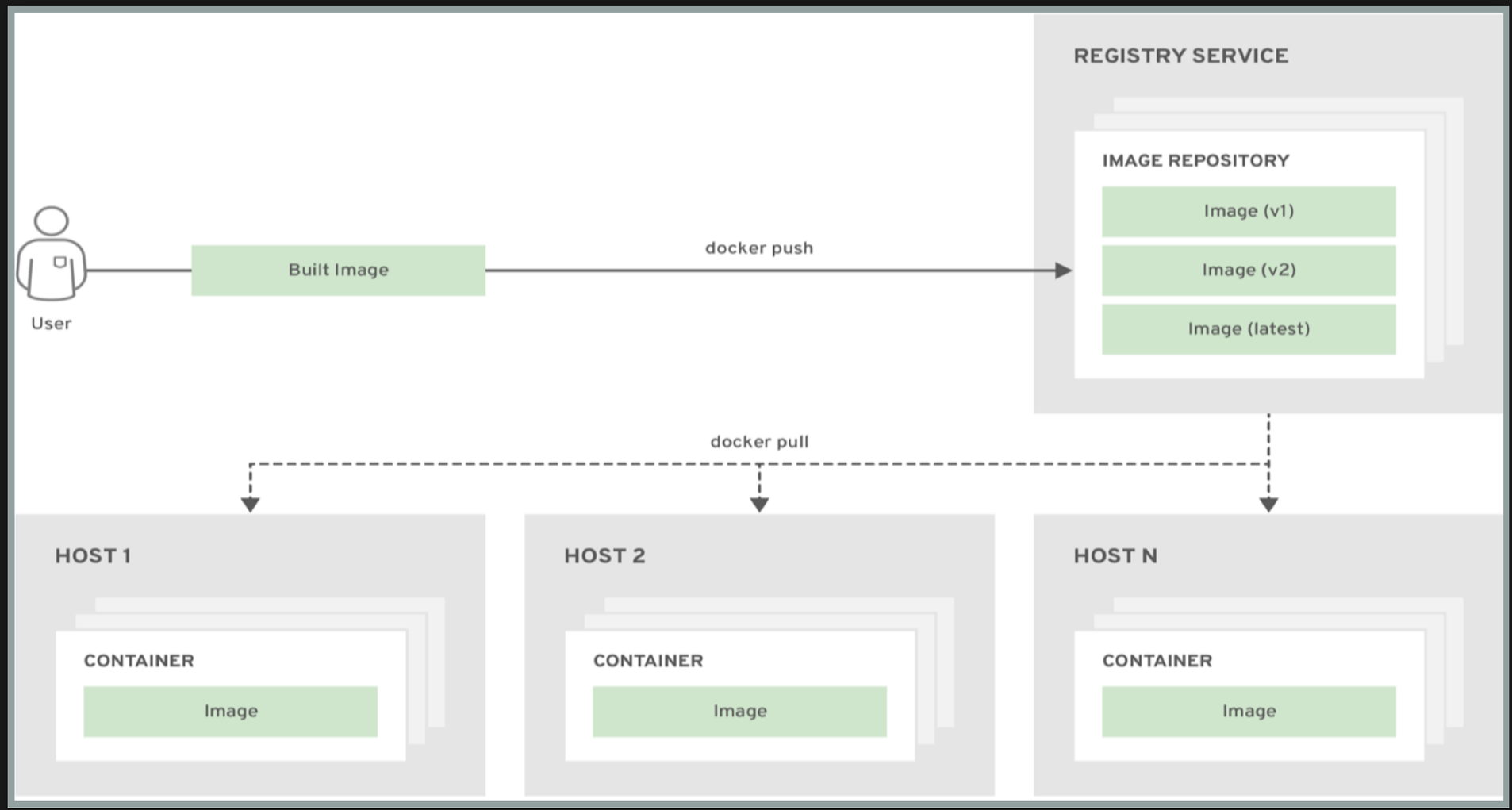
DOCKER ARCHITECTURE



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



INSTALL DOCKER

RUN THE FOLLOWING COMMANDS

- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | - apt-key add -
- add-apt-repository "deb [arch=amd64] https://-download.docker.com/linux/ubuntu $(lsb_release -cs) - stable"
- apt-get update
- apt-cache policy docker-ce
- apt-get install -y docker-ce
- systemctl status docker

RUN YOUR FIRST CONTAINER

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

WHAT HAPPENS IN THE ABOVE COMMAND

- 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
- Creates a new container based on the image and prepares - to start
- Gives it a virtual IP on a private network inside docker - engine
- Opens up port 80 on host and forwards to port 80 on - container
- Starts container

DOCKER COMMANDS

- List running Containers: docker container ls
- List running and stopped Containers: docker container ls - -a
- Verify installation: docker version
- Display configuration: docker info
- Run Container: docker container run --publish - :
- Stop Container: docker container stop
- Delete Container: docker container rm
- Delete Running Container: docker container rm -f -
- Pull Image: docker pull
- List all local images: docker image ls
- Remove local image: docker image rm

INSIDE CONTAINER

- List Process in Container: docker container top -
- Details of Container config: docker inspect -
- Performance Stats for all containers: docker container - stats
- Log of Container: docker container logs

SHELL INSIDE CONTAINER

- Start new container interactively: docker container run - -it centos
- Get shell inside running container: docker container exec - -it
  - e.g. docker container exec -it distracted_darwin /bin/bash

LAB - FIRST CONTAINER

- Run Container
- Run Container in Background
- Pull Image
- What's Going On In Container
- Get Shell Inside Container
- Container Logs
- Stop Containers
- Delete Containers
- Manage Multiple Containers

RUN CONTAINER

DevOps Course by Ali Kahoot

You will run a container from nginx image:

```
$ 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 ls -a
```

RUN CONTAINER IN BACKGROUND

DevOps Course by Ali Kahoot

You will run a nginx container in background:

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

-d specifies to run the container in background 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
```

PULL IMAGE

DevOps Course by Ali Kahoot

You will pull a centos image

```
$ docker pull centos
```

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

```
$ docker pull centos:7
```

WHAT'S GOING ON IN CONTAINER

DevOps Course by Ali Kahoot

List process of a container

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

List details of container config:

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

Display logs of container:

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

Display performance stats of all container:

```
$ docker container stats
```

GET SHELL INSIDE CONTAINER

DevOps Course by Ali Kahoot

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
# display running and stopped containers
$ docker container ls -a
```

Get shell inside running container

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

CONTAINER LOGS

DevOps Course by Ali Kahoot

Get container logs of a running container:

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

STOP CONTAINER

DevOps Course by Ali Kahoot

Stop all running containers:

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

DELETE CONTAINER

DevOps Course by Ali Kahoot

Delete stopped containers:

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

MANAGE MULTIPLE CONTAINERS

DevOps Course by Ali Kahoot

You will run a nginx and httpd container. You 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

Run httpd container:

```
$ docker container run -d publish 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
```

List all containers:

```
$ docker container ls -a
```

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 - app
- Can attach containers to more than one virtual network - (or none)
- Skip virtual networks and use host IP (--net=host)
- 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
- Use --link to enable DNS on default bridge network

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
- Detach a network from container: docker network disconnect

LAB - NETWORKS

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

CREATE NETWORK

Create a new virtual network:

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

By default bridge driver type is used

LIST NETWORK

List all virtual networks:

```
$ docker network ls
```

INTERCOMMUNICATION OF CONTAINERS

You will run two interactive containers and access each other using the specified DNS name.

Run first container:

```
$ 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
```

Access centos1 from shell of centos2 using ping:

```
$ ping centos1
```

Go back to shell of centos1 and run the command:

```
$ ping centos2
```

DELETE CONTAINERS

Delete centos1 and centos2 containers:

```
$ docker container rm centos1 centos2
Or
$ docker container rm centos1
$ docker container rm centos2
```

DELETE NETWORK

Delete virtual network new-network:

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

List network to verify new-network has been deleted

```
$ docker network ls
```

PERSISTENT DATA

- Containers are usually immutable and ephemeral
- What about databases, or unique data?
- Docker gives us a feature "persistent data" to resolve - this
- Two ways:
  - Volumes: make special location outside of container UFS
  - Bind Mounts: link container path to host path

PERSISTENT DATA: VOLUMES

- Override with docker run -v /path/in/container
- Bypasses Union File System and stores in alt location on - host
- Connect to none, one, or multiple containers at once
- By default they only have a unique ID, but you can assign - name then it's a "named volume"

PERSISTENT DATA: BIND MOUNTING

- Maps a host file or directory to a container file or - directory
- Basically just two locations pointing to the same file(s)
- Bypasses Union File System
- run -v /Users/stuff:/path/container (mac/linux)
- run -v //c/Users/stuff:/path/container (windows)

LAB - PERSISTENT DATA

1. Upgrade postgres using Named Volume
2. Setup Apache Server using Bind Mounting

UPGRADE POSTGRES USING NAMED VOLUME

DevOps Course by Ali Kahoot

You will create a named volume, then you will run a postgres version 9.6.1 container and attach that named volume to it, then you will stop this container, next you will run a postgres version 9.6.2 container and attach the named volume to it and check the logs to see to verify postgres has been upgraded.

Create named volume:

```
$ docker volume create psql
```

List volumes:

```
$ docker volume list
```

You should see the psql in the list

Run postgres version 9.6.1 container:

```
$ docker container run -d --name postgres1 -v psql:/var/lib/postgres/data postgres:9.6.1
```

Check logs and verify if postgres is configured:

```
$ docker container logs postgres1
```

You should look for log "LOG: database system is ready to accept connections" Stop the container:

```
$ docker container stop postgres1
```

Run postgres version 9.6.2 container and attach the psql volume to it:

```
$ docker container run -d --name postgres2 -v psql:/var/lib/postgres/data postgres:9.6.2
```

Check logs to verify if the psql volume is attached and database is ready meaning you have successfully upgraded from 9.6.1 to 9.6.2

```
$ docker container logs postgres2
```

You should look for log "LOG: database system is ready to accept connections"

Stop postgres2 container:

```
$ docker stop postgres2
```

Cleanup:

```
$ docker container rm postgres1 postgres2
```

Delete Volume:

```
$ docker volume rm psql
```

SETUP APACHE SERVER USING BIND MOUNTING

DevOps Course by Ali Kahoot

You will run an apache server container and bind mount document root i.e. /usr/local/apache2/htdocs/ to your host directory Run an apache server container:

```
$ docker run -dit --name web-server -p 8080:80 \
-v /host/path/website/:/usr/local/apache2/htdocs/ httpd:2.4
```

Create a html page named index.html

```
$ vi /host/path/website/index.html
```

Add the following content to index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Learn Docker</title>
</head>
<body>
<h1>Learn Docker With Us</h1>
</body>
</html>
```

Next point your browser to localhost:8080/index.html and you should be presented with the page we created
Cleanup Container:

```
$ docker container stop web-server
$ docker container rm web-server
```

Go to the host directory that you mounted and list files, you should see index.html still there

```
$ cd /host/path/website/
$ ls
```

DevOps Course by Ali Kahoot