

# Containerization

# Introduction

Your Name

Background – Development / Infrastructure

Experience on Containers

Experience on Cloud

# Agenda

- Introduction
- Docker Components
- Classroom Environment
- Containers
- Docker – Images
- Docker - Building Images
- Deep Dive – Images
- Deep Dive – Containers
- MicroServices Example
- Container Network Model
- Docker Volumes

# **Session: 1**

## **Introduction**

Before proceeding with the new terms and technologies let's have a look on the history/traditional approaches used for the application since ages.

## What is Container?

# A History Lesson

- The problem got the solution by a technology called “Hypervisor-Based Virtualization”.
- One physical server can contain multiple running applications.
- Each application need a VM to run the application binaries.

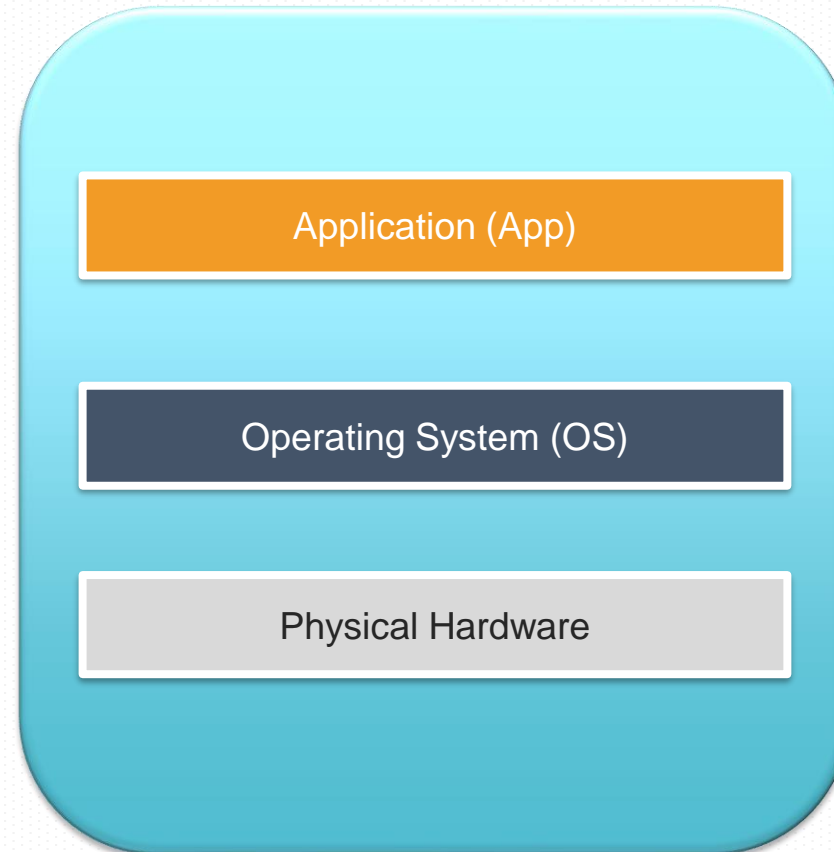
# A History Lesson

- In the traditional ages of development and deployment of application

Unused Resources

Difficult Migrations

Vendor dependency



Slow Deployment time

Difficult to Scale

Huge cost in Infrastructure

**Traditional Approach**

# Virtualization

OverHead

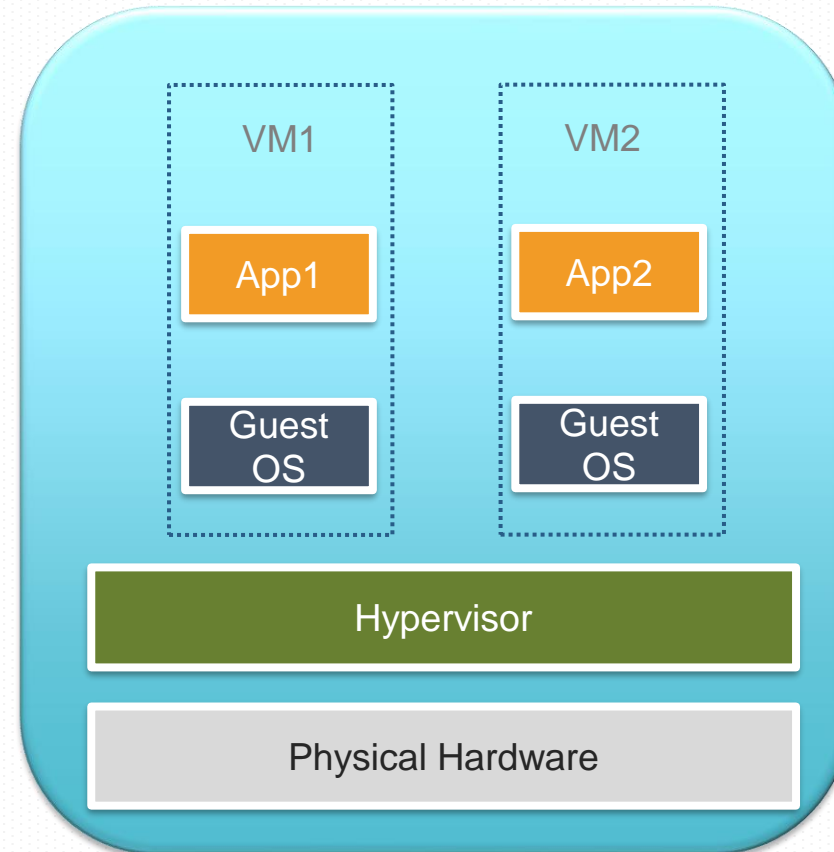
Portability issues

Boot-time still in Minutes

Scaling issue in Hybrid Env

Migrations still failing

Costly Solution



**Server Virtualization**

Better Resource Pooling

Easier to Scale

Flexibility & Easy Migration

Faster Deployments

Faster Boot time



# Containerization

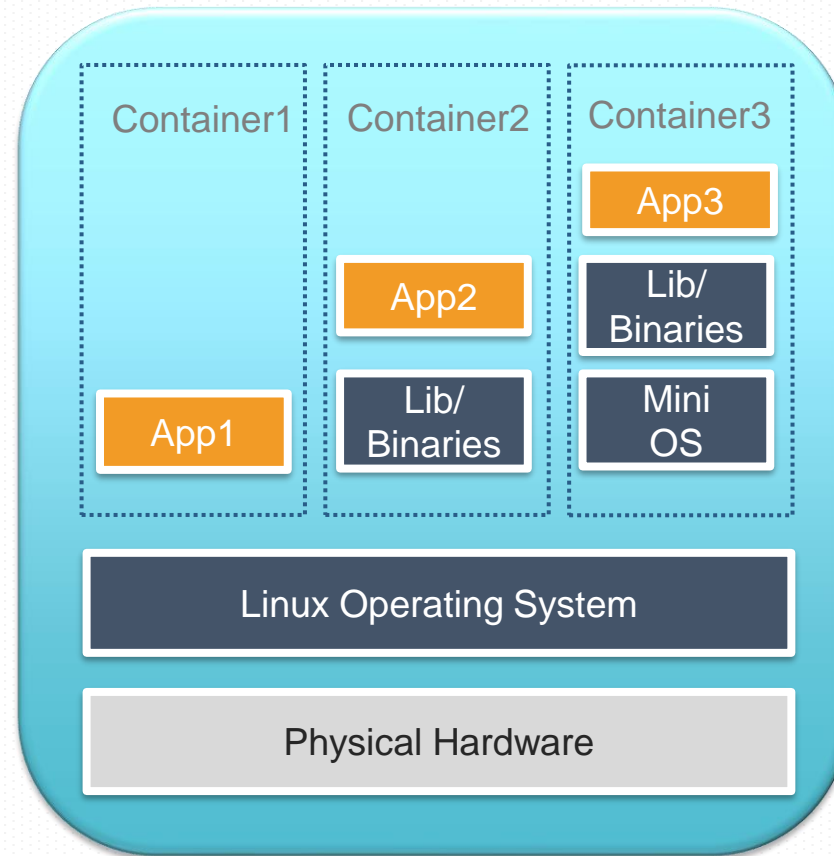
Less OverHead

Highly Portable

Scaling in Hybrid Env

High Migrations success ratio

Cost Effective Solution



**Containerization**

Much Better Resource Pooling

Extended Scaling

Flexibility & Easy Migration

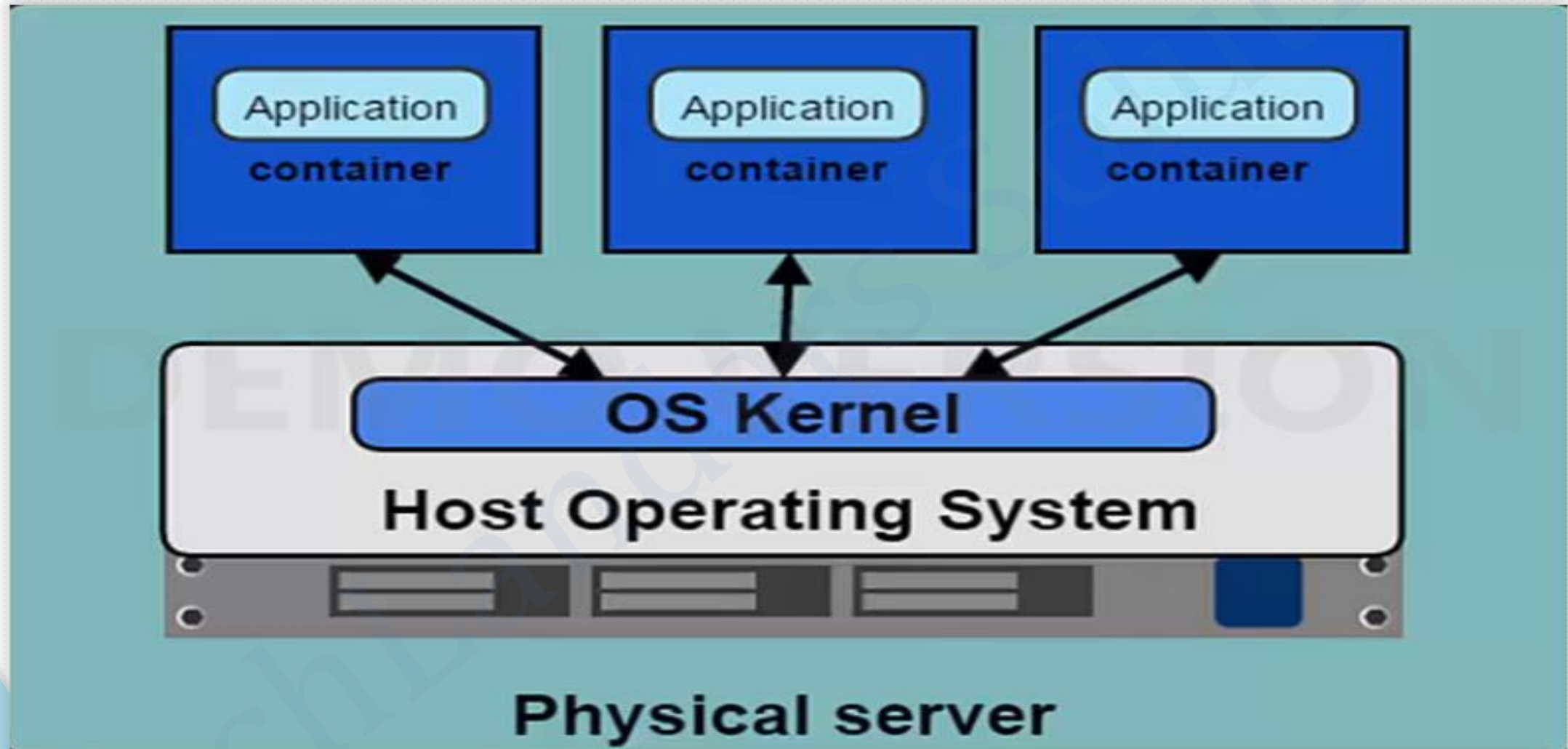
Faster Deployments (Seconds)

Faster Boot time (Seconds)

# Introducing Containers

- Container based virtualization uses the kernel on the host's operating system to run multiple guest instances
- Each guest instance is called a "Container"
- Each container has its own
  - Root Filesystem
  - Processes
  - Memory
  - Devices
  - Network Ports
- From outside its looks like a VM but its not a VM

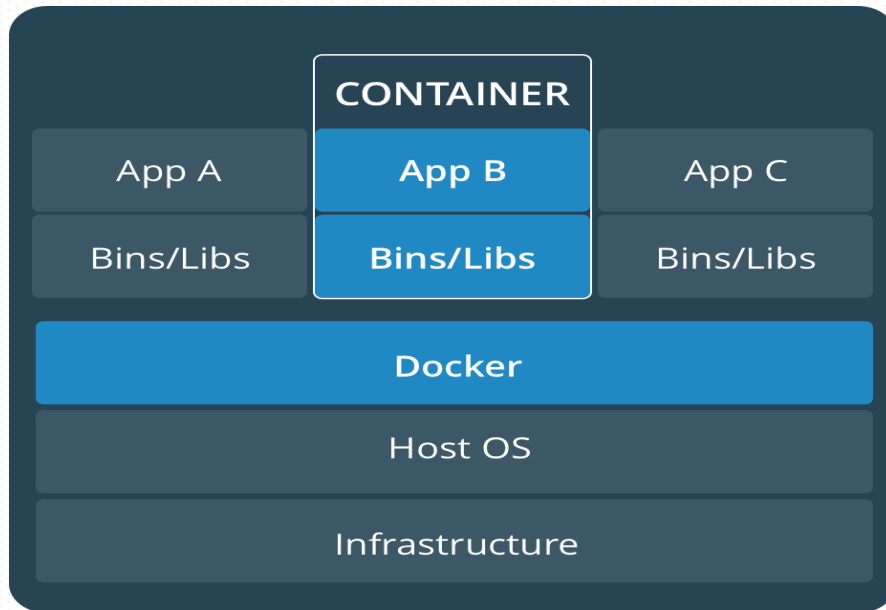
# Overview of Containers



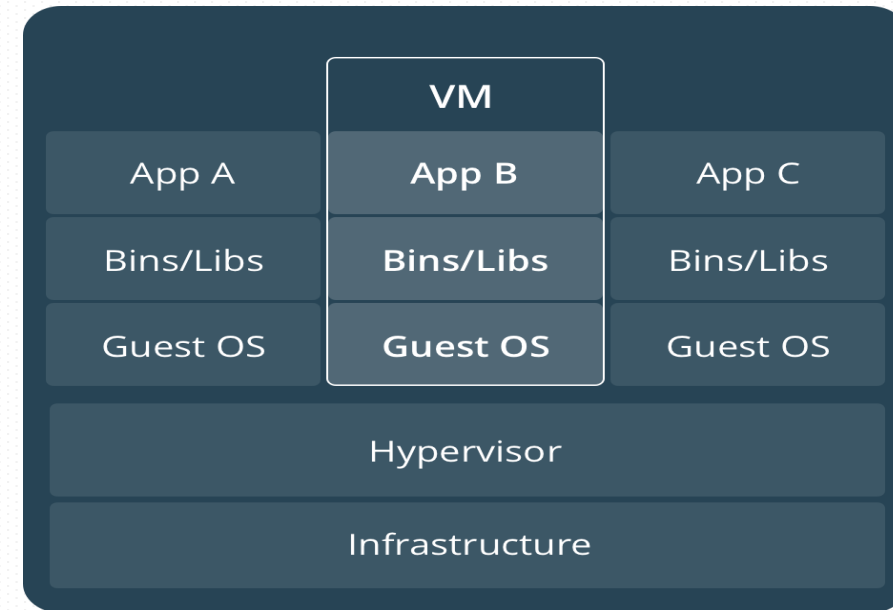
# Containers VS VM's

- Container are more light weight
- No need to install dedicated guest OS, no virtualization like VM is required
- Stop/Start time is very fast
- Less CPU, RAM, Storage Space required
- More containers per machine than VM's
- Great Portability

# Containers VS VM's

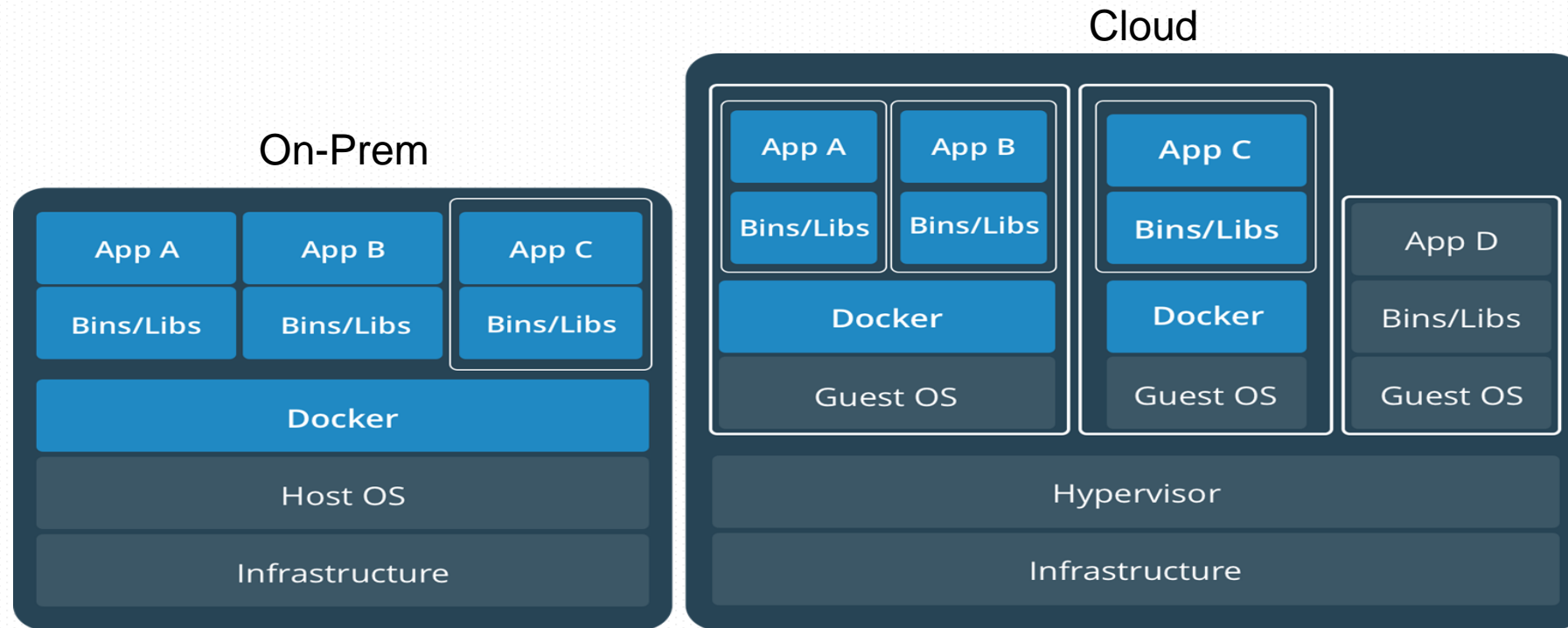


Containers are an app level construct



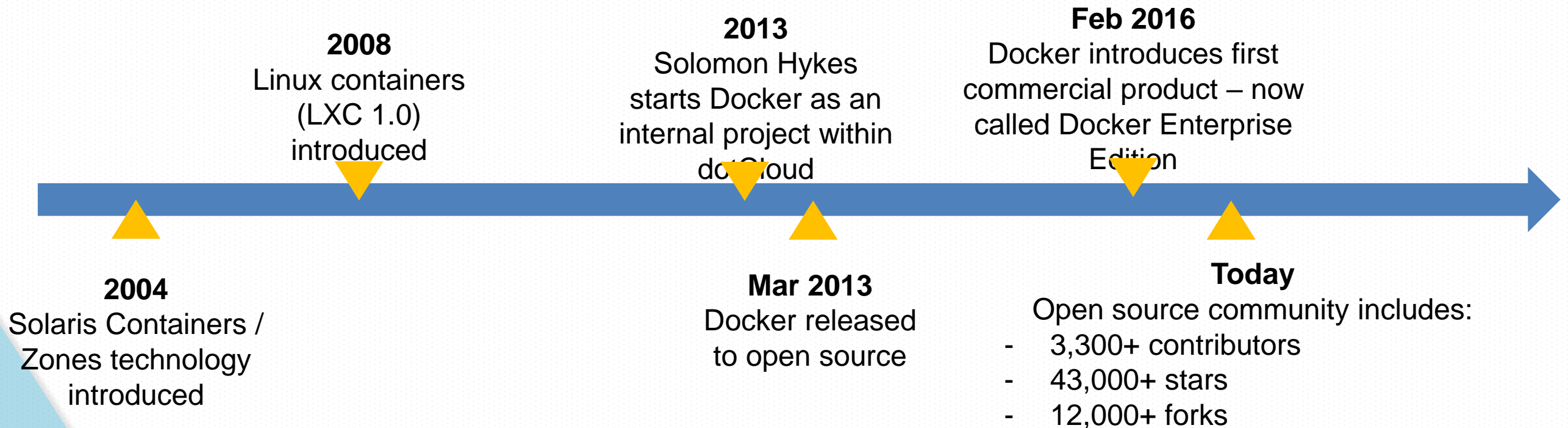
VMs are an infrastructure level construct to turn one machine into many servers

# Containers and VM's together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

# Origins of Docker Project



# Origins of Docker Project

- 'dotCloud' was operating a PaaS platform, using a custom container engine.
- This engine was based on 'OpenVZ' (and later, LXC) and AUFS.
- It started (circa 2008) as a single Python script.
- By 2012, the engine had multiple (~10) Python components. (and ~100 other micro-services!)
- End of 2012, 'dotCloud' refactors this container engine.
- The codename for this project is "Docker."





# About Docker Inc.


- Docker Inc. Formerly 'dotCloud' Inc, used to be a French company
- Docker Inc. is the primary sponsor and contributor to the Docker Project:
  - Hires maintainers and contributors.
  - Provides infrastructure for the project.
  - Runs the Docker Hub.
- HQ in San Francisco.
- Backed by more than 100M in venture capital.


# Deployment Problem


Multiplicity of Stacks

 **Static website**  
`nginx 1.5 + modsecurity + openssl + bootstrap 2`

 **Background workers**  
`Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs`

 **User DB**  
`postgresql + pgv8 + v8`

 **Queue**  
`Redis + redis-sentinel`

 **Analytics DB**  
`hadoop + hive + thrift + OpenDK`

 **Web frontend**  
`Ruby + Rails + sass + Unicorn`

 **API endpoint**  
`Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client`

Do services and apps  
interact  
appropriately?

Multiplicity of  
hardware  
environments

 **Development VM**

 **QA server**

**Customer Data Center**



**Public Cloud**

**Disaster recovery**

**Production Servers**



**Production Cluster**
















**Contributor's laptop**



Can I migrate  
smoothly and  
quickly?



# Matrix Checks

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

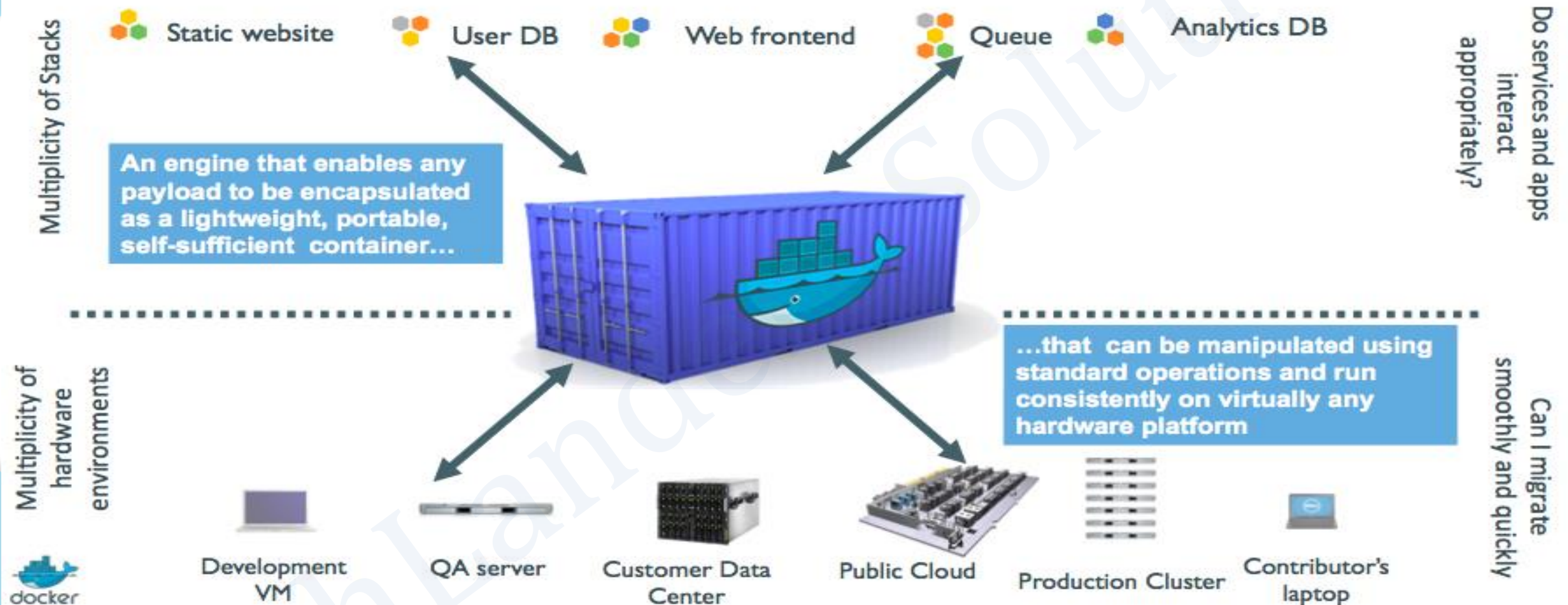


# Intermodal Shipping Containers

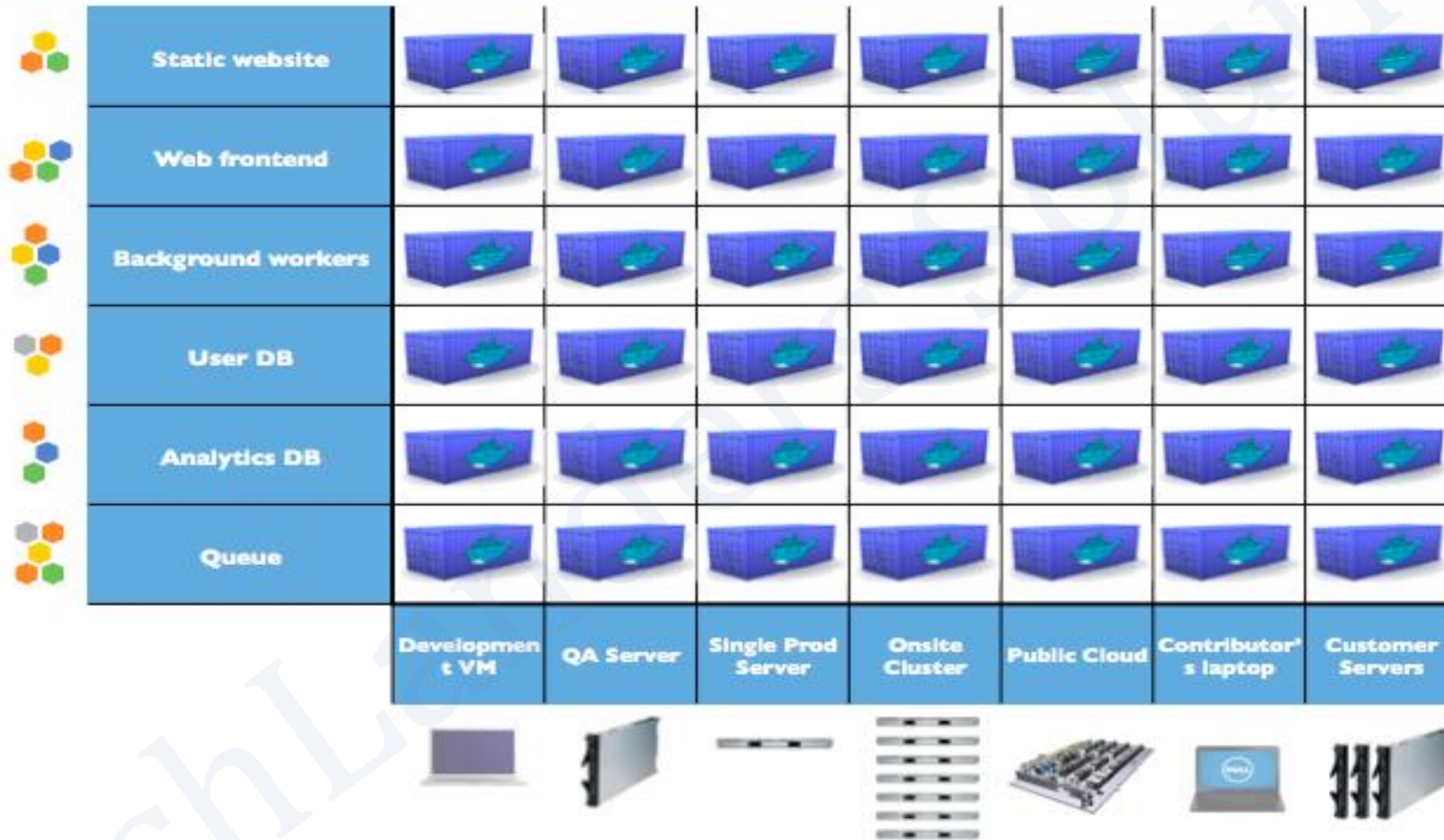




# Shipping Container for Applications



# Eliminate the Matrix



# Results

## Speed

- No OS to boot = applications online in seconds

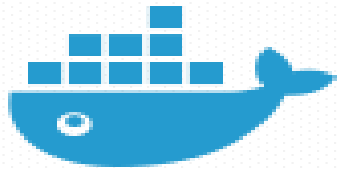
## Portability

- Less dependencies between process layers = ability to move between infrastructure

## Efficiency

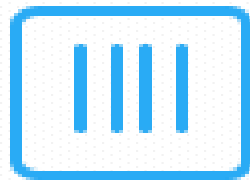
- Less OS overhead
- Improved VM density

# Adoption in Just 4 years



**14M**

Docker  
Hosts



**900K**

Docker  
apps



**77K%**

Growth in  
Docker job  
listings



**12B**

Image pulls  
Over 390K%  
Growth



**3300**

Project  
Contributors



# Why Docker?



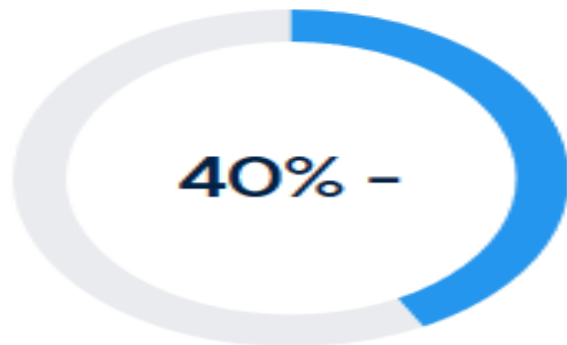
Faster Time to Market



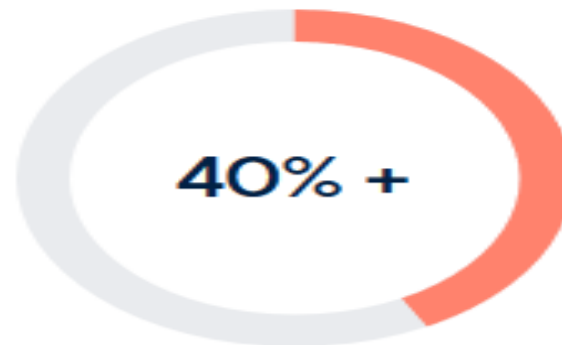
Developer Productivity



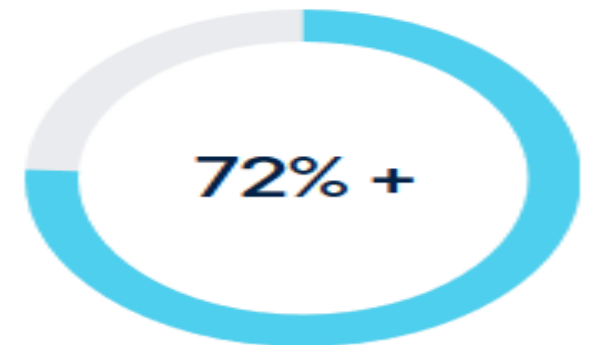
Deployment Velocity



IT Infrastructure Reduction



IT Operational Efficiency



Faster Issue Resolution

# Session: 2

## Docker Components

# Docker Overview

- ▶ Docker is an open platform for developing, shipping, and running applications.
- ▶ Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- ▶ With Docker, you can manage your infrastructure in the same ways you manage your applications.
- ▶ By using Docker's methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

# Docker Platform

- ▶ Docker provides the ability to package and run an application in a loosely isolated environment called a container. The **isolation** and **security** allow you to run many containers simultaneously on a given host.
- ▶ Because of the **lightweight** nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

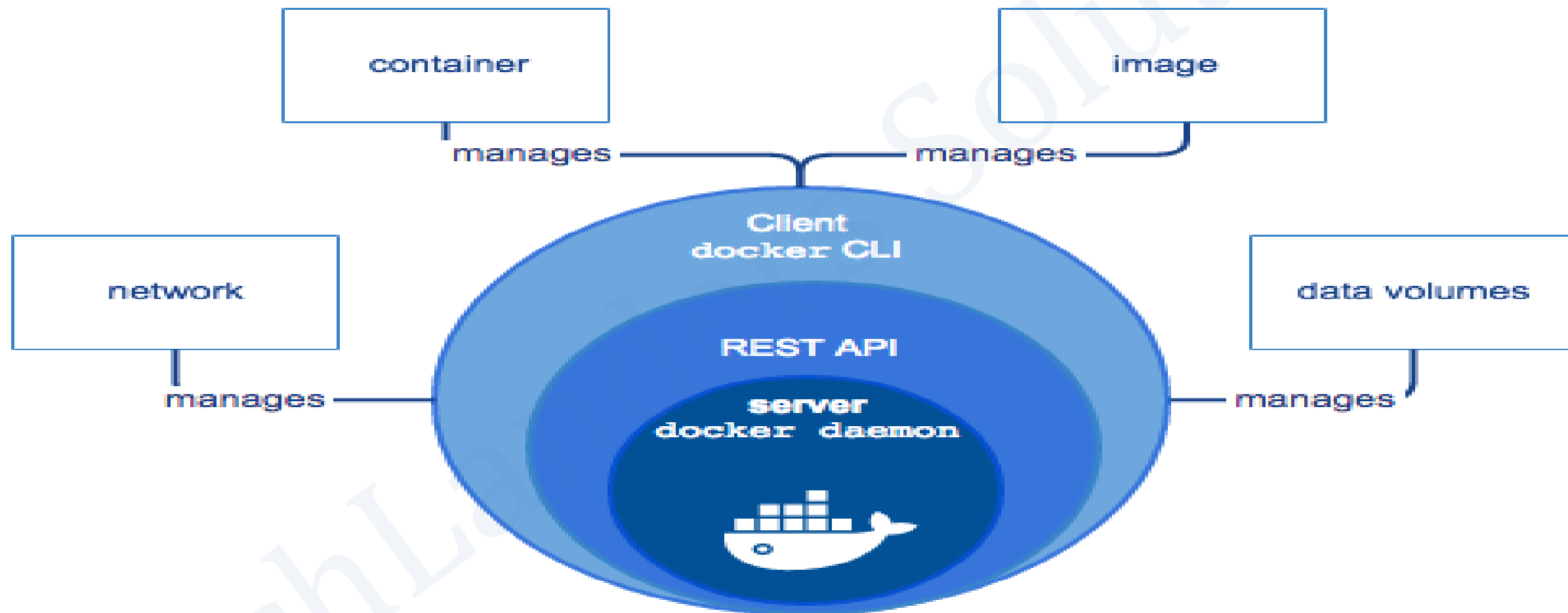
# Docker Test

- ▶ `docker --version`
- ▶ Test the docker functioning: `docker run hello-world`

# Docker Platform

- ▶ Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

# Docker Engine

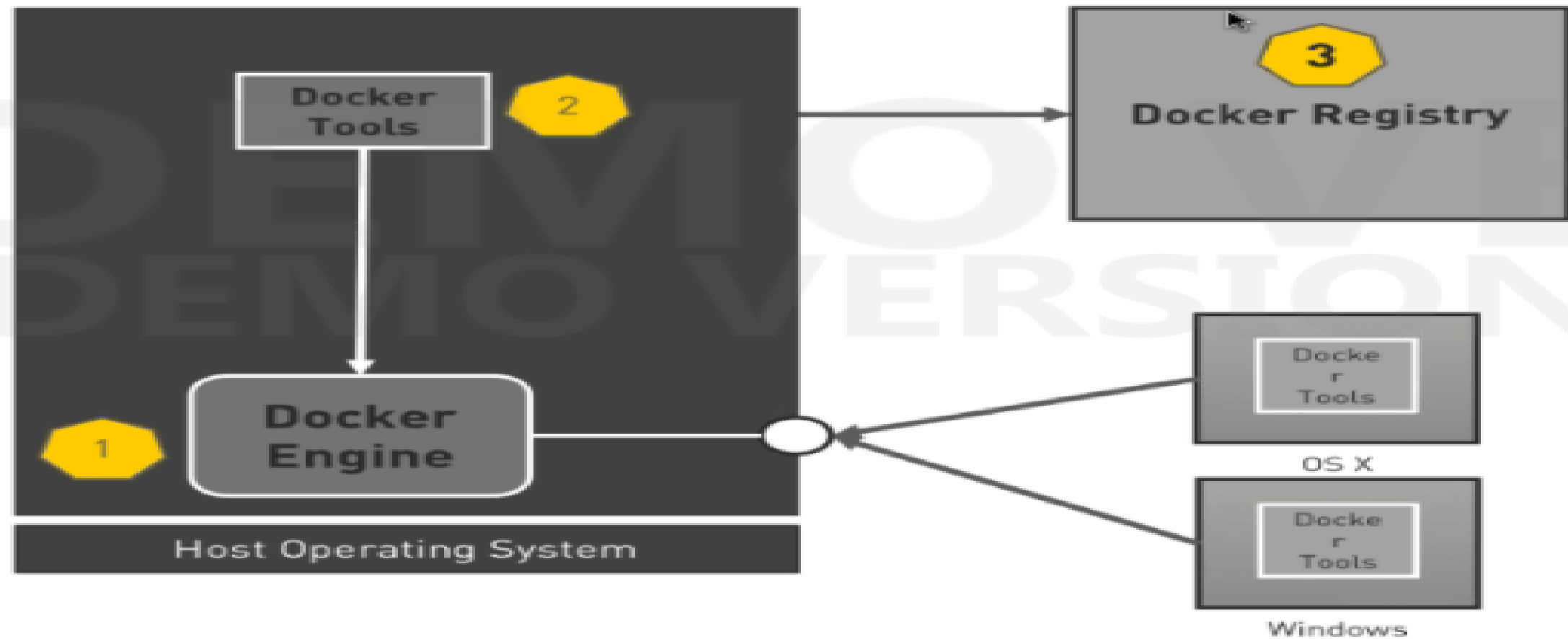


# Docker Engine

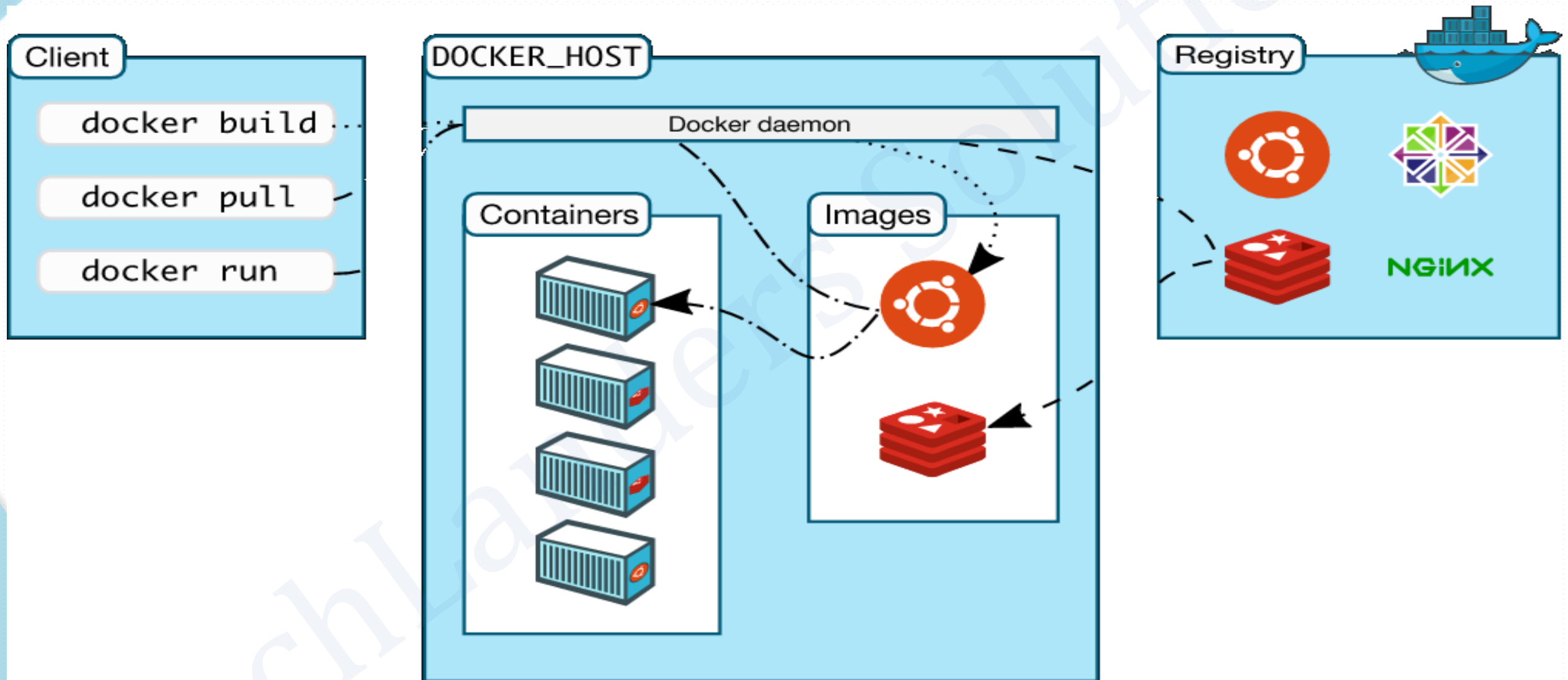
- ▶ Docker Engine is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.
- ▶ The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands
- ▶ The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes



# Docker Architecture



# Docker Architecture



# Docker Architecture

- ▶ Docker uses a client-server architecture.
- ▶ The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- ▶ The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- ▶ The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

# Docker Architecture

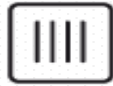
- ▶ The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- ▶ The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

# Docker Architecture



## **Image**

The basis of a Docker container. The content at rest.



## **Container**

The image when it is 'running.' The standard unit for app service



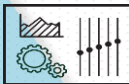
## **Engine**

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.



## **Registry**

Stores, distributes and manages Docker images



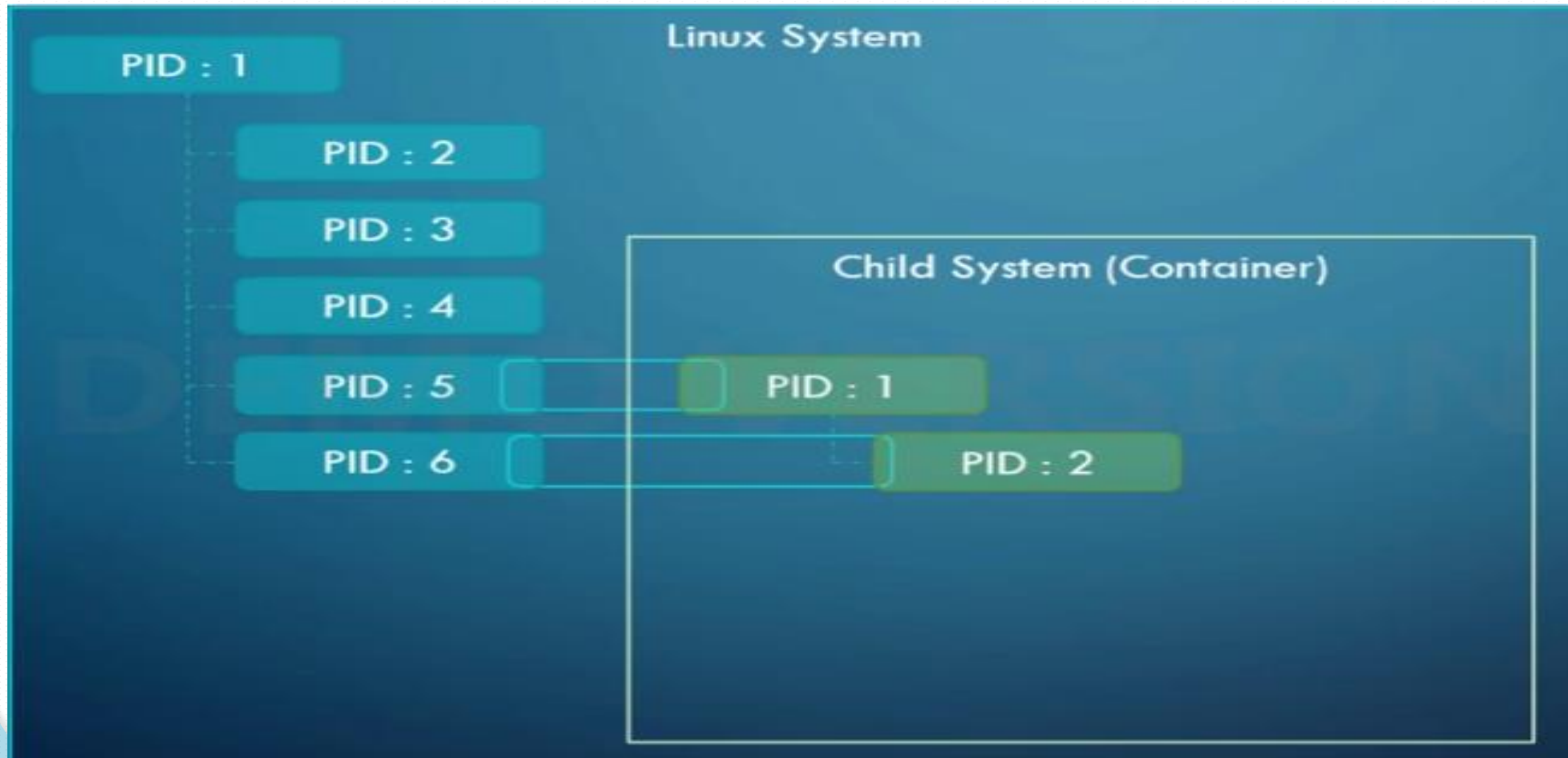
# Docker Architecture

- Lets understand how does applications works in isolation under the hood.
- Docker uses namespace which is a nothing but an isolated environment like VM, but on top of VM called container.



# Docker Architecture

- The demonstration will be given once we understand the container operations.



# Docker Images

- ▶ A Docker image is a read-only template with instructions for creating a Docker container.
- ▶ For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others.
- ▶ A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.
- ▶ Docker images are the build component of Docker.



# Docker Containers

- ▶ A Docker container is a running instance of a Docker image.
- ▶ You can run, start, stop, move, or delete a container using Docker API or CLI commands.
- ▶ When you run a container, you can provide configuration metadata such as networking information or environment variables.
- ▶ Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.
- ▶ Docker containers are the run component of Docker.

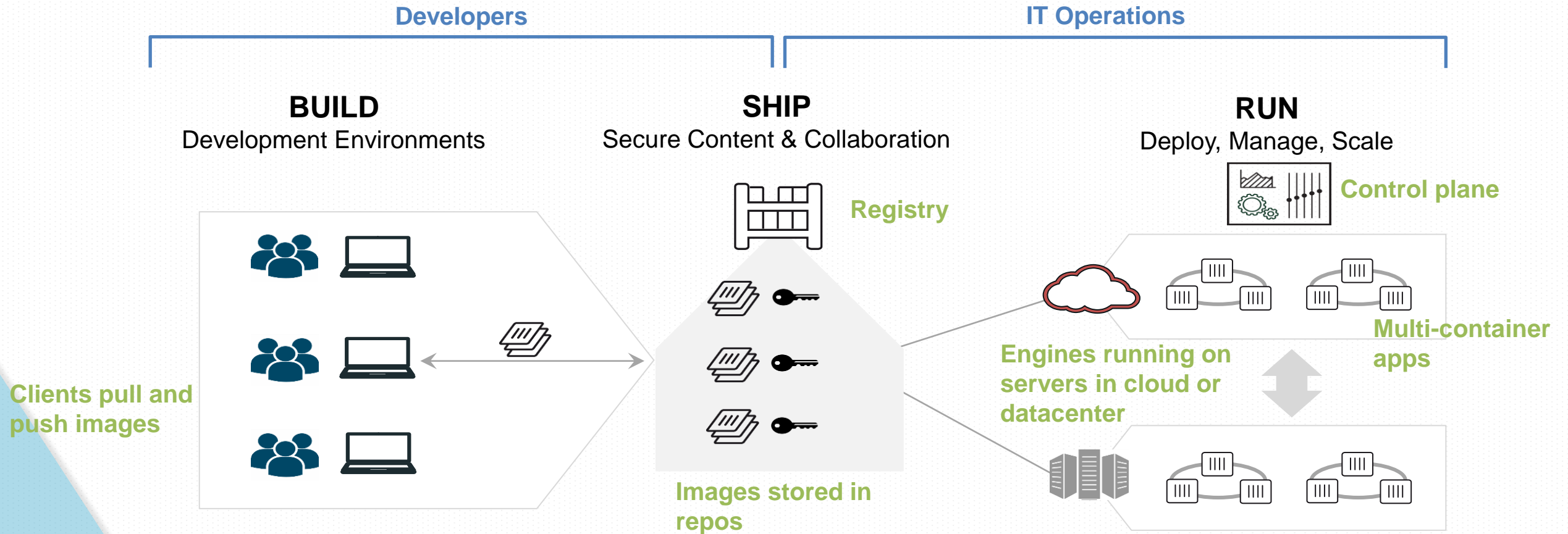
# Docker Registries

- ▶ A docker registry is a library of images.
- ▶ A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
- ▶ Docker registries are the distribution component of Docker.
- ▶ “Docker Hub” is known as global registry.

# Docker Features

- Lightweight
  - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.
- Open
  - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.
- Secure
  - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

# Container as a Service



# Session: 3

## Environment

# Docker Engine Install Demo

- ▶ Docker Engine/Client would be installed on Training Environment as demo LAB.
- ▶ <https://docs.docker.com/engine/installation/linux/centos/>
- ▶ <https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/>

# Docker Installation Key Points

- ▶ docker.service Systemd File:

```
[root@TechLanders lib]# more /usr/lib/systemd/system/docker.service
```

- ▶ Docker Socket file:

```
[root@TechLanders lib]# file /var/run/docker.sock  
/var/run/docker.sock: socket
```

- ▶ Docker PID file and Docker Container PID file (This is Docker Daemon which will have pid1)

```
root@TechLanders run]# cat /var/run/docker.pid  
3715
```

```
[root@TechLanders libcontainerd]# cat /var/run/docker/containerd/docker-containerd.pid  
4251
```

- ▶ Docker Detailed Information:

```
[root@TechLanders libnetwork]# docker info
```

# Manage Docker as a non-root user

- ▶ The docker daemon binds to a Unix socket instead of a TCP port.
- ▶ By default that Unix socket is owned by the user "root" and other users can only access it using sudo.
- ▶ The docker daemon always runs as the root user.
- ▶ If you don't want to use sudo when you use the docker command, add users to Unix group called "docker"  
example: `usermod -aG docker <user-name>`
- ▶ When the docker daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.



# Session: 4

## Containers

# How Container works

- ▶ A container uses the host machine's Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started.
- ▶ Each container is built from an image.
- ▶ The image defines the container's contents, which process to run when the container is launched, and a variety of other configuration details.
- ▶ The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS) in which your application runs.

# How Container works

- ▶ When you use the "docker run" CLI command, the Docker Engine client instructs the Docker daemon to run a container.

```
docker run ubuntu ps ax
```

- ▶ This example tells the Docker daemon to run a container using the centos Docker image, to remain in the foreground in interactive mode (-i), provide a tty terminal (-t) and to run the /bin/bash command.

```
docker run -i -t centos /bin/bash
```

Because if you exit the current running process /bin/bash (pid 1), container will stop/exit. Use "Ctrl pq" to safe exit without stopping the container.

# How Container works

```
[root@TechLanders libcontainerd]# docker run -i -t centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
d9aaf4d82f24: Pull complete
Digest: sha256:eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e
Status: Downloaded newer image for centos:latest
[root@9a06b1a61fc5 /]#
```

```
[root@TechLanders overlay]# ls -lrt /var/lib/docker/image/overlay2/repositories.json
-rw-----. 1 root root 545 Sep 18 03:17 /var/lib/docker/image/overlay2/repositories.json
```

# How Container works

```
[root@TechLanders overlay]# cat repositories.json
```

```
{"Repositories":{"centos":{"centos:latest":"sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768","centos@sha256:eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e":"sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768"},"hello-world":{"hello-world:latest":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37","hello-world@sha256:1f19634d26995c320618d94e6f29c09c6589d5df3c063287a00e6de8458f8242":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37"}}}}
```

```
[root@TechLanders overlay]# docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	latest	<b>196e0ce0c9fb</b>	3 days ago	197MB
hello-world	latest	05a3bd381fc2	5 days ago	1.84kB

```
[root@TechLanders overlay]# docker image inspect centos
```

# More Useful - Container

- ▶ Do something in our container:

Lets suppose we try to use “talk” for communication.

- Let’s check how many packages are installed:

```
rpm -qa | wc -l
```

# A non interactive - Container

- ▶ In your Docker environment, just run the following command:

```
docker run jpetazzo/clock
```

- This container just displays the time every second.
- This container will run forever.
- To stop it, press ^C.
- Docker has automatically downloaded the image jpetazzo/clock.

# Run in background - Container

- ▶ Containers can be started in the background, with the -d flag (daemon mode):

```
docker run -d jpetazzo/clock
```

- We don't see the output of the container.
- But don't worry: Docker collects that output and logs it!
- `docker ps -a`
- `docker logs <container-id>`
- Docker gives us the ID of the container.



# List Running Containers

- ▶ With `docker ps`, just like the UNIX `ps` command, lists running processes.

`docker ps`

`docker ps -l`

`docker ps -a`

`docker ps -q`

- The (truncated) ID of our container.
- The image used to start the container.
- That our container has been running (Up) for a couple of minutes.
- Now, start multiple containers and use “`docker ps`” to list them.

# Stop our Container

- There are two ways we can terminate our detached container.
  - Killing it using the docker “kill” command.
  - Stopping it using the docker “stop” command.
- The first one stops the container immediately, by using the KILL signal.
- The second one is more graceful. It sends a TERM signal, and after 10 seconds, if the container has not stopped, it sends KILL.

# Removing Container

- Let's remove our container:

```
docker rm <yourContainerID>
```

# Attaching to a Container

- You can attach to a container:  
  
    `docker attach <containerID>`
- The container must be running.
- *There can be multiple clients attached to the same container.*

# Restarting a Container

- When a container has exited, it is in stopped state.
- It can then be restarted with the “start” command.

`docker start <containerID>`

- The container must be running.
- You can also use restart command  
`docker restart <container-id>`

# LAB1

- Create one Ubuntu Container in interactive and terminal mode
- Exit out of the container
- Check the status of container
- Restart the container
- Get attached to the container
- Get out of container without exiting ,, stop the container
- Remove the container
- Create the container now in detached mode and follow the same steps ..

# LAB2

- Create a new nginx container using ( -d ) option.
- Docker run -d nginx
- Now check the difference using docker ps -a command
- Try to access this container
- Docker attach <container-ID>  
cat /etc/os-release (THIS WILL NOT WORK) (BECAUSE WE ARE NOT USING INETRACTIVE IN THE CREATION)
- Use docker exec -it <CONATINER-ID> /bin/bash
- Remove everything
- Docker stop
- Docker rm
- Docker run -dt nginx (YOU NEED TO PROVIDE A TERMINAL HERE) → THIS IS TRUE WITH BASE IMAGES
- Docker attach <container-ID>  
cat /etc/os-release

# LAB3

- *On your host machine install httpd package*  
#For centos
  - Yum install httpd -y#For ubuntu :
  - `apt-get install apache2`
- *Verify installation:*  
rpm -qa | grep -i httpd  
#For ubuntu  
find / -name apache2 or `dpkg -list | grep -i apache2`
- *Create a container :*
- #For ubuntu :  
`docker run -it --name c1 ubuntu`  
`docker run -it centos`



# LAB3 cont..

*Check the package inside it:*

```
dpkg --list | grep -i apache
```

\*Install the package if not there :

```
apt-get update -y
```

```
apt-get install apache2
```

```
dpkg --list | grep apache2
```

#For centos :

```
rpm -qa | grep -i httpd
```

•If facing an error while installing httpd in centos container run below :

```
sudo sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
```

```
sudo sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*
```

```
sudo yum update -y
```