Desmistificando Microsserviços e DevOps: Projetando Arquiteturas Efetivamente Escaláveis

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1004] - Seminários em SI 3
https://github.com/IF1004/IF1004

ASSERT Lab
Advanced System and Software Engineering Research Technologies

Centro de Informática UFPE 40 anos

Licença do material

Este Trabalho foi licenciado com uma Licença

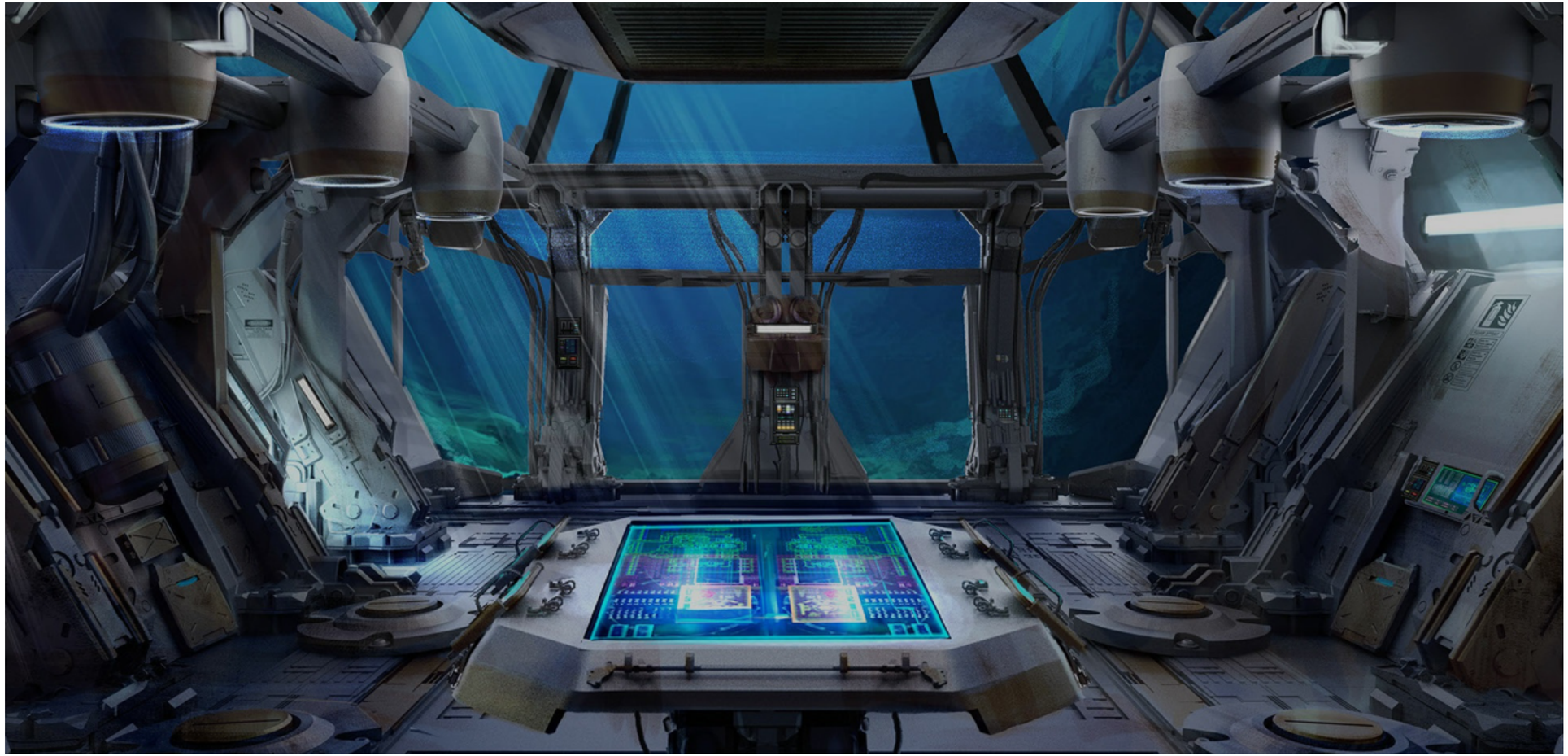
Creative Commons - Atribuição-NãoComercial-Compartilhalgual 3.0 Não Adaptada

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

Resources

- There is no textbook required. However, the following are some books that may be recommended:
  - Docker Fundamentals, 2017 Docker Inc
  - DevOps: A Software Architect's Perspective (SEI Series in Software Engineering)
  - Site Reliability Engineering: How Google runs Production Systems. Edited by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy.
  - DevOps na prática: entrega de software confiável e automatizada
  - Continuous Integration



Containerizing with Docker

What is complexity costing us?

- Dependency conflicts, infrastructure mismatches, and lack of scalability are all examples of problems with
- Standardization and Encapsulation
- Luckily, this problem is not new

Devops circa 1912

Encapsulation

- Encapsulation eliminates friction across
  ture, a rdizat
  scale

Photo Roel Hemkes, CC-BY 2.0

Photo Roel Hemkes, CC-BY 2.0

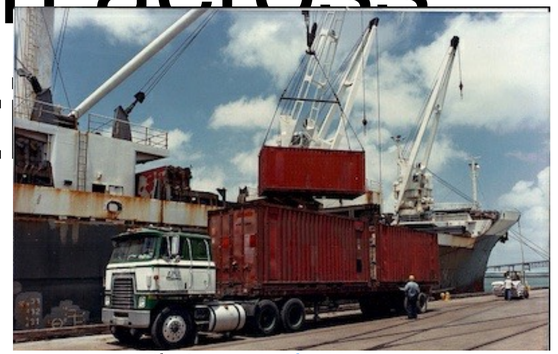
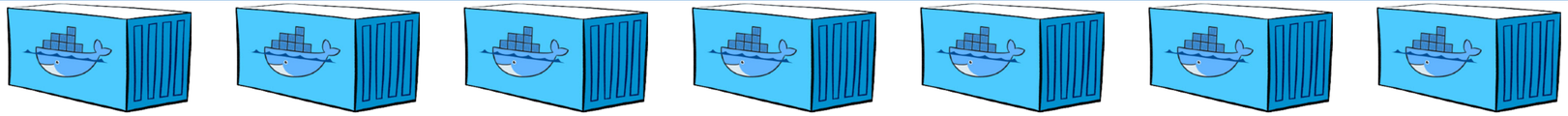
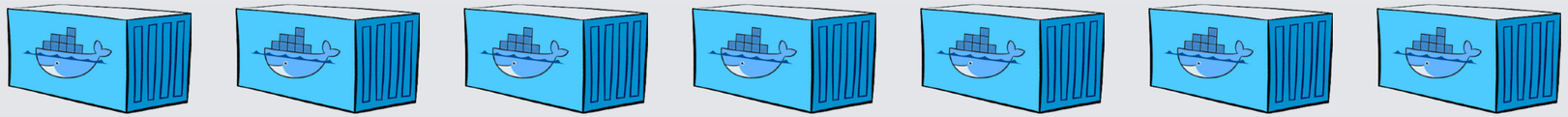
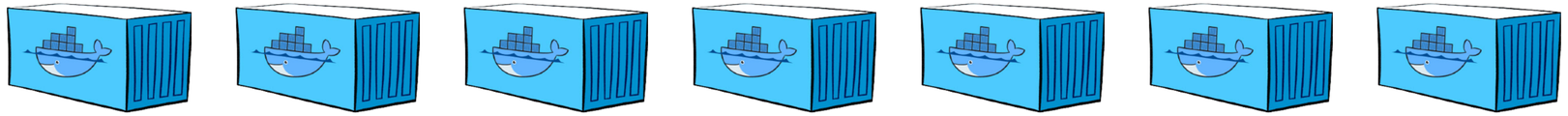
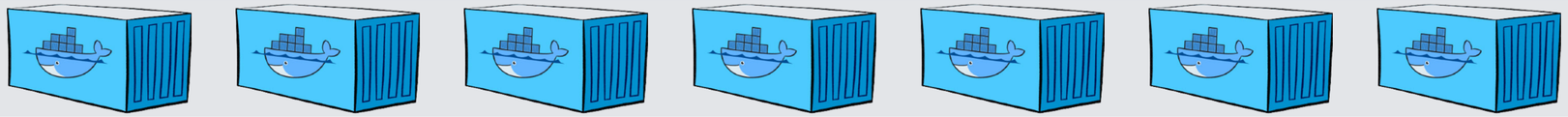
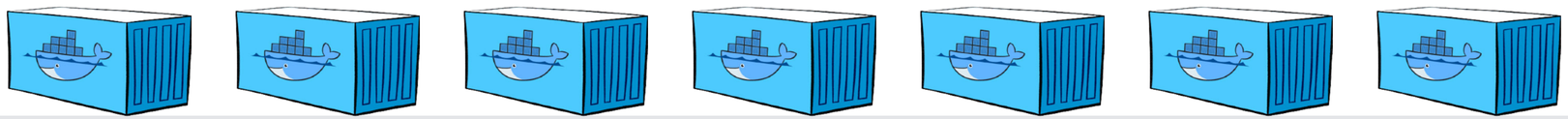
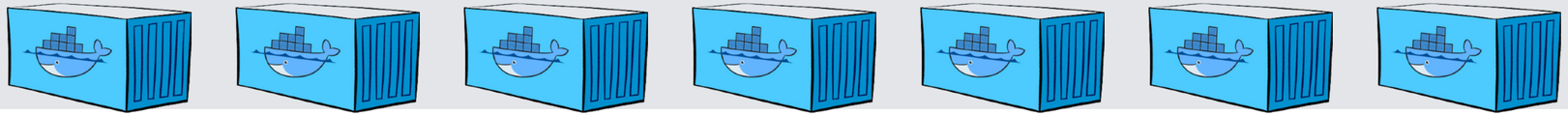


Photo Roy Luck, CC-BY 2.0

Deployment Nightmare

	Development VM	QA Server	Single Pod Server	Onsite Cluster	Public Cloud	Contributor's Laptop	Customer's server
Static Website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Any app, anywhere

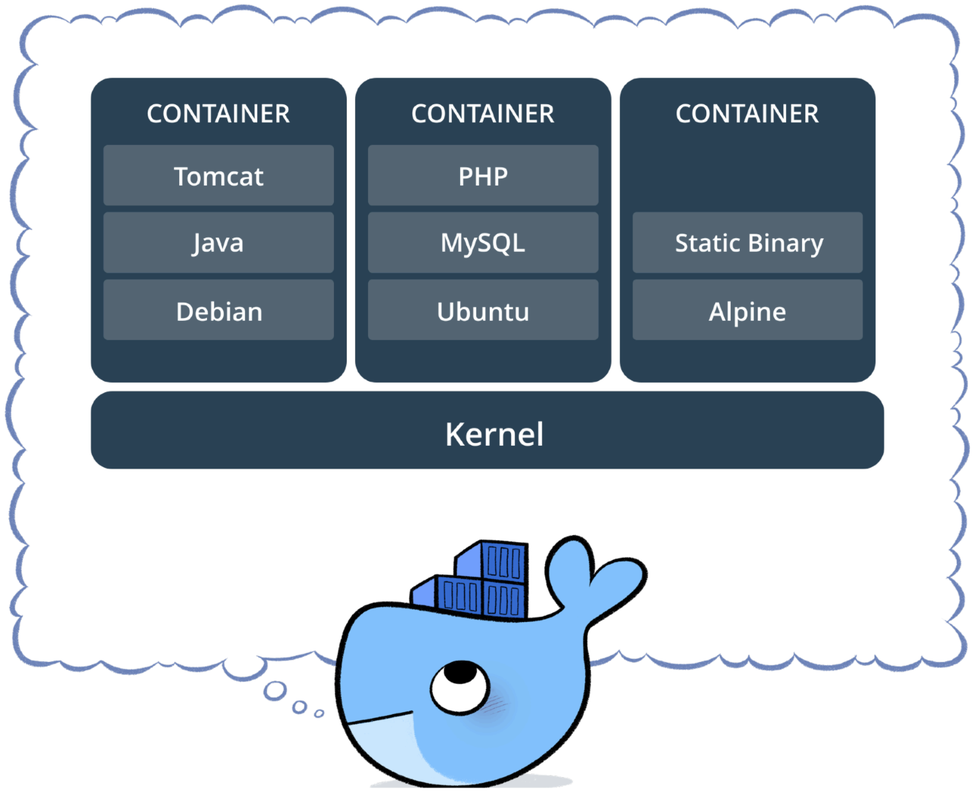
	Development VM	QA Server	Single Pod Server	Onsite Cluster	Public Cloud	Contributor's Laptop	Customer's server
Static Website							
Web frontend							
Background Workers							
User DB							
Analytics DB							
Queue							

What are containers?

- CONTAINERS ARE PROCESSES
- Containers are processes sandboxed by:
  - Kernel namespaces
  - Root privilege management
  - System call restrictions
  - Private network stacks
  - etc

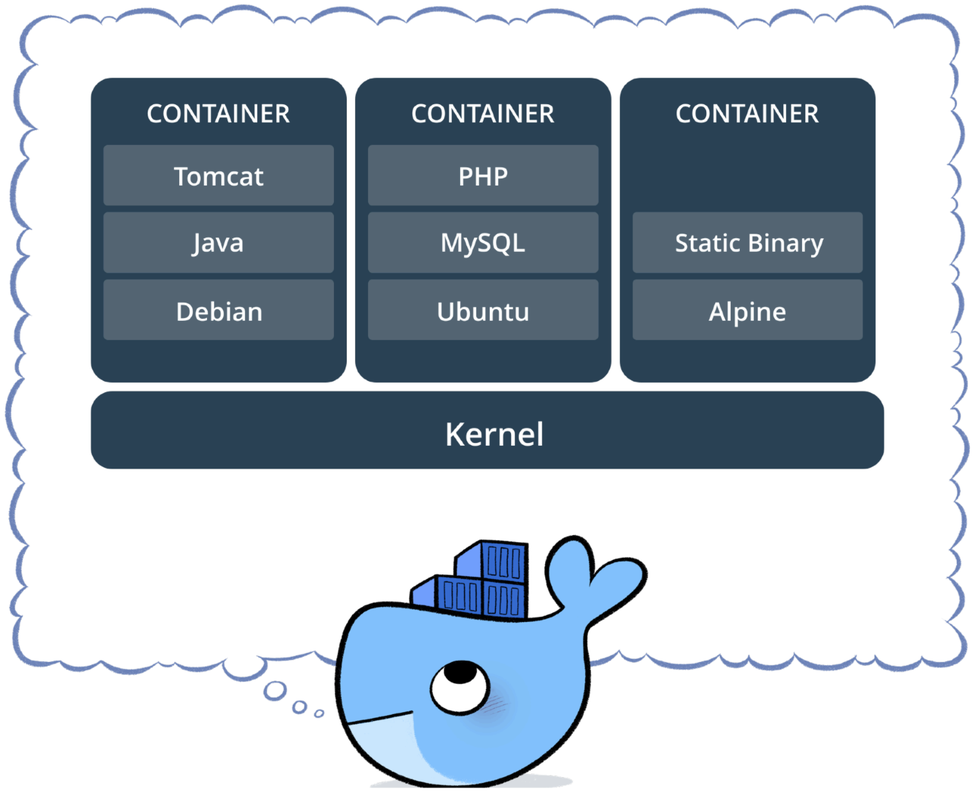
What are containers?

- Containers provide private spaces on top of the operating system
- This technique is also called operating system virtualization
  - The kernel of the operating system provides isolated virtual spaces
  - Each of these virtual spaces is called a container or virtual engine (VE)
- Containers allow processes to run on an isolated environment on top of the host operating system



What are containers?

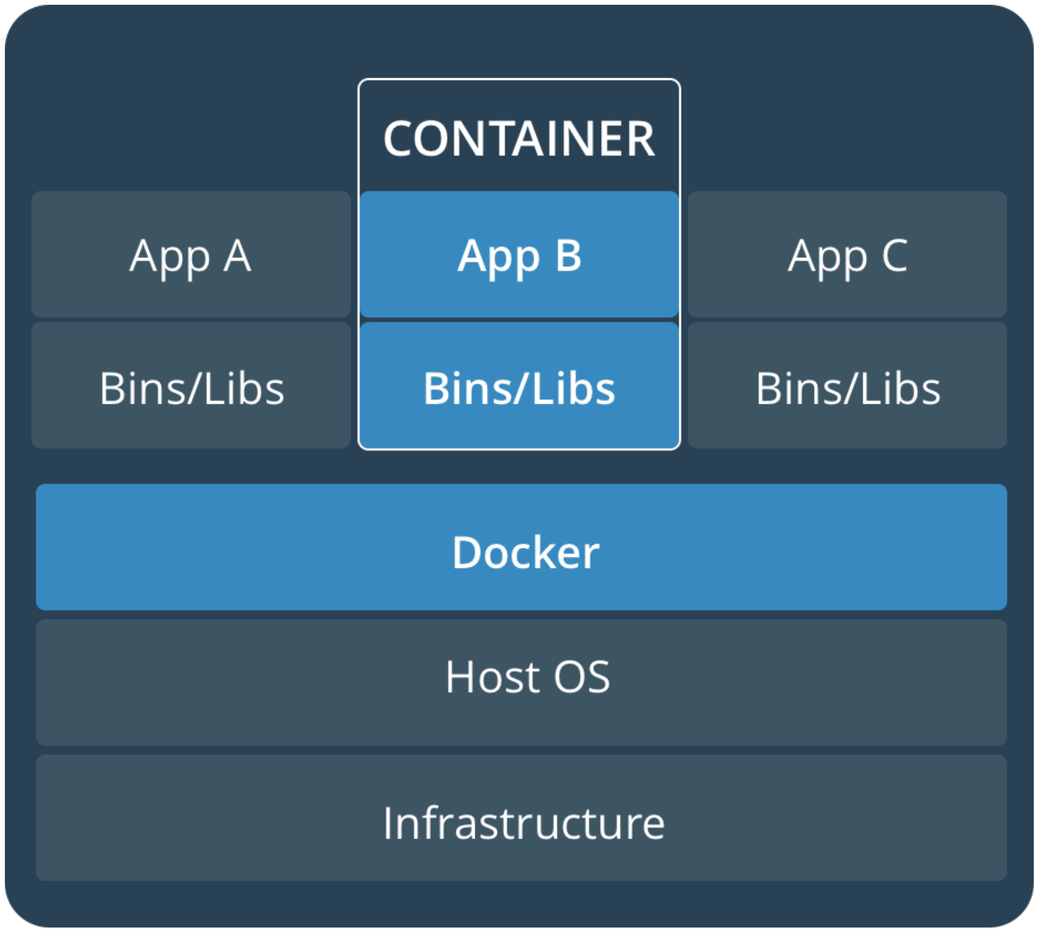
- Containers are easy mechanisms to build, ship, and run compartmentalized software components
- Generally, containers package all the binaries and libraries that are essential for running an application
- Containers reserve their own filesystem, IP address, network interfaces, internal processes, namespaces, OS libraries, application binaries, dependencies, and other application configurations

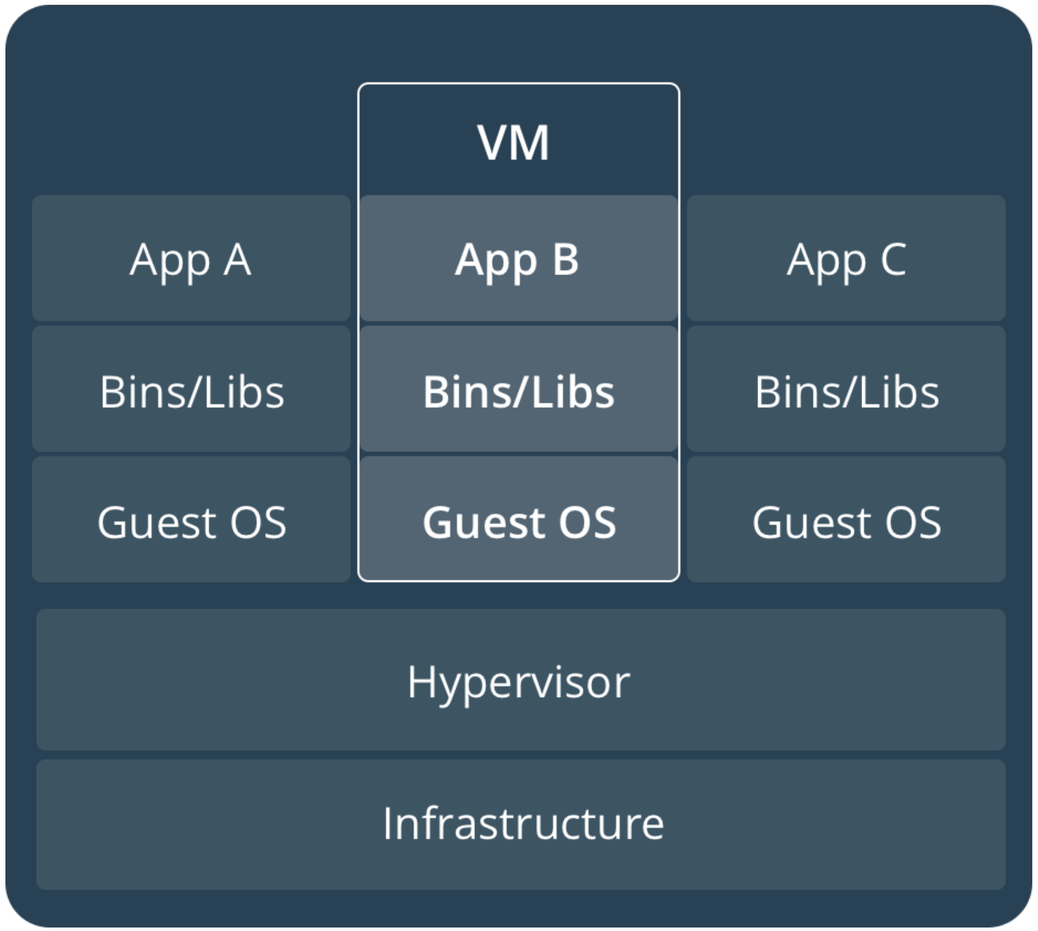


Big white shark?

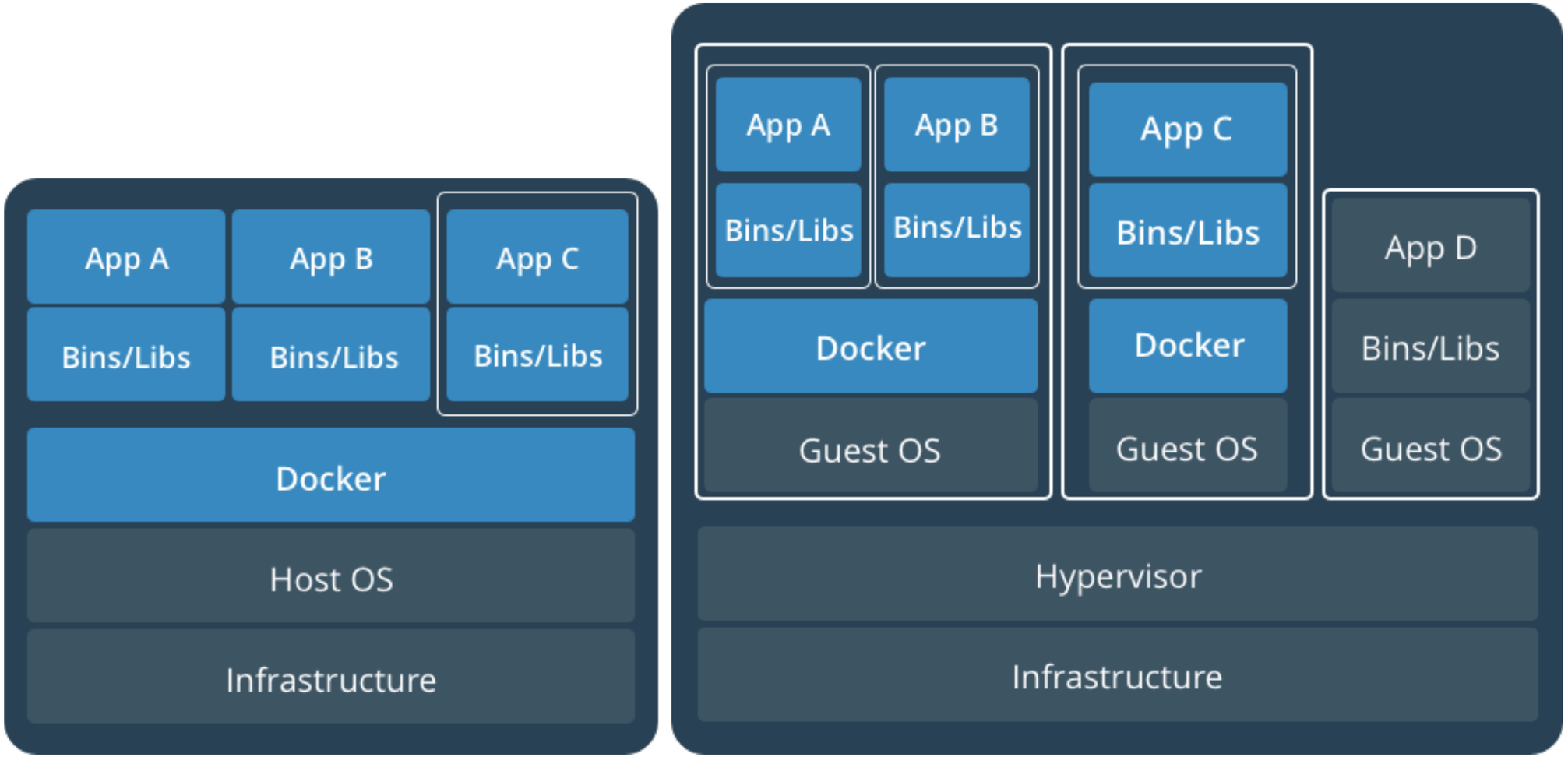
- There are billions of containers used by organizations
- Moreover, there are many large organizations heavily investing in container technologies
- Docker is far ahead of the competition, supported by many large operating system vendors and cloud providers
- Lmctfy, SystemdNspawn, Rocket, Drawbridge, LXD, Kurma, and Calico are some of the other containerization solutions
- Open container specification is also under development

The difference between VMs and containers

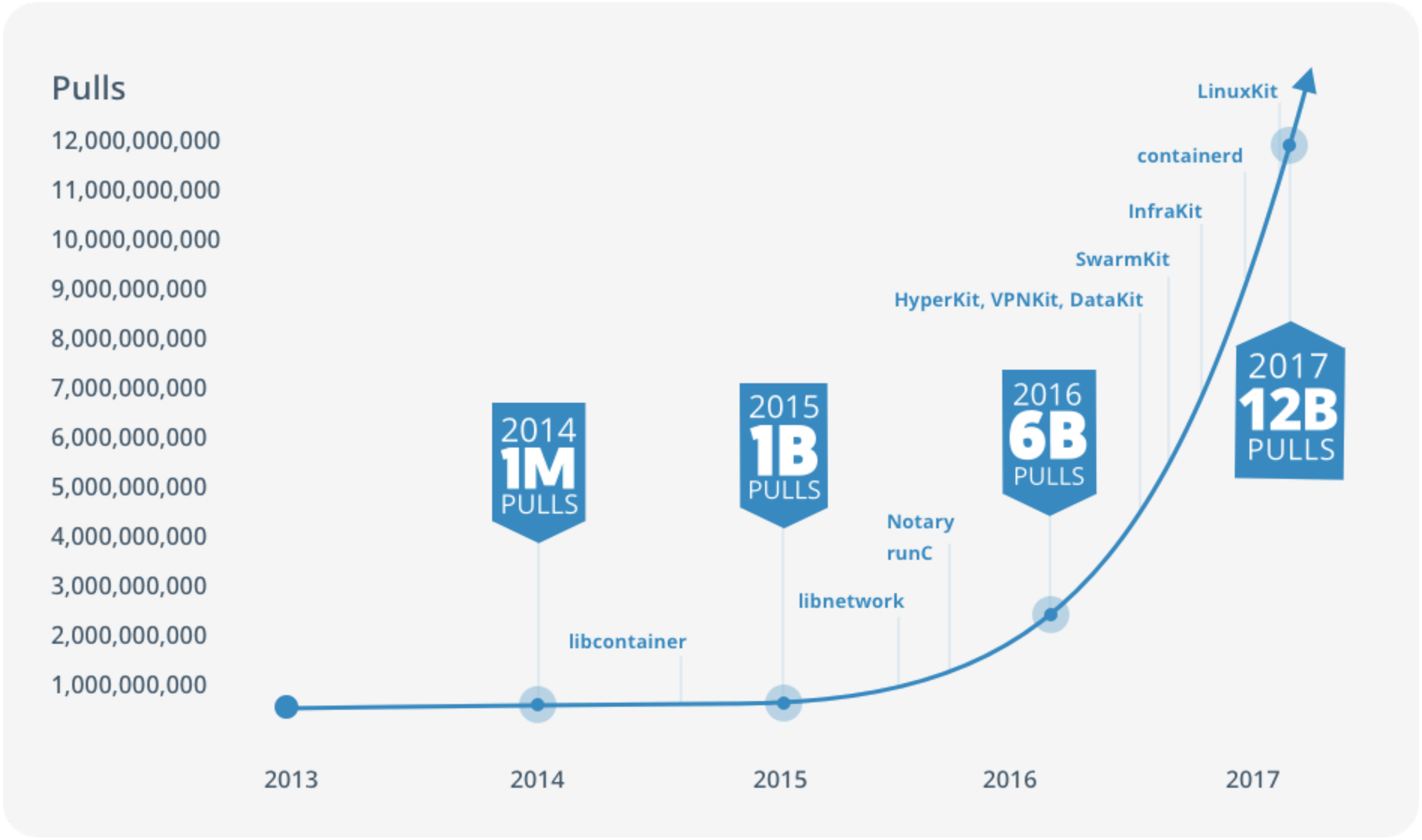




Containers and Virtual Machines Together



Container Standards and Industry Leadership



The benefits of containers

- Self-contained: package the essential application binaries and their dependencies together
- Lightweight: containers, in general, are smaller in size with a lighter footprint.
  - The simplest Spring Boot microservice packaged with an Alpine container with Java 8 would only come to around 170 MB in size
- Scalable: container images are smaller in size and there is no OS booting at startup
- Portable: are built with all the dependencies, they can be ported across multiple machines or across multiple cloud providers
- Lower license cost: Many software license terms are based on the physical core

The benefits of containers

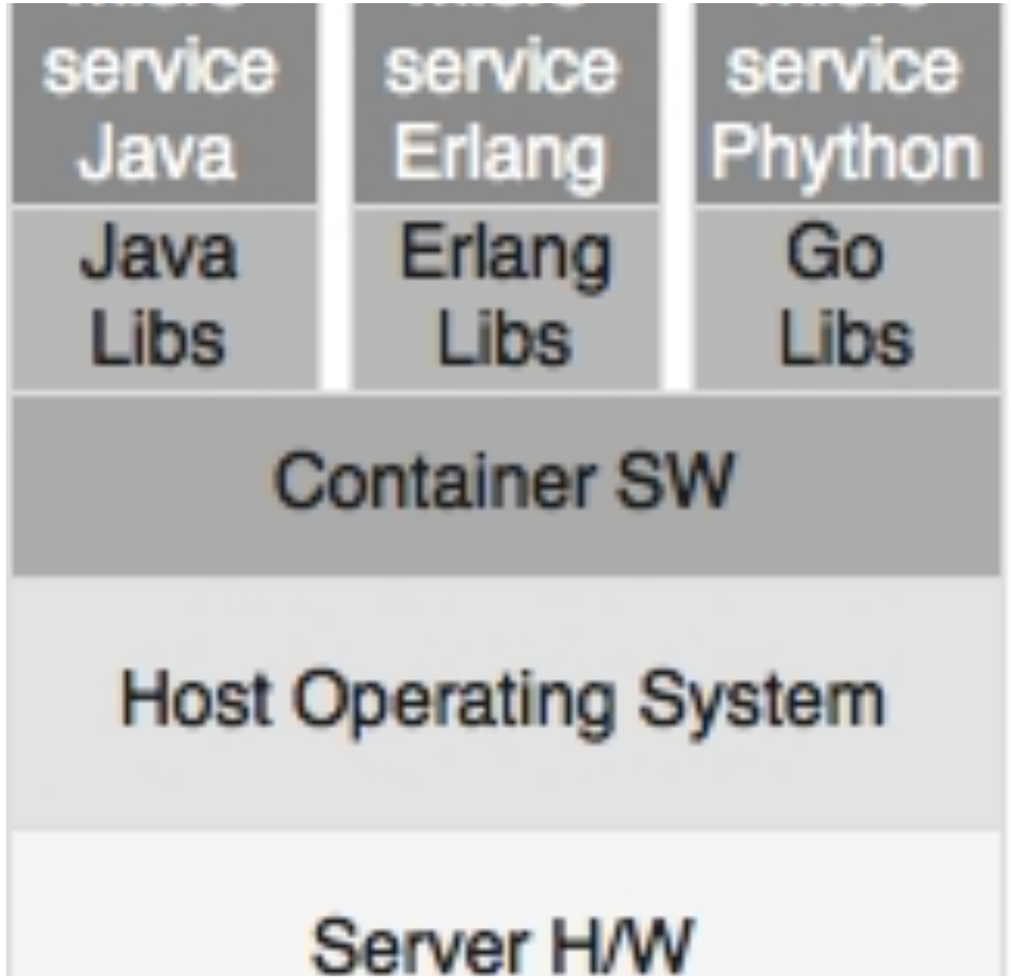
- DevOps: the lightweight footprint of containers makes it easy to automate builds and publish and download containers from remote repositories
- Version controlled: support versions by default
- Reusable: container images are reusable artifacts
- Immutable containers: containers are created and disposed of after usage, they are never updated or patched
  - Immutable containers are used in many environments to avoid complexities in patching deployment units
  - Patching results in a lack of traceability and an inability to recreate environments consistently

Microservices and containers

- Microservices can run without containers, and containers can run monolithic applications
- Containers are good for monolithic applications
  - but the complexities and the size of the monolith application may kill some of the benefits of the containers

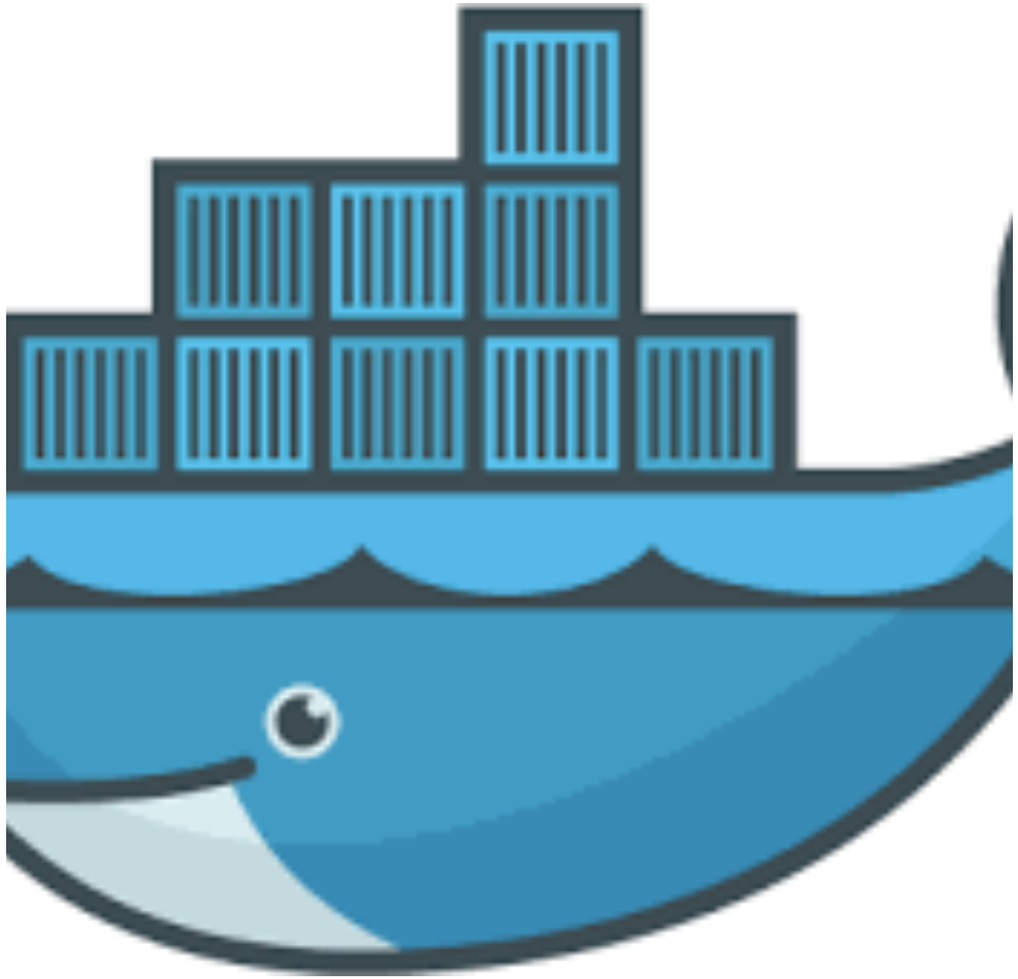
Microservices and containers

- The real advantage of containers can be seen when managing many polyglot microservices
- Eliminate the need to have different deployment management tools to handle polyglot microservices
- Not only abstract the execution environment but also how to access the services
- Irrespective of the technologies used, containerized microservices expose REST APIs
- Once the container is up and running, it binds to certain ports and exposes its APIs
- As containers are self-contained and provide full stack isolation among services, in a single VM or bare metal, one can run multiple heterogeneous microservices and handle them in a uniform way



Introduction to Docker

- Containers have been in the business for years, but the popularity of Docker has given containers a new outlook
- As a result, many container definitions and perspectives emerged from the Docker architecture
- Docker is so popular that even containerization is referred to as dockerization
- Docker is a platform to build, ship, and run lightweight containers based on Linux kernels



Security

"Gartner asserts that applications deployed in containers are more secure than applications deployed on the bare OS."

http://blogs.gartner.com/joerg-fritsch/can-you-operationalize-docker-containers/

Safer applications

All Linux isolation capabilities

- pid namespace
- mnt namespace
- net namespace
- uts namespace
- user namespace
- pivot_root
- uid/gid drop
- cap drop
- all cgroups
- selinux
- apparmor
- seccomp

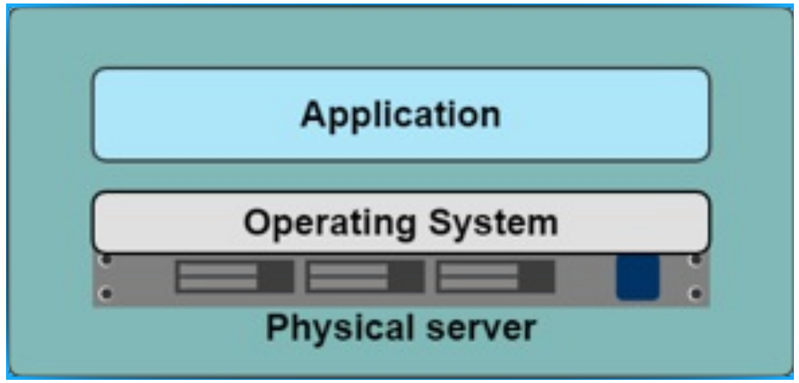
+

1. Out of the box default settings and profiles
2. Granular controls to customize settings

= **Safer Apps**

Encapsulation I: Physical servers

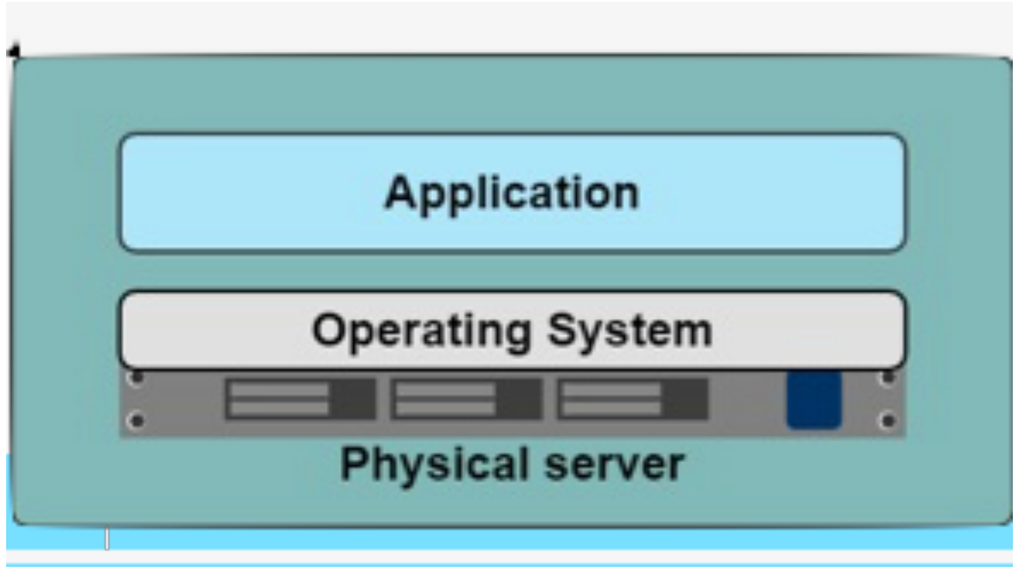
- The Dark Ages



- One application, one physical server

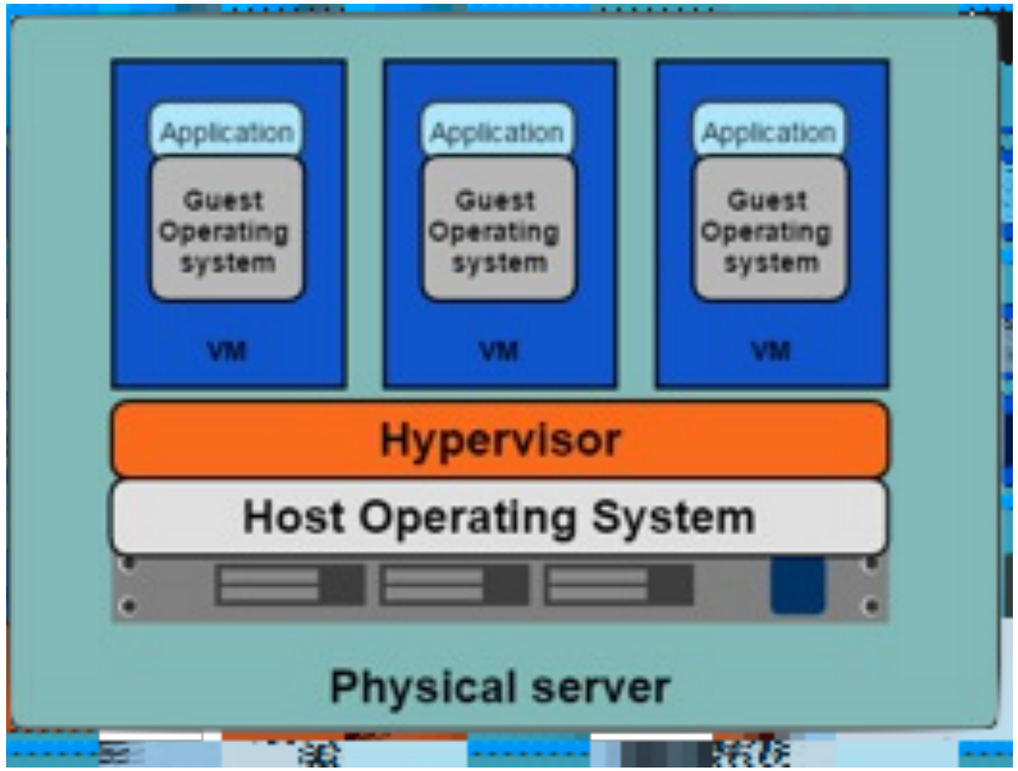
Encapsulation I: Physical servers

- Limits of physical encapsulation
  - Slow deployment
  - Huge costs
  - Provisioning speed limited by physical logistics
  - Difficult to scale
  - Difficult to migrate Vendor lock-in



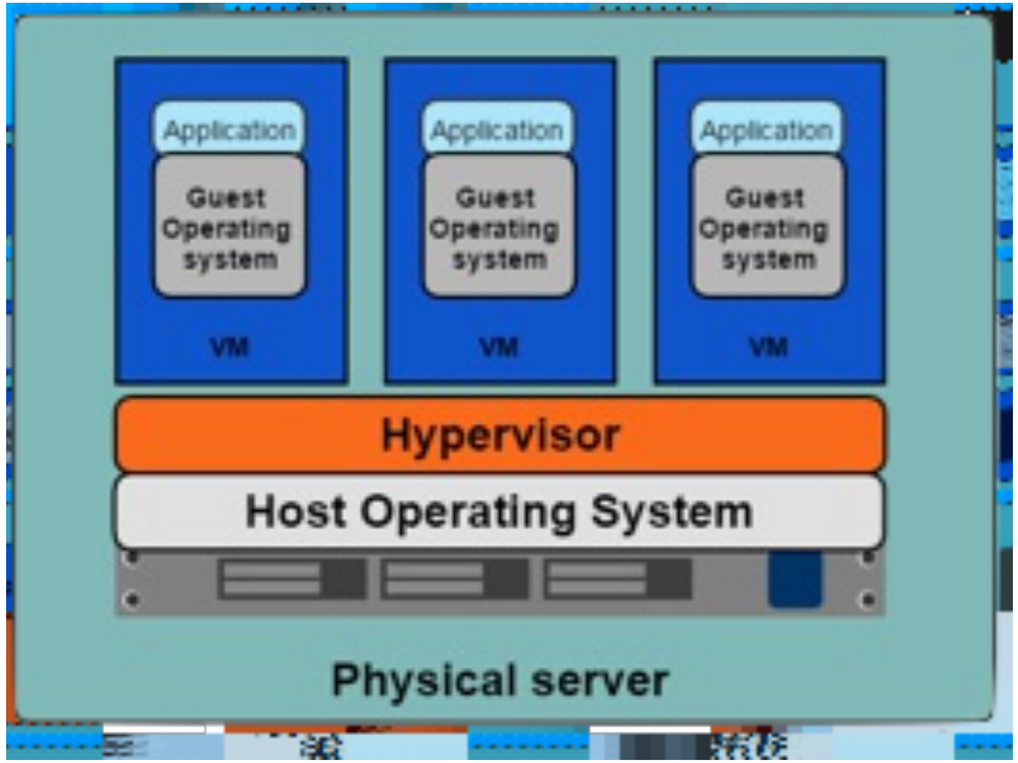
Encapsuation II: VM

- Multiple apps on one server
- Elastic, real time provisioning
- Scalable pay-per-use cloud models viable



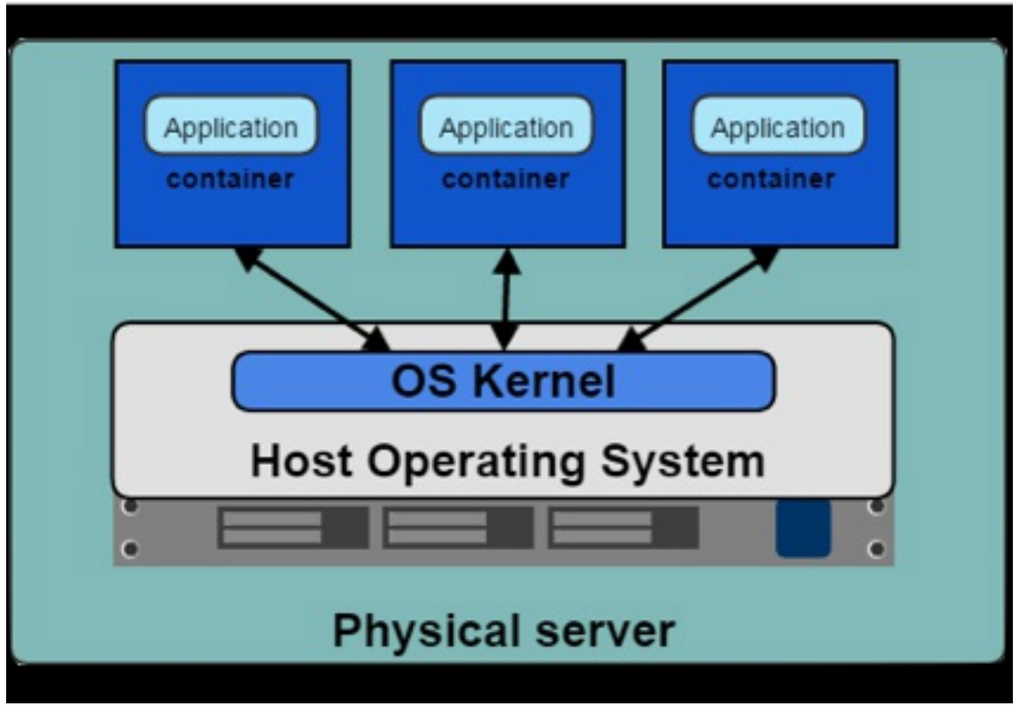
Encapsuation II: VM

- VM Limitations
  - VMs require CPU & memory allocation
  - Significant overhead from guest OS



Encapsulation III: Containers

- Containers leverage kernel features to create extremely light-weight encapsulation:
  - Kernel namespaces
  - Network namespaces
  - Linux containers
  - cgroups & security tools
  - Results in faster spool-up and denser servers



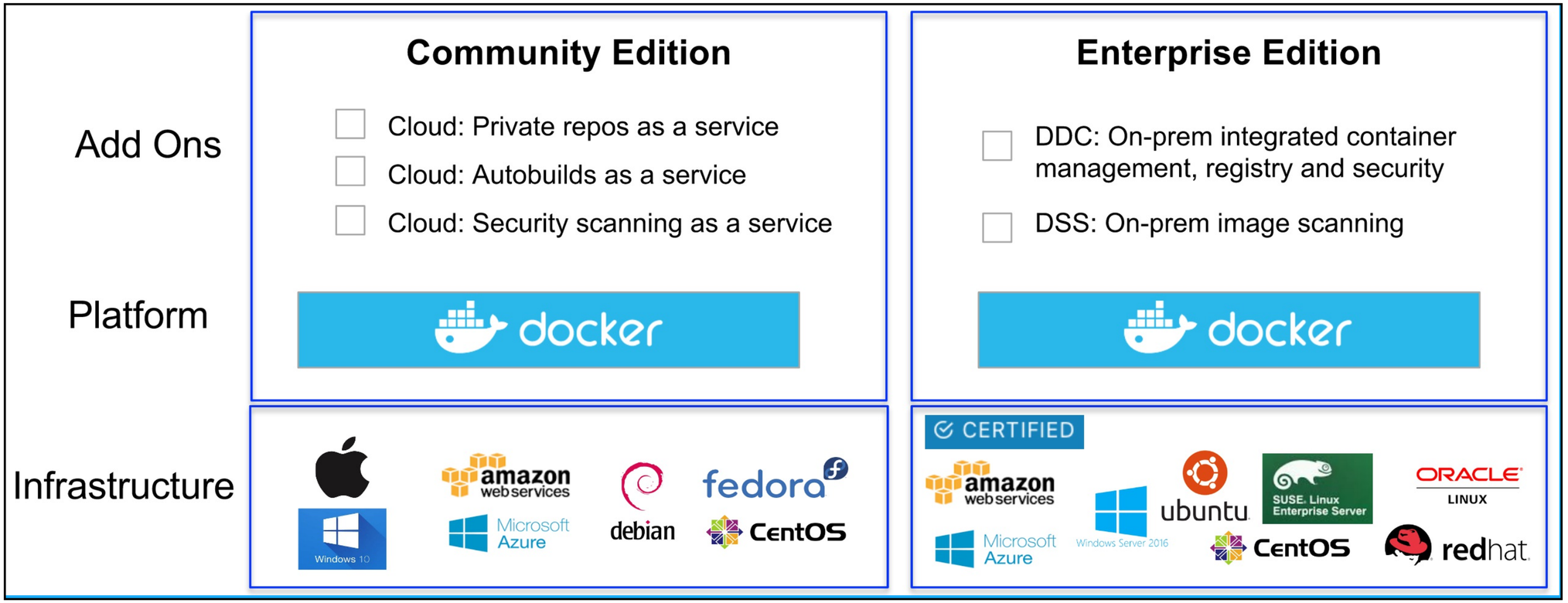
Introduction to Docker

- The most basic thing Docker provides is a
- framework for service encapsulation
- But what are the implications of this for developers, ops, and orgs?

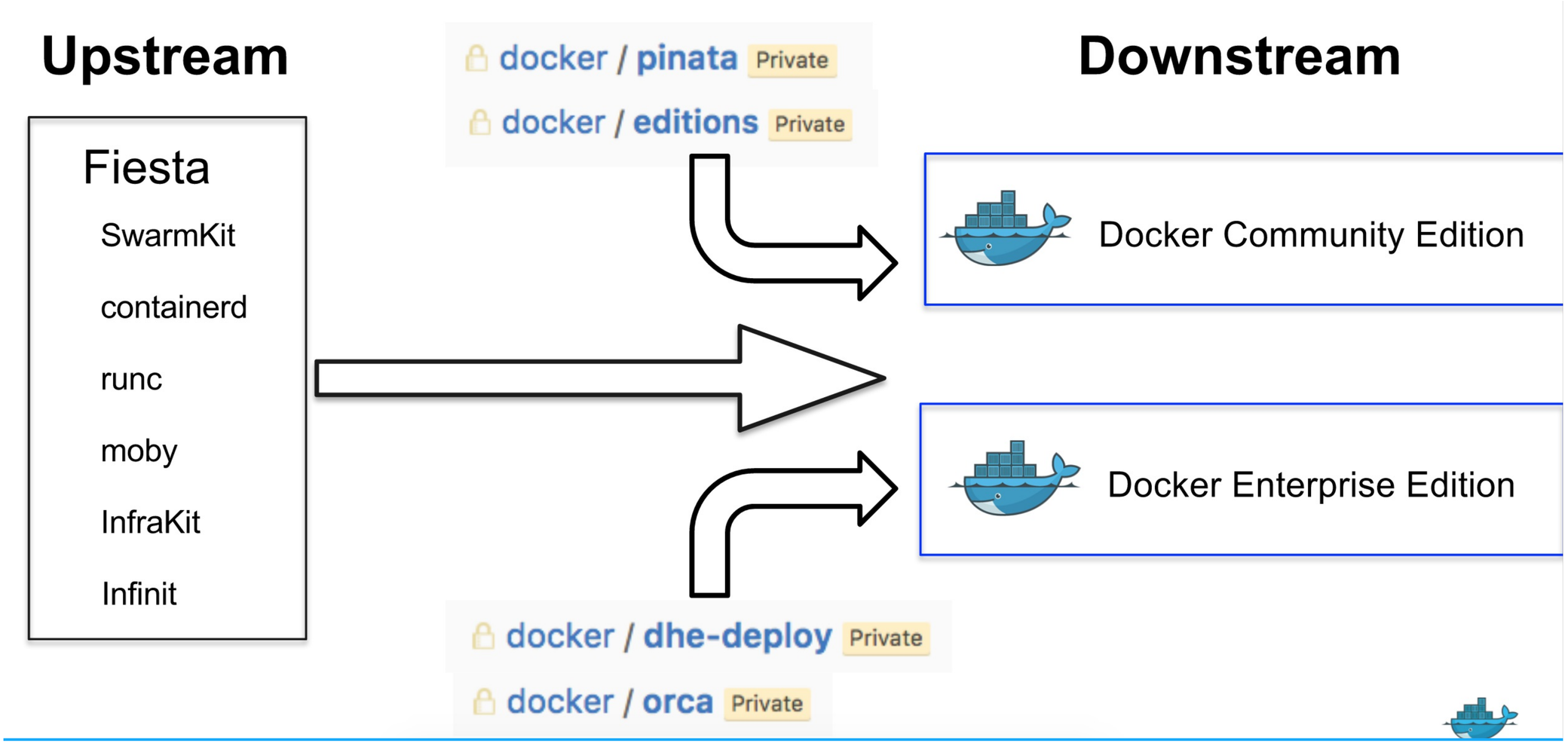
Distributed application architecture

- Encapsulation supercharges:
  - Monolith Densification
  - Service-Based Architecture
  - Devops

Docker Product Offerings

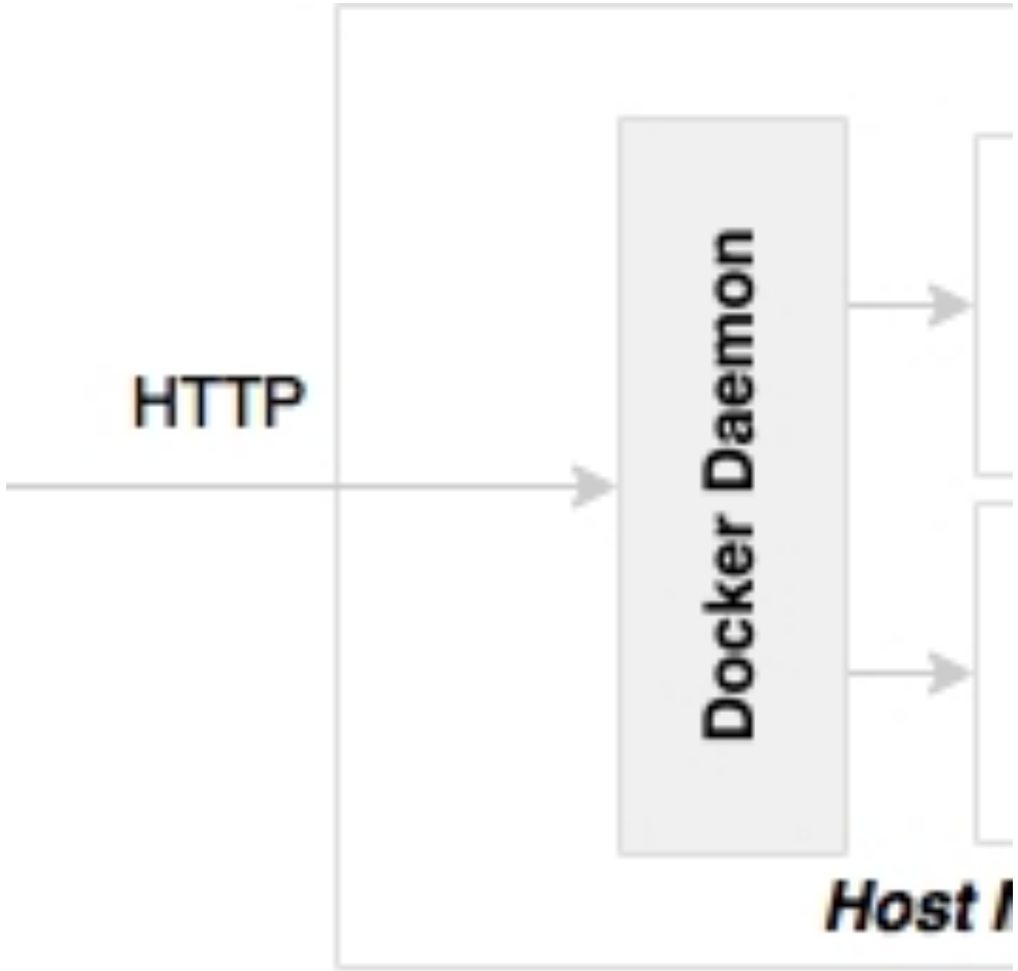


Components of CE/EE



The key components of Docker

- The Docker daemon is a server-side component that runs on the host machine responsible for building, running, and distributing Docker containers
  - exposes APIs for the Docker client to interact with the daemon. These APIs are primarily REST-based endpoints
- The Docker client is a remote command-line program that interacts with the Docker daemon through either a socket or REST APIs
  - Docker users use the CLI to build, ship, and run Docker containers

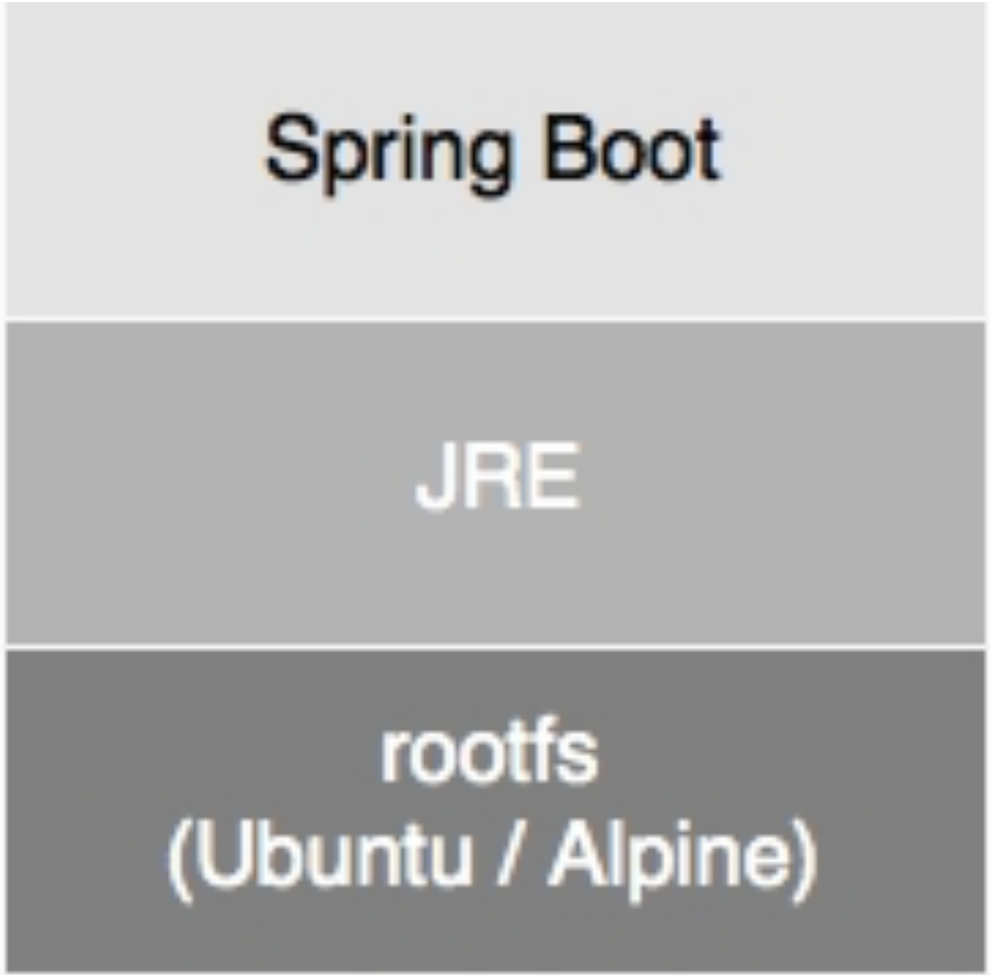


Docker concepts

- A Docker image is the read-only copy of the operating system libraries, the application, and its libraries
- Once an image is created, it is guaranteed to run on any Docker platform without alterations
- In Spring Boot microservices, a Docker image packages operating systems such as Ubuntu, Alpine, JRE, and the Spring Boot fat application JAR file
  - Docker images are based on a layered architecture in which the base image is one of the flavors of Linux
  - Each layer, as shown in the preceding diagram, gets added to the base image layer with the previous image as the parent layer
  - Docker uses the concept of a union filesystem to combine all these layers into a single image, forming a single filesystem

Docker image

- Every time we rebuild the application, only the changed layer gets rebuilt, and the remaining layers are kept intact
- Multiple containers running on the same machine with the same type of base images would reuse the base image, thus reducing the size of the deployment
- For instance, in a host, if there are multiple containers running with Ubuntu as the base image, they all reuse the same base image



Docker image

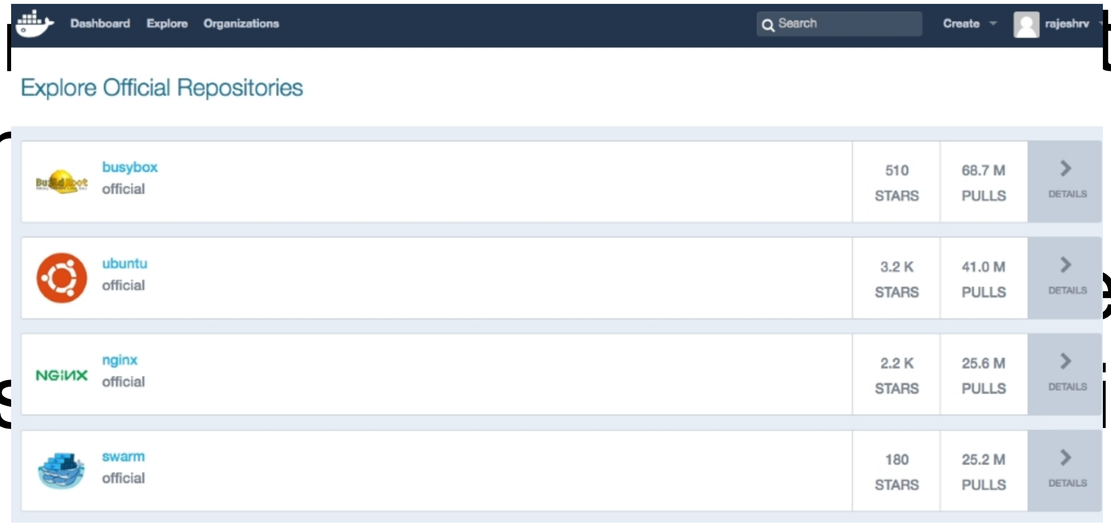
- IMAGES ARE LAYERED FILESYSTEMS
- Provide filesystem for container process
- Image = stack of immutable layers
- Start with a base image
- Add layer for each change

Docker containers

- Docker containers are the running instances of a Docker image
- Containers use the kernel of the host operating system when running
  - Hence, they share the host kernel with other containers running on the same host
- The Docker runtime ensures that the container processes are allocated with their own isolated process space using kernel features such as **cgroups** and the kernel **namespace** of the operating system
-

The Docker registry

- The Docker registry is a central place where Docker images are published and downloaded from
- The URL https://hub.docker.com is the central registry provided by Docker
- The Docker r… that one can download ar…
- Docker also … specific to the accounts … istry



Dockerfile

- A Dockerfile is a build or scripting file that contains instructions to build a Docker image
- There can be multiple steps documented in the Dockerfile, starting from getting a base image
- The docker build command looks up Dockerfile for instructions to build
- One can compare a Dockerfile to a pom.xml file used in a Maven build

Dockerfile

- FROM command defines base image
- Each subsequent command adds a layer
- docker image build ... builds image from Dockerfile

```
# Comments begin with the pound sign
FROM ubuntu:16.04
RUN apt-get update
ADD /data /myapp/data
...
```

The future of containerization – unikernels and hardened security

- Containerization is still evolving, but the number of organizations adopting containerization techniques has gone up in recent times
- Currently, Docker images are generally heavy
  - In an elastic automated environment, where containers are created and destroyed quite frequently, size is still an issue
  - A larger size indicates more code, and more code means that it is more prone to security vulnerabilities

The future of containerization – unikernels and hardened security

- The future is definitely in small footprint containers
- Docker is working on unikernels, lightweight kernels that can run Docker even on low-powered IoT devices
  - Unikernels are not full-fledged operating systems, but they provide the basic necessary libraries to support the deployed applications

The future of containerization – unikernels and hardened security

- The security issues of containers are much discussed and debated
- The key security issues are around the user namespace segregation or user ID isolation
- If the container is on root, then it can by default gain the root privilege of the host
- Using container images from untrusted sources is another security concern
- Docker is bridging these gaps as quickly as possible, but there are many organizations that use a combination of VMs and Docker to circumvent some of the security concerns

Homework 13

- Work through the exercises 1 to 18 in the Docker Fundamentals Exercises book