

Docker

Welcome

Safe Harbour

- All images used are taken with consensus for this education purpose and does not associate with any obligation of Intellectual Property.



AGENDA

- Dockers & Containers
 - Architecture
 - Coexistence
- Playing with Dockers – 1
- Docker Images and Containers
- Docker Hub

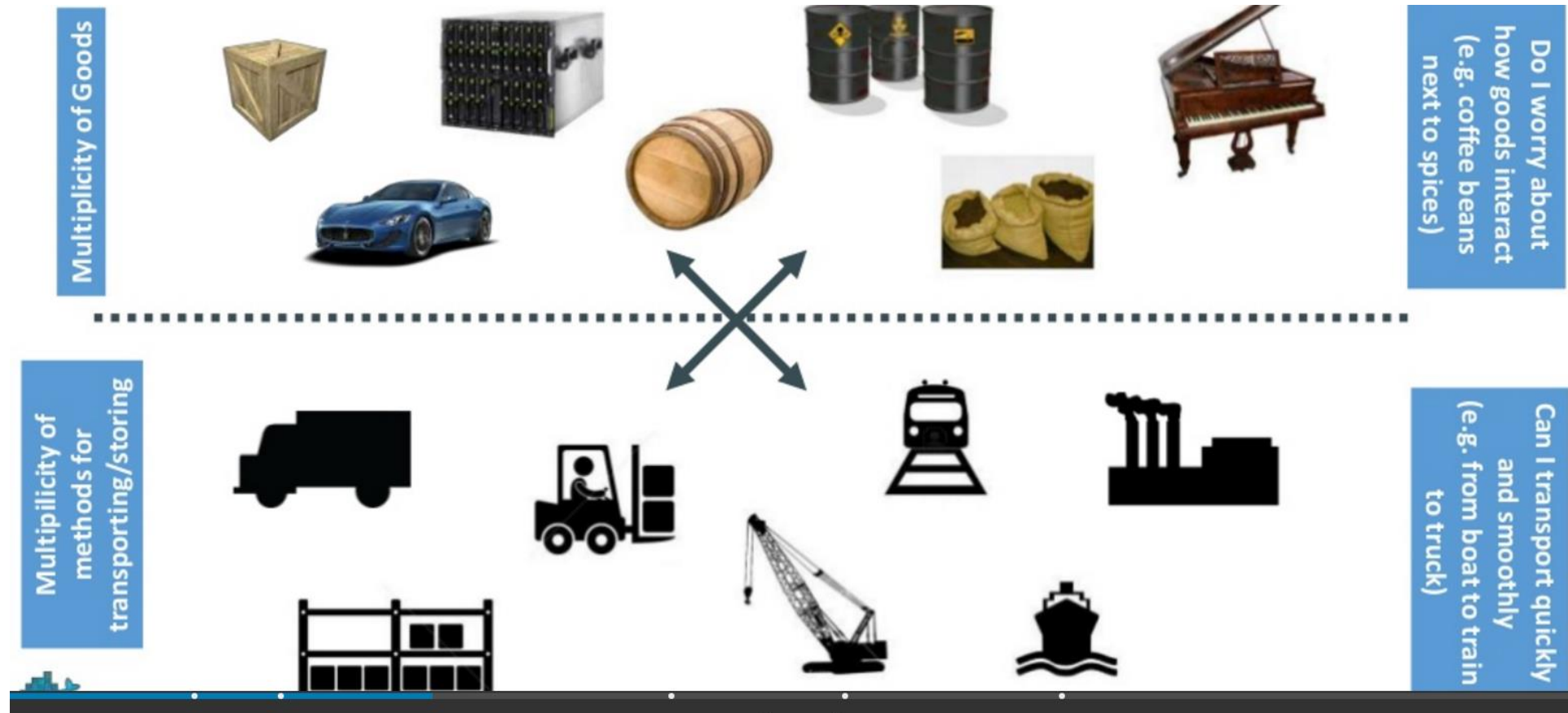
SESSION 1

- Containers and Architecture



a helpful tool for packing, shipping, and running applications within “containers

Understanding....an analogy ...cargo transport pre-1960




















































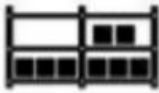





What are the possibilities

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

SOLUTION—shipping containers



This solved the problem

How does this container idea translate to our problem

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
							

Container and VM

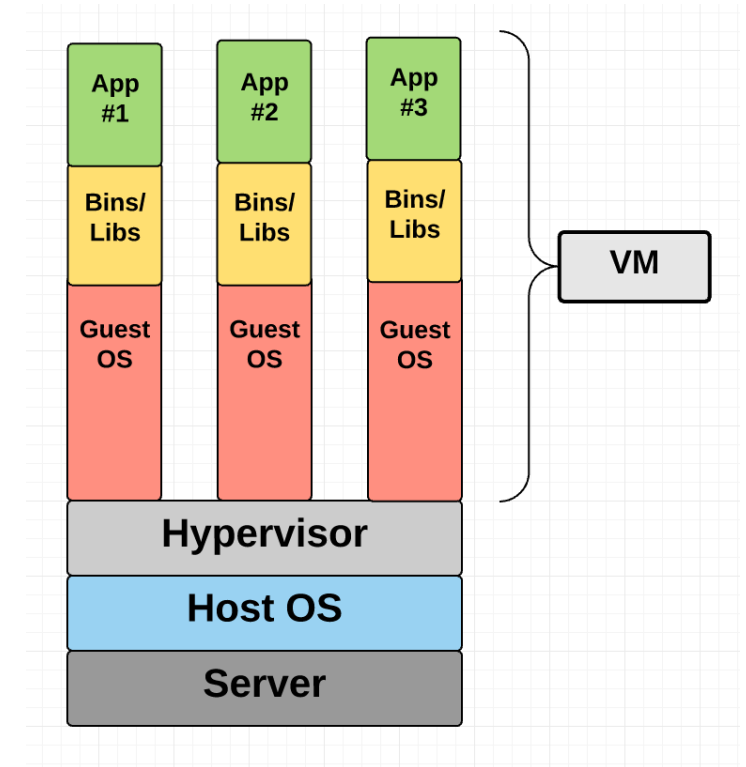
- **Core Objective:-**

to isolate an application and its dependencies into a self-contained unit that can run anywhere.

- remove the need for physical hardware, allowing for more efficient use of computing resources, both in terms of energy consumption and cost effectiveness.
- main difference between containers and VMs is in their architectural approach

VM

- A VM is essentially an emulation of a real computer that executes programs like a real computer. VMs run on top of a physical machine using a *“hypervisor”*.
A hypervisor, in turn, runs on either a host machine or on *“bare-metal”*.
- A **hypervisor** is a piece of software, firmware, or hardware that VMs run on top of. The hypervisors themselves run on physical computers, referred to as the *“host machine”*.
 - The host machine provides the VMs with resources, including RAM and CPU
- The benefit of a hosted hypervisor is that the underlying hardware is less important.
 - The host’s operating system is responsible for the hardware drivers instead of the hypervisor itself, and is therefore considered to have more *“hardware compatibility.”*



VM

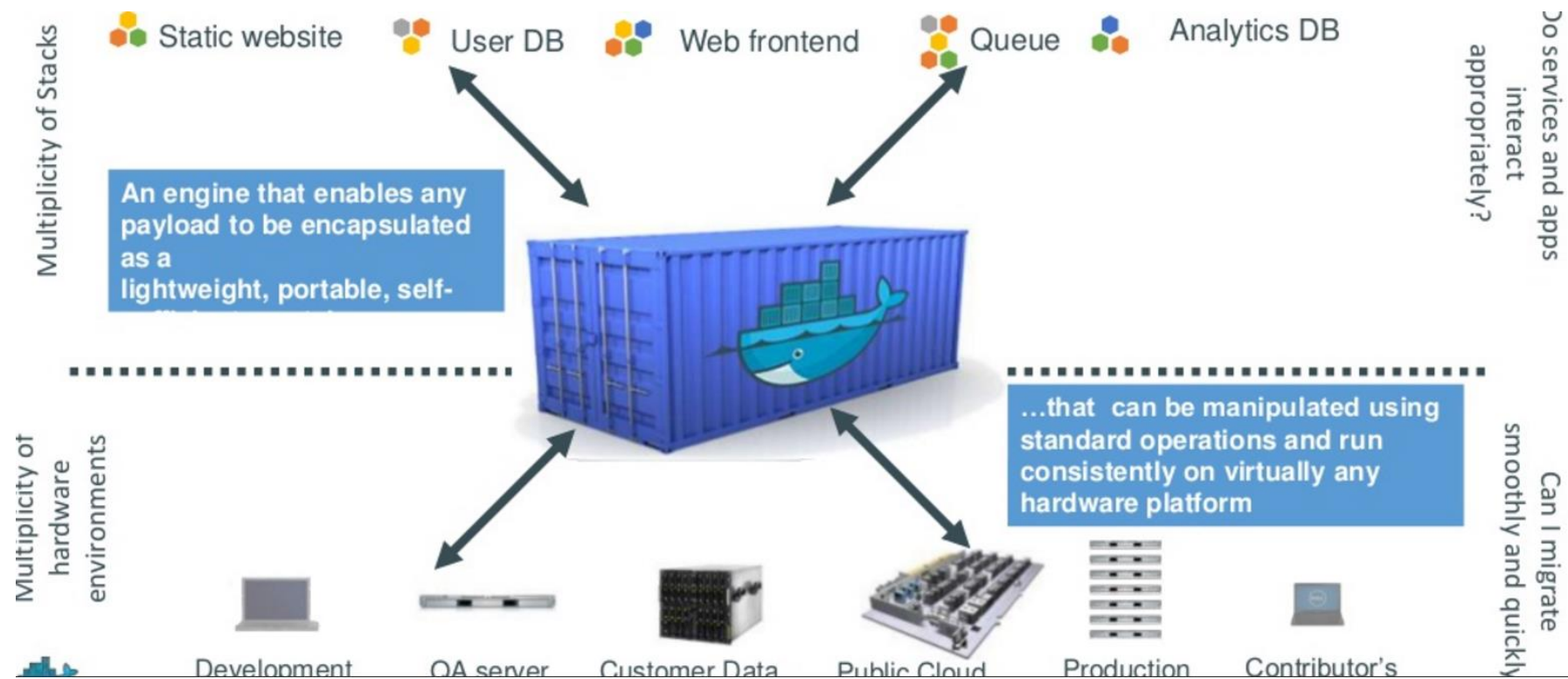
- The VM that is running on the host machine (again, using a hypervisor) is also often called a “*guest machine*.”
 - This guest machine contains both the application and whatever it needs to run that application (e.g. system binaries and libraries).
 - It also carries an entire virtualized hardware stack of its own, including virtualized network adapters, storage, and CPU — which means it also has its own full-fledged guest operating system.
 - From the inside, the guest machine behaves as its own unit with its own dedicated resources. From the outside, we know that it’s a VM — sharing resources provided by the host machine.

VM – bare metal

- A bare metal hypervisor environment tackles the performance issue by installing on and running from the host machine's hardware. Because it interfaces directly with the underlying hardware, it doesn't need a host operating system to run on.

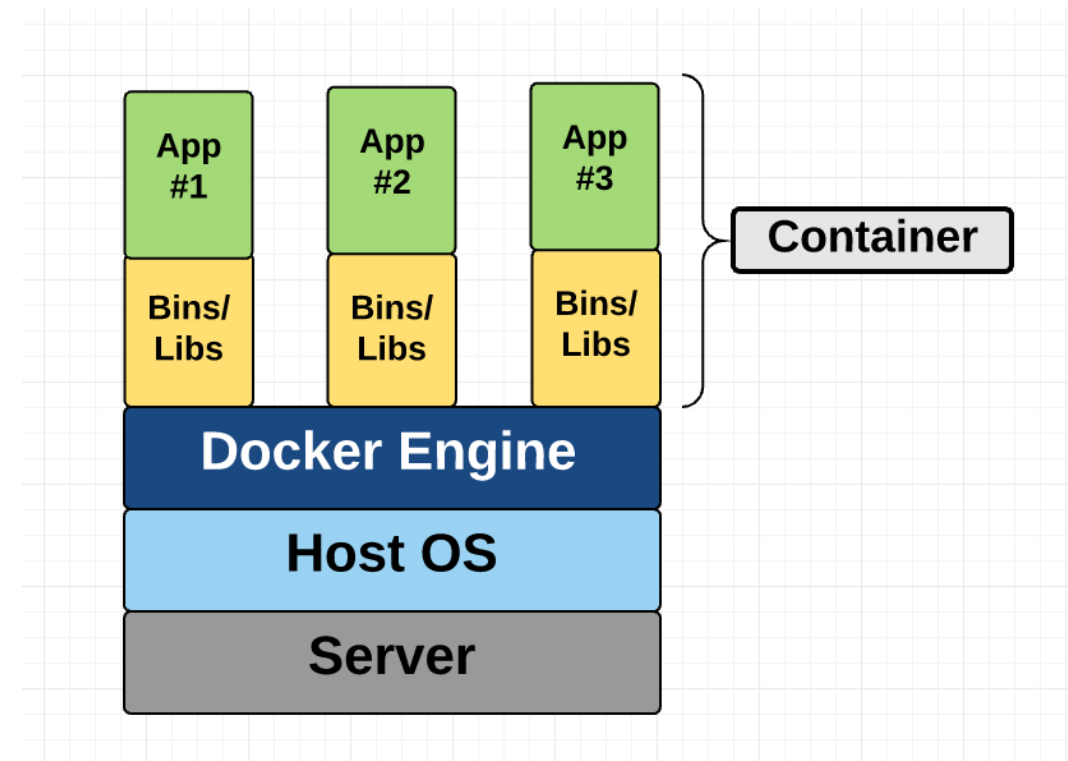
CONTAINER

- Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization by abstracting the “user space”

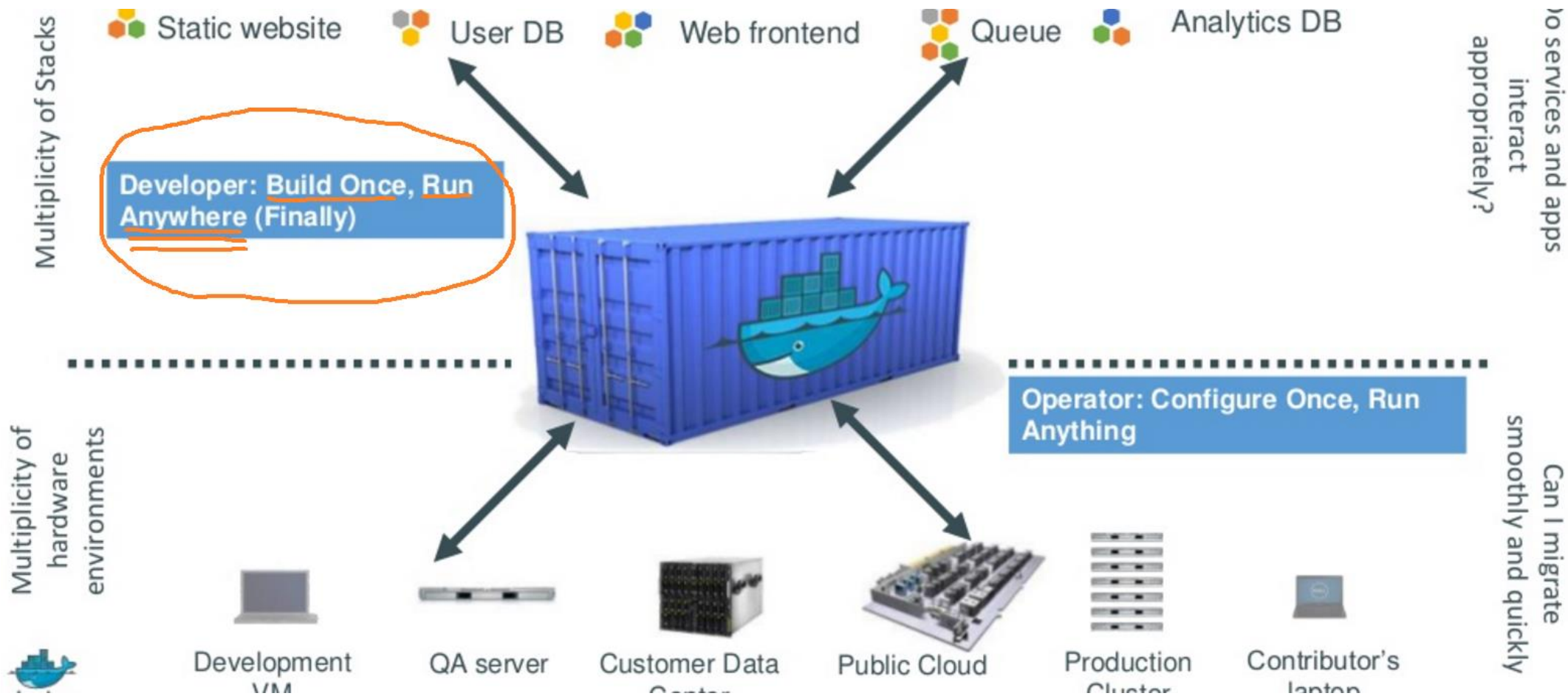


World of Containers

- For all intent and purposes, containers look like a VM.
For example, they have private space for processing, can execute commands as root, have a private network interface and IP address, allow custom routes and ip tables rules, can mount file systems, and etc.
- The one big difference between containers and VMs is that ***containers *share* the host system's kernel with other containers.***



Container :: Do once run anywhere



Docker's container --- the concept (and relation to our shipping container)

	Physical Containers	Docker
Content Agnostic	The same container can hold almost any type of cargo	Can encapsulate any payload and its dependencies
Hardware Agnostic	Standard shape and interface allow same container to move from ship to train to semi-truck to warehouse to crane without being modified or opened	Using operating system primitives (e.g. LXC) can run consistently on virtually any hardware—VMs, bare metal, openstack, public IAAS, etc.—without modification
Content Isolation and Interaction	No worry about anvils crushing bananas. Containers can be stacked and shipped together	Resource, network, and content isolation. Avoids dependency hell
Automation	Standard interfaces make it easy to automate loading, unloading, moving, etc.	Standard operations to run, start, stop, commit, search, etc. Perfect for devops: CI, CD, autoscaling, hybrid clouds
Highly efficient	No opening or modification, quick to move between waypoints	Lightweight, virtually no perf or start-up penalty, quick to move and manipulate
Separation of duties	Shipper worries about inside of box, carrier worries about outside of box	Developer worries about code. Ops worries about infrastructure.

LXC = Linux container
technology to run multiple Linux on one
host, etc.

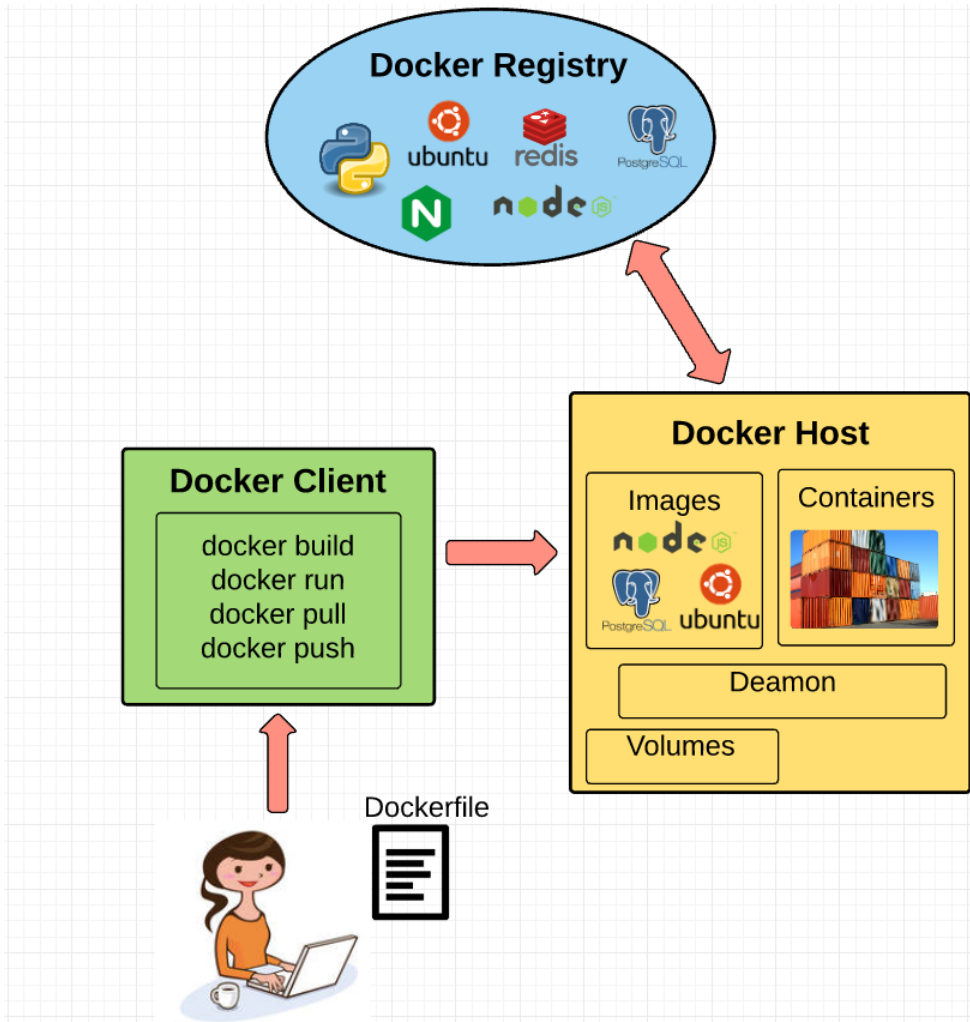
CI/CD =continuous integration & continuous delivery
....go [here to see on AWS](#) (dealing with automatic builds.)

Docker container—developer viewpoint

Build once...run anywhere

- A clean, safe, hygienic and portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
- Automate testing, integration, packaging...anything you can script
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That's the power of Docker

Docker Terminology



Docker engine is the layer on which Docker runs. It's a lightweight runtime and tooling that manages containers, images, builds, and more. It runs natively on Linux systems and is made up of:

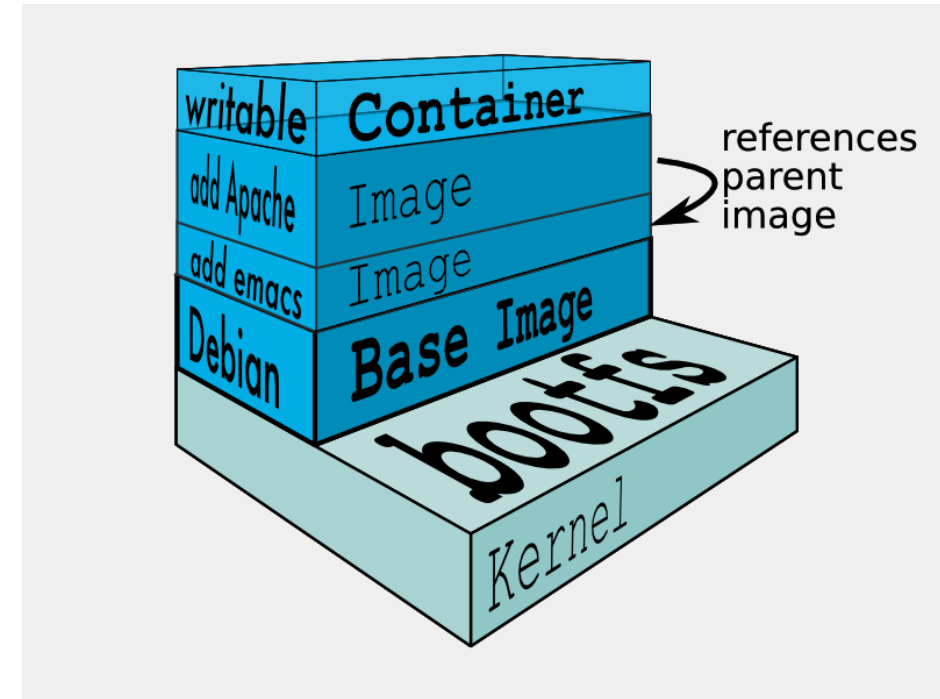
Docker Daemon that runs in the host computer.

Docker Client that then communicates with the Docker Daemon to execute commands.

REST API for interacting with the Docker Daemon remotely.

Docker Images

- Docker Images are read-only templates that you build from a set of instructions written in your Docker file. Images define both what you want your packaged application and its dependencies to look like *and* what processes to run when it's launched.
- The Docker image is built using a Docker file. Each instruction in the Docker file adds a new “layer” to the image, with layers representing a portion of the image's file system that either adds to or replaces the layer below it. Layers are key to Docker's lightweight yet powerful structure. Docker uses a Union File System to achieve this:



Container Abstract

- The term “container” is really just an abstract concept to describe how a few different features work together to visualize a “container”

NAME SPACES

Namespaces provide containers with their own view of the underlying Linux system, limiting what the container can see and access.

NET,PID, MNT,UTS, IPC, USER

CGROUPS

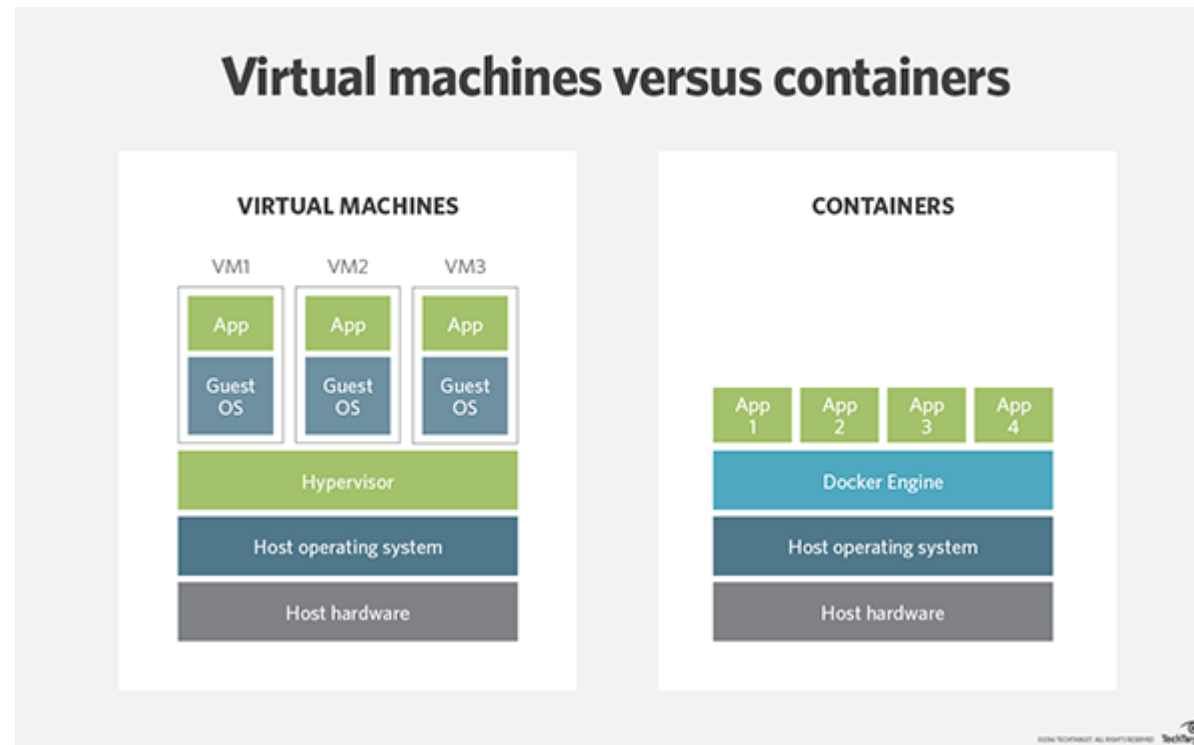
Control groups (also called cgroups) is a Linux kernel feature that isolates, prioritizes, and accounts for the resource usage (CPU, memory, disk I/O, network, etc.) of a set of processes.

FILESYSTEM

ISOLATED, UFS

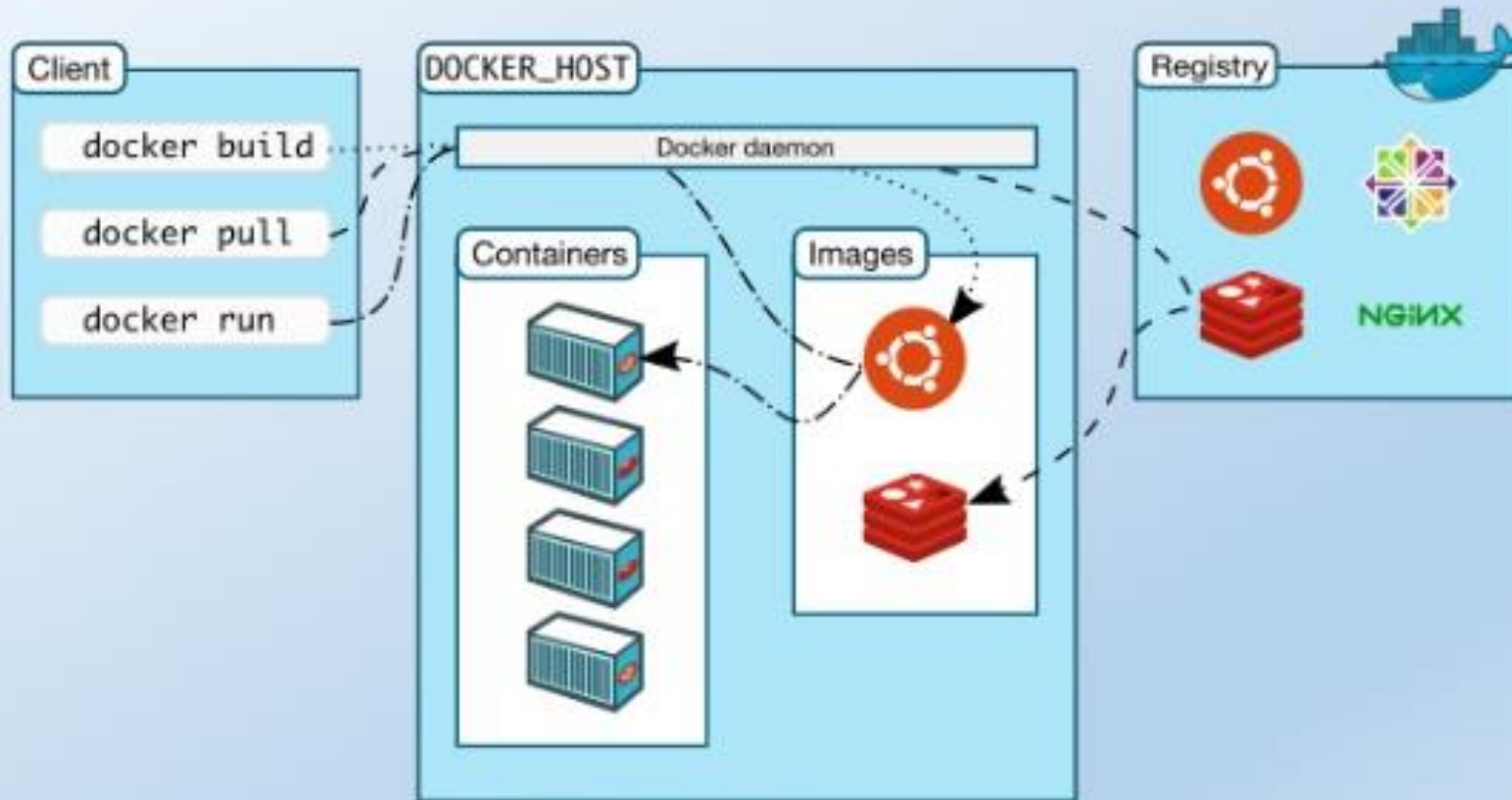
Co-existence

- if you need to run multiple applications on multiple servers, it probably makes sense to use VMs. On the other hand, if you need to run many *copies* of a single application, Docker offers some compelling advantages.



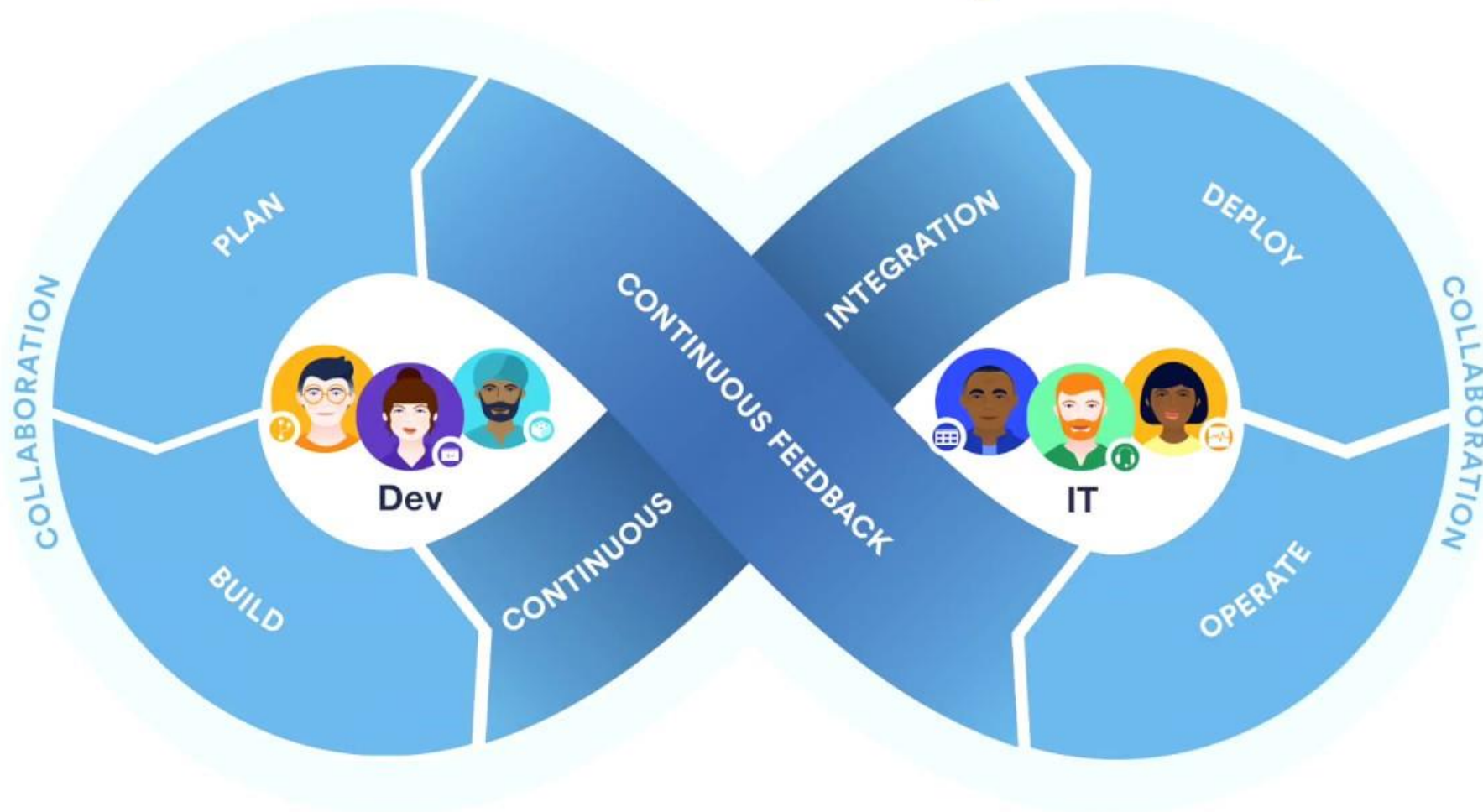
Architecture

Docker Architecture



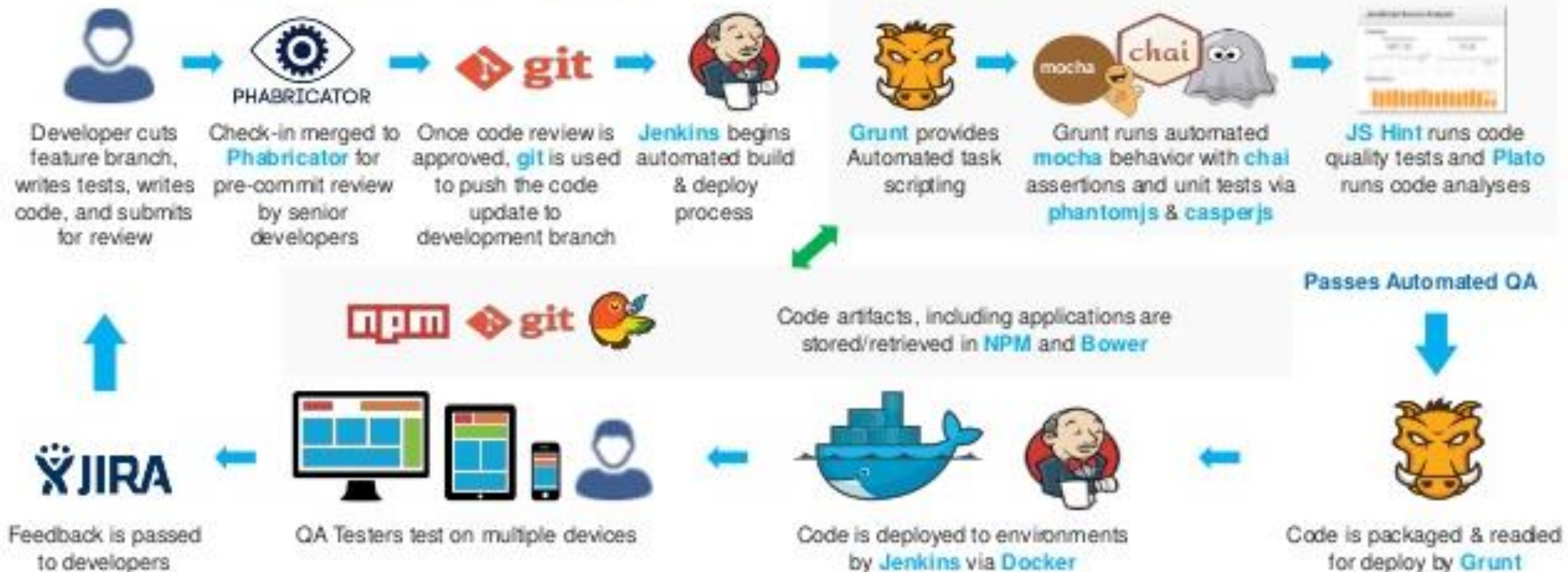
CI . CD and CF

Dev and IT better together



CI in DevOps

Modern Web Architecture requires a journey toward the next generation of agile development methods, DevOps capabilities, and quality-first engineering principles



Session 2

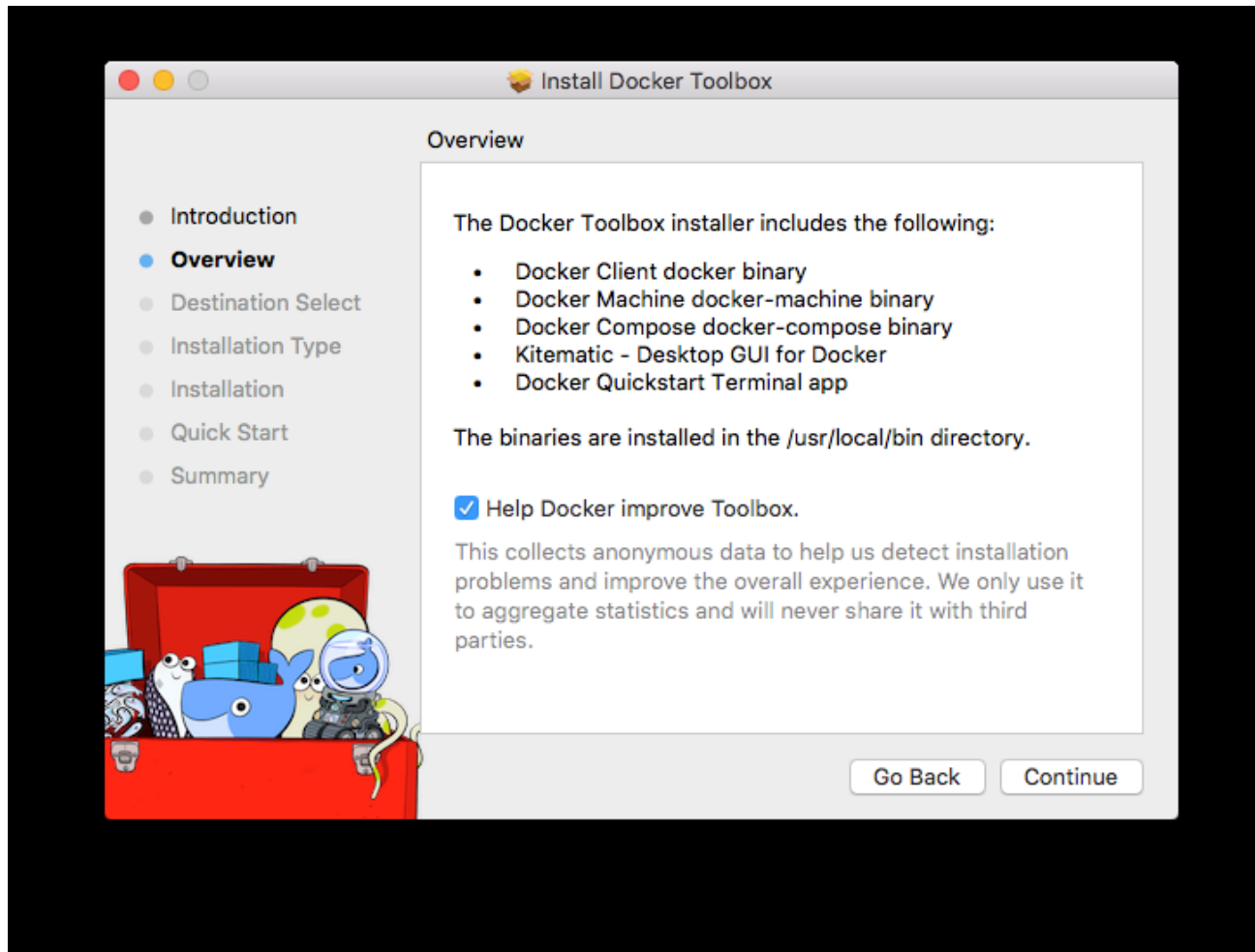
- Installation
- Playing with Docker

Docker Software

Capabilities	Community Edition	Enterprise Edition Basic	Enterprise Edition Standard	Enterprise Edition Advanced
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓	✓
Image management			✓	✓
Container app management			✓	✓
Image security scanning				✓

Docker Toolbox

- <https://docs.docker.com/toolbox/overview/#whats-in-the-box>



Check for Virtualization Enabled ?

-- My Computer , System Properties
(Virtualization Enabled)

--Windows Feature On or Off

-- Oracle VM – Check Features

Docker Tool Box

- First Login to Docker Terminal
 - Be in Oracle Free WIFI or Mobile WIFI
 - Creates Docker Engine
 - Docker Whale and IP for \$
 - Docker CLI

Docker CLI

- Docker
(Docker Client)
- Docker Engine
(Docker Hardware)
- Docker File

