Introduction to Docker

Contents

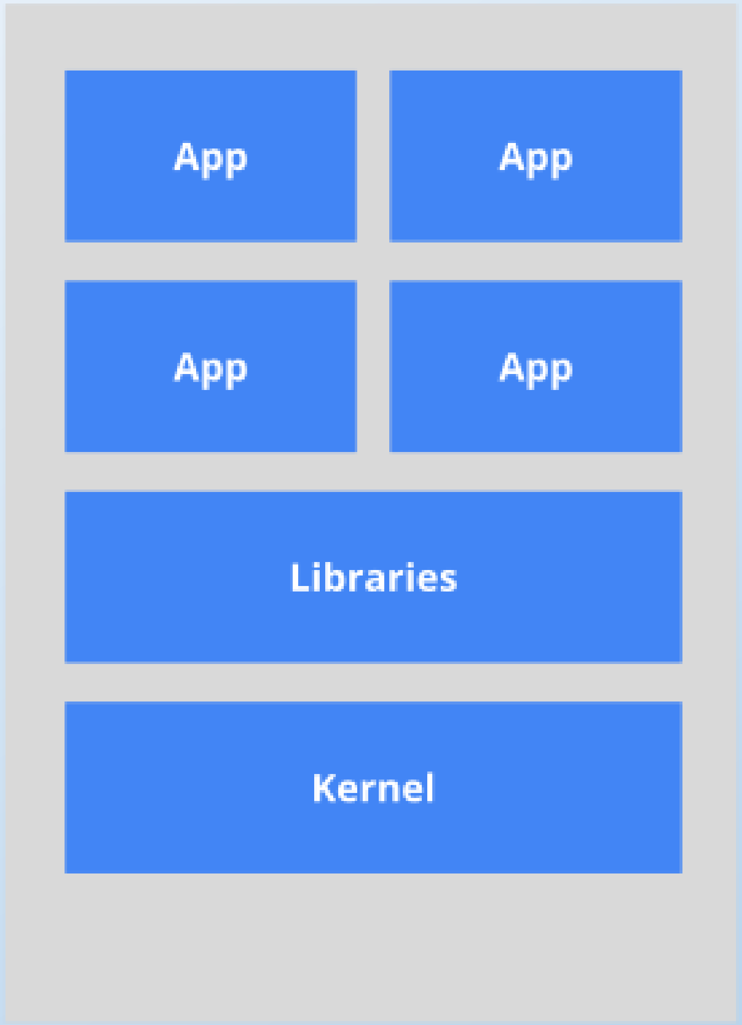
Containers

- LXC (Linux Containers) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel.

- The Linux kernel provides the cgroups functionality that allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any virtual machines, and namespace isolation functionality that allows complete isolation of an applications' view of the operating environment, including process trees, networking, user IDs and mounted file systems

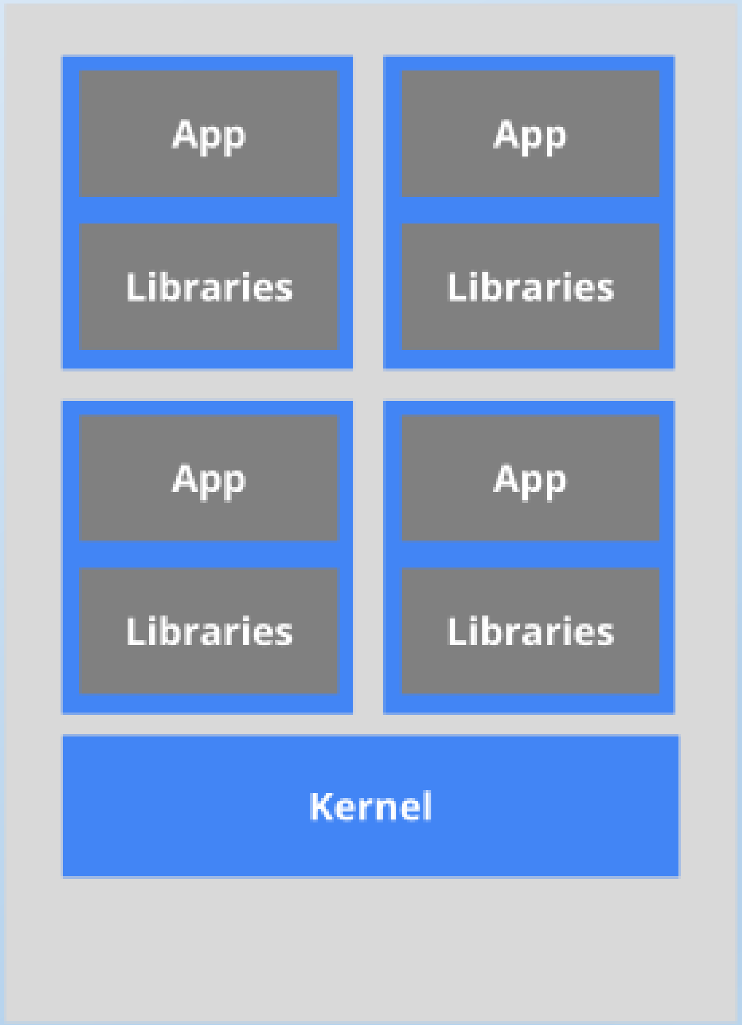
Why Containers

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

What is Docker

- Docker is an open-source project that automates the deployment of applications inside software container
- Docker containers wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server.
- This guarantees that it will always run the same, regardless of the environment it is running in.

docker

The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

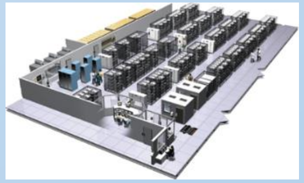
Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud



Disaster recovery

Production Cluster

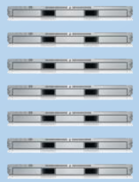
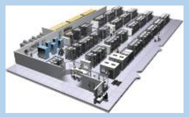
Production Servers

Contributor's laptop

Can I migrate smoothly and quickly?

The Matrix From Hell

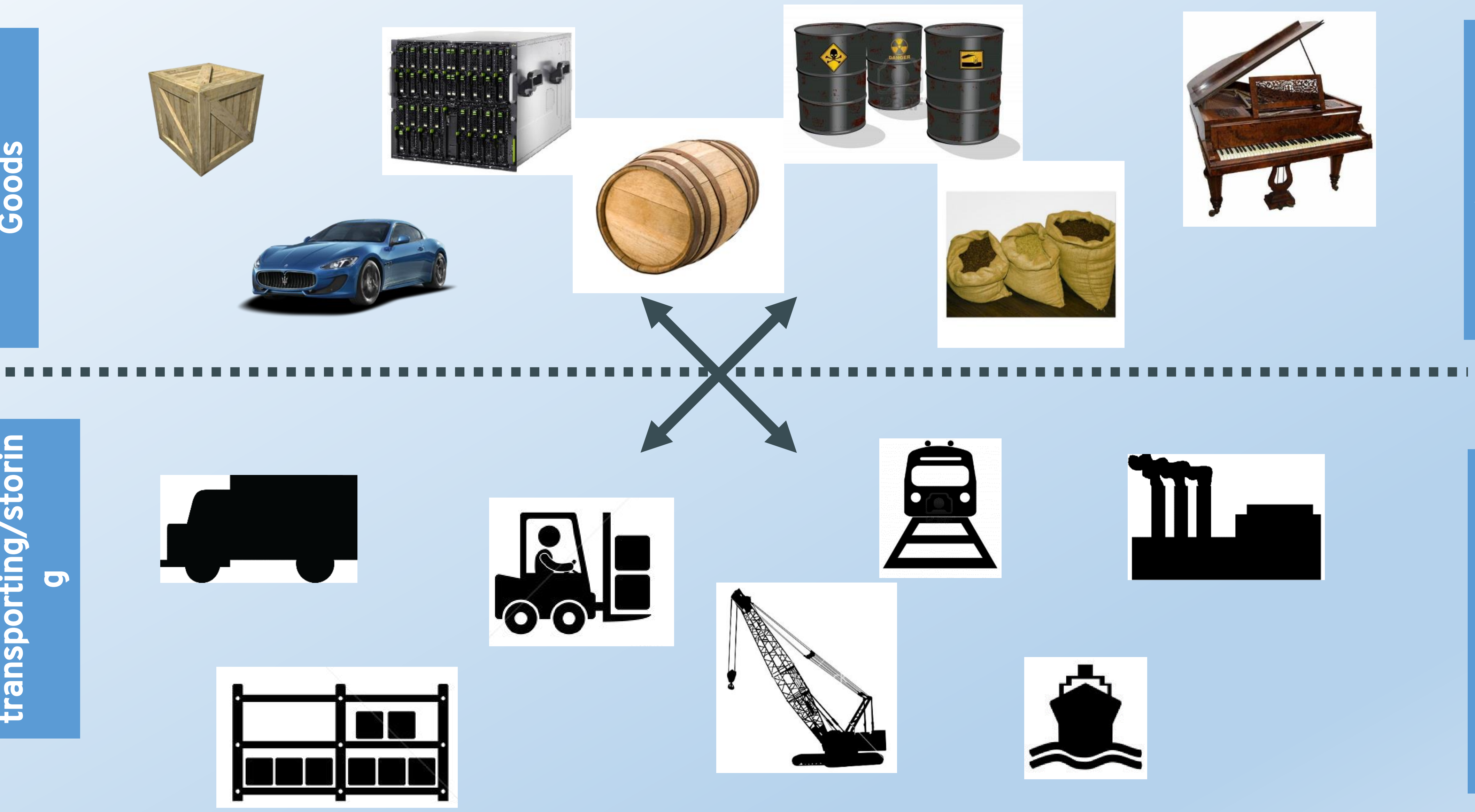
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor’s laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Cargo Transport Pre-1960

Multiplicity of Goods

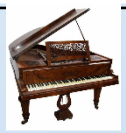
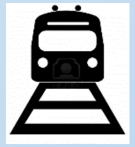
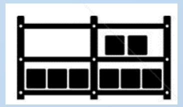
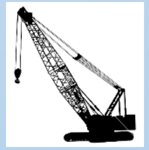
Do I worry about how goods interact (e.g. coffee beans next to spices)



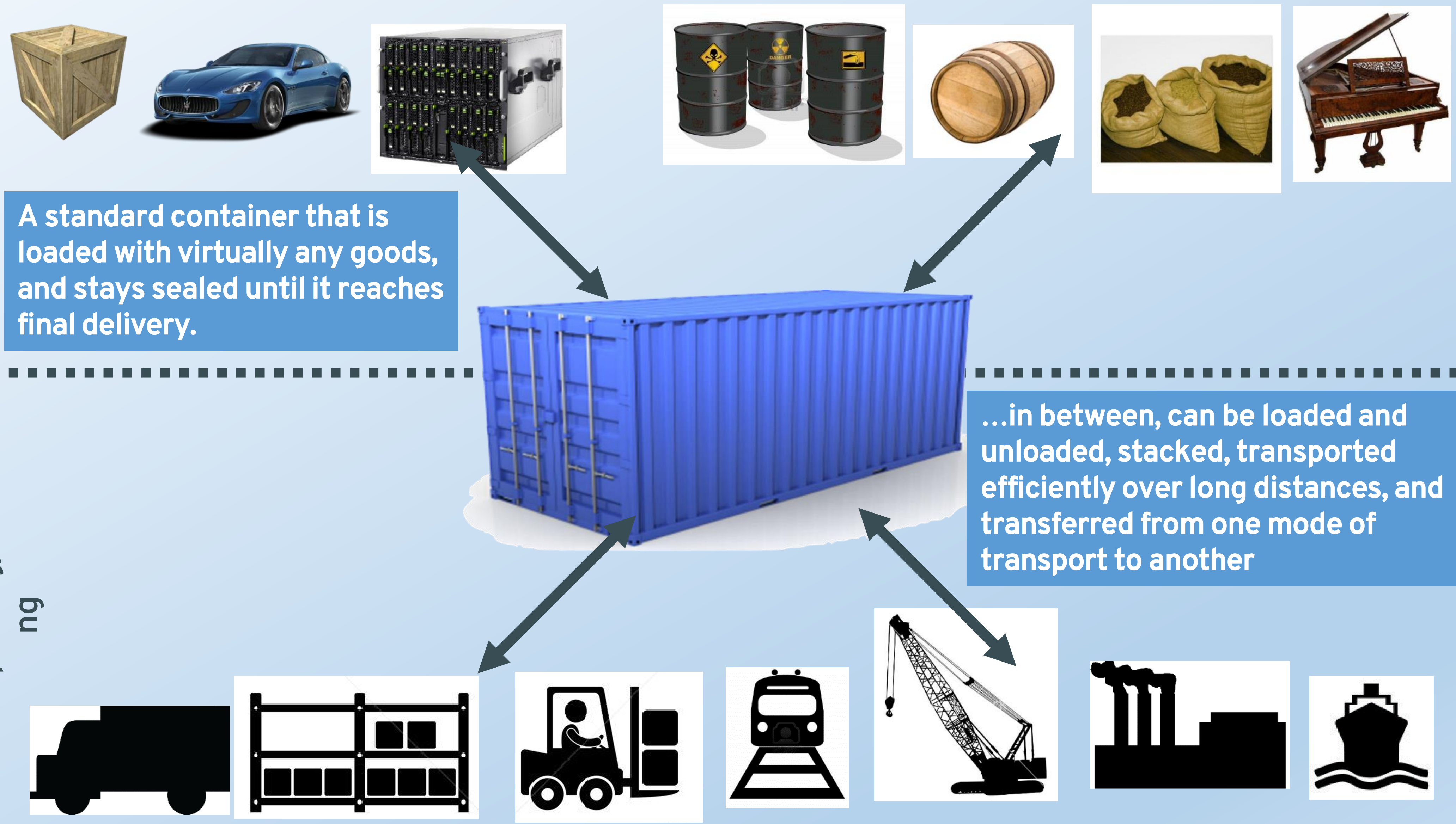
Multipilicity of methods for transporting/storin g

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Also a matrix from hell

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

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.

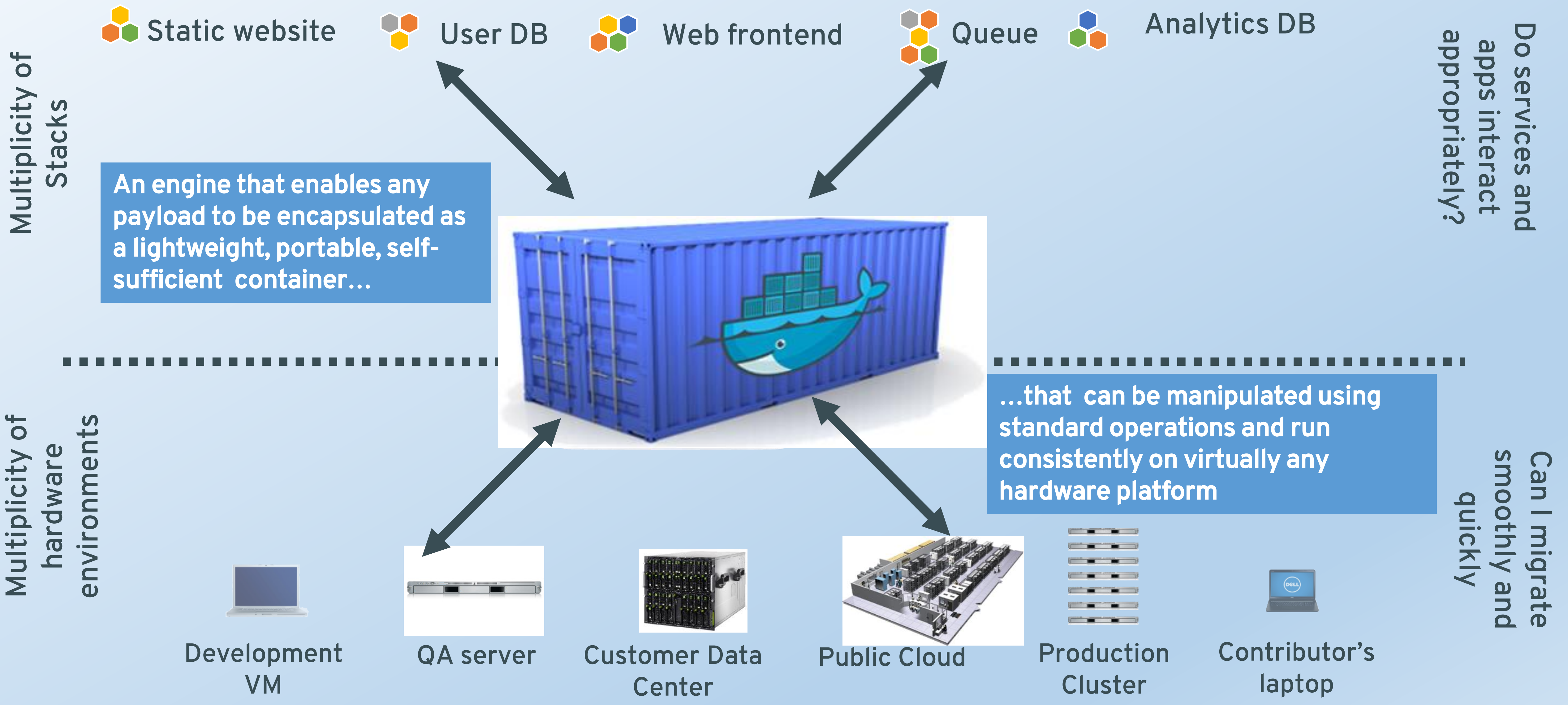
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multiplicity of methods for transporting/storing

…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)

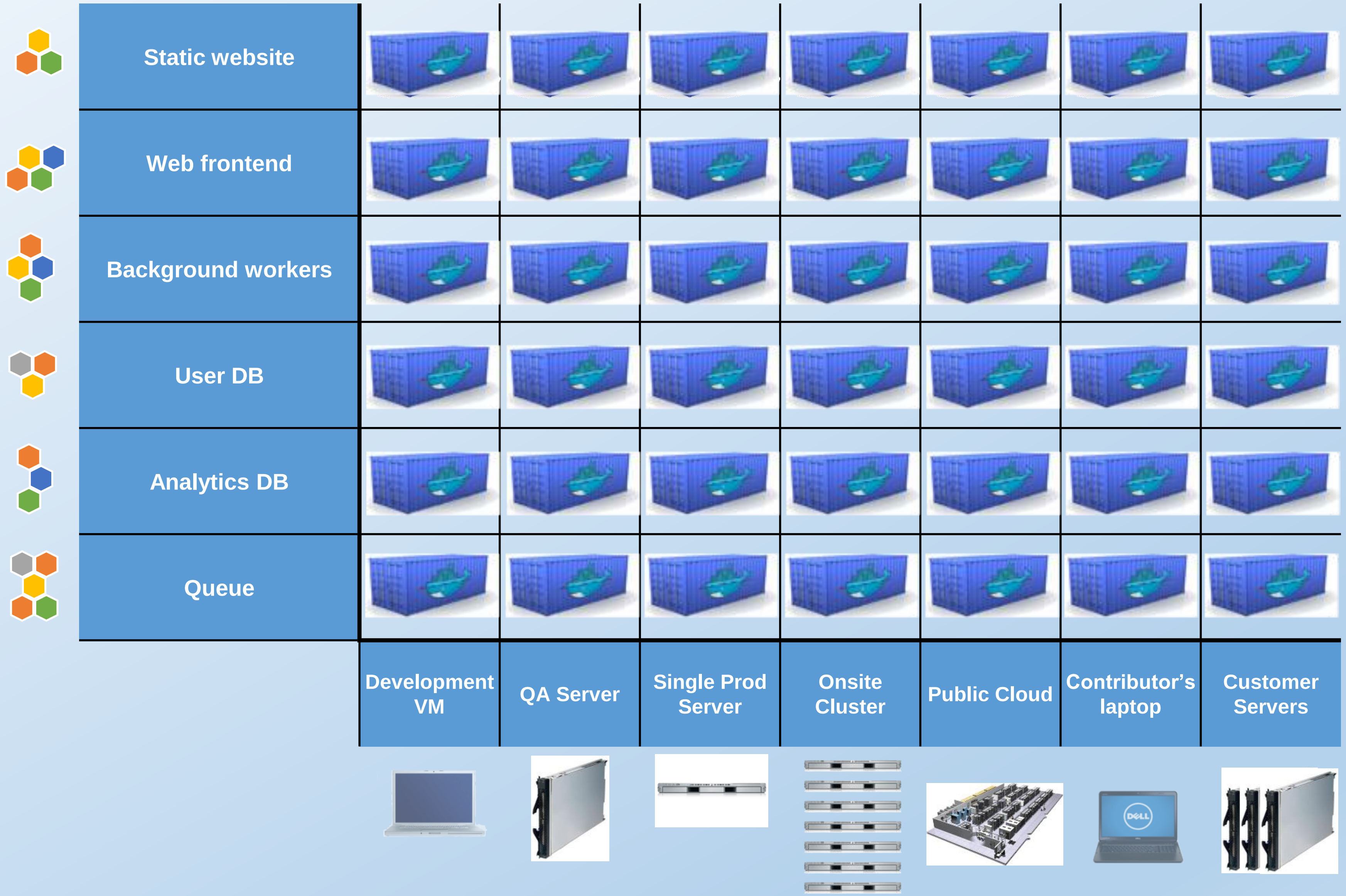
Docker is a shipping container system for code



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

Docker eliminates the matrix from Hell



Why Developers Care

- 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

Why Developers Care

- 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

Why Devops Cares?

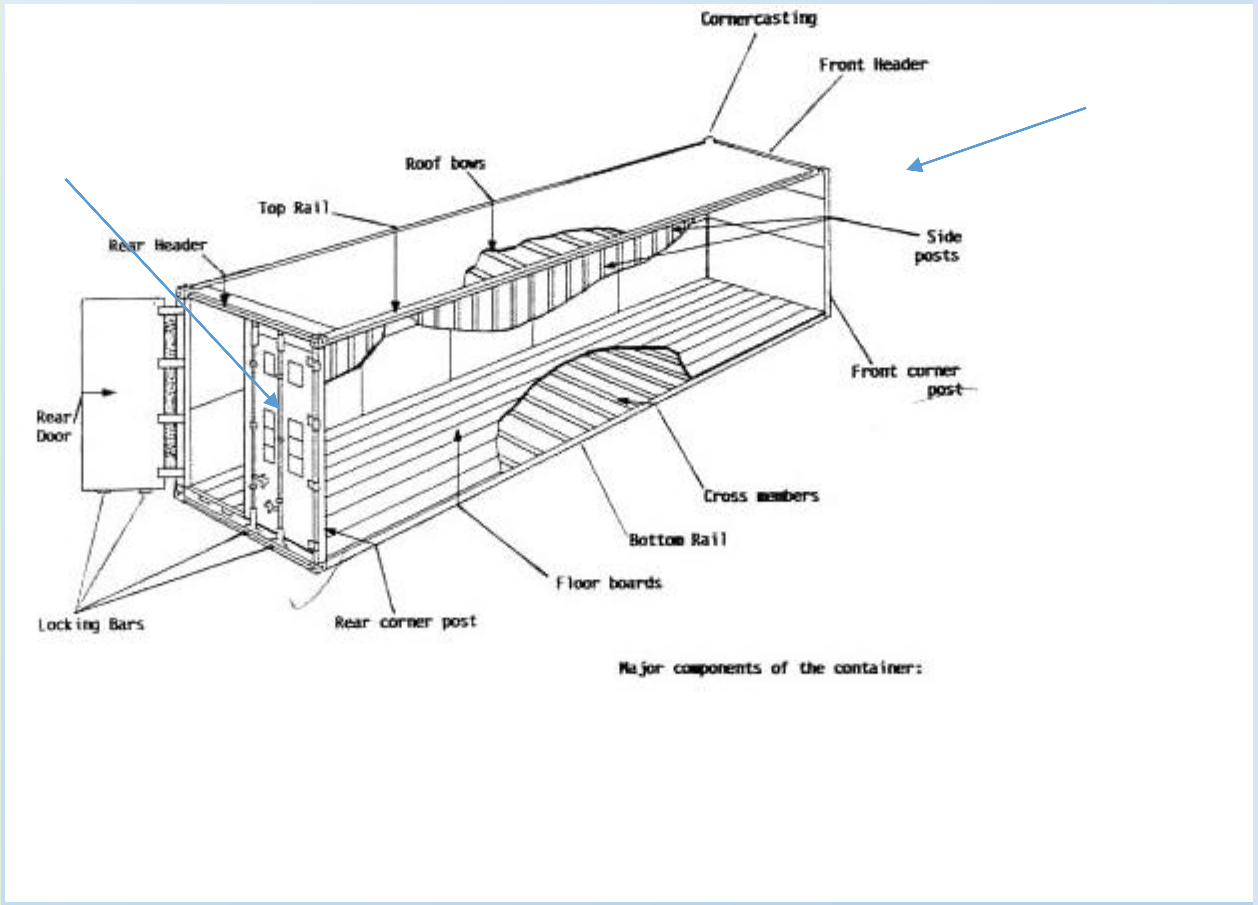
- Configure once…run anything
  - Make the entire lifecycle more efficient, consistent, and repeatable
  - Increase the quality of code produced by developers.
  - Eliminate inconsistencies between development, test, production, and customer environments

Why Devops Cares?

- Support segregation of duties
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems
- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs

Why it works—separation of concerns

- The Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same



- The Administrator
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

More technical explanation

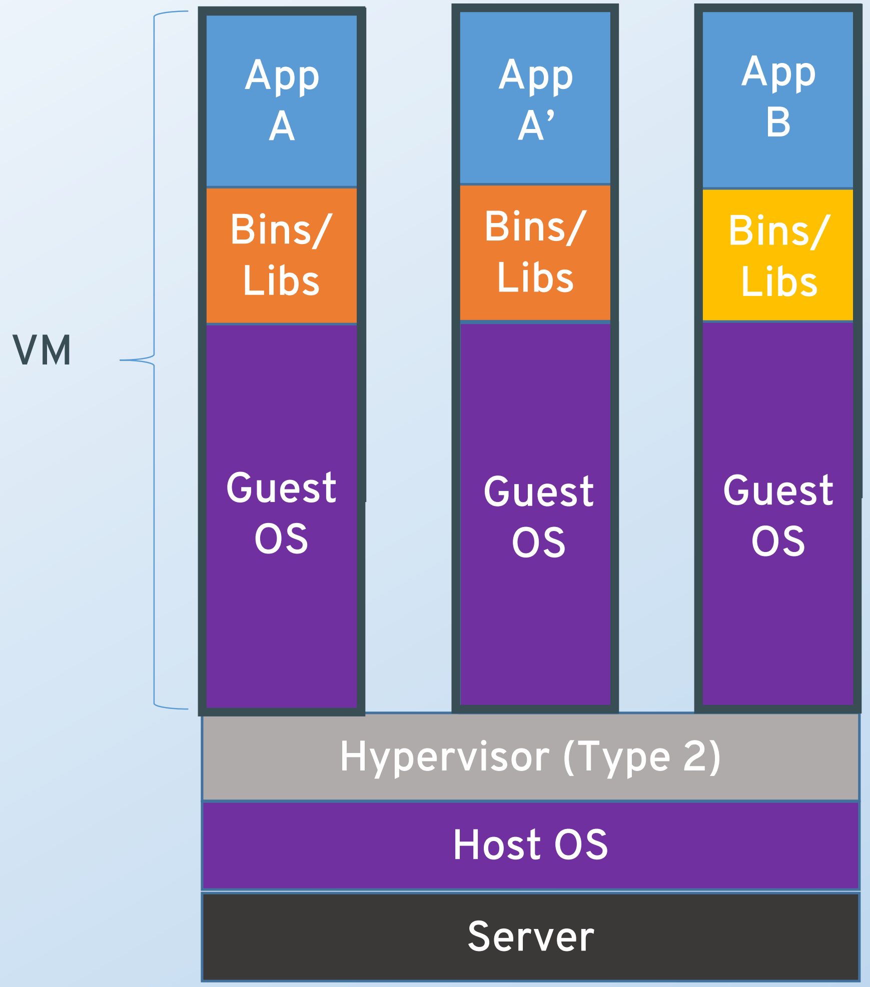
WHY

- Run everywhere
  - Regardless of kernel version (2.6.32+)
  - Regardless of host distro
  - Physical or virtual, cloud or not
  - Container and host architecture must match*
- Run anything
  - If it can run on the host, it can run in the container
  - i.e. if it can run on a Linux kernel, it can run

WHAT

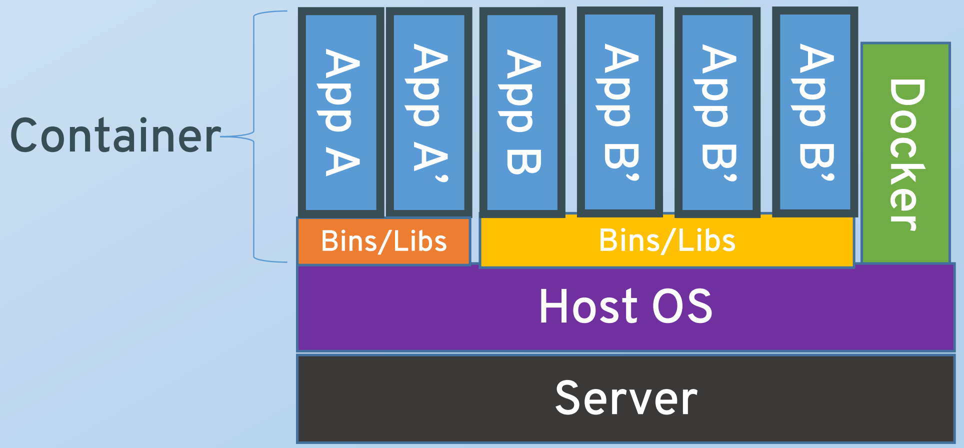
- High Level—It’s a lightweight VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - Can have its own /sbin/init (different from host)
- Low Level—It’s chroot on steroids
  - Can also *not* have its own /sbin/init
  - Container=isolated processes
  - Share kernel with host
  - No device emulation (neither HVM nor PV) from host)

Containers vs. VMs



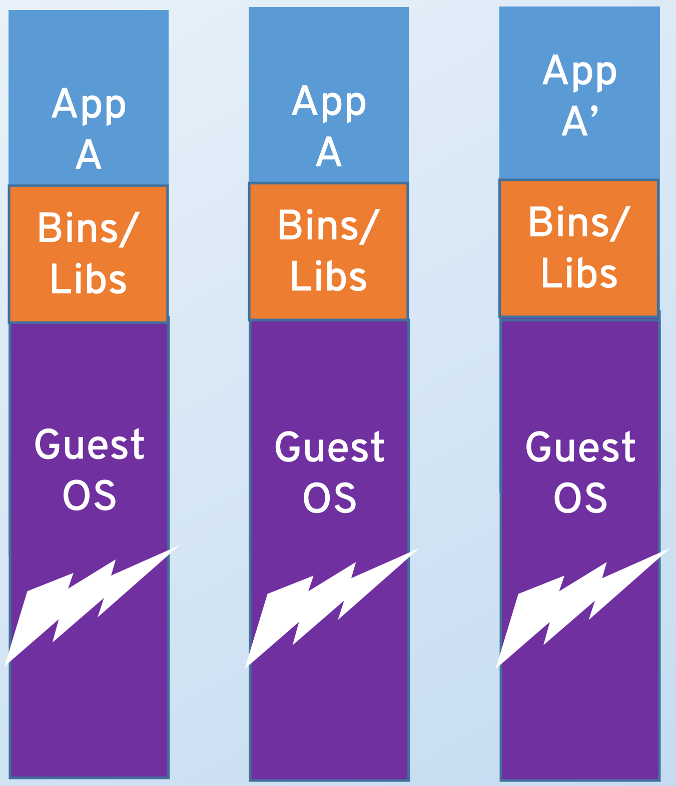
Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Why are Docker containers lightweight?

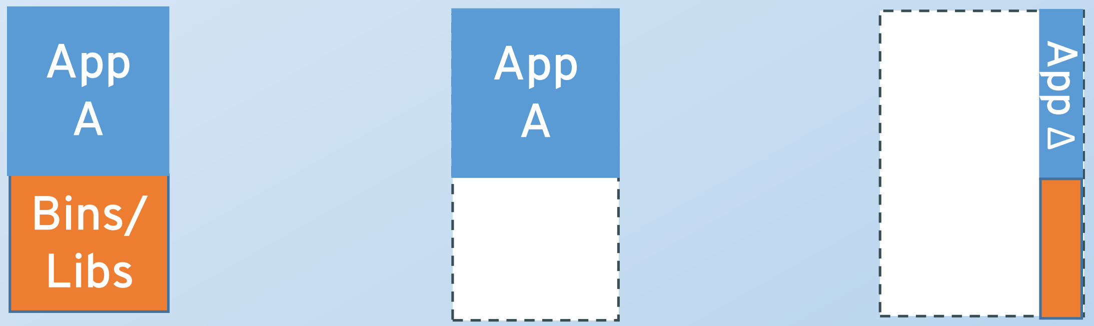
VMs



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App

(No OS to take up space, resources, or require restart)

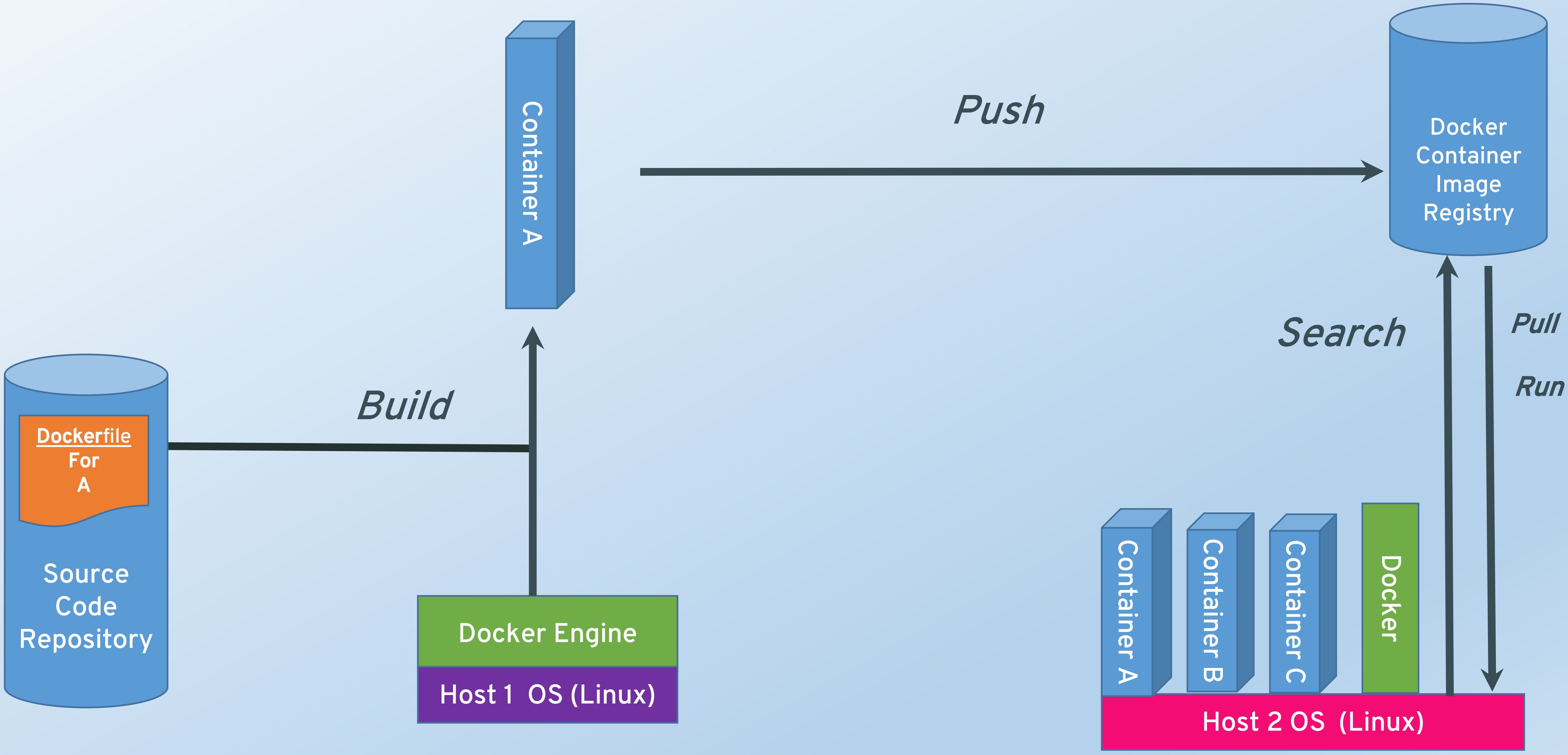
Copy of App

No OS. Can Share bins/libs

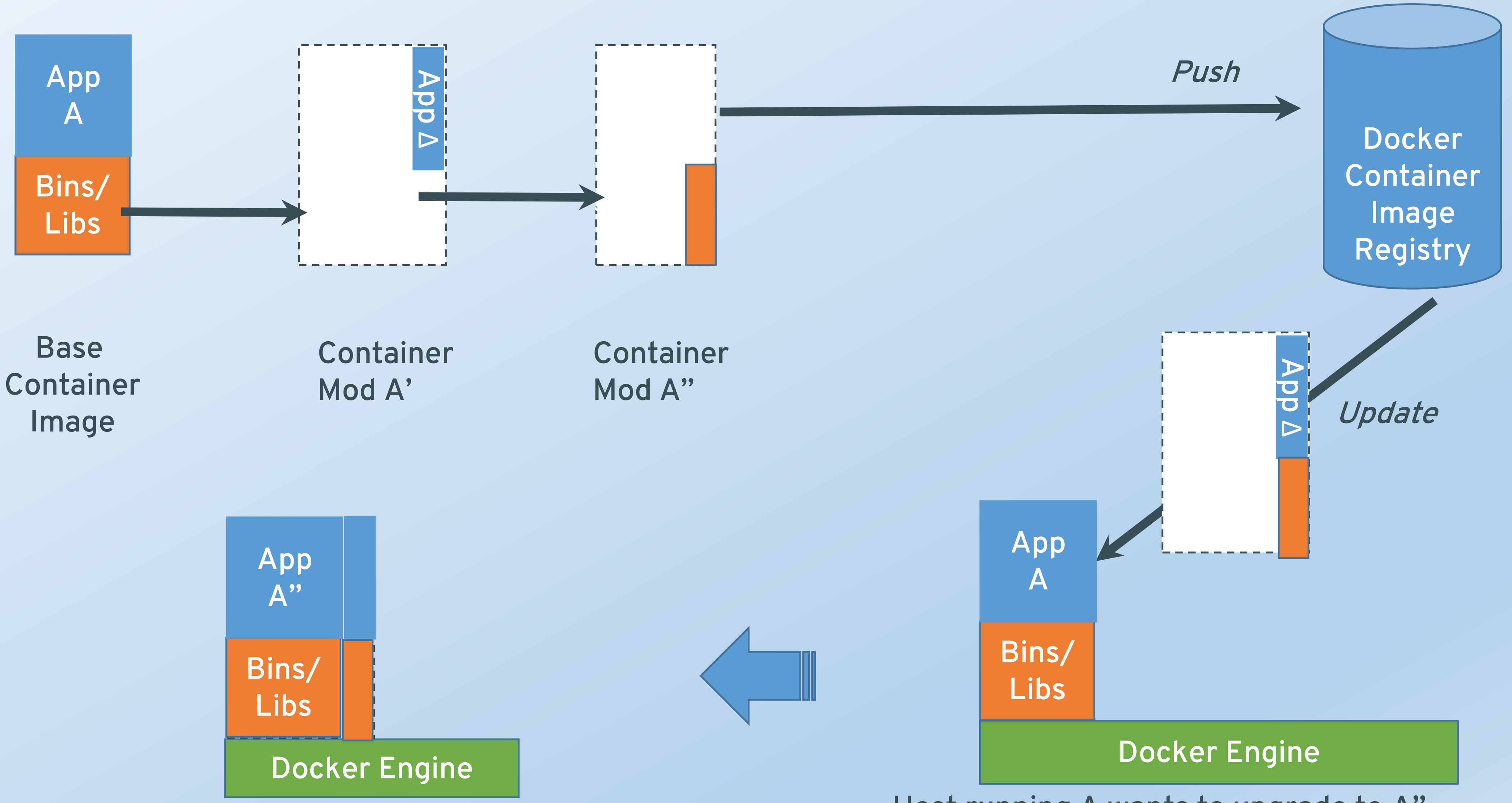
Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

What are the basics of the Docker system?



Changes and Updates



Base Container Image

Container Mod A’

Container Mod A”

Host is now running A”

Host running A wants to upgrade to A”. Requests update. Gets only diffs

Ecosystem Support

- Operating systems
  - Virtually any distribution with a 2.6.32+ kernel
  - Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
  - CoreOS—Small core OS purpose built with Docker
- OpenStack
  - Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana release
- Private PaaS
  - OpenShift
  - Solum (Rackspace, OpenStack)
  - Other TBA
- Public PaaS
  - Deis, Voxoz, Cocaine (Yandex), Baidu PaaS

Ecosystem Support

- Public IaaS
  - Native support in Rackspace, Digital Ocean,+++
  - AMI (or equivalent) available for AWS & other
- DevOps Tools
  - Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible +++
- Orchestration tools
  - Mesos, Heat, ++
  - Shipyard & others purpose built for Docker
- Applications
  - 1000’s of Dockerized applications available at index.docker.io

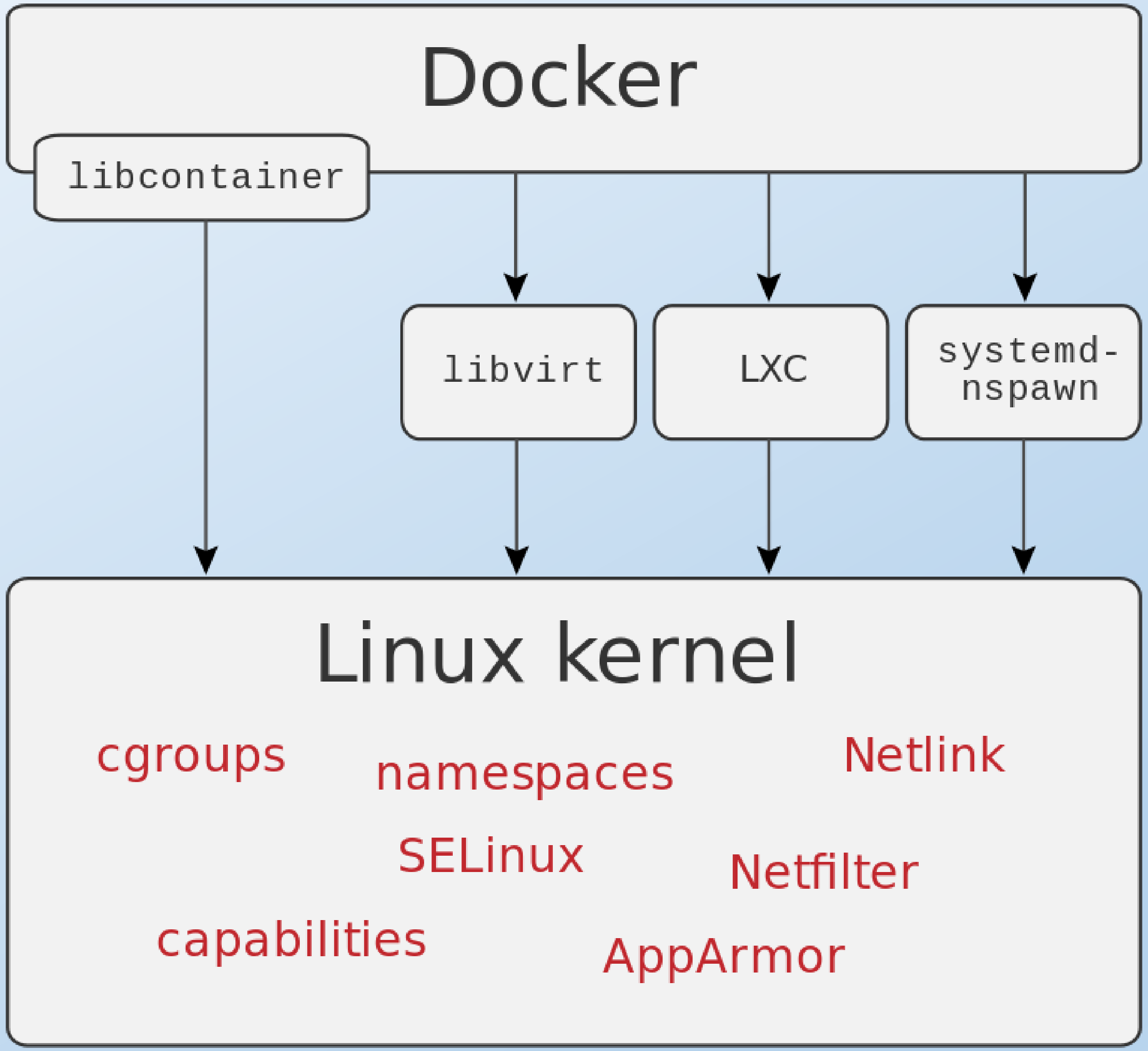
Use Cases

Use Case	Examples
Clusters	Building a MongoDB cluster using docker
	Production Quality MongoDB Setup with Docker
	Wildfly cluster using Docker on Fedora
Build your own PaaS	OpenSource PaaS built on Docker, Chef, and Heroku Buildpacks
Web Based Environment for Instruction	JiffyLab – web based environment for the instruction, or lightweight use of, Python and UNIX shell
Easy Application Deployment	Deploy Java Apps With Docker = Awesome
	How to put your development environment on docker
	Running Drupal on Docker
	Installing Wordpress on Docker

Use Cases

Use Case	Examples
Create Secure Sandboxes	Docker makes creating secure sandboxes easier than ever
Create your own SaaS	Memcached as a Service
Automated Application Deployment	Multi-cloud Deployment with Docker
Continuous Integration and Deployment	Next Generation Continuous Integration & Deployment with dotCloud's Docker and Strider
	Testing Salt States Rapidly With Docker
Lightweight Desktop Virtualization	Docker Desktop: Your Desktop Over SSH Running Inside Of A Docker Container

Docker Architecture



Namespaces

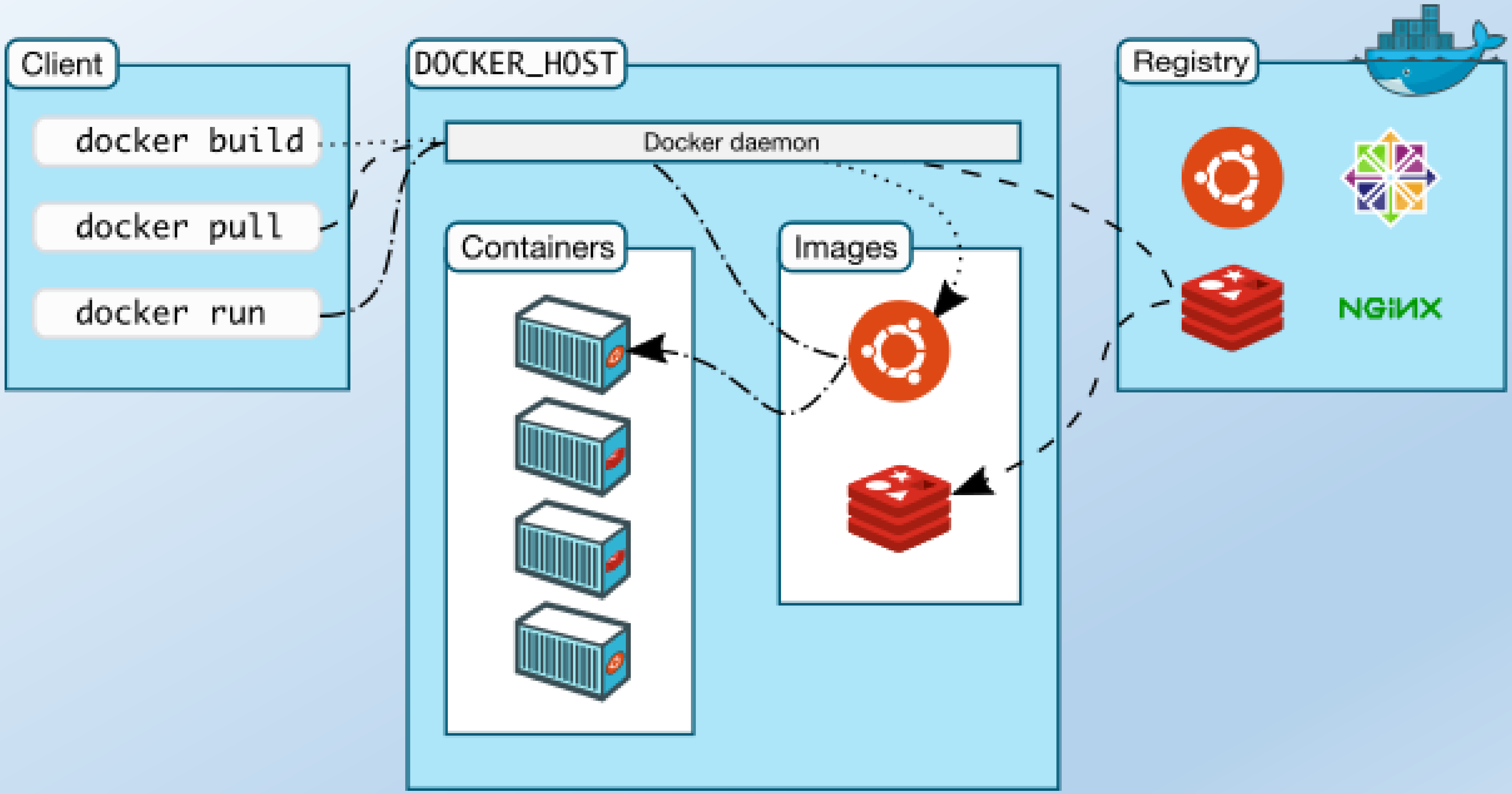
- Docker takes advantage of a technology called namespaces to provide the isolated workspace we call the container.
- When you run a container, Docker creates a set of namespaces for that container.
- Some of the namespaces that Docker Engine uses on Linux are:

1. The pid namespace: Process isolation (PID: Process ID).
2. The net namespace: Managing network interfaces (NET: Networking).
3. The ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).
4. The mnt namespace: Managing mount-points (MNT: Mount).
5. The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System)

Control groups

- Docker Engine on Linux also makes use of another technology called cgroups or control groups.
- A key to running applications in isolation is to have them only use the resources you want.
- This ensures containers are good multi-tenant citizens on a host.
- Control groups allow Docker Engine to share available hardware resources to containers and, if required, set up limits and constraints.
- For example, limiting the memory available to a specific container.

Docker Architecture

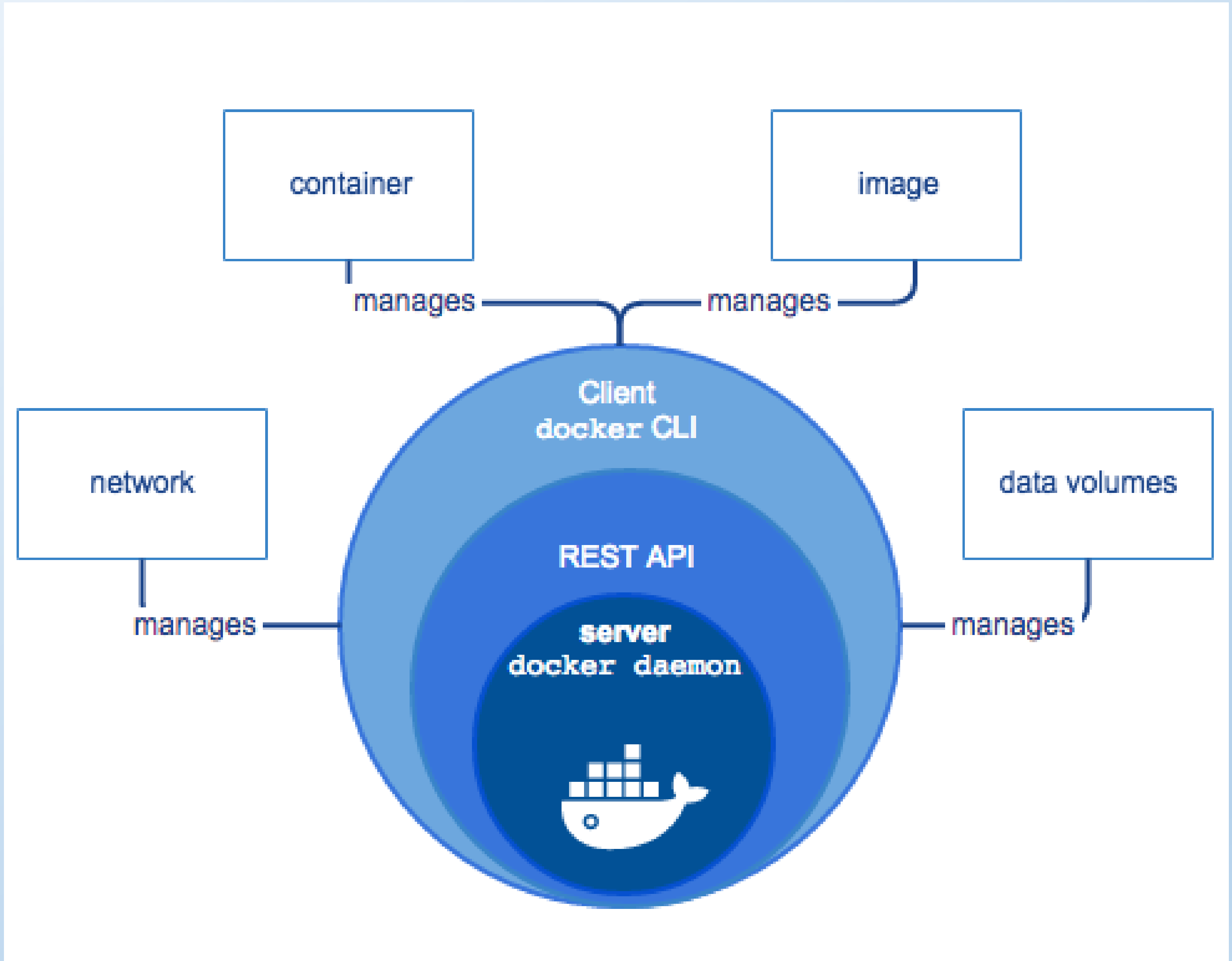


Docker Engine

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.

Docker Engine



Docker images

- A Docker image is a read-only template. For example, an image could contain an Ubuntu operating system with Apache and your web application installed.

- Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.

- Docker images are the build component of Docker.

Docker File

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- Using docker build users can create an automated build that executes several command-line instructions in succession.
- The docker build command builds an image from a Dockerfile and a context.

Docker File - Example

- Instructions

INSTRUCTION arguments

Eg. RUN echo 'we are running some # of cool things!’

- Parser directives:

FROM ImageName

Example:

```
#Comment
FROM windowsservercore
COPY testfile.txt c:\
RUN dir c:\
```

To Build image using this file:

```
docker build -f /path/to/a/Dockerfile  .
```

Docker Hub

- Docker registries hold images.
- These are public or private stores from which you upload or download images.
- The public Docker registry is provided with the Docker Hub. (hub.docker.com)
- It serves a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created.
- Docker registries are the distribution component of Docker.

Docker CLI

- The CLI makes use of the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands.

- Many other Docker applications make use of the underlying API and CLI.

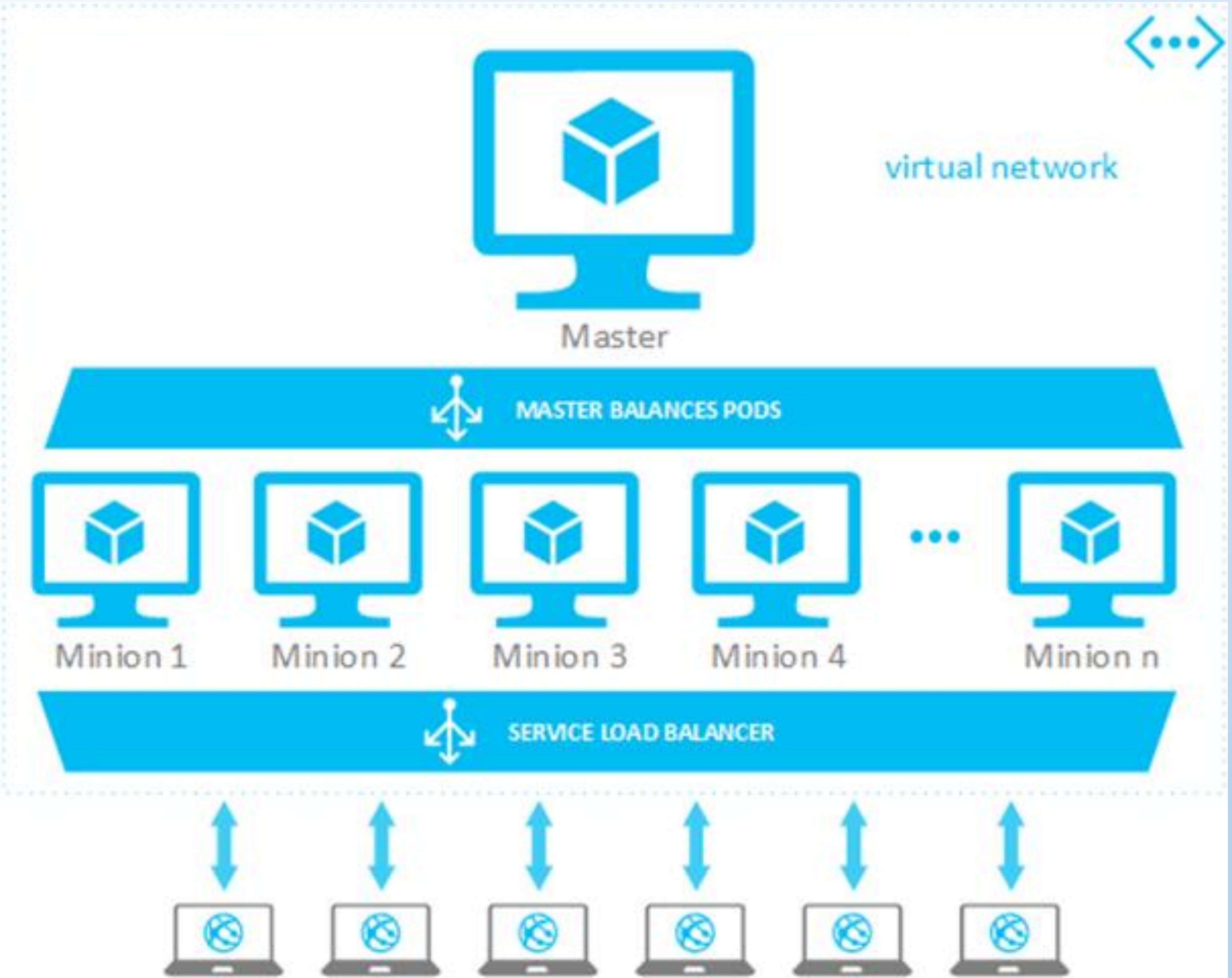
- The CLI is also used to issue commands.

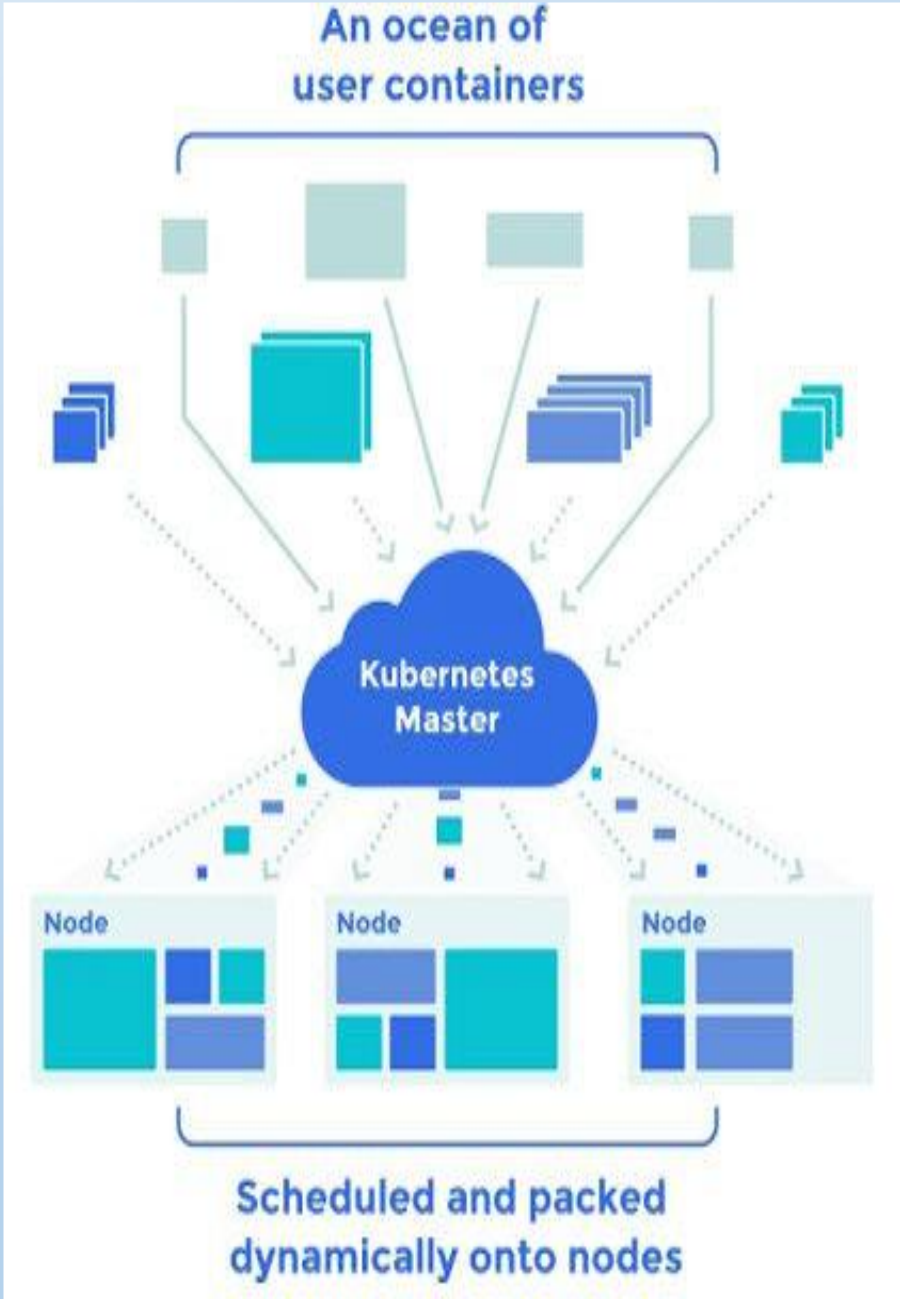
Kubernetes

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

- With Kubernetes, you can:
  - Deploy your applications quickly and predictably.
  - Scale your applications on the fly.
  - Seamlessly roll out new features.
  - Optimize use of your hardware by using only the resources you need

Kubernetes Architecture




An ocean of user containers
Kubernetes Master
Node
Node
Node
Scheduled and packed dynamically onto nodes

Kubernetes Features

Kubernetes is:

- portable: public, private, hybrid, multi-cloud
- extensible: modular, pluggable, hookable, composable
- self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

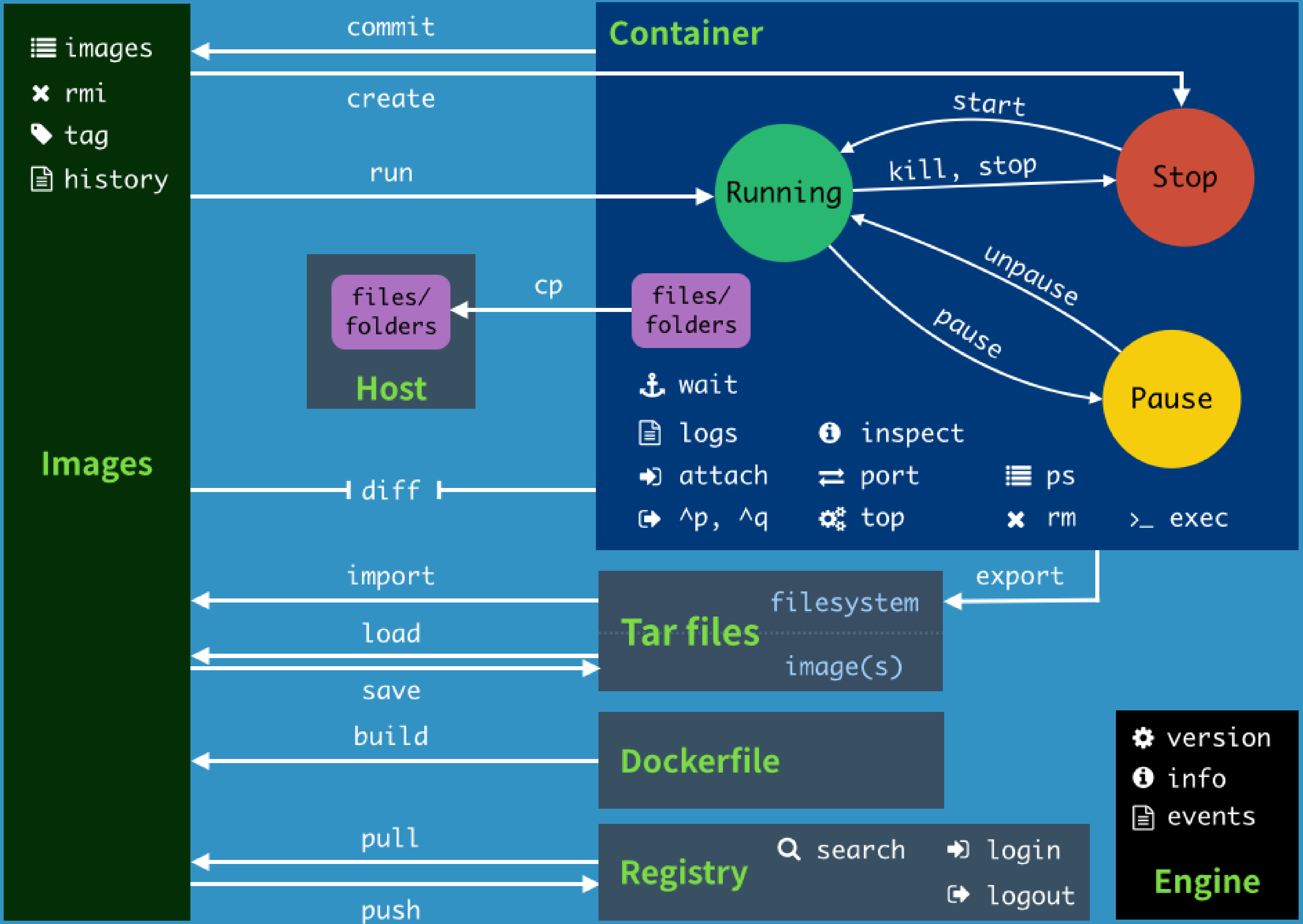
The Kubernetes project was started by Google in 2014.

Docker commands

docker [OPTIONS] COMMAND [arg...]

- docker run
- docker ps
- docker stop|start|restart
- docker build
- docker rm
- docker rmi
- docker logs

Docker Commands Diagram



@fntsrlike

Demo Time!

Q&A

docker

Thank you