Introduction to Docker

Contents

Why all the excitement?

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

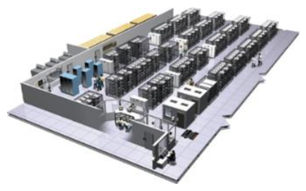
Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Public Cloud



Production Cluster

Disaster recovery

Customer Data Center

Production Servers

Contributor's laptop

Can I migrate smoothly and quickly?

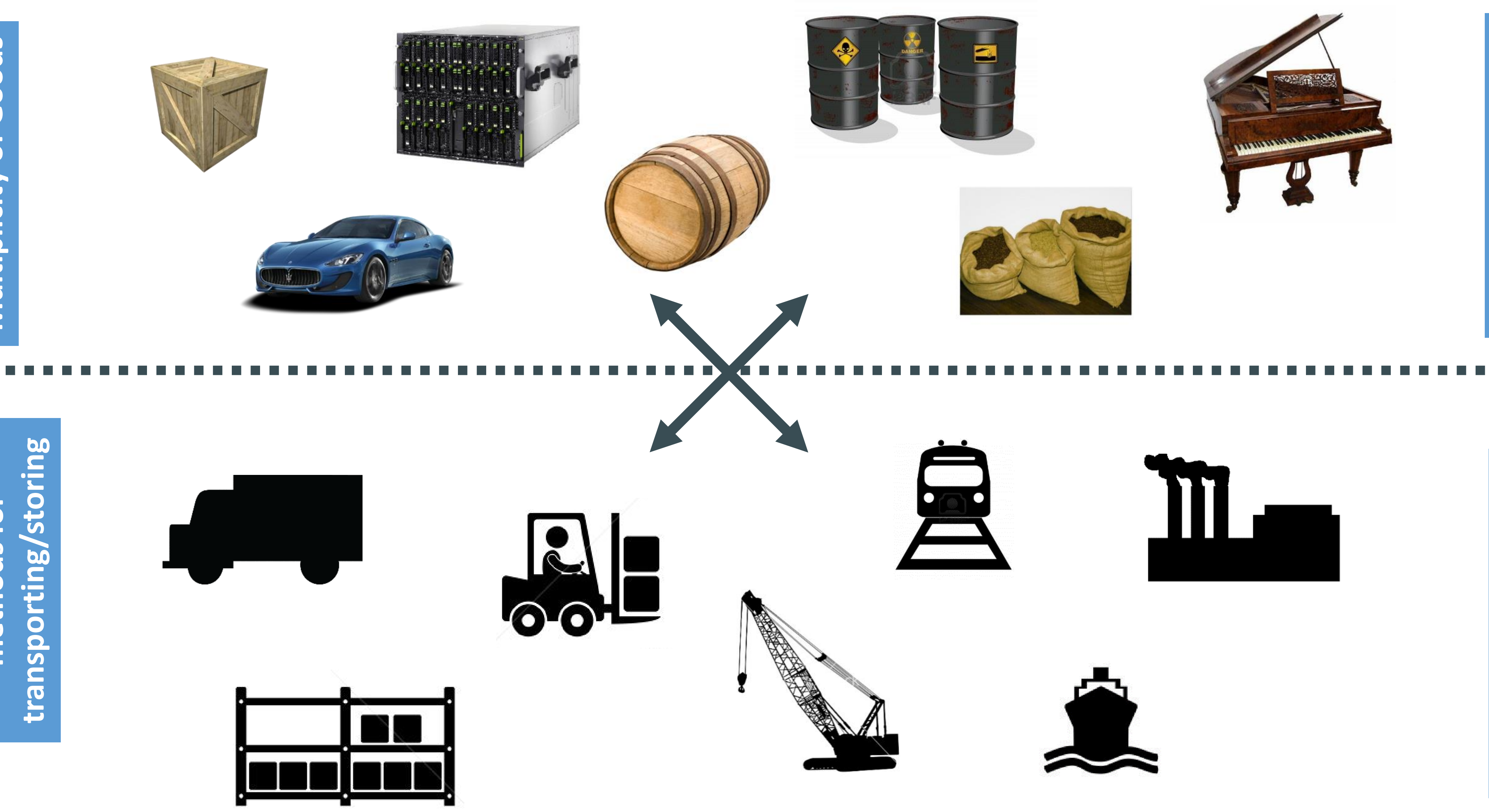
docker

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

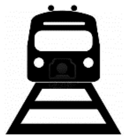
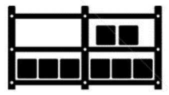


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing

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

Also a matrix from hell

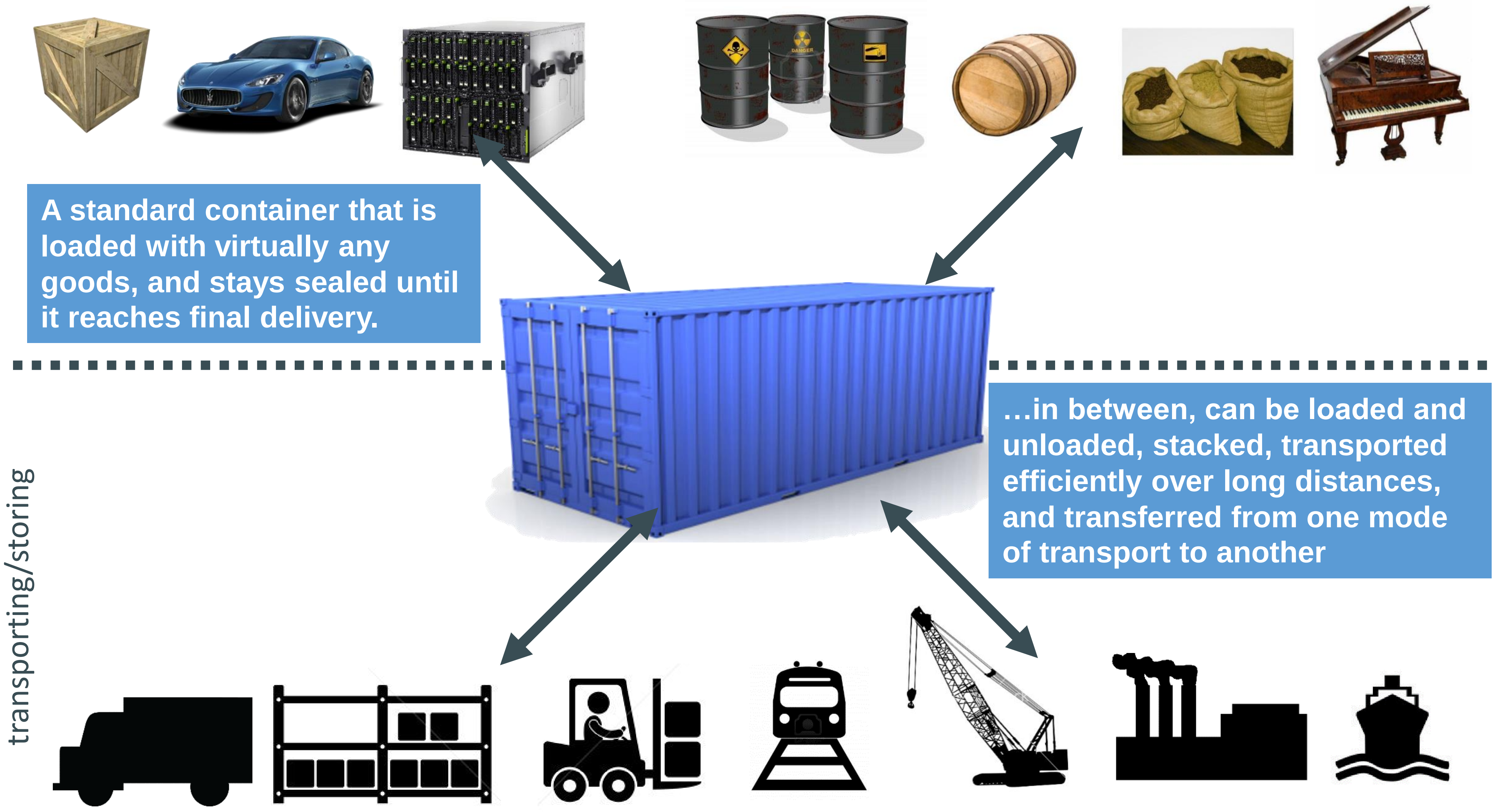
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Multiplicity of Goods

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

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

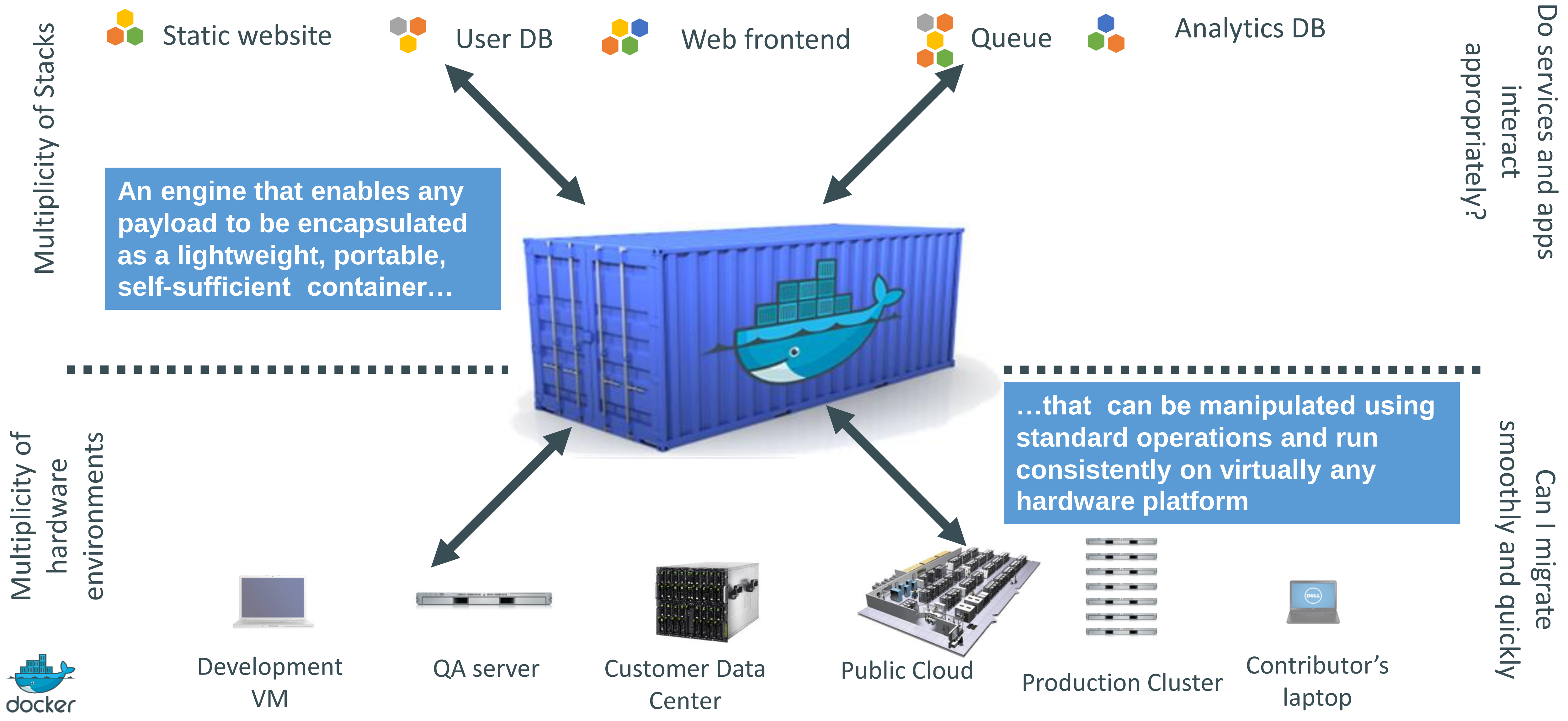


…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

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

	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	📦	📦	📦	📦	📦	📦	📦

Why Developers Care

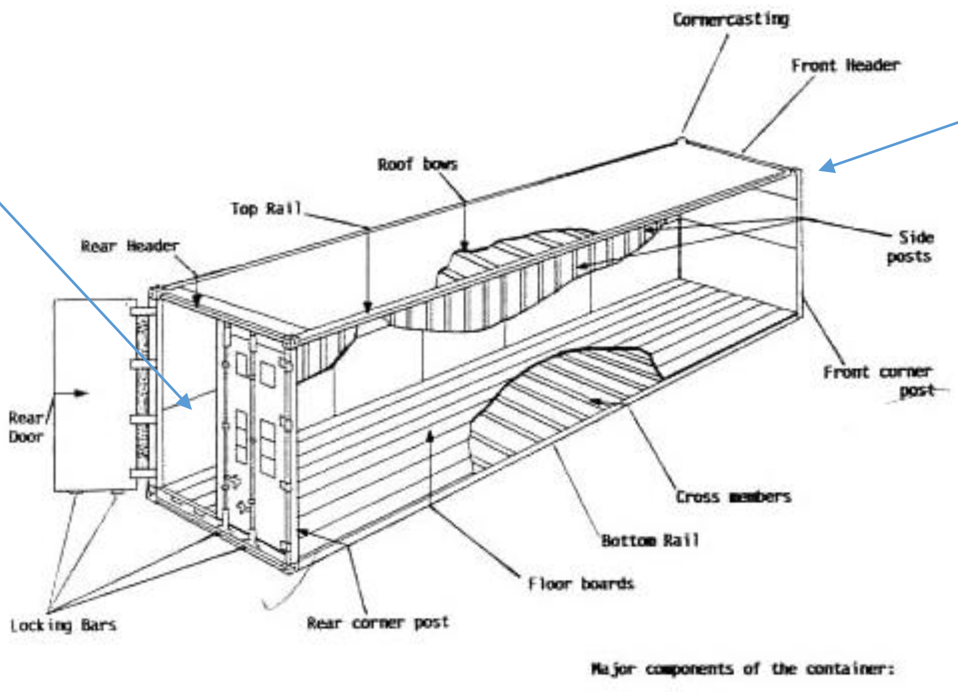
- Build once…(finally) run anywhere
  - A clean, safe, hygienic and portable runtime environment for your app.
  - No worries about missing dependencies, packages and other pain points during subsequent deployments.
  - Run each app in its own isolated container, so you can run various versions of libraries 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

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
  - 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

- Dan 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



- Oscar the Ops Guy
    - 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

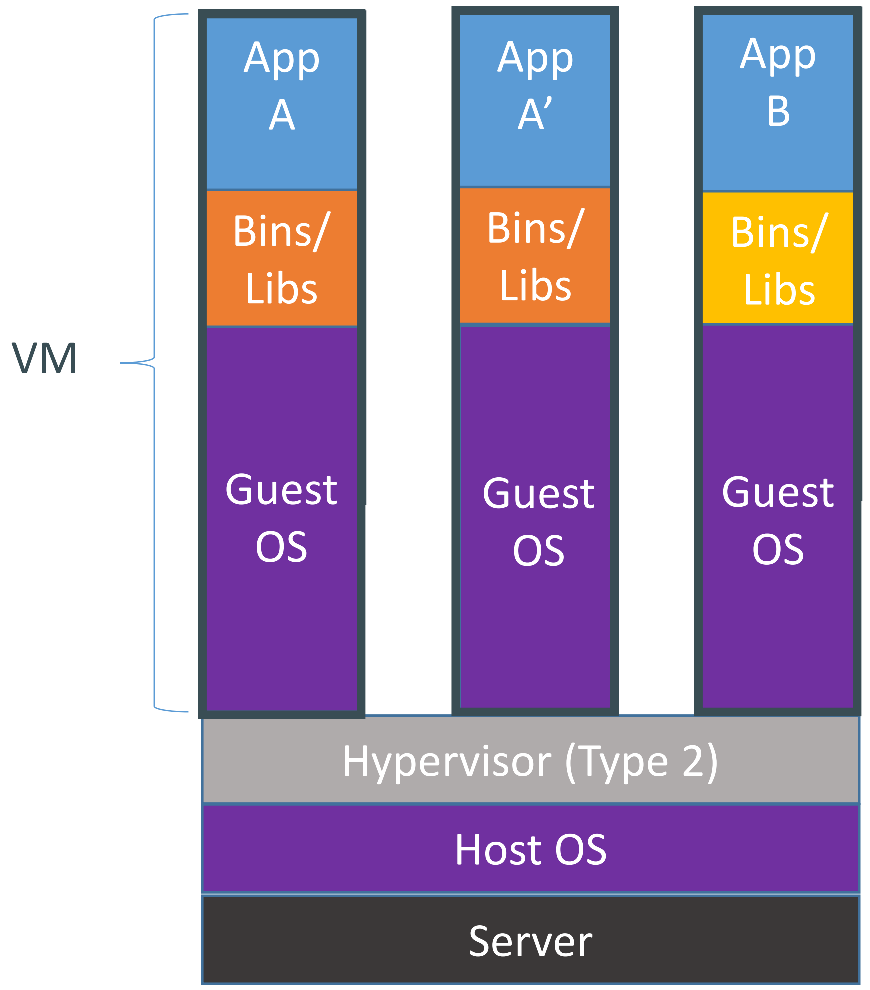
WHY

- Run everywhere
  - Regardless of kernel version (2.6.32+)
  - Regardless of host distro
  - Physical or virtual, cloud or not
- 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

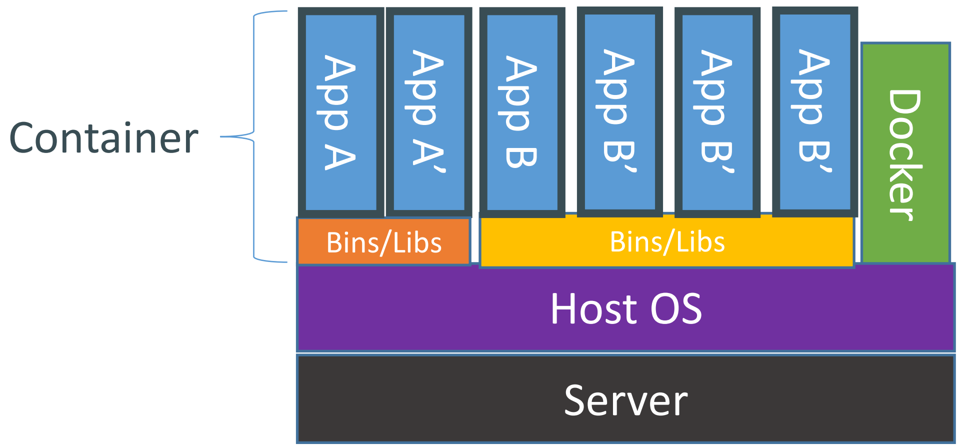
- High Level—It's a lightweight VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - <<machine container>>

Containers vs. VMs

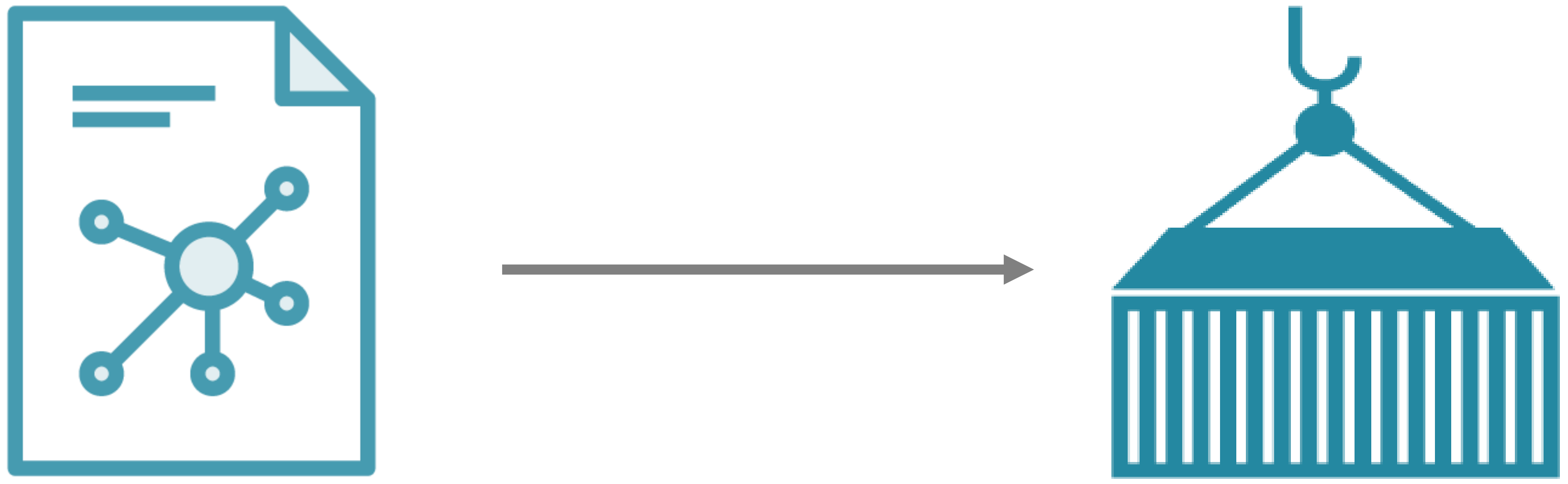


Containers are isolated, but share OS and, where appropriate, bins/libraries

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



The Role of Images and Containers



Docker Image

Example: Ubuntu with Node.js and Application Code

Docker Container

Created by using an image. Runs your application.

Using Docker: Build, Ship, Run Workflow

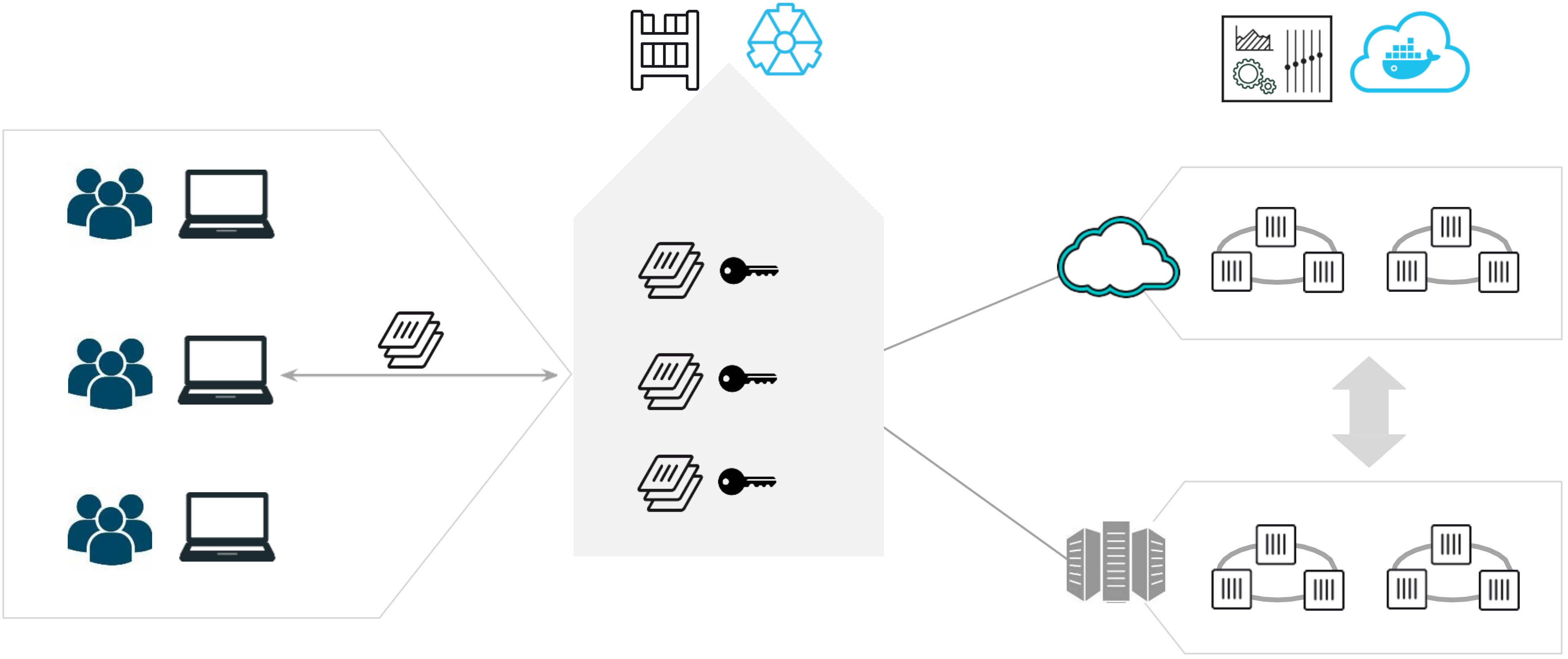
Developers

IT Operations

BUILD
Development Environments

SHIP
Create & Store Images

RUN
Deploy, Manage, Scale



Some Docker vocabulary

Docker Image

The basis of a Docker container. Represents a full application

Docker Container

The standard unit in which the application service resides and executes

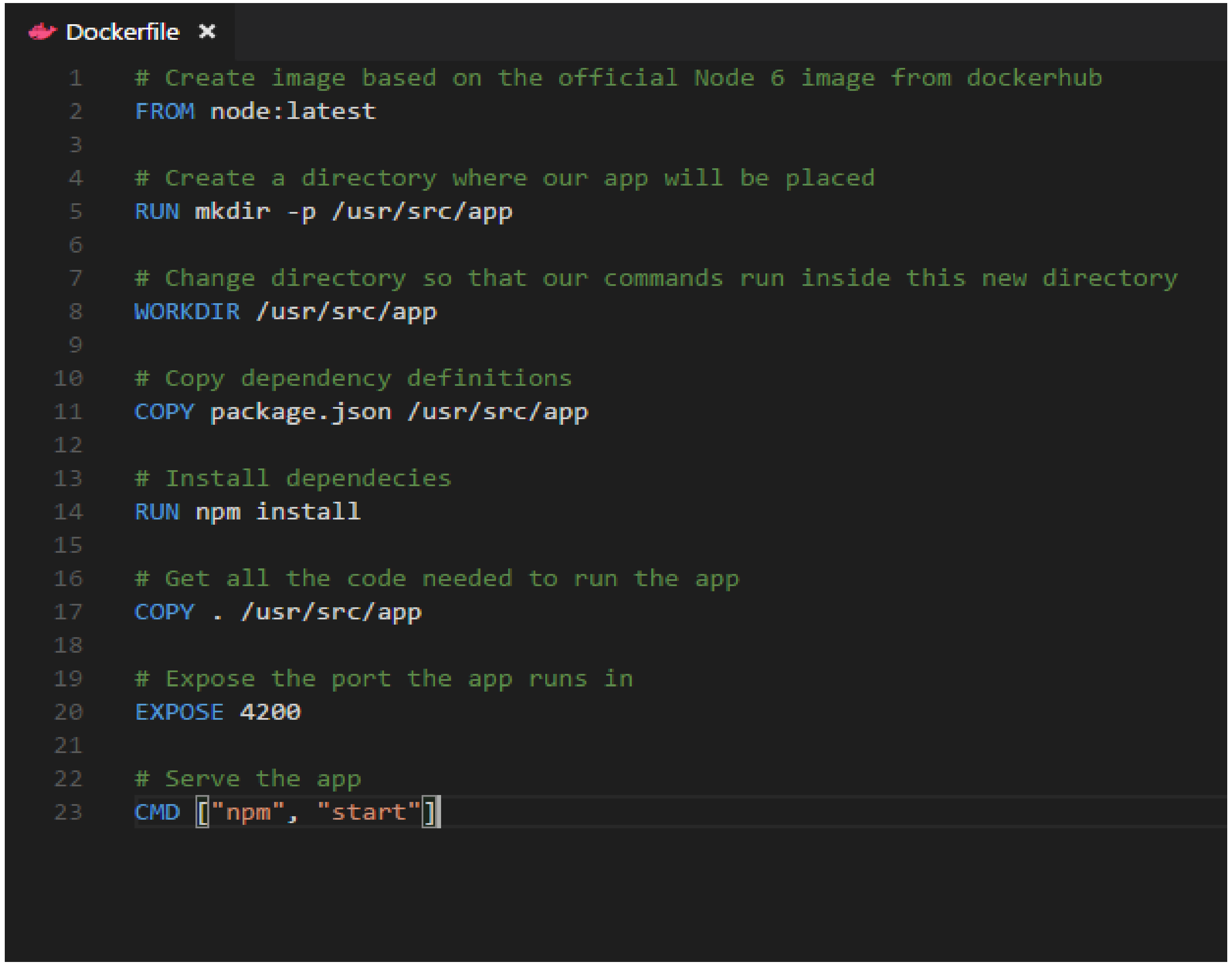
Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))

Cloud or server based storage and distribution service for your images

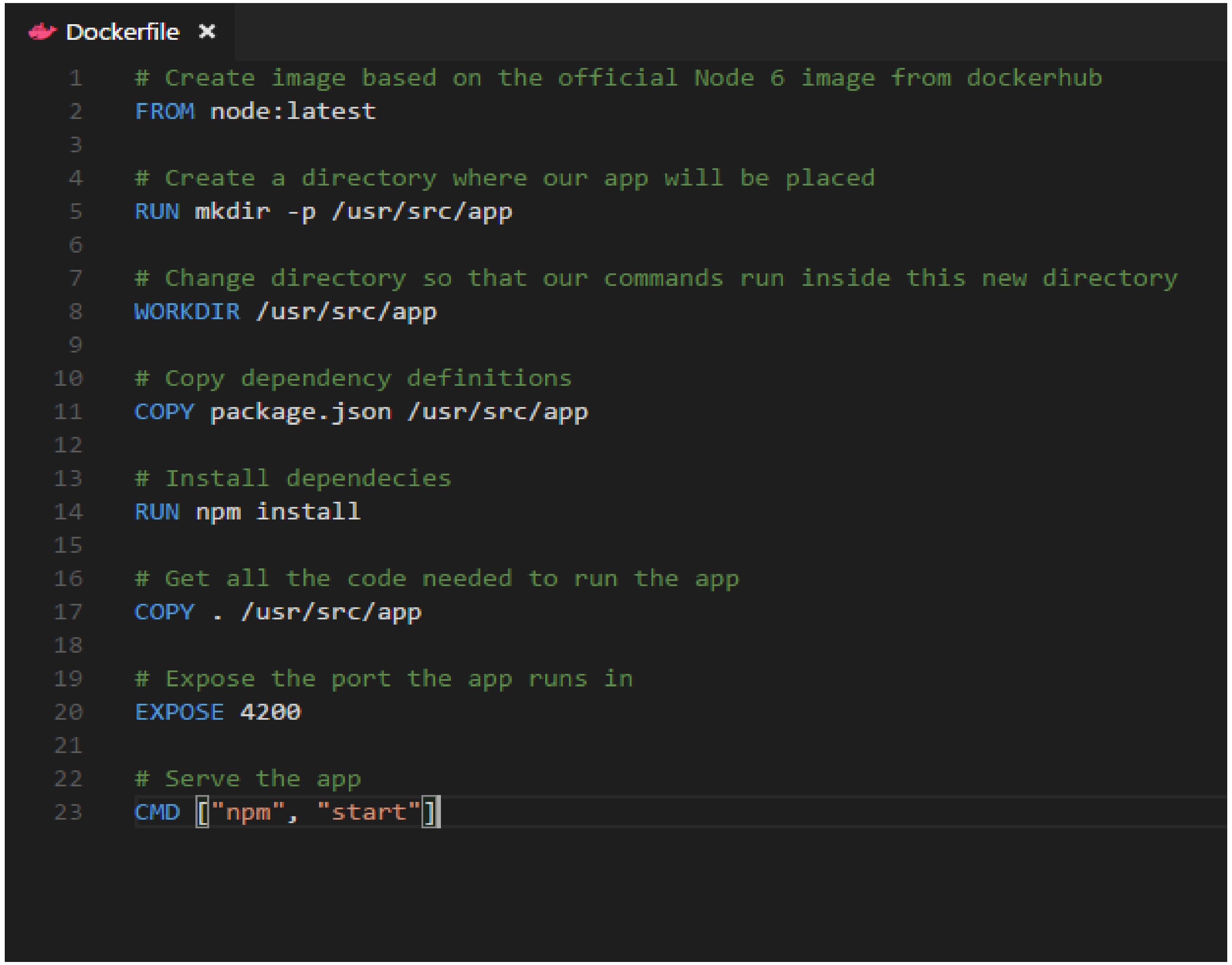
Dockerfile – Linux Example



```dockerfile
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

- Instructions on how to build a Docker image
- Looks very similar to “native” commands
- Important to optimize your Dockerfile

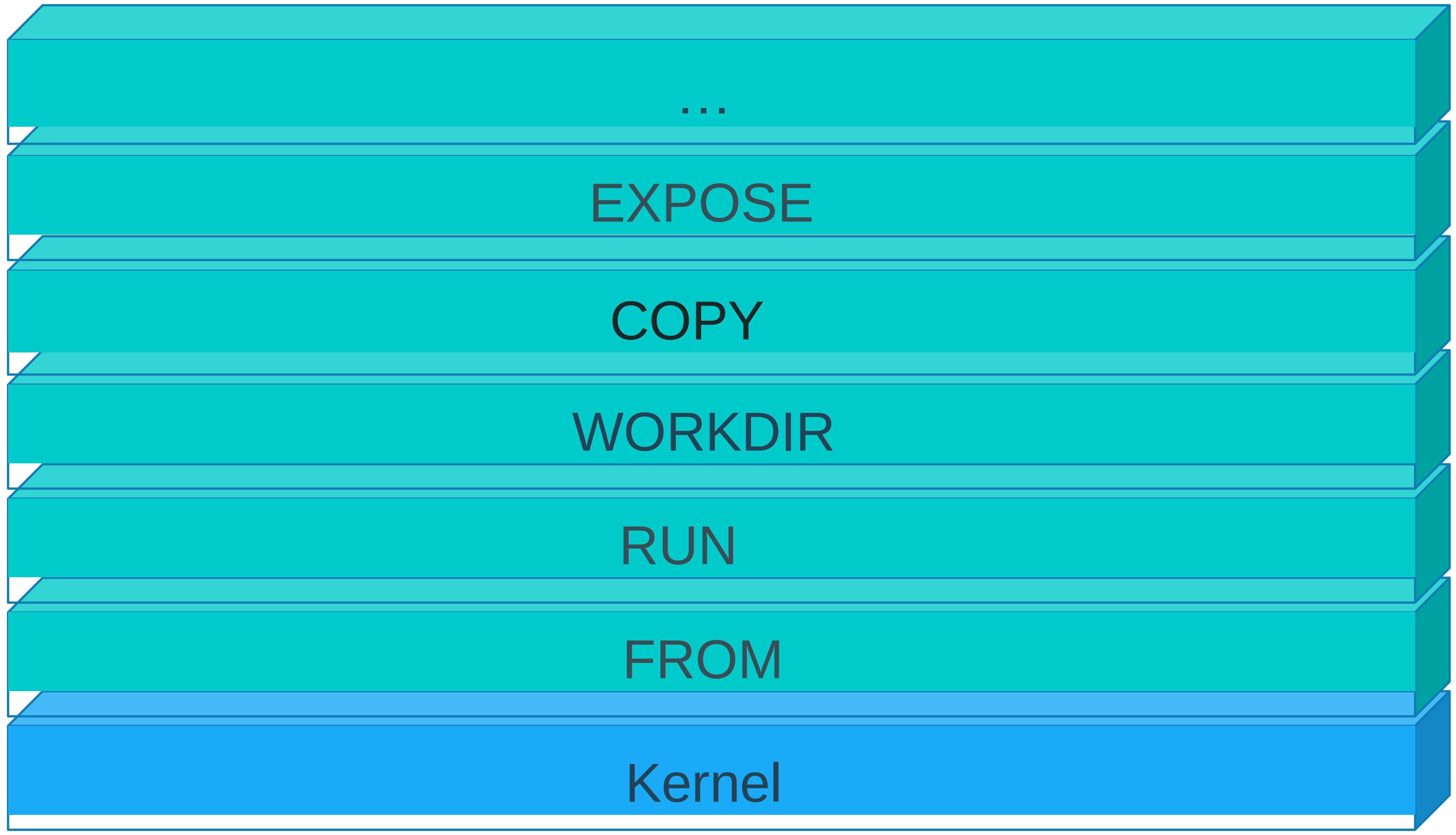
Let’s Go Back to Our Dockerfile



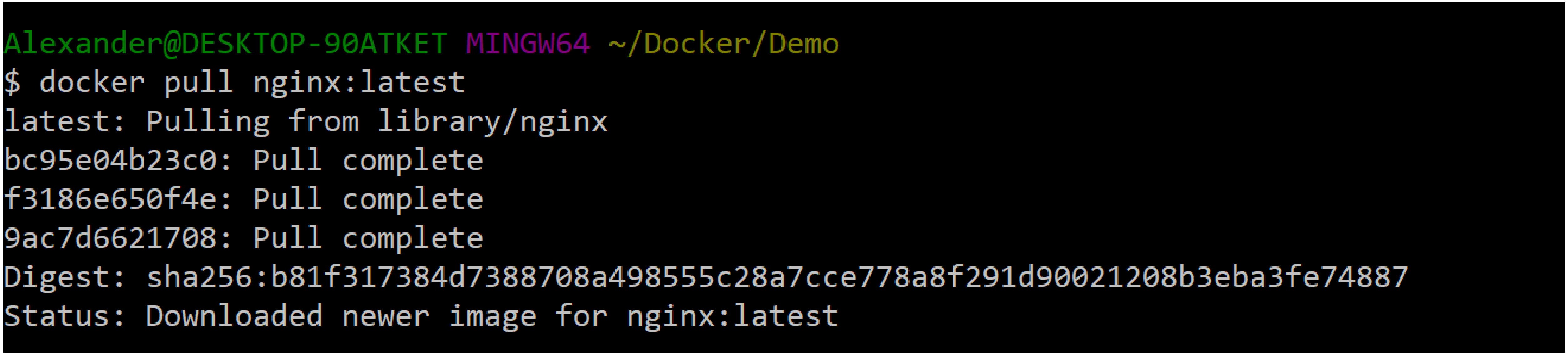
Dockerfile ×

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

Each Dockerfile Command Creates a Layer



Docker Image Pull: Pulls Layers



```
Alexander@DESKTOP-90ATKET MINGW64 ~/Docker/Demo
$ docker pull nginx:latest
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
f3186e650f4e: Pull complete
9ac7d6621708: Pull complete
Digest: sha256:b81f317384d7388708a498555c28a7cce778a8f291d90021208b3eba3fe74887
Status: Downloaded newer image for nginx:latest
```

Docker Volumes

- Volumes mount a directory on the host into the container at a specific location

- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it

- Can be created in a Dockerfile or via CLI

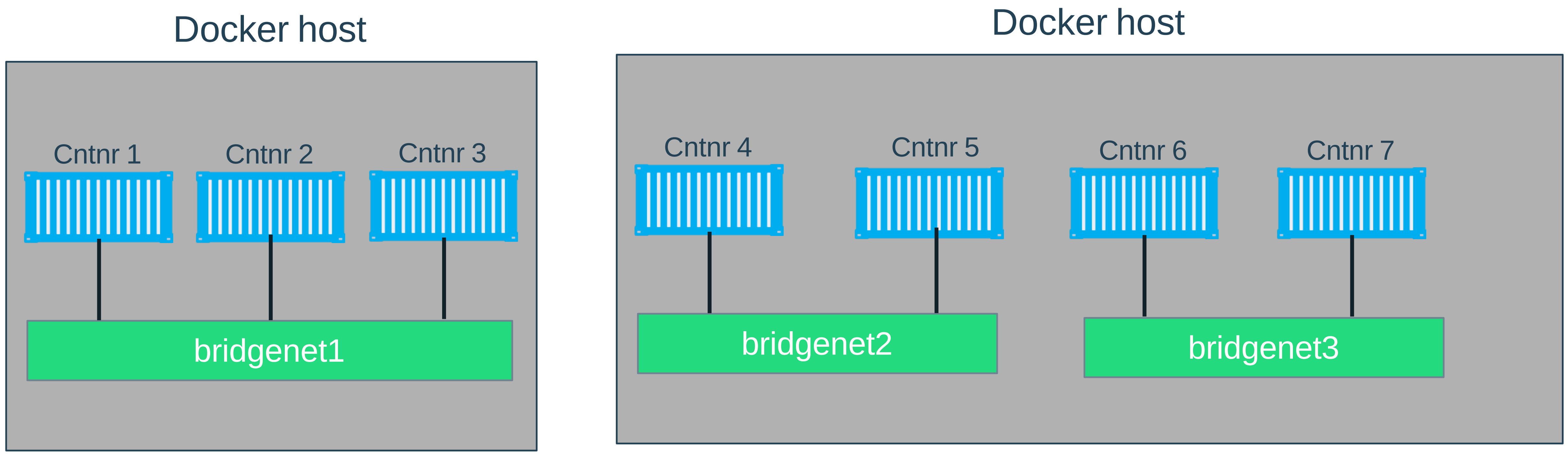
Why Use Volumes

- Mount local source code into a running container

```
docker container run -v $(pwd):/usr/src/app/
myapp
```

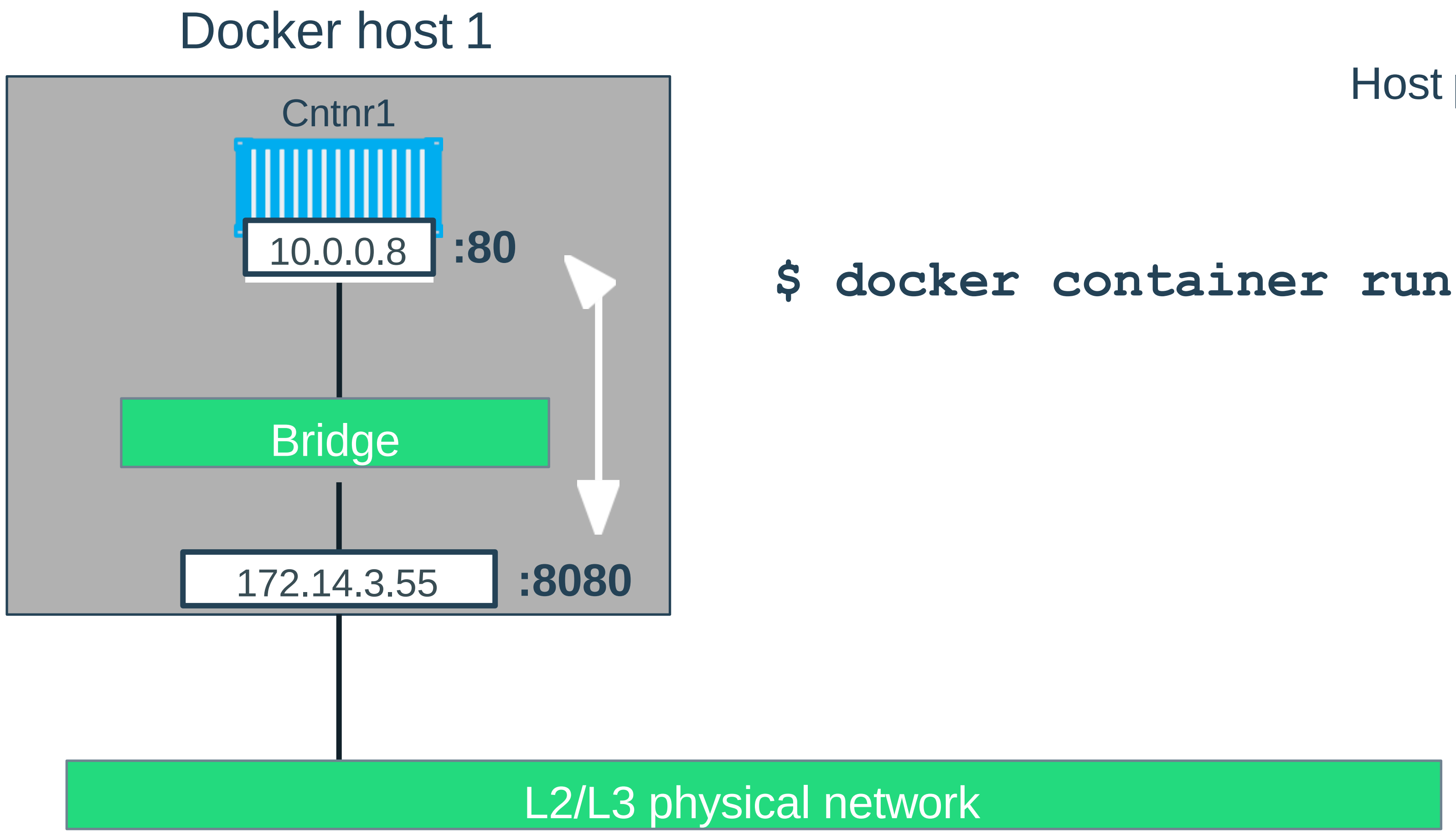
- Improve performance
  - As directory structures get complicated traversing the tree can slow system performance
- Data persistence

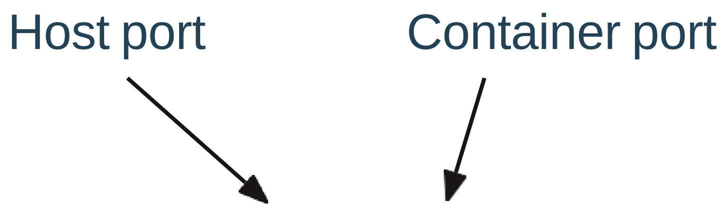
What is Docker Bridge Networking



```
docker network create -d bridge --name bridgenet1
```

Docker Bridge Networking and Port Mapping

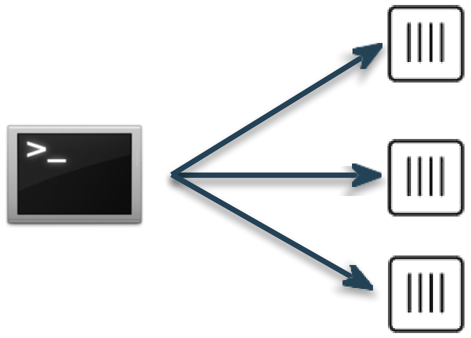




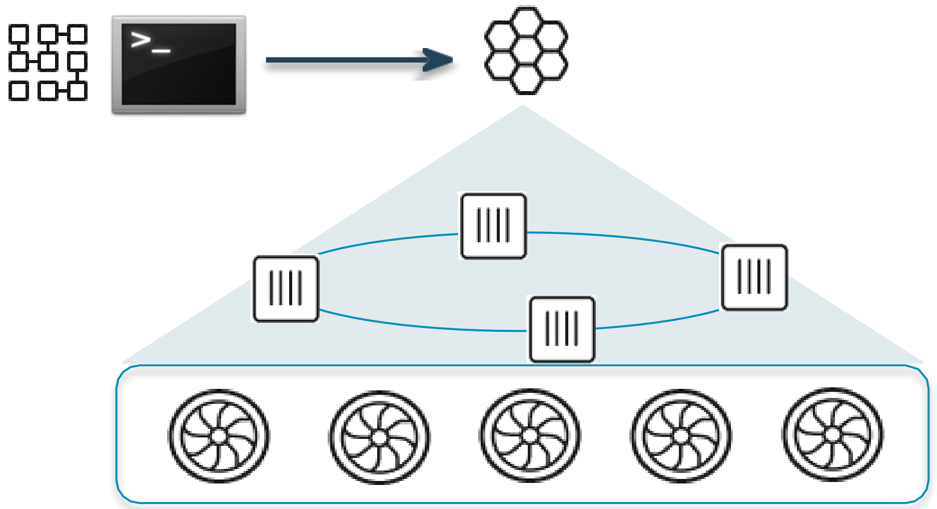
```
$ docker container run -p 8080:80 ...
```

Docker Compose: Multi Container Applications

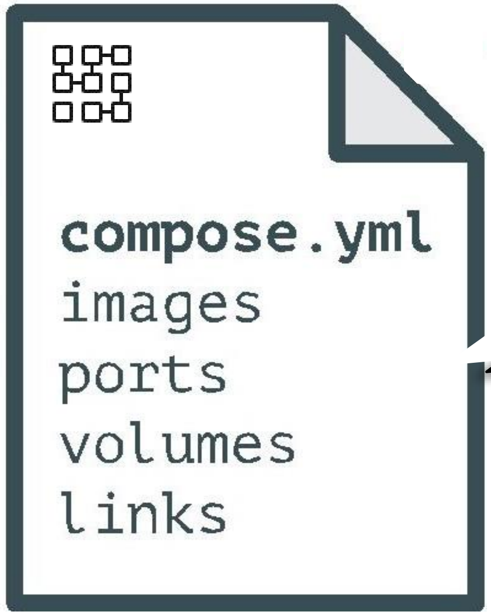
- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane

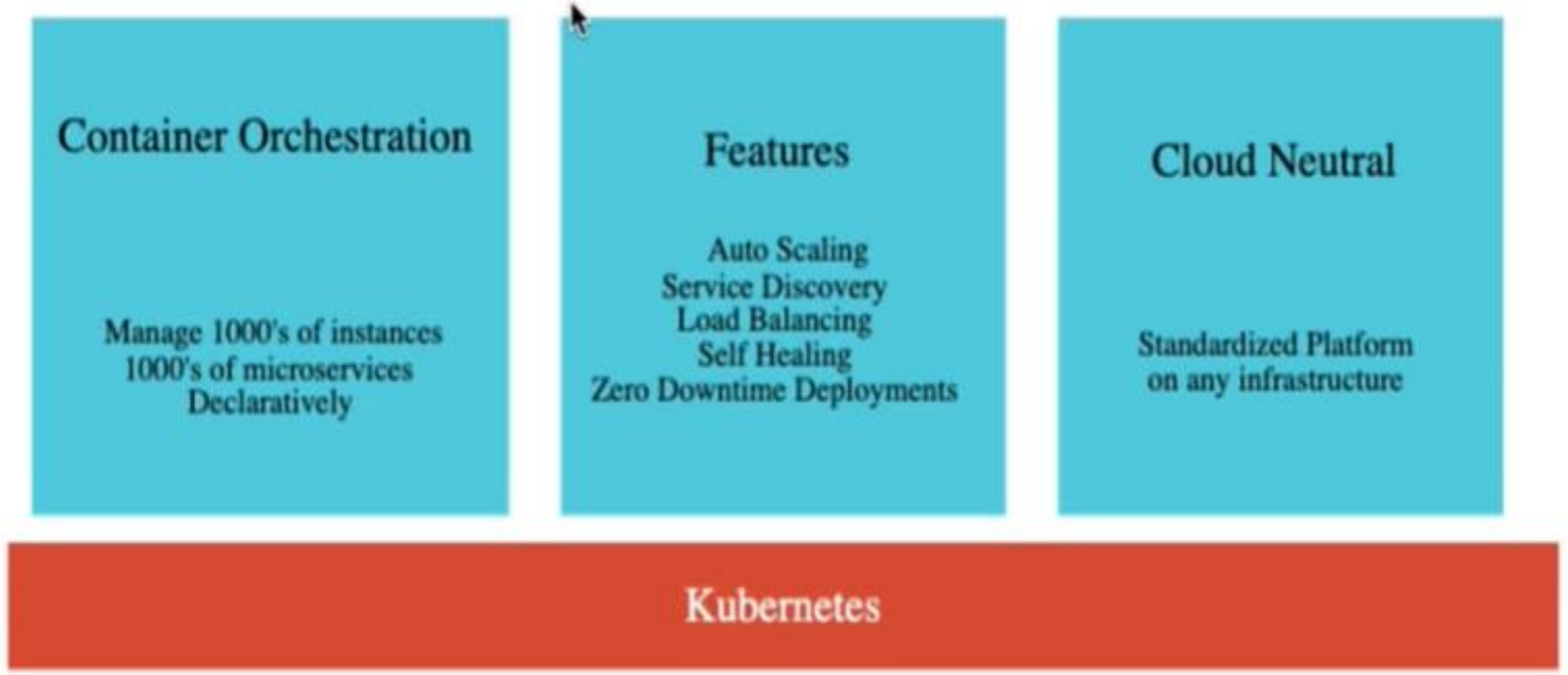


Docker Compose: Multi Container Applications



```
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
angular: # name of the first service
build: client # specify the directory of the Dockerfile
ports:
- "4200:4200" # specify port forwarding

express: #name of the second service
build: api # specify the directory of the Dockerfile
ports:
- "3977:3977" #specify ports forewarding

database: # name of the third service
image: mongo # specify image to build container from
ports:
- "27017:27017" # specify port forewarding
```

What is Kubernetes



Container Orchestration

Manage 1000's of instances
1000's of microservices
Declaratively

Features

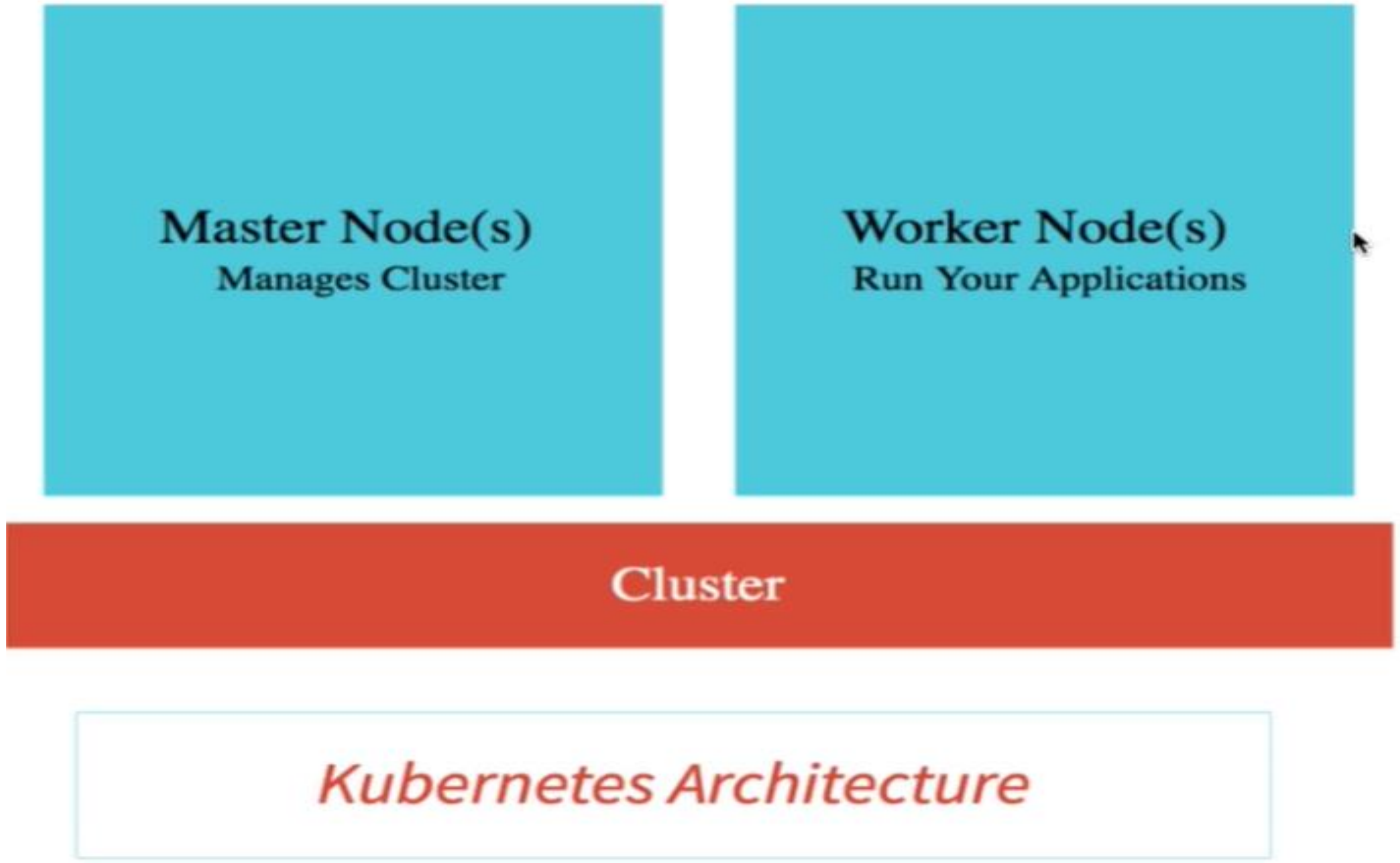
Auto Scaling
Service Discovery
Load Balancing
Self Healing
Zero Downtime Deployments

Cloud Neutral

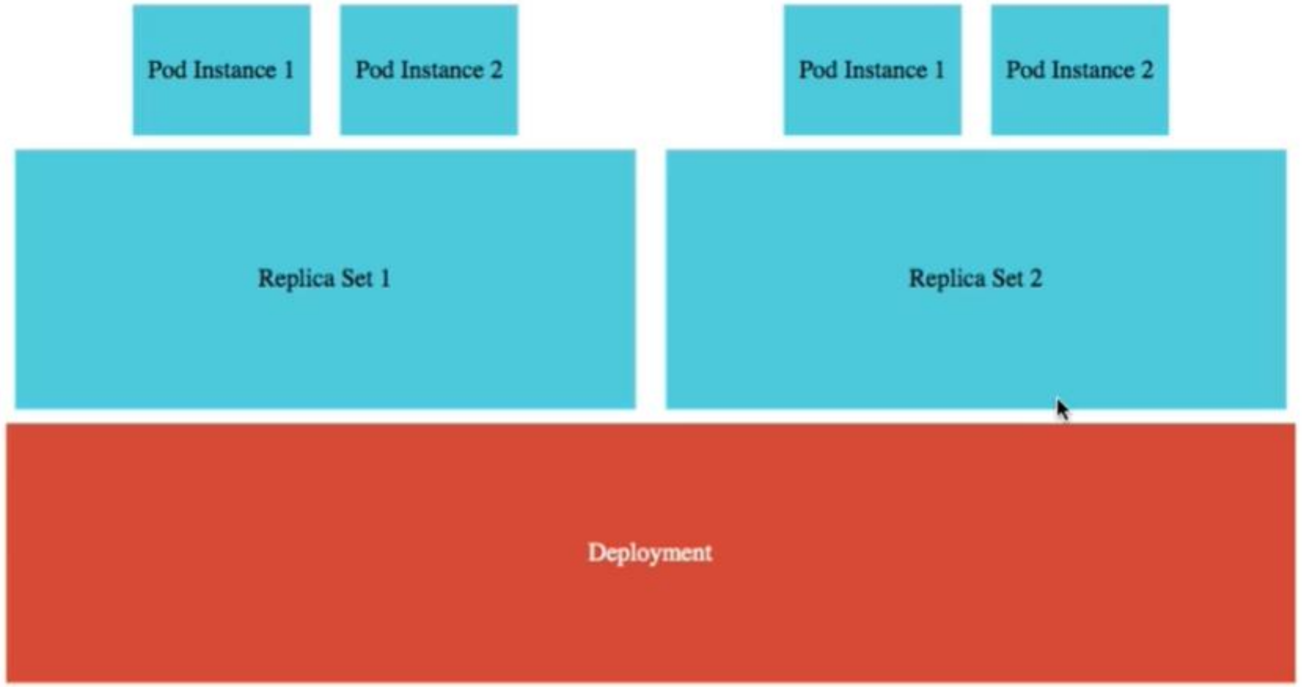
Standardized Platform
on any infrastructure

Kubernetes

Kubernetes Architecture



Kubernetes Architecture



Kubernete Deployments