Persistent

Overview of Docker

Training Goals

At the end of this whole Docker course, you will be able to understand:

- What is Docker – Docker fundamentals and Architecture, Docker installation.
- What are Docker images and registry?
- Deploying applications using Docker
- Deploying Java applications and multi-container application stack
- Docker APIs, Orchestration, production deployment

Pre-requisites

- Working knowledge of any Unix/Linux OS

- Some basic knowledge about Java application stack and databases is necessary but not mandatory.

Set-up

Windows

- On Windows platform
  - ssh client like putty
  - Edit Plus Editor/Notepad++

Linux

- On Linux platform
  - Linux Box Server
  - Putty connection on m/c

Objectives :

At the end of this module, you will be able to Learn :

- What is Docker?
- Why Docker is being used?
- How Docker is different from VMs, Vagrant, Chef and Puppet?
- What Docker Isn’t?
- Architecture of Docker
- Installation of Docker

What is Docker ?

What is Docker ?

- Docker is an open platform to build, ship and run any applications anywhere.
- Docker allows you to package an application with all it’s dependencies into a standard unit.
- Docker container wrap-up a piece of software in a complete file system that contains everything it needs to run – code, runtime, system tools and libraries.

Build + Ship + Run = docker

Docker tools

Docker Platform Consists of the following tools:

- Docker Engine – This is a lightweight runtime, that builds and runs Docker containers.
- Docker Hub – This is a registry where the Docker images are stored and also shared with other Docker users.
- Docker Machine – This is a tool which let's user install Docker Engine on Virtual hosts.
- Docker Swarm - Provides native clustering capabilities and turns a group of Docker engines into a single, virtual Docker Engine.
- Docker Compose - is a tool for defining and running multi-container Docker applications.

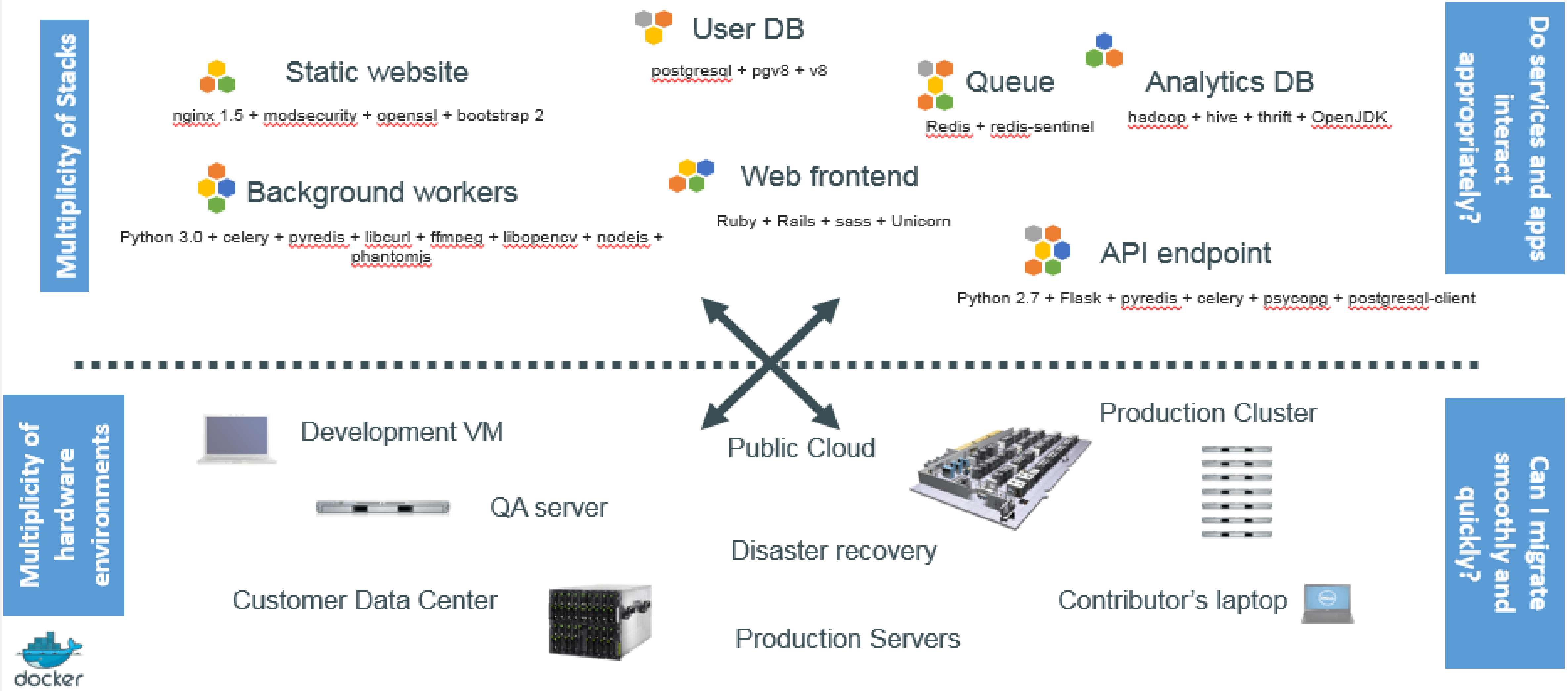
Docker characteristics

Lightweight	Open	Secure
• Containers running on a single machine, share the same kernel, and make efficient use of system resources.	• Based on Open standards, runs on major Linux distributions and Windows flavors.	• Containers isolate applications from each other.

Why Docker?

- Shipping code to server is becoming too hard.
- Software stacks are becoming more complex
  - Static Web sites
  - User Database
  - Analytics database
  - Queues, background workers
  - Web front ends
  - API end points

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 Servers

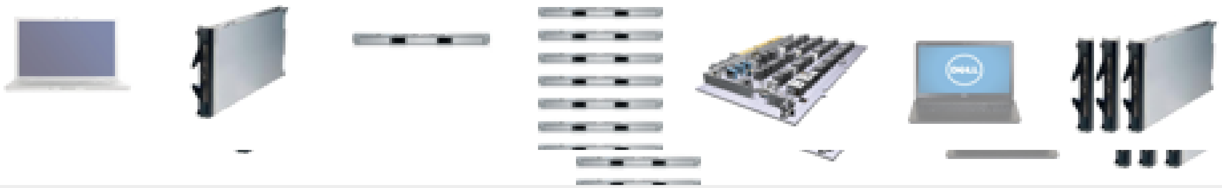
Production Cluster

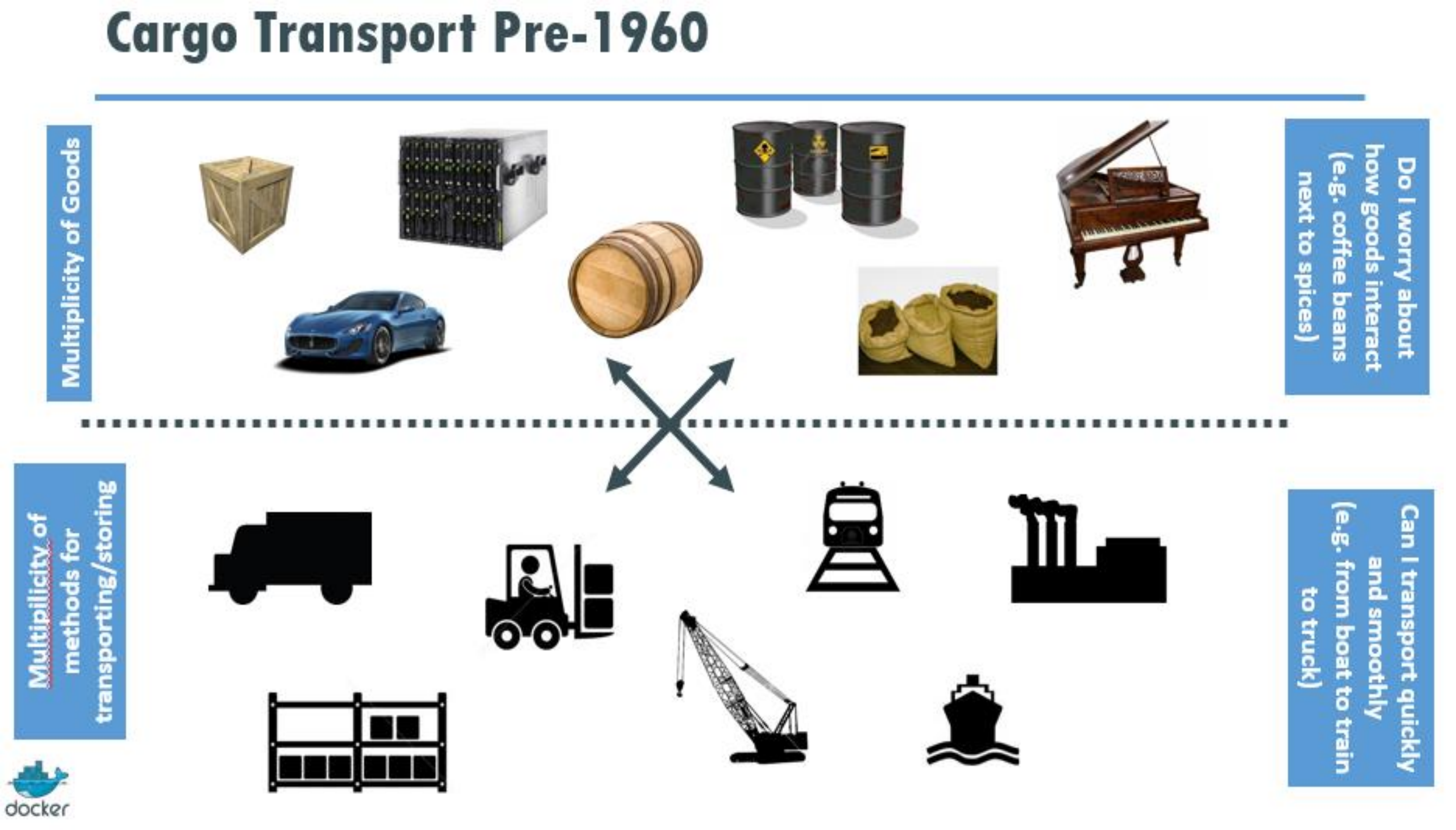
Contributor's laptop

Can I migrate smoothly and quickly?

Results in N X N compatibility nightmare

	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

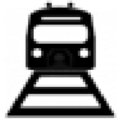
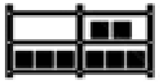
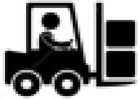
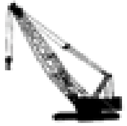
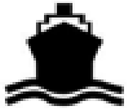
Multiplicity of Goods

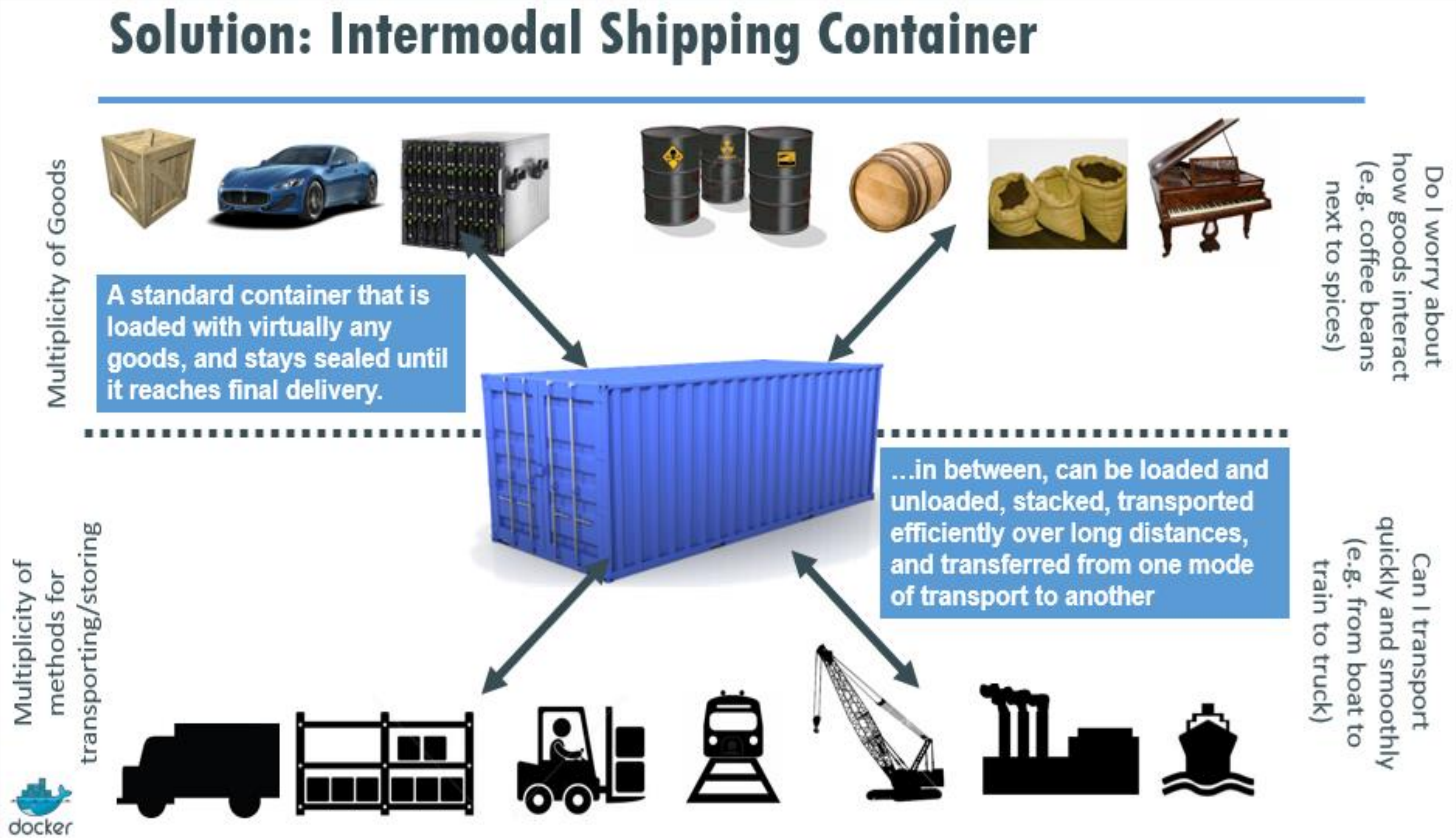
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing

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

Also an NxN Matrix

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



Solution: Intermodal Shipping Container

Multiplicity of Goods

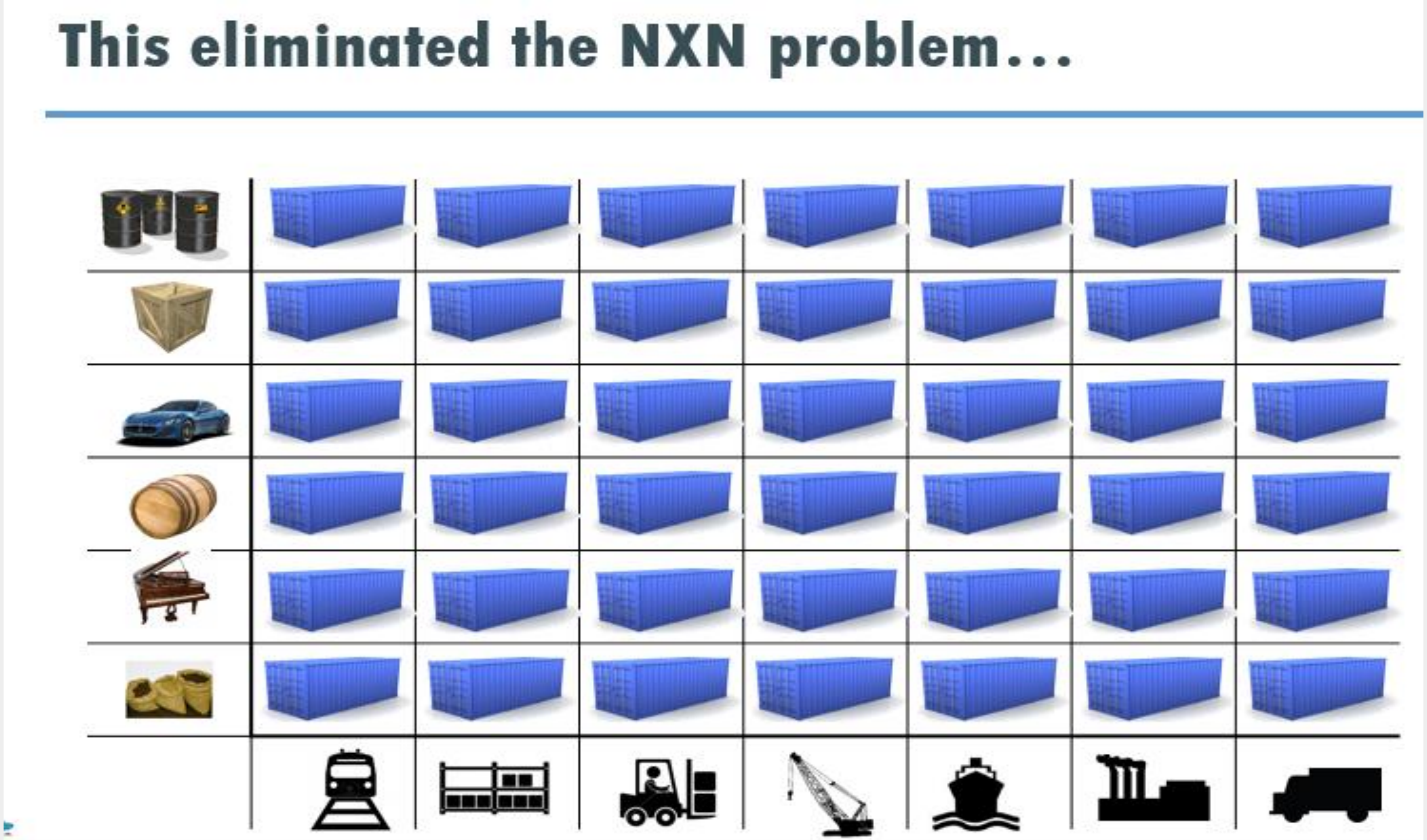
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

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

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

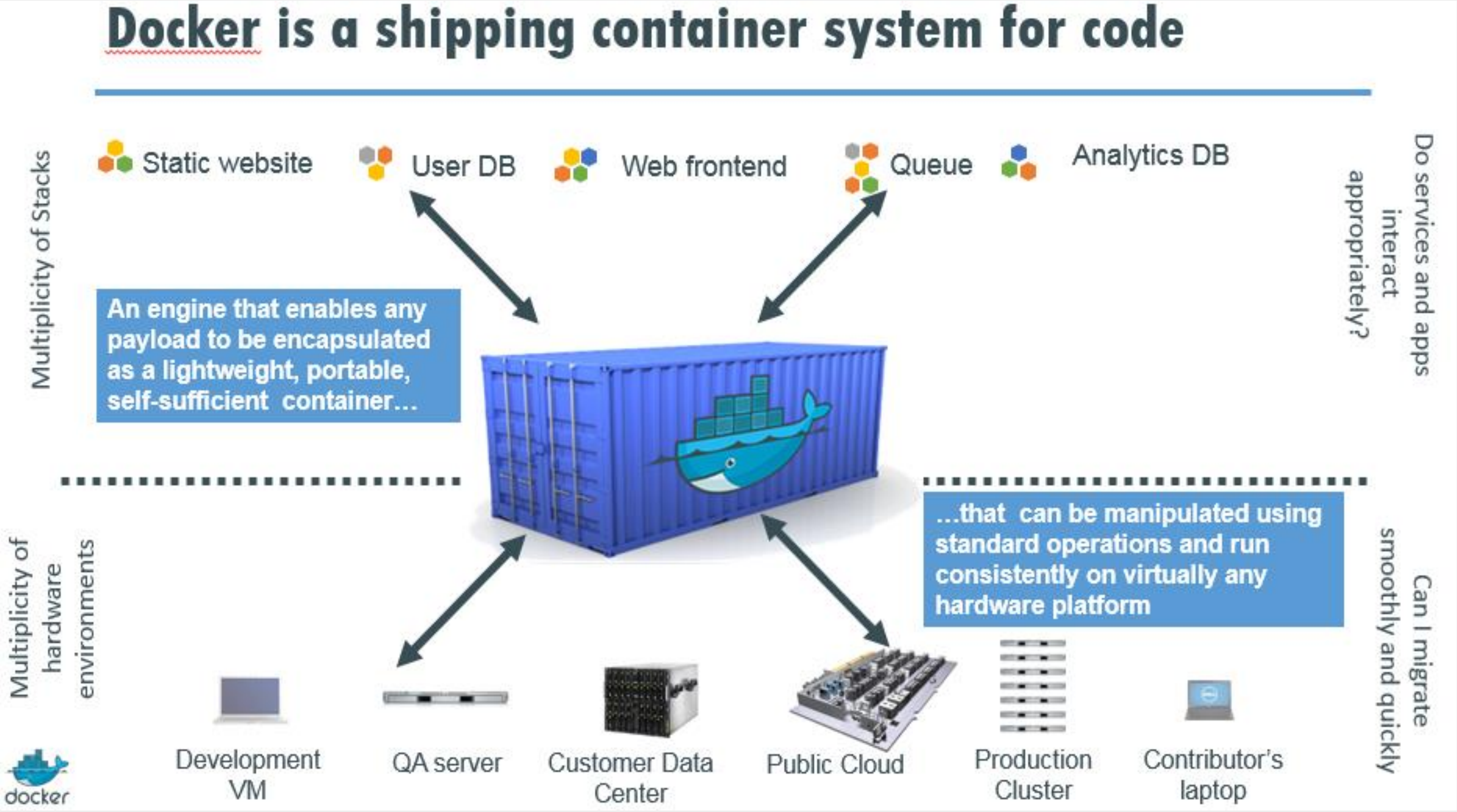
Multiplicity of methods for transporting/storing

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



This eliminated the NXN problem…

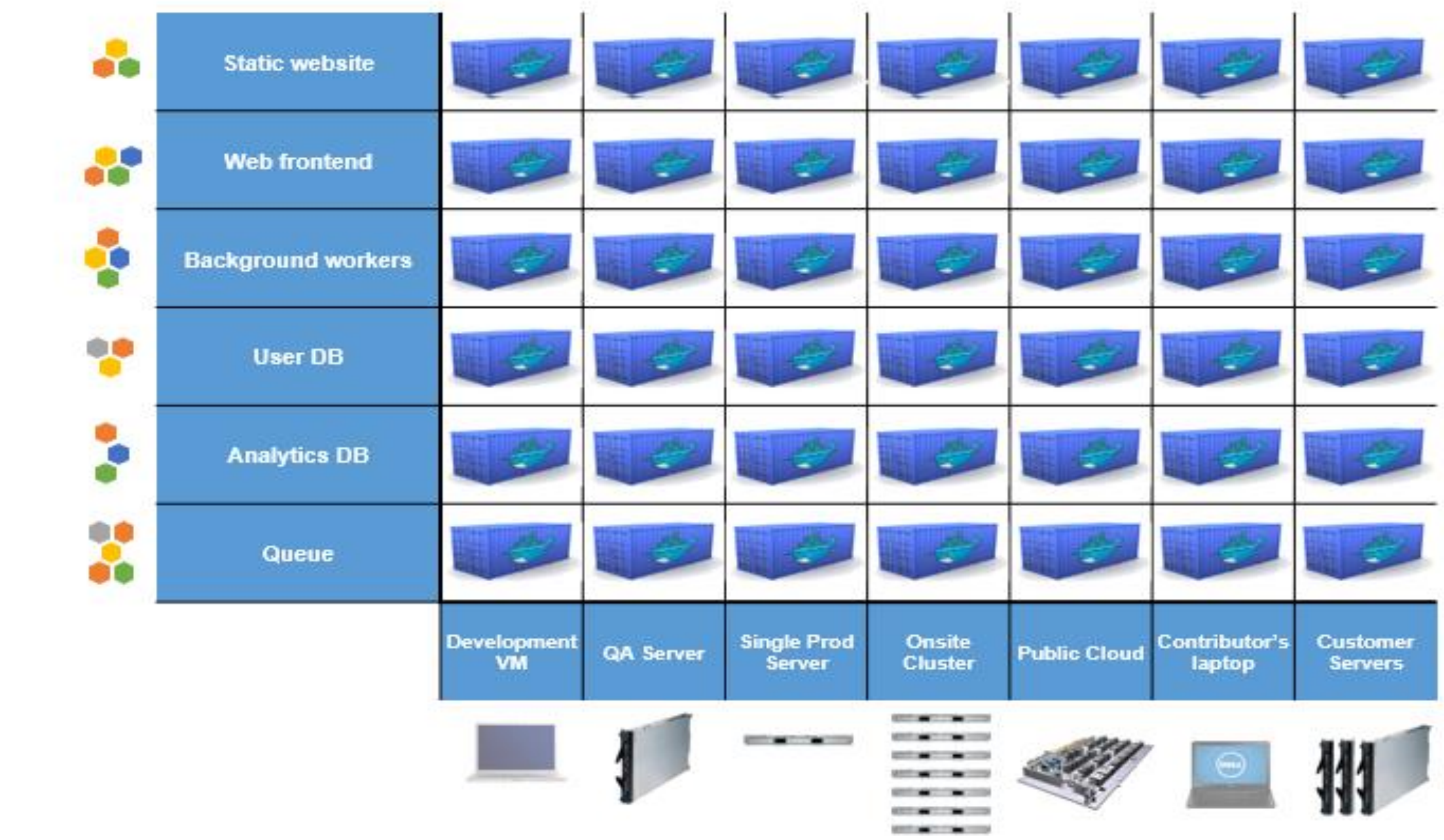
Docker is a shipping container system for code



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 solves the NXN problem



Static website

Web frontend

Background workers

User DB

Analytics DB

Queue

Development VM

QA Server

Single Prod Server

Onsite Cluster

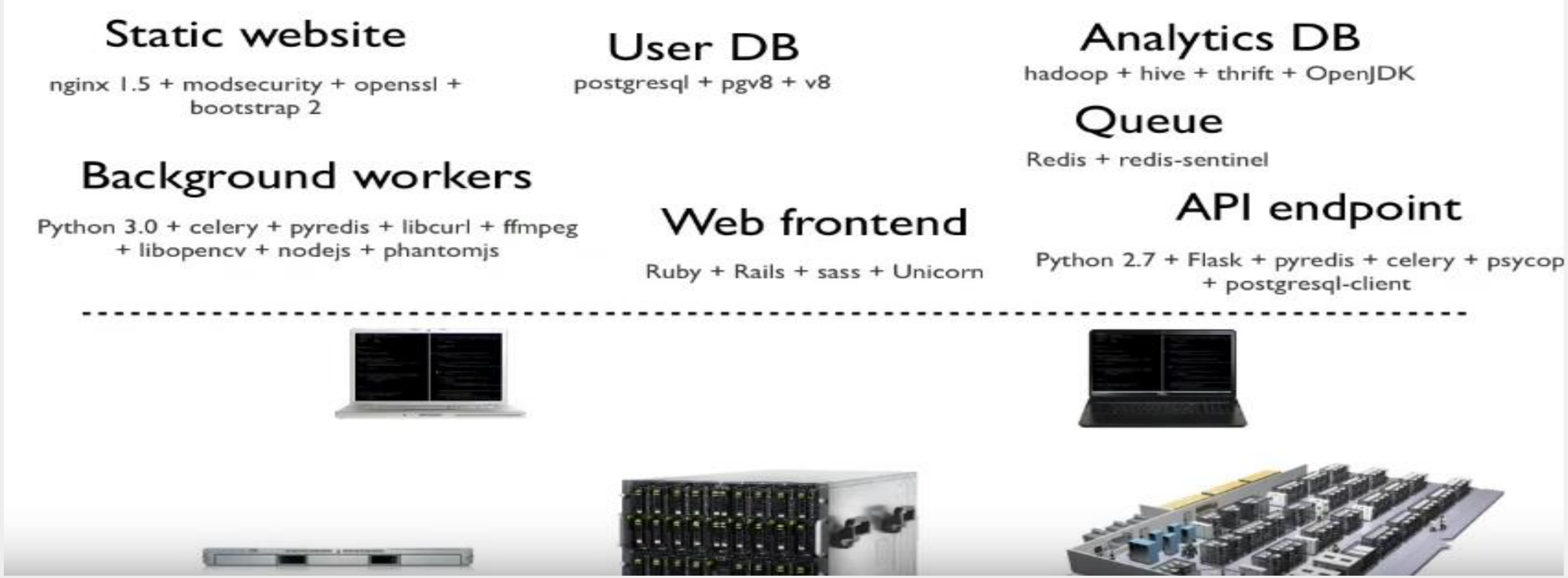
Public Cloud

Contributor's laptop

Customer Servers

Why Docker? …

- Application stacks need to run on multiple hardware platforms like Developer’s laptop, test machines in local data centers or cloud and Production environments on public cloud



Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Analytics DB

hadoop + hive + thrift + OpenJDK

Queue

Redis + redis-sentinel

Background workers

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

Web frontend

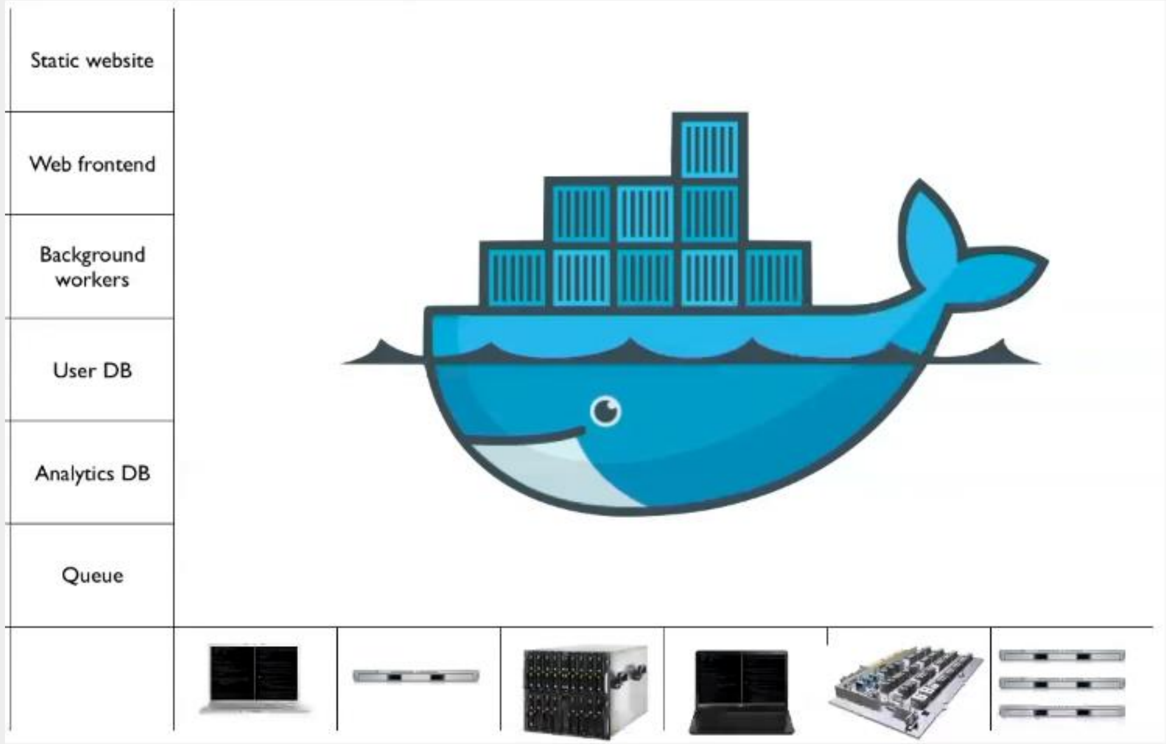
Ruby + Rails + sass + Unicorn

API endpoint

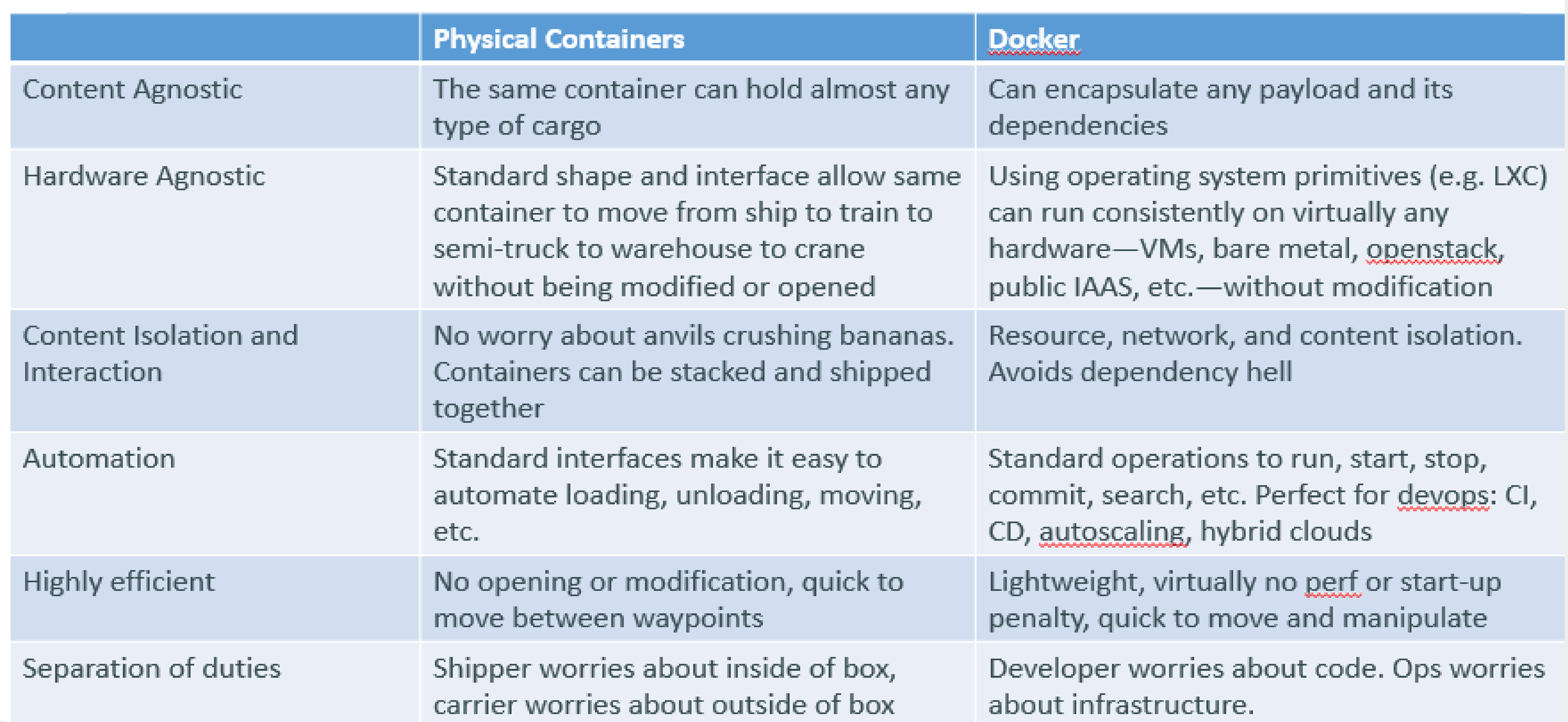
Python 2.7 + Flask + pyredis + celery + psycop + postgresql-client

Why Docker? …

- Analogy of Shipping industry
- Shipping of different types of goods from one part of world to another.
- Shipping industry solved this problem with standard container.



Why Container Matters?



	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.

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
  - 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 Care?

- 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 Docker? …

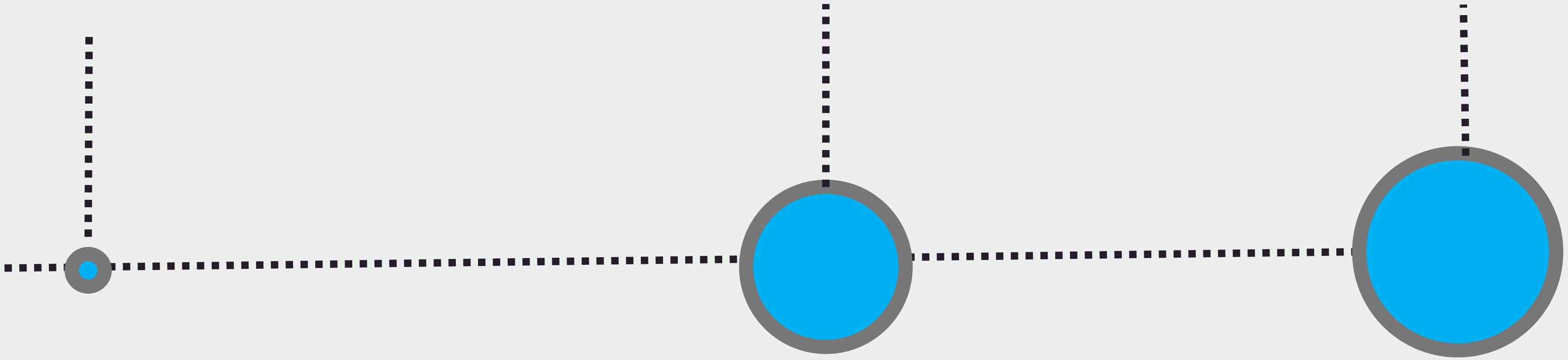
Standard Container Format

Ops

Developer

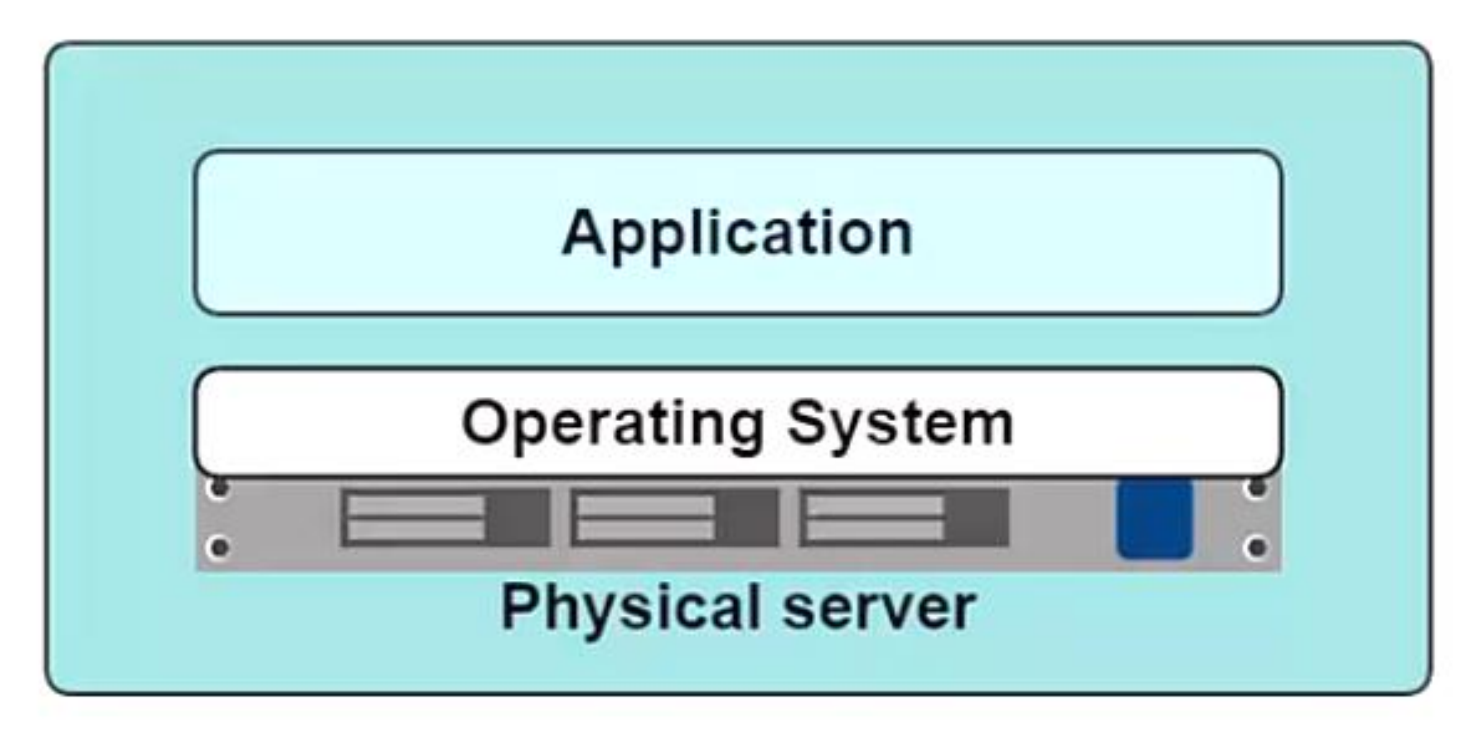
History of Docker

- Docker was introduced to the world by Solomon Hykes Founder of dotCloud in Python developers conference in March 2013.
- Within few weeks of this announcement, there was surprising amount of press and the project was quickly open sourced and made available on GitHub.
- Within few months the adoption Docker, has increased exponentially.



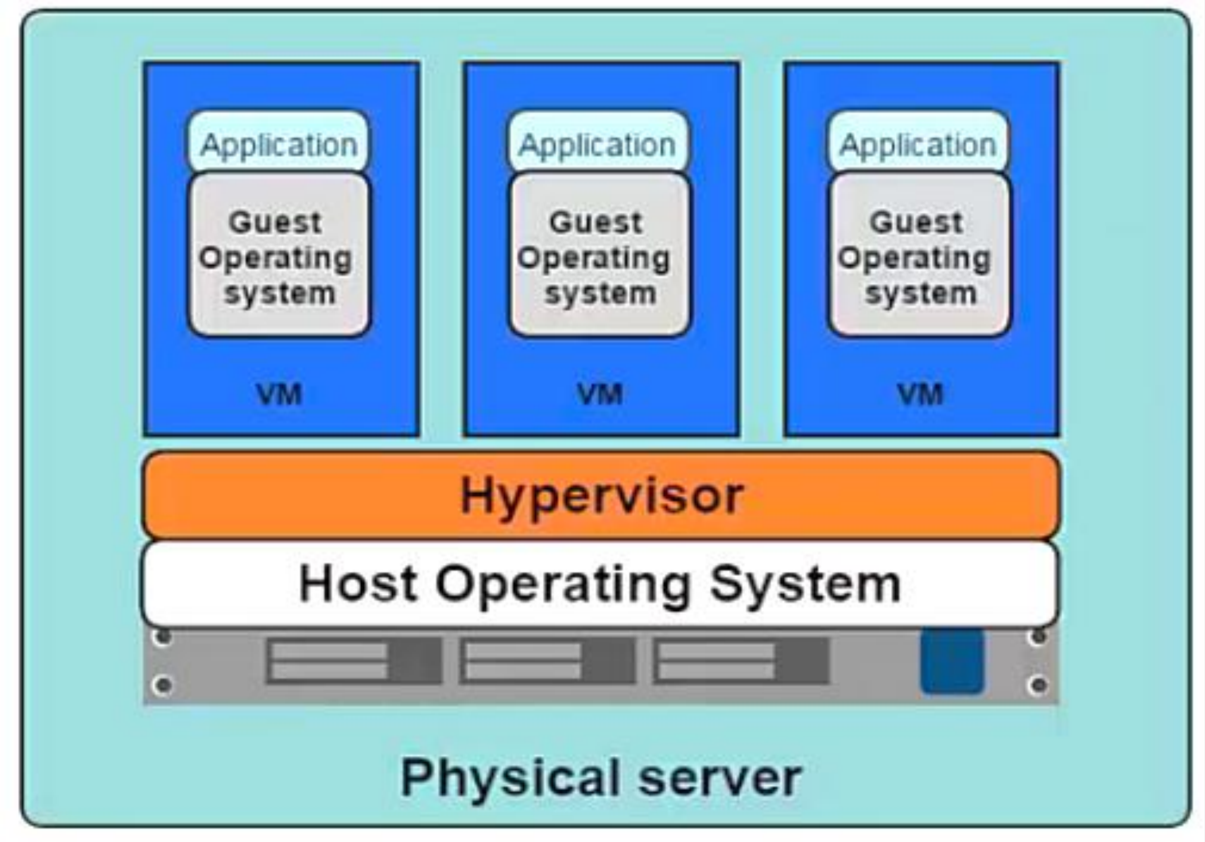
Docker is not a virtual machine.

- Historically one application was deployed on one physical server.
- The disadvantages of this approach were
  - slow deployment times,
  - huge costs,
  - wasted resources,
  - difficult to scale,
  - difficult to migrate
  - vendor lock-in.



Docker is not a virtual machine.

- To address the problems with physical server-based approach, Virtual machines were adopted.
- One physical server can contain multiple virtual machines.
- Each application runs in a virtual machine(VM)



Docker is not a virtual machine.

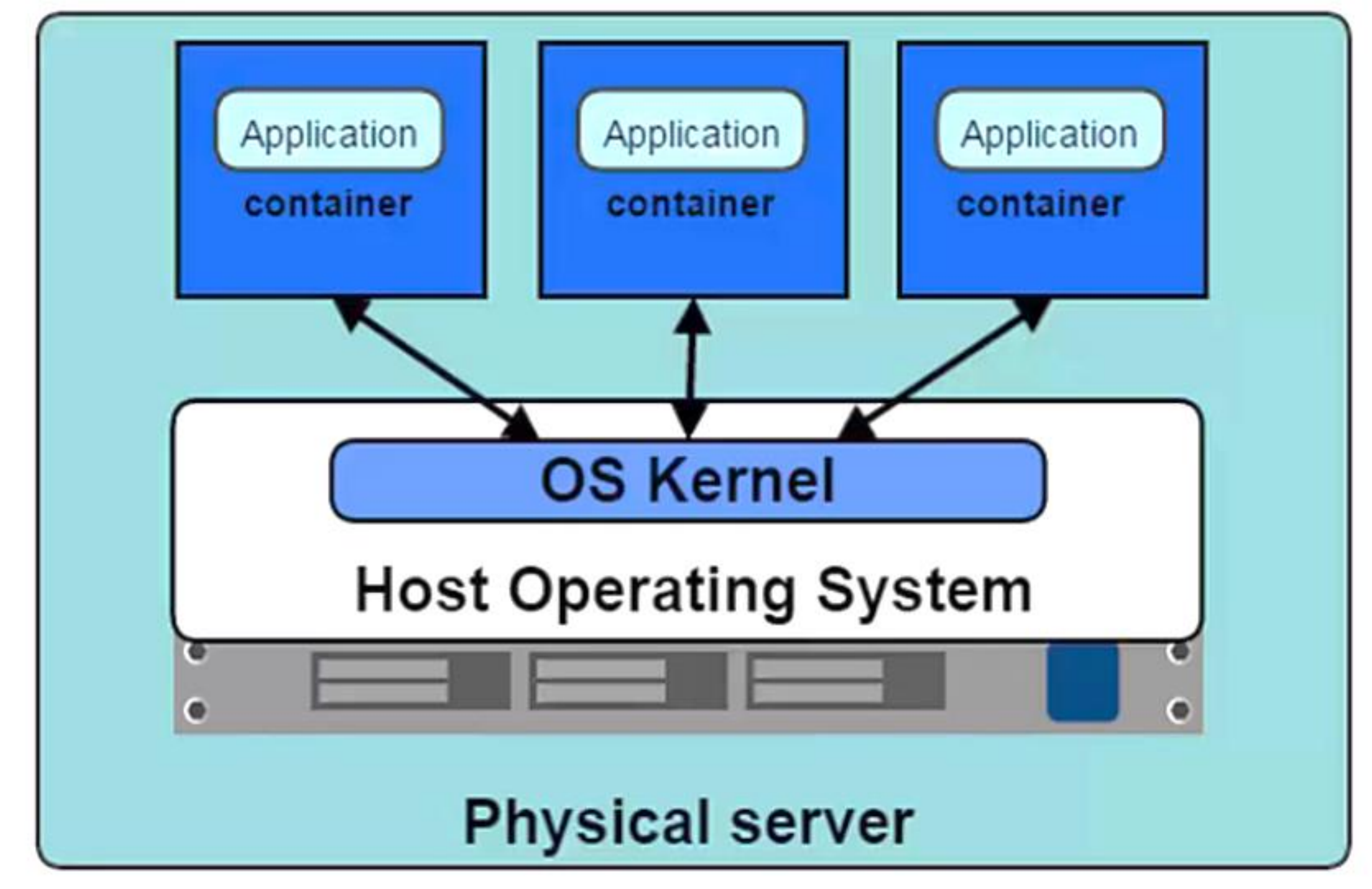
VMs

- Each VM still requires – CPU allocation, storage, RAM, an entire guest operating system
- More VMs you run, the more resources you need.
- Guest OS means wasted resources
- Application portability not guaranteed.

Containers

- Container based virtualization uses the kernel on the host’s OS to run multiple guest instances.
- Each guest instance is called as container
- Each container has its own – root file system, processes, memory, devices and network ports

Docker is not a virtual machine.



Docker is not a virtual machine.

- Containers like Docker are more lightweight as compared to VMs.
- There is no need to install guest OS on each container.
- Lesser space, CPU, RAM required for containers as compared to VMs.
- More containers per machine as compared to VMs.
- Greater portability

What Docker is not?

For some of the tool categories, Docker doesn’t directly replace them but can be used with conjunction to achieve great results.

- Virtualization platform like VMWare or KVM.
- Cloud platforms (Openstack, Cloudstack, etc.)
- Configuration management (puppet, chef) – Docker significantly improves ability to manage applications and their dependencies but does not directly replaces traditional configuration management.
- Deployment management environment (Vagrant) - Vagrant is a virtual machine management tool and often used to simulate server stacks. e.g. running a Linux stack on Windows box.

Reference Material : Websites & Blogs

- https://www.docker.com/
- https://training.docker.com/self-paced-training
- https://www.youtube.com/watch?v=Q5POuMHxW-0

Docker up and Running by Karl Matthias and Sean kane

Key Contacts

Docker Interactive

Dattatray Kulkarni

dattatray_kulkarni@persistent.co.in

Asif Immanad

asif_immanad@persistent.co.in

Vice President

Shubhangi Kelkar

shubhangi_kelkar@persistent.co.in

Thank you!

Persistent University