Readings

- A quick introduction to Docker
  http://blog.scottlowe.org/2014/03/11/a-quick-introduction-to-docker/

- Docker Get Started, Part 1: Orientation and setup
  https://docs.docker.com/get-started/

- Microservices
  https://martinfowler.com/articles/microservices.html

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 + OpenJ

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 Cente

Public Cloud

Production Cluster

Disaster recovery

Contributor's laptop

Production Servers

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



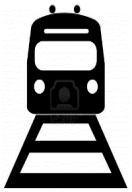


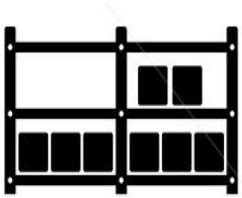


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing







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

NxN Matrix?

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

Solution: Shipping Container

Multiplicity of Goods





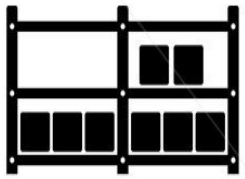


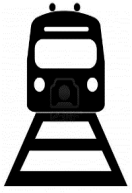
Do I worry about how goods interact (e.g. coffee beans next to spices)

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

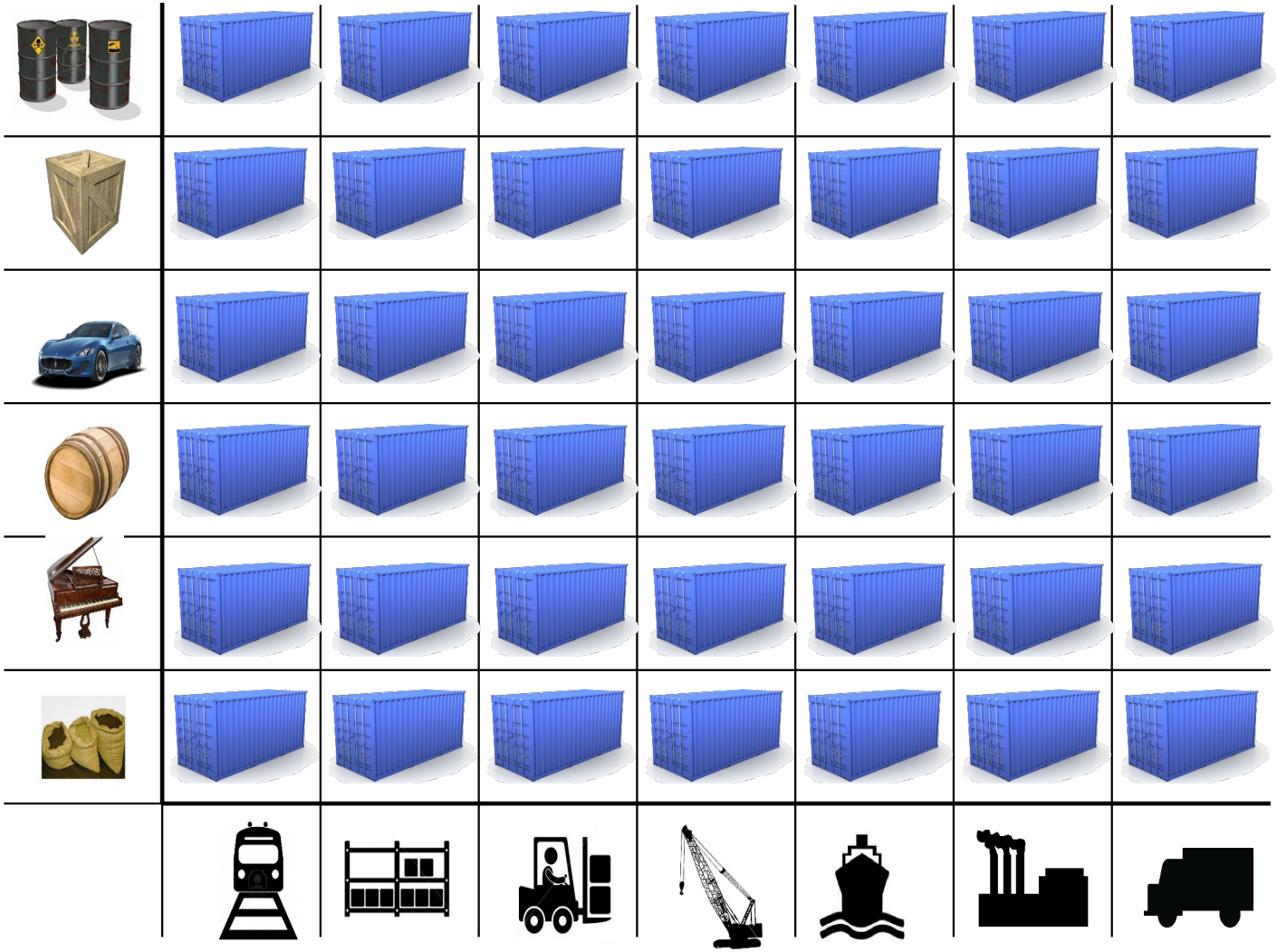
Multiplicity of methods for transporting/storing




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

This eliminated the NXN problem…

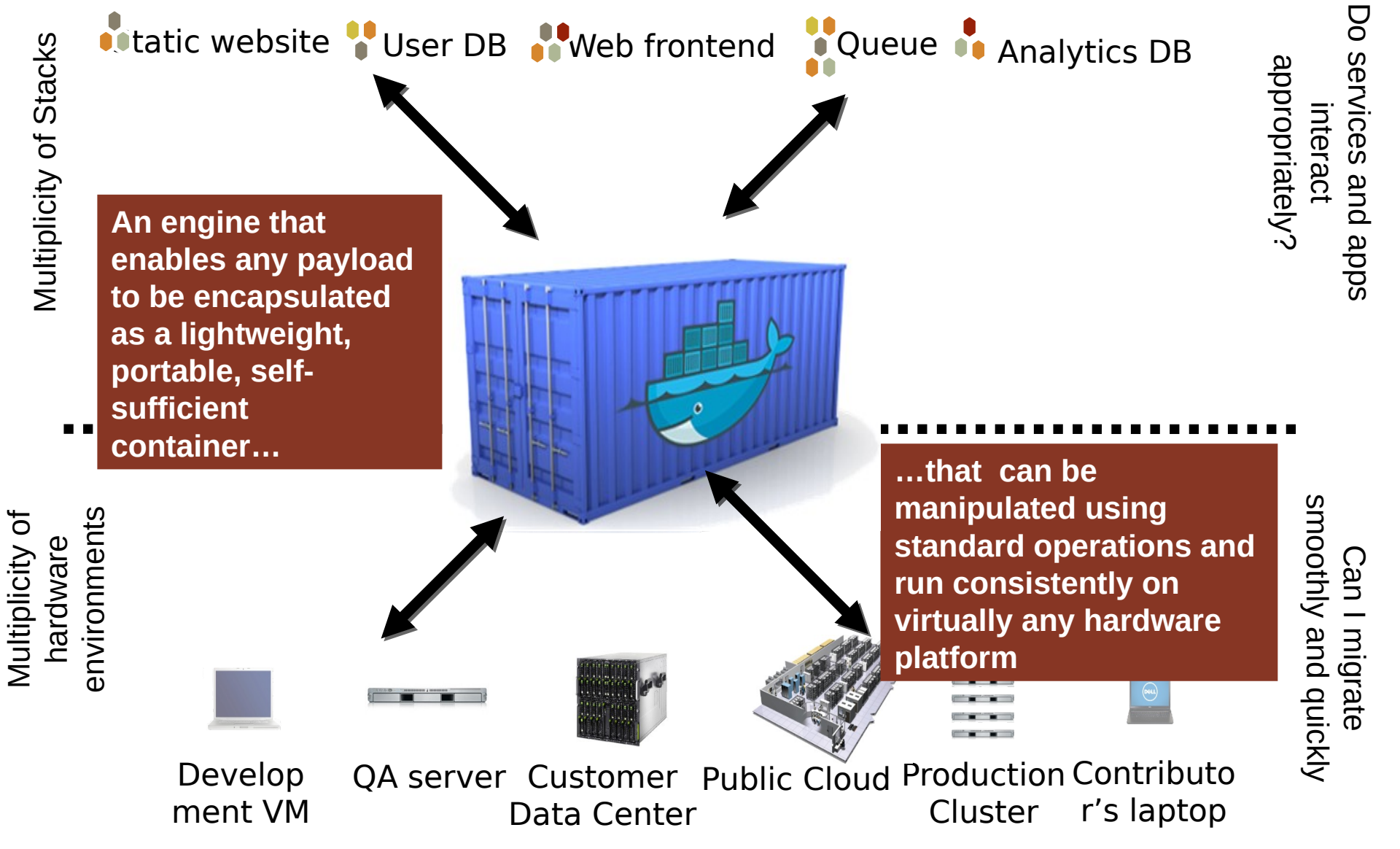


- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

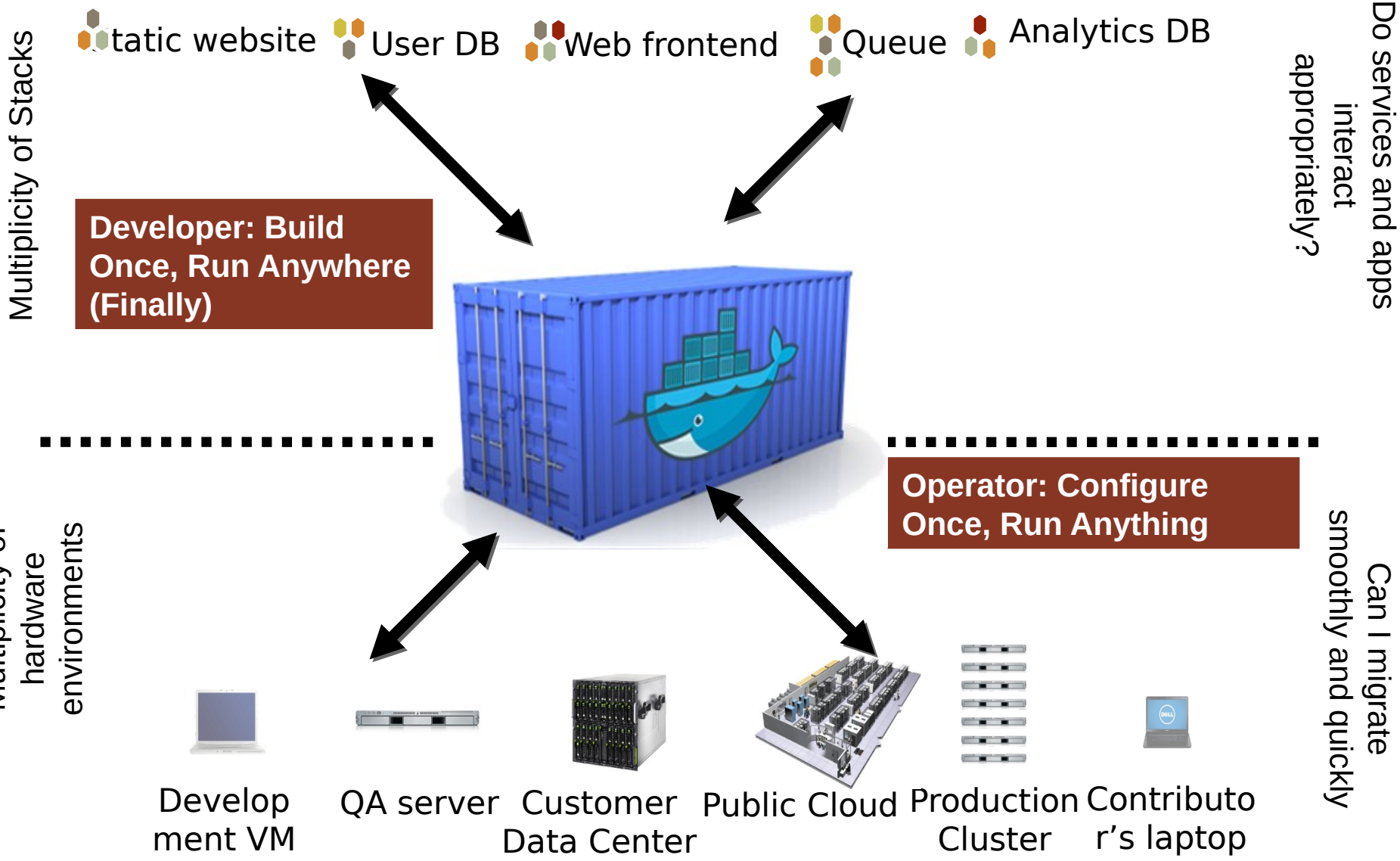
→ massive globalizations

- 5000 ships deliver 200M containers per year

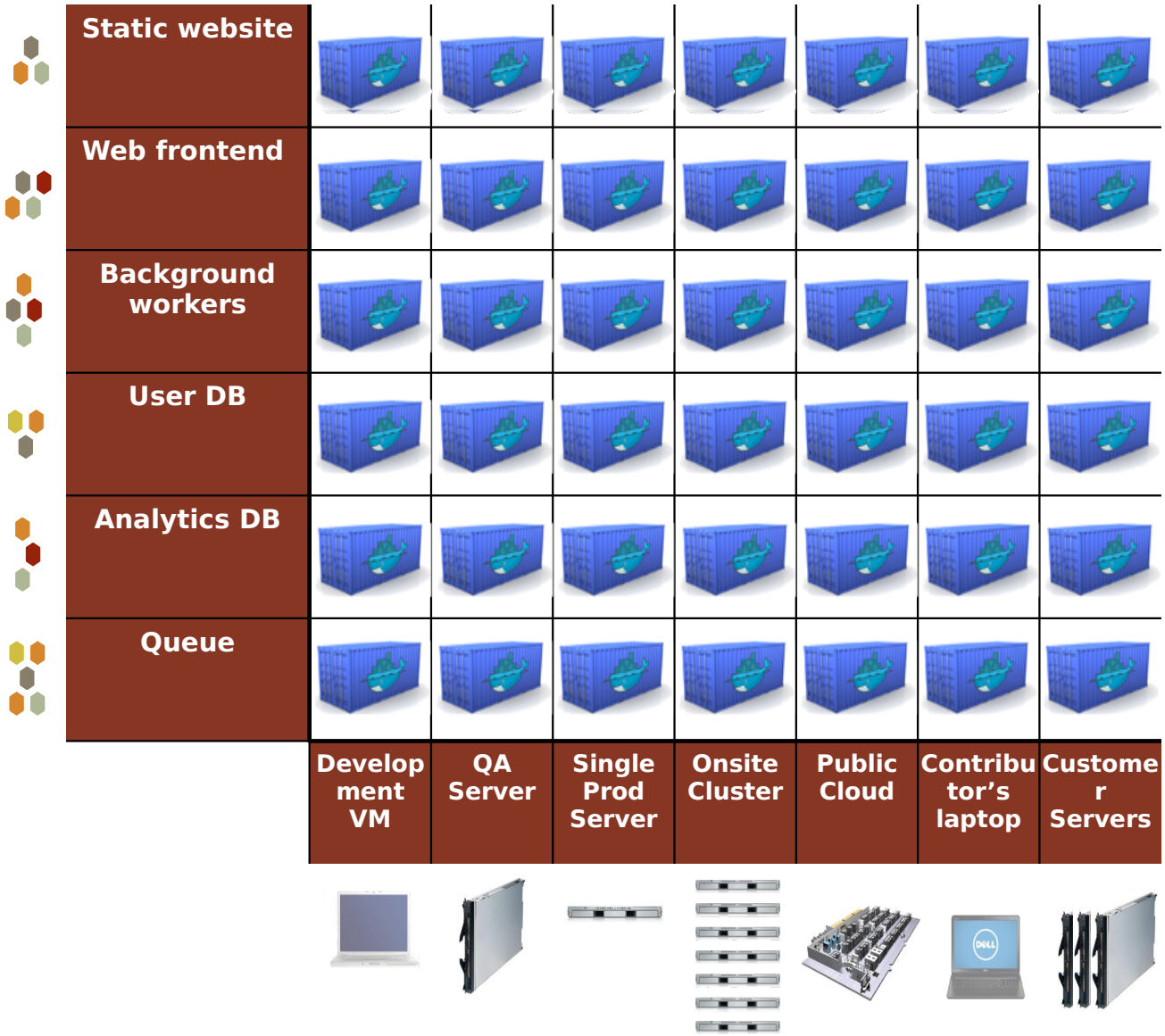
Docker is a shipping container system for code



Or…put more simply



Docker solves the NXN problem



Container

- Container is a type of virtualization that allows multiple isolated applications to run on a single host operating system.

- Containers are lightweight and portable, and they provide a way to package an application and its dependencies together, so it can run consistently across different environments.

- Containers are different from virtual machines (VMs) as they do not require a separate operating system to be installed for each container. This avoids the overhead of running multiple separate operating system.

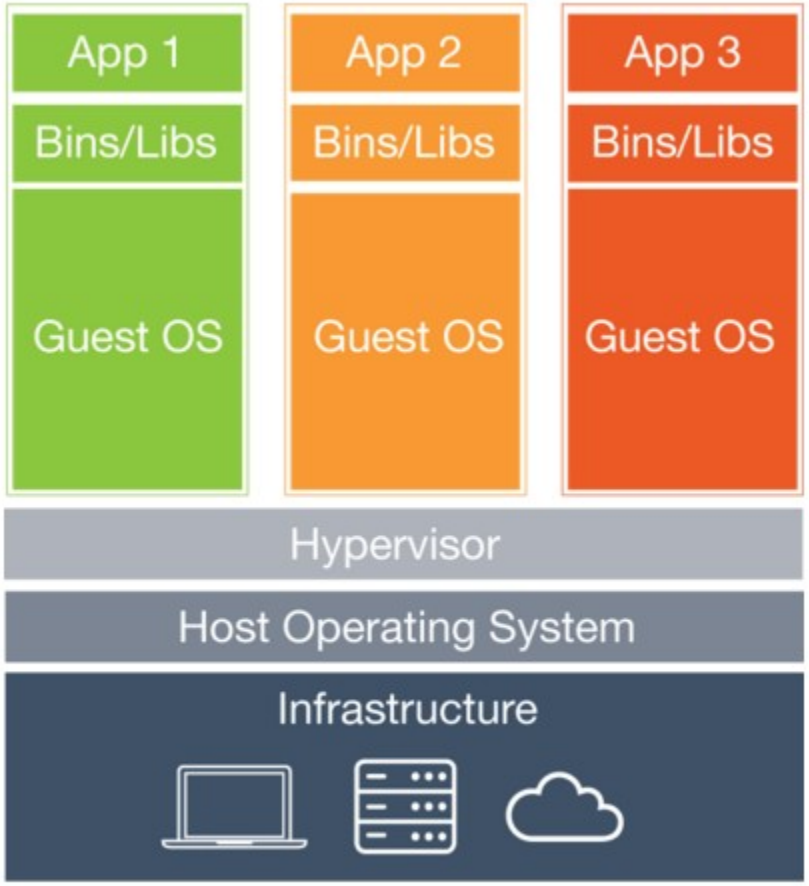
Container (cont)

- The core component of container virtualization is the container engine, which is a software that manages the creation and execution of containers.
- The container engine uses the host operating system's kernel to create isolated environments for each container.
- These isolated environments, called containers which share the host operating system's kernel and libraries, but have their own file system, network stack, and process space.
- Containers are created from container images, which are pre-configured and bundled with all the necessary dependencies for the application to run.
- Container images can be stored in a container registry, and can be easily pulled and run on any host that supports the container engine.

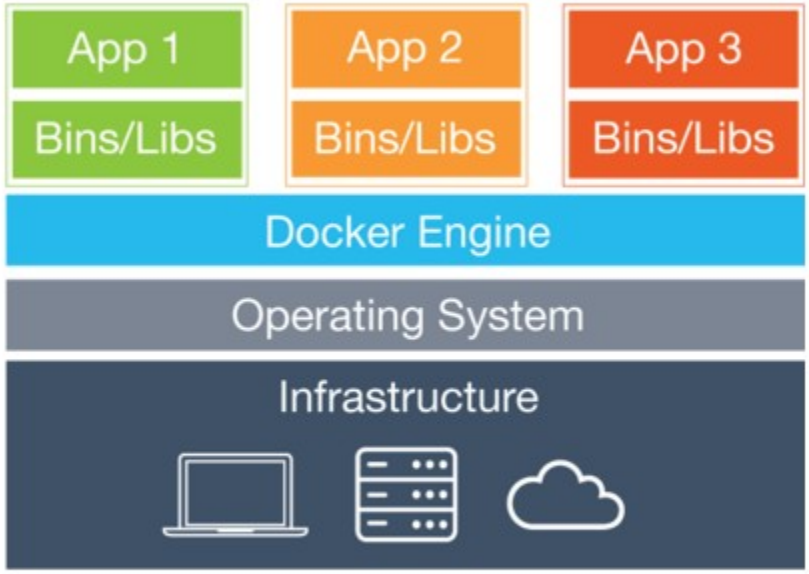
LXC

- In Linux, the most common container technology is LXC (Linux Containers).
- LXC uses Linux kernel features such as control groups and namespaces to create isolated environments for containers.
- Control groups, also known as cgroups, are used to limit, prioritize and account system resources such as CPU, memory, and I/O.
- Namespaces, on the other hand, are used to provide isolated environments for container's process, network and file system.

Virtual Machines vs Containers



Virtual Machines



Containers

Virtual Machines vs Containers (Cont.)

- Containers share resources with the host OS, which makes them an order of magnitude more efficient
- Applications running in containers incur little to no overhead compared to applications running natively on the host OS.
- The fundamental goals of VMs and containers are different
  - the purpose of a VM is to fully emulate a foreign environment
  - the purpose of a container is to make applications portable and self-contained

Containers over VMs

1. Cloud-native applications: Containerized applications are designed to run in distributed environments and can be easily deployed to different cloud platforms, making them well suited for cloud-native applications.
2. Microservices: Containers are well suited for microservices architecture, as they provide isolation and are lightweight, making it easy to deploy and scale individual services.
3. Resource efficiency: Containers are more lightweight and efficient than VMs, as they share the host operating system's kernel and libraries, which reduces the resources required to run the application.
4. Rapid Development and Testing: Containers allow developers to quickly create and test new applications, as they can be easily created and destroyed, which can help to speed up the development process.

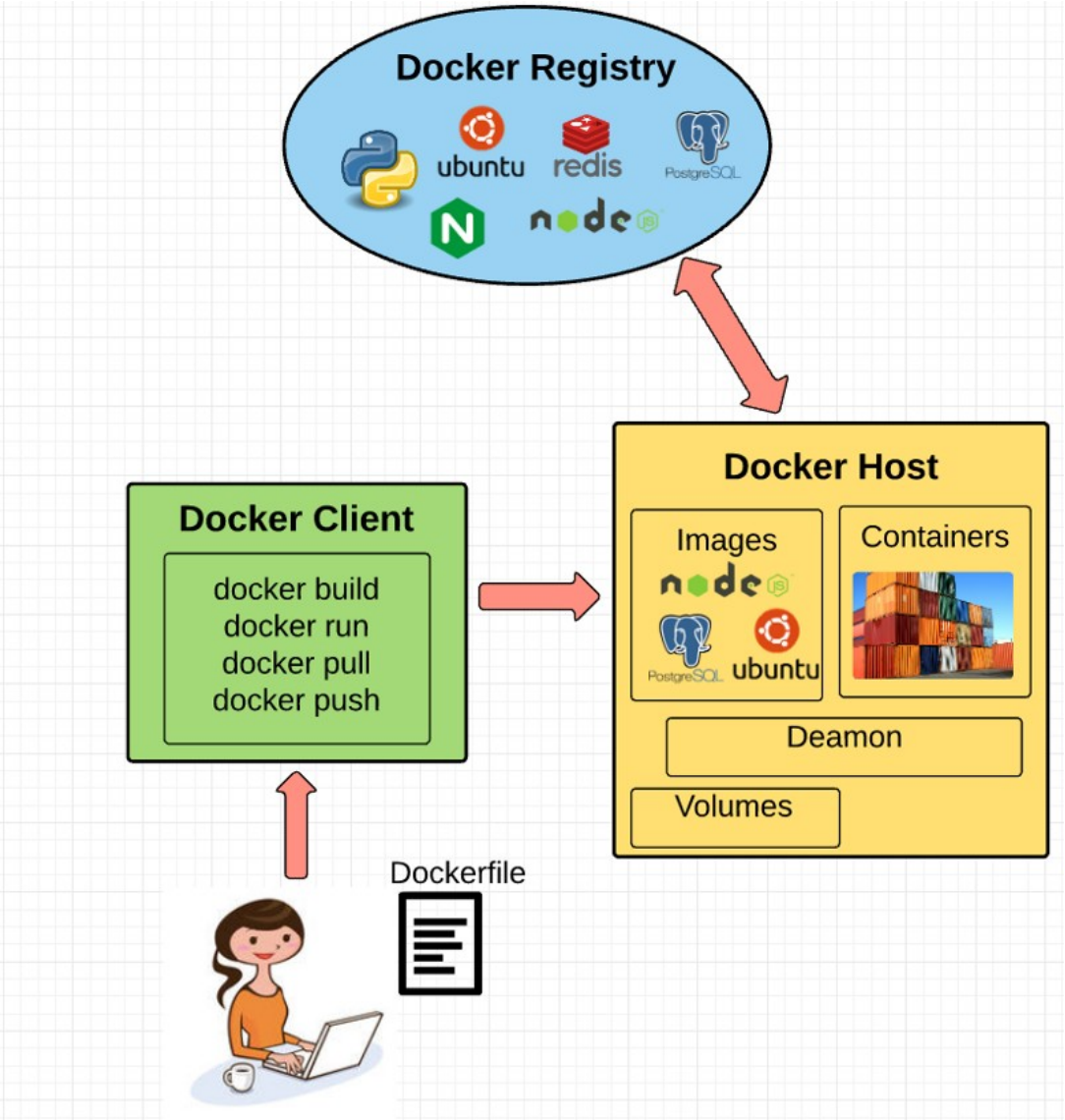
VMs over Containers

1. Legacy Applications: If an application is not designed to run in a containerized environment, it may be more efficient to run it in a virtual machine, as it will not require any modification.
2. High Security Requirements: In some cases, such as for sensitive data or applications with strict regulatory compliance requirements, virtual machines may be a better option as they provide a higher level of isolation and security than containers.
3. Hardware-intensive Workloads: Applications that require direct access to physical hardware resources such as GPUs, high-speed networking, or specialized devices may be better suited for virtual machines as they can provide direct access to these resources.
4. Different Operating Systems: If your application requires different operating systems for different components, it may be more efficient to use VMs, as each VM can run a different operating system.
5. High Performance Computing: Applications that need high-performance computing resources like large memory, high-speed storage, or complex network configurations are better suited for virtual machines.

Docker

- Docker is an open-source platform that is widely used container engine on Linux.
- Docker provides a simple and efficient way to create, deploy, and run containerized applications.
- It also provides an easy-to-use command-line interface and an API that allows you to automate the management of your containers.

Docket Engine Overview



Container Orchestration

- Container orchestration is the process of automating the deployment, scaling, and management of containerized applications.
- It involves the use of software tools that can manage and coordinate the scheduling, scaling, and deployment of containers across a cluster of machines.
- With container orchestration, administrators can define the desired state of their applications, such as the number of replicas of a container, and the orchestration tool will automatically ensure that the desired state is met by creating, updating, or deleting containers as needed.
- Some of the key features of container orchestration include:
  - Automatic scaling: The ability to automatically scale the number of containers in response to changes in demand.
  - Self-healing: The ability to automatically recover from failures, such as if a container crashes, by restarting it or rescheduling it on another node.
  - Load balancing: The ability to automatically distribute incoming traffic across multiple containers.
  - Service discovery: The ability to automatically discover and connect to other services in the system.

Docker Swarm

- Docker Swarm is a native clustering and orchestration solution for Docker.
- It allows you to create and manage a cluster of Docker nodes as a single virtual system.
- It provides features such as service discovery, load balancing, and scaling, which makes it easy to build, ship and run distributed applications.
- It also allows you to schedule the placement of containers across the nodes in the cluster, and to ensure that the desired number of replicas of a container are running at all times.

Kubernetes

- Kubernetes (often shortened to "K8s") is an open-source container orchestration system that automates the deployment, scaling, and management of containerized applications.
- It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF).
- Kubernetes manage containerized applications in a clustered environment. It abstracts away the underlying infrastructure and provides a consistent set of APIs for deploying, scaling, and managing containerized applications.
- Kubernetes uses a declarative configuration model, where the desired state of the system is defined in configuration files.
- The declarative configuration model is a way of describing the desired state of a system, rather than the steps needed to achieve that state.
- This approach allows administrators to specify the desired state of their applications and infrastructure, and then rely on Kubernetes to automatically make the necessary changes to achieve that state.

Kubernetes (Cont.)

- Kubernetes provides several key features:
  - Automatic scaling of application replicas based on resource usage
  - Self-healing capabilities to automatically replace and reschedule failed containers
  - Automated rollouts and rollbacks of application updates
  - Service discovery and load balancing for application components
  - Automated storage provisioning and management
  - Automatic binpacking of containers onto nodes in the cluster

Docker Swarm Vs Kubernetes

•Kubernetes and Docker Swarm are both popular container orchestration systems. However, they have some key differences:

1. Kubernetes is an open-source platform that was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Docker Swarm, on the other hand, is a native clustering and orchestration solution for Docker.
2. Kubernetes provides a more powerful and flexible orchestration engine, with more advanced features such as automatic scaling, self-healing, and rollouts, and rollbacks. Docker Swarm is more limited in terms of its orchestration capabilities, but it is simpler to set up and use.

3. Kubernetes provides a declarative configuration model where the desired state of the system is defined in configuration files, while Docker Swarm uses a more imperative model where the desired state is defined through commands.

4. Kubernetes supports a wide range of deployment options including on-premise, in the public cloud, and hybrid deployments. Docker Swarm is typically used for deployments on a single cluster or a single cloud provider.

5. Kubernetes has a large and active community, with many contributors and a wide range of third-party tools and plugins available. Docker Swarm has a smaller community and fewer third-party tools and plugins.

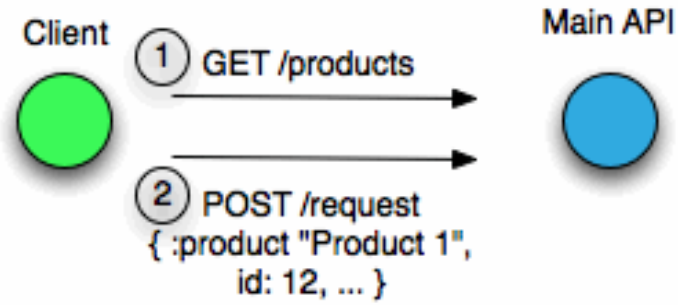
6. Kubernetes is more complex to set up and manage, it is typically used for large-scale production deployments, while Docker Swarm is simpler and more suited for small-scale and development deployments.

Microservice Architecture

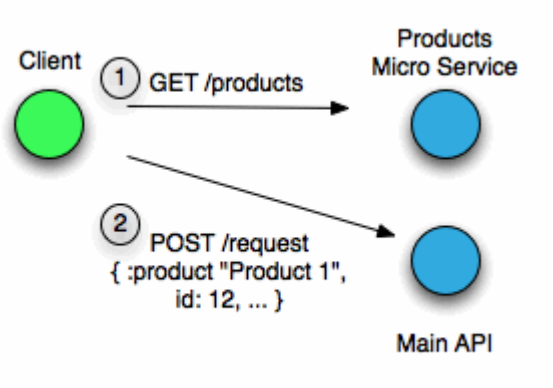
“Microservice architectural is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.”

Microservice Architecture (cont.)

Traditional development

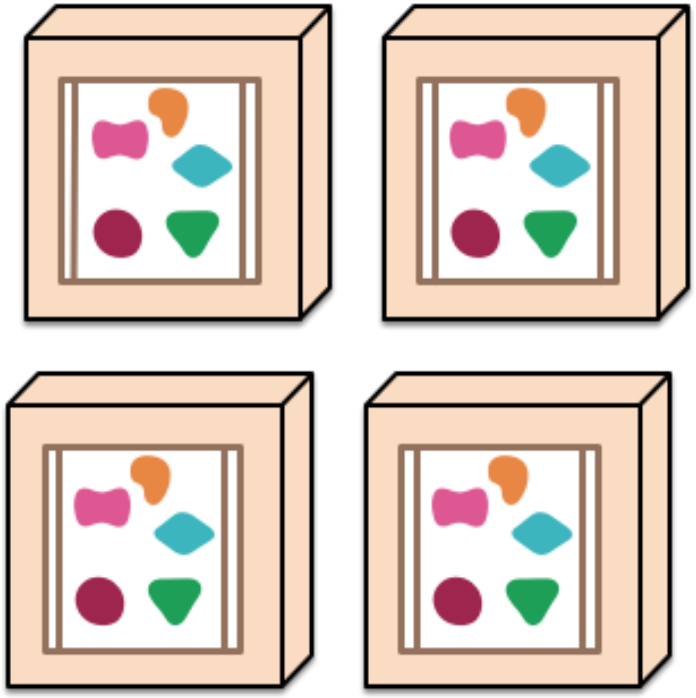


Microservices Architecture

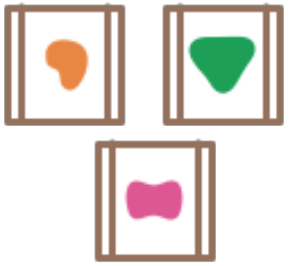


A monolithic application puts all its functionality into a single process...

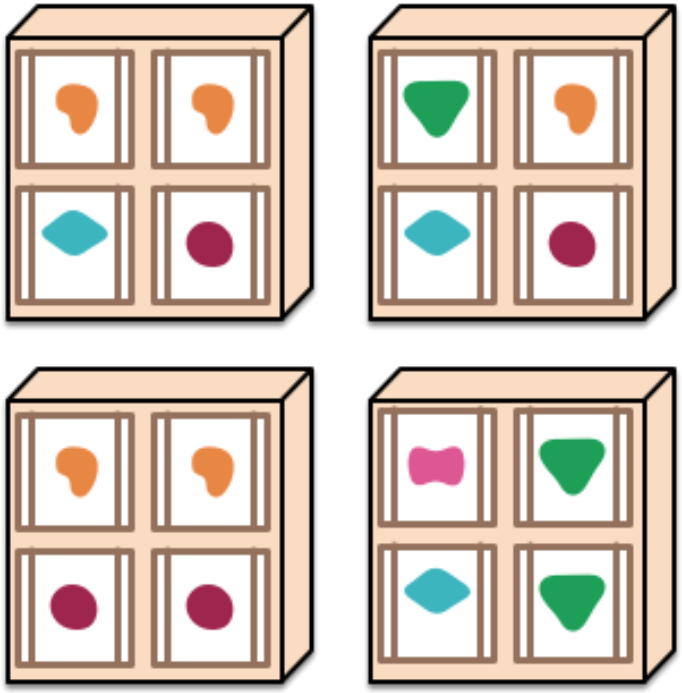
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Coming up next!

Working with Docker . . .