Iceland Liechtenstein Norway grants

Computing Innovation for Technology Entrepreneurship

Information and Communications Technology based Innovation

CITE Computing Innovation for Technology Entrepreneurship

Innovative Trends: Cloud Computing

From Monoliths to Containerization

Alin ALECU

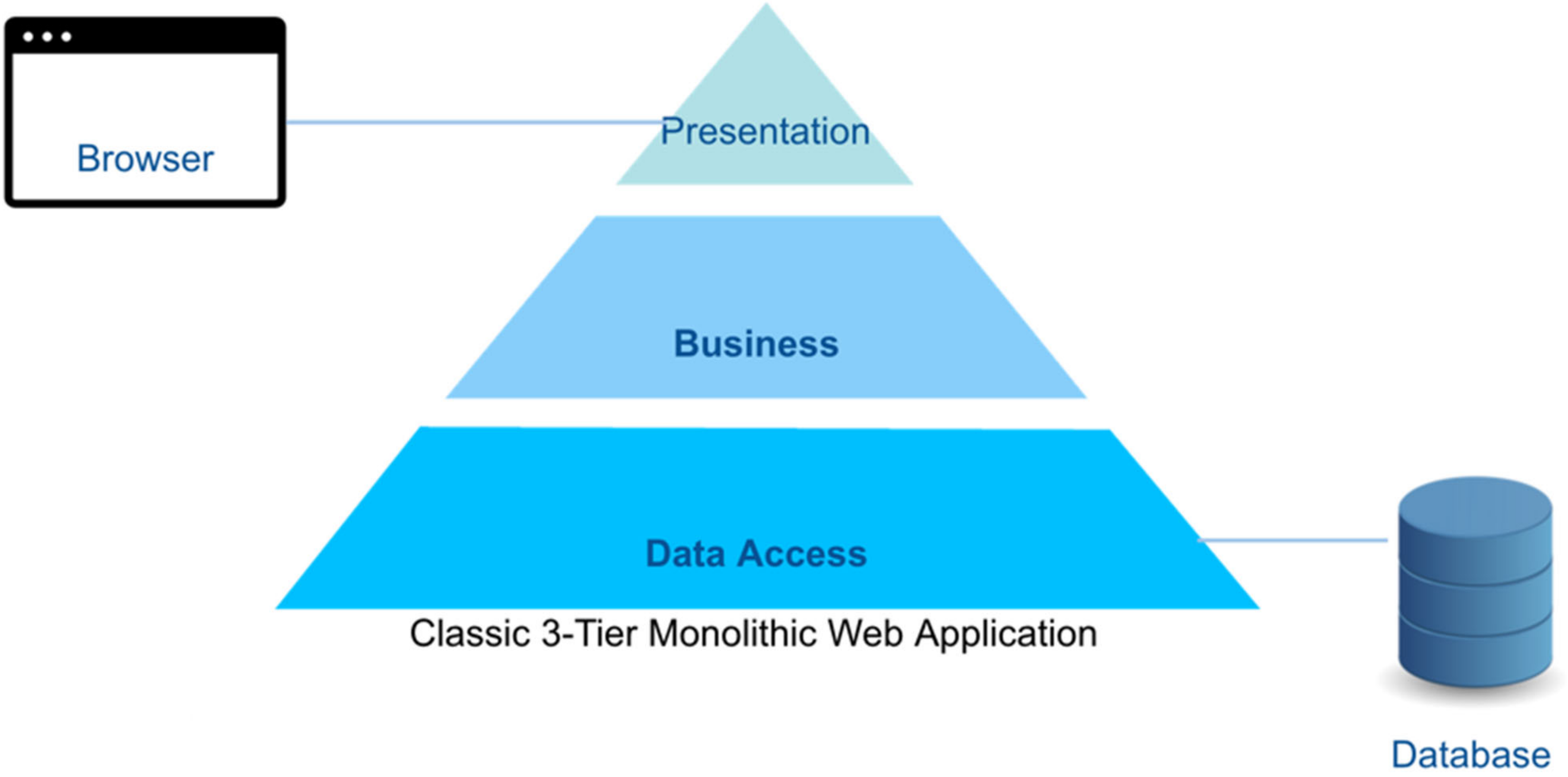
Outline

- Monoliths
- Service Oriented Architectures
- MicroServices
- Containerization
- Orchestration

Outline

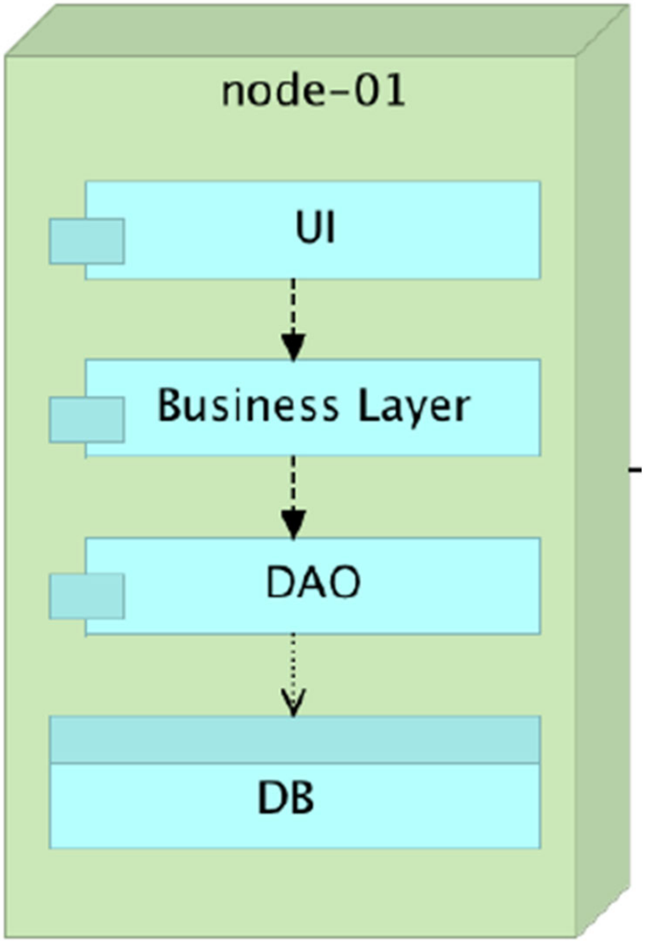
- **Monoliths**
- Service Oriented Architectures
- MicroServices
- Containerization
- Orchestration

Monolithic Applications



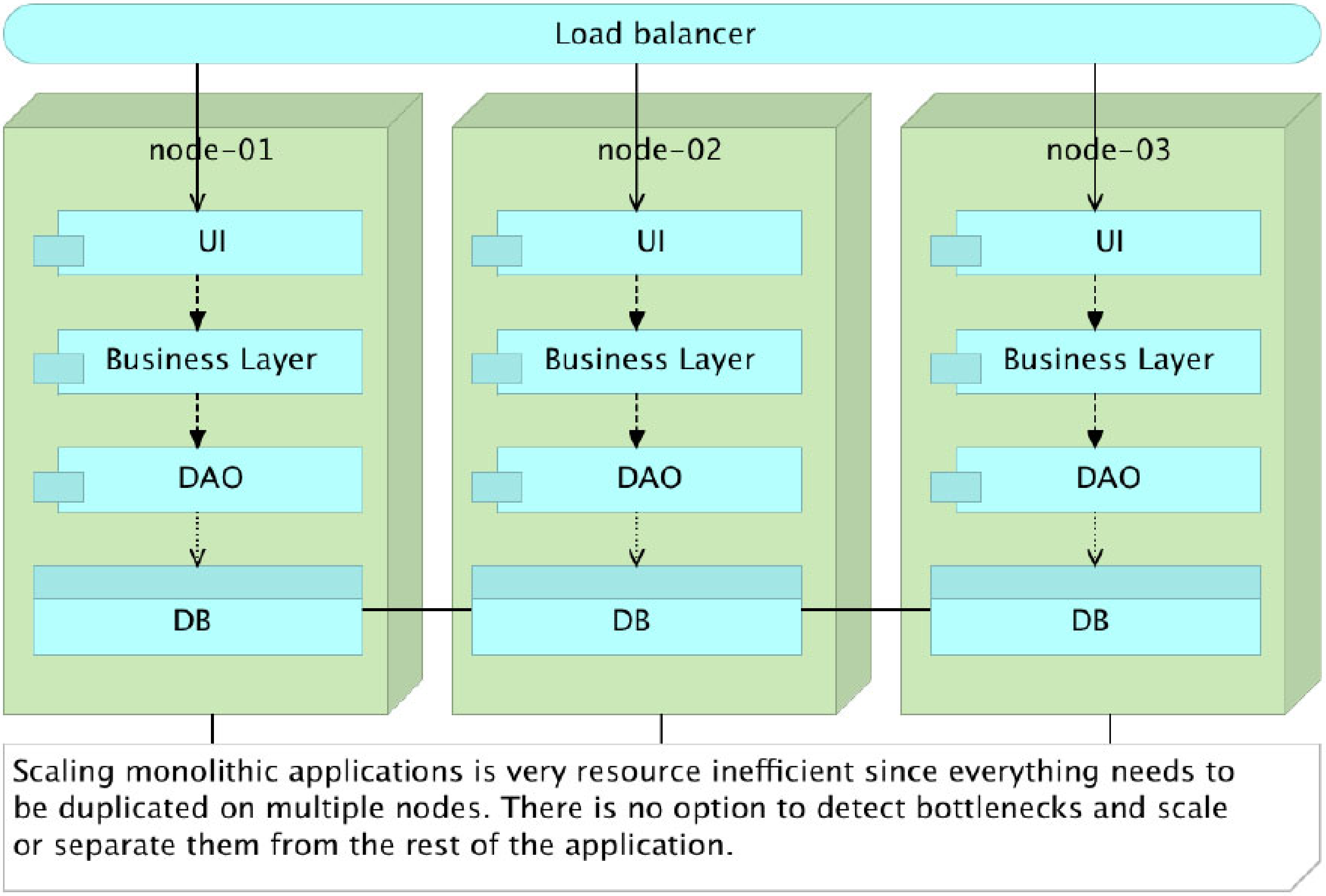
Classic 3-Tier Monolithic Web Application

Monolithic Applications



- When an application is relatively small, splitting it into horizontal layers is a good idea.
- It provides a separation that makes development faster and easier as well as well as a separation based on the type of task the code should do

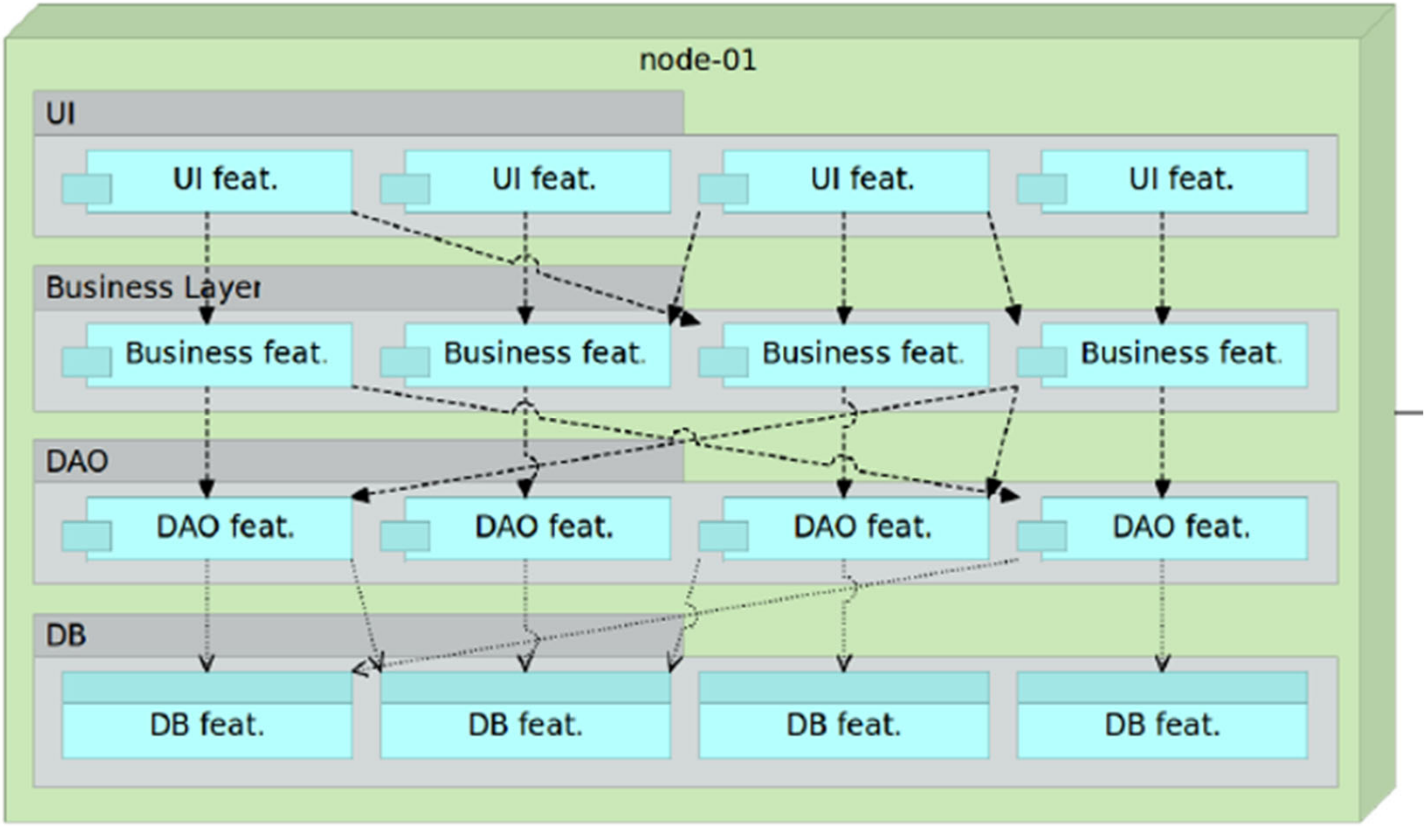
Scaling Monolithic Applications



Scaling monolithic applications is very resource inefficient since everything needs to be duplicated on multiple nodes. There is no option to detect bottlenecks and scale or separate them from the rest of the application.

Iceland
Liechtenstein
Norway grants

Monolithic Applications with increased number of features



- When an application becomes bigger and the number of features increases, initial design based on horizontal layers becomes less efficient.
- Tight coupling between separate layers, longer paths for potentially simple solutions, increased complexity

Outline

- Monoliths
- **Service Oriented Architectures**
- MicroServices
- Containerization
- Orchestration

Service-Oriented Architectures (SOA)

- ❑ Lookup for the tomato sellers

  Yellow Pages: contain companies that are selling tomatoes, their location, and contact information.

- ❑ Find the service offered according to my needs

  Where, when and how can I buy tomatoes?

- ❑ Buy the tomatoes

  Do the transaction

- ❑ Lookup for the Service Provider

  Registry: contain providers that are selling services, their location, and contact information.

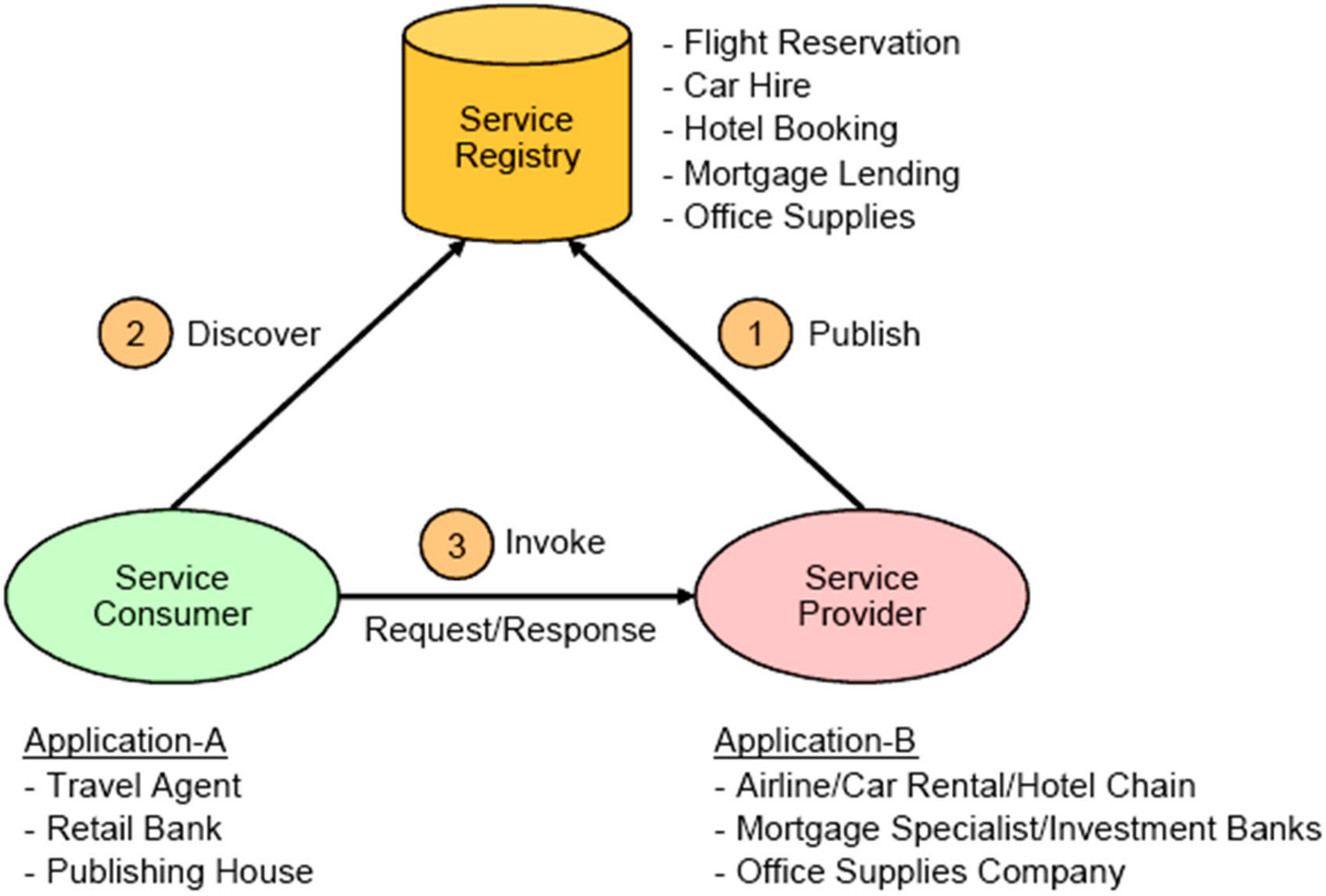
- ❑ Find the service offered according to my needs

  Where, when and how can I get the service?

- ❑ Access the service

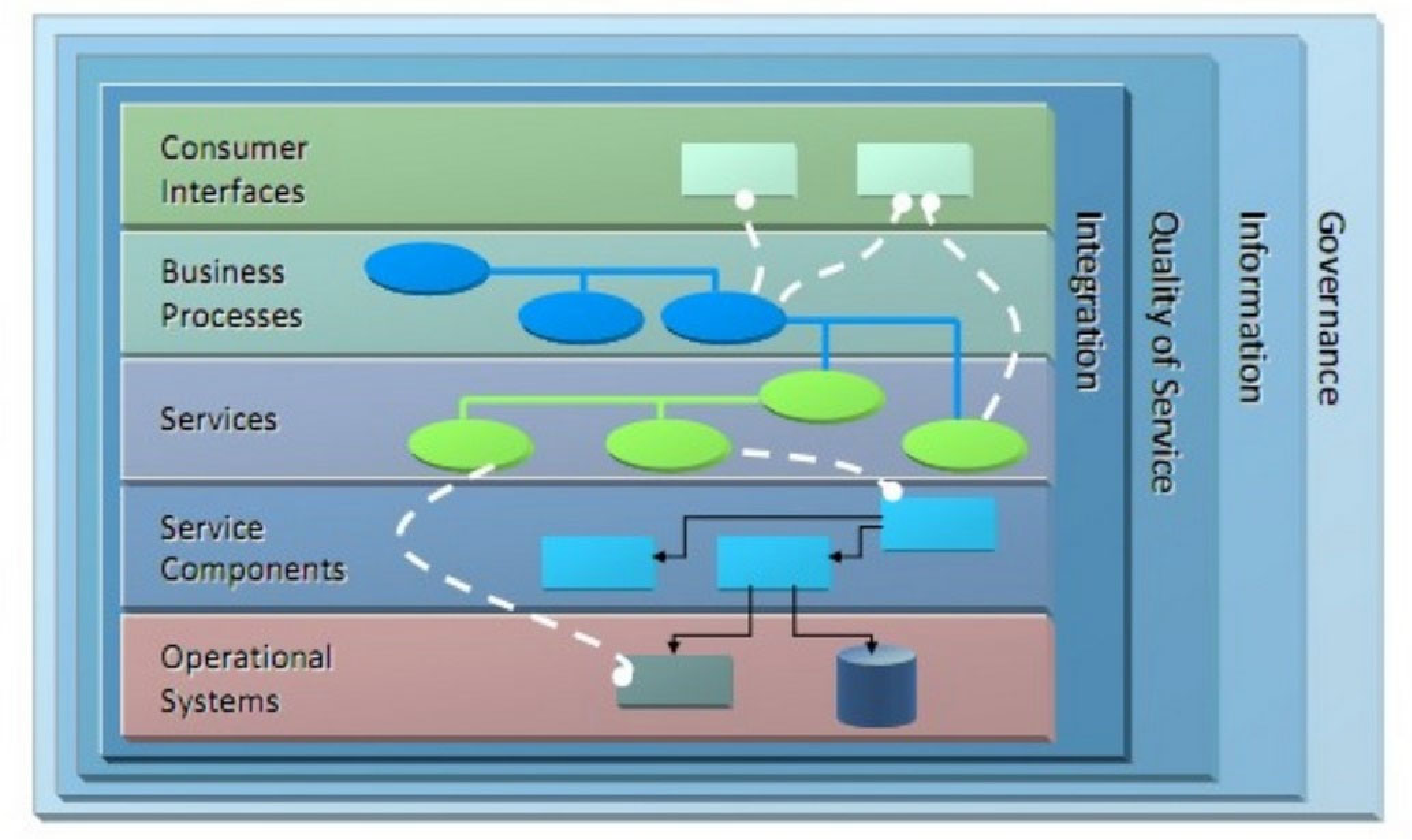
  do the transaction

Service-Oriented Architectures (SOA): Components and Operations

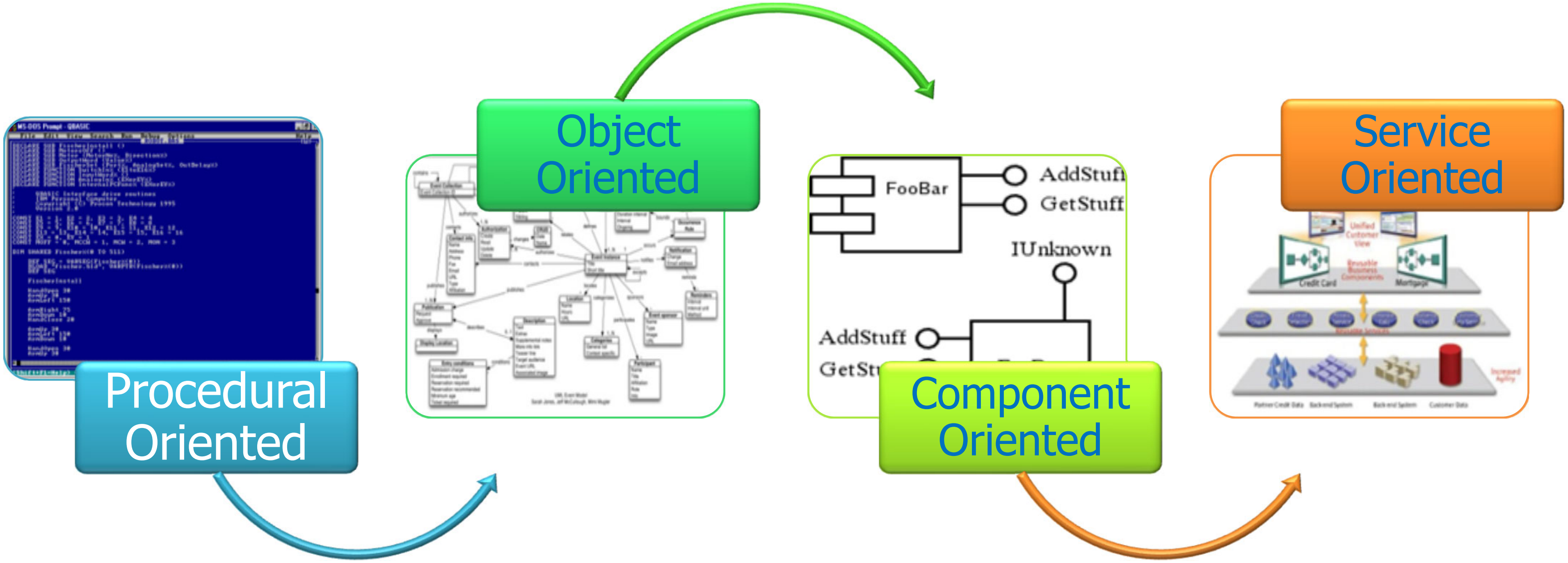


Iceland Liechtenstein Norway grants

Open Group SOA Reference Architecture (SOA RA)



Historically, how did we get to SOA?



Procedural Oriented

Object Oriented

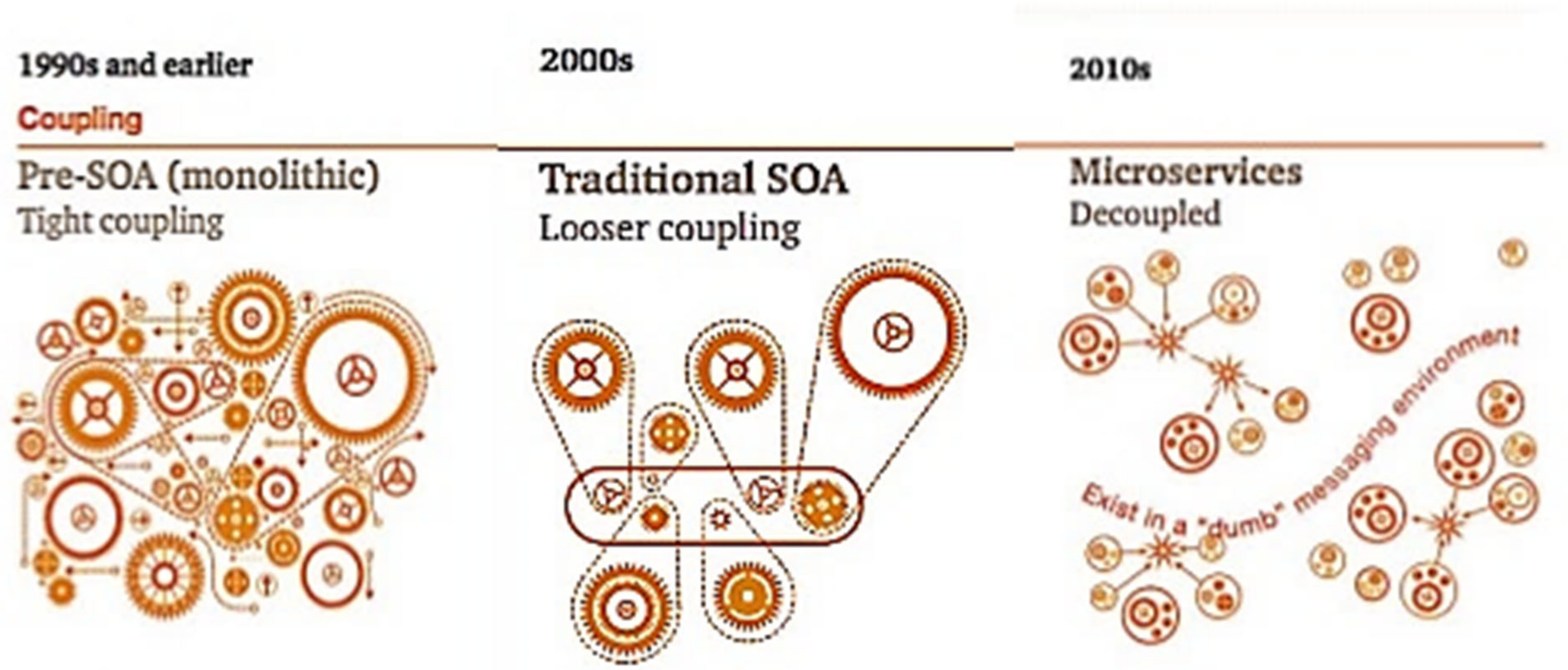
Component Oriented

Service Oriented

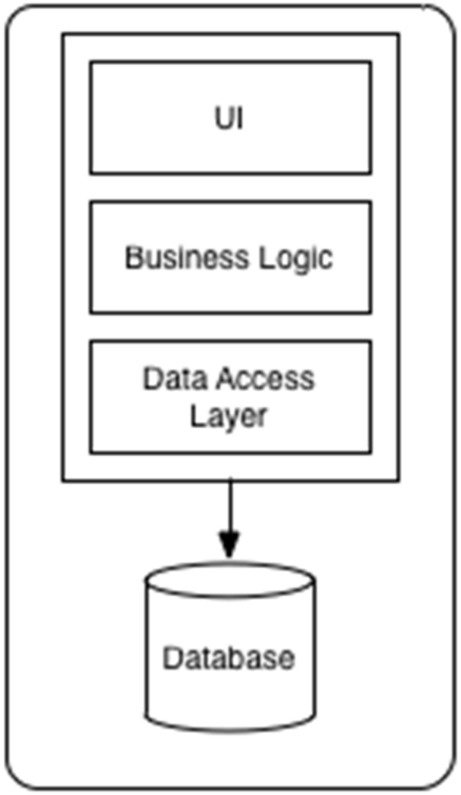
Outline

- Monoliths
- Service Oriented Architectures
- **MicroServices**
- Containerization
- Orchestration

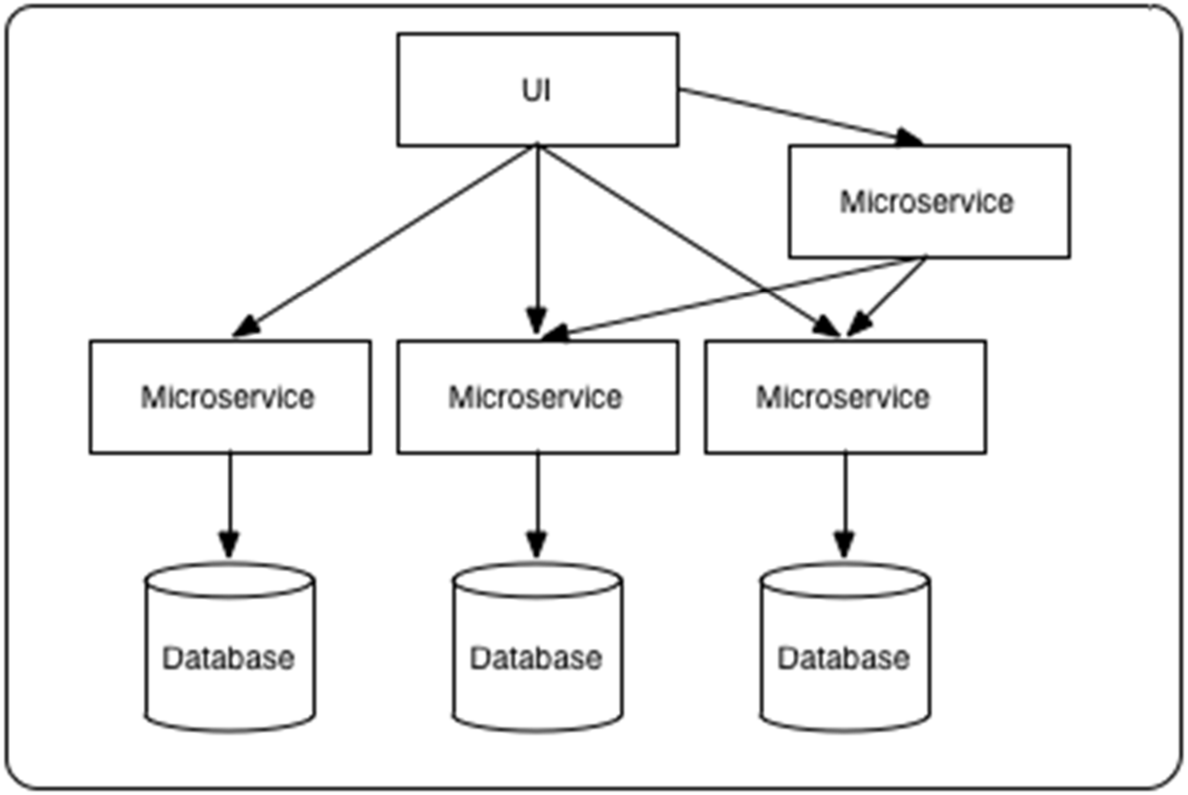
From Monolith to SOA and on to Microservices



Monolith vs Microservices

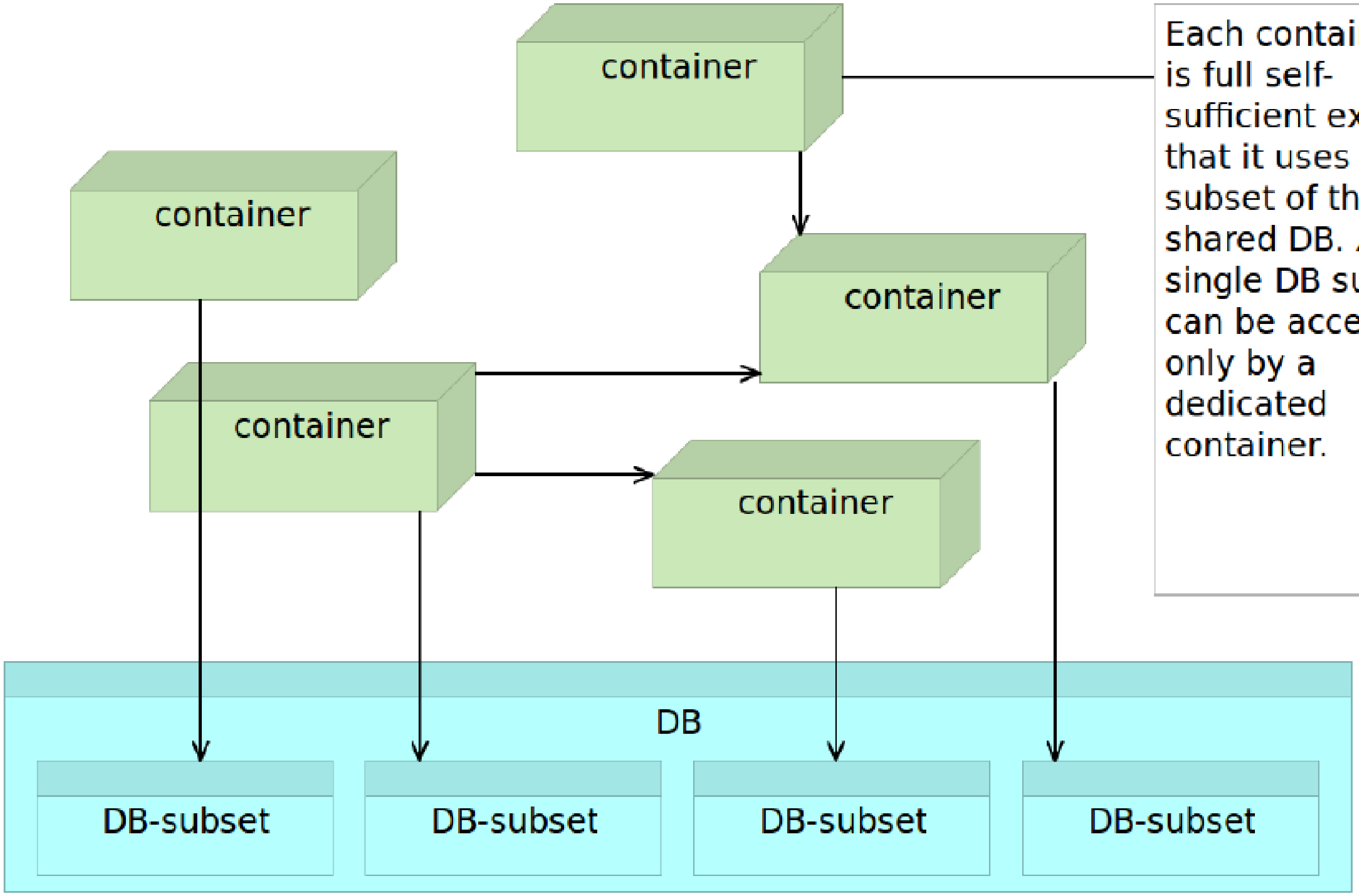


Monolithic Architecture



Microservices Architecture

Example: microservices accessing the shared database



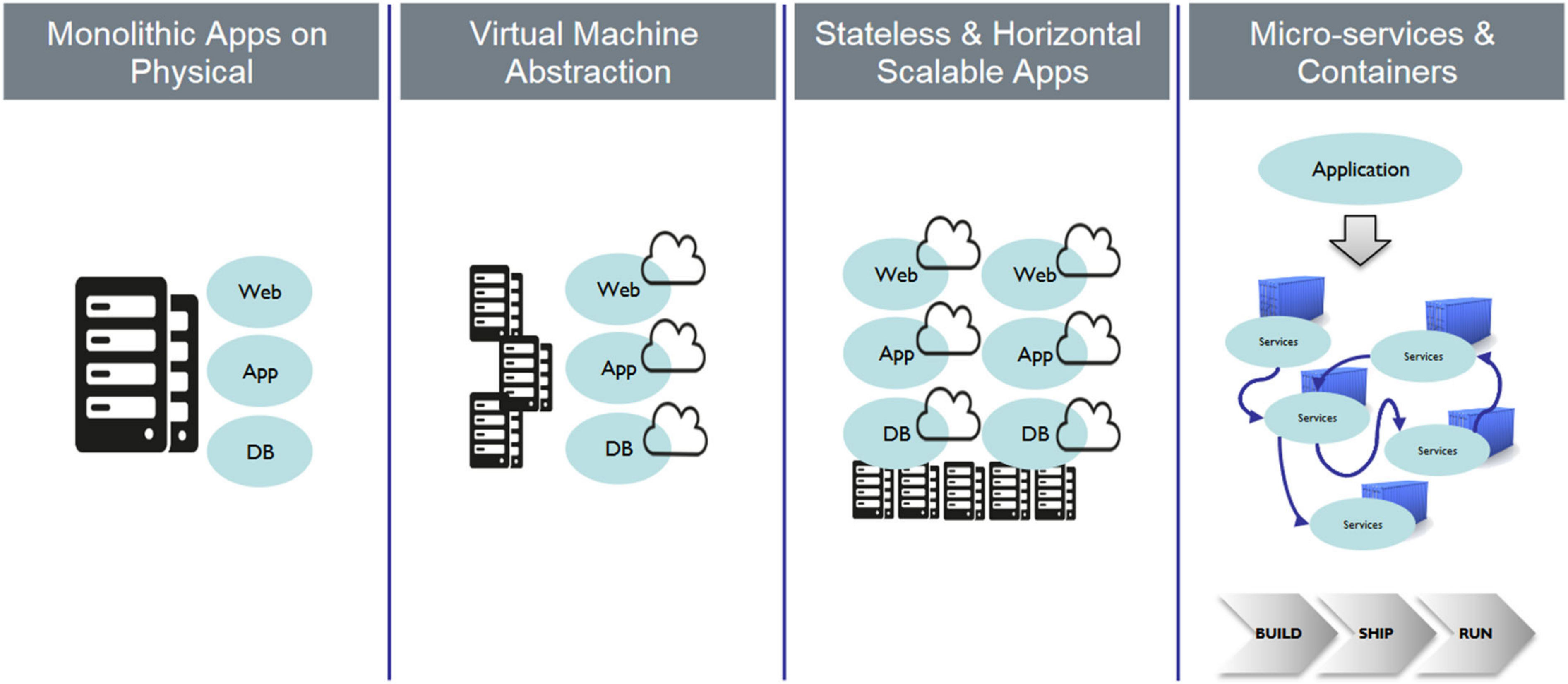
Each container is full self-sufficient except that it uses a subset of the shared DB. A single DB subset can be accessed only by a dedicated container.

Outline

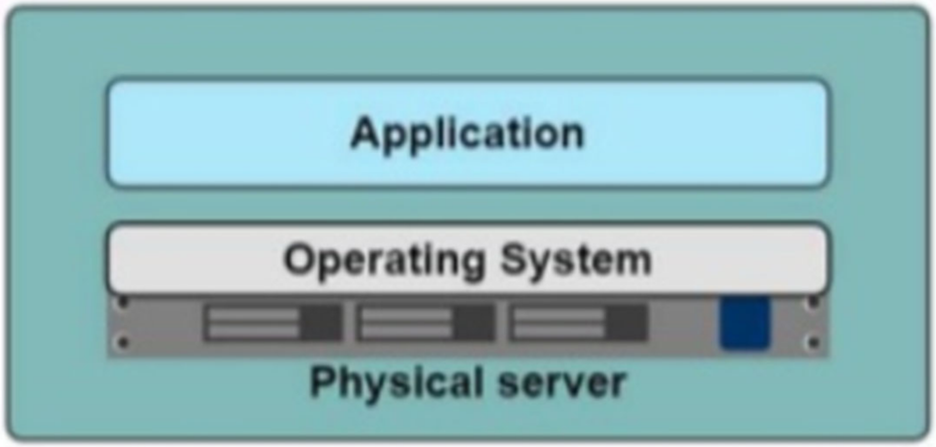
- Monoliths
- Service Oriented Architectures
- MicroServices
- **Containerization**
- Orchestration

Iceland Liechtenstein Norway grants

Application Deployment History


Monolithic Apps on Physical
Web
App
DB
Virtual Machine Abstraction
Web
App
DB
Stateless & Horizontal Scalable Apps
Web
Web
App
App
DB
DB
Micro-services & Containers
Application
Services
Services
Services
Services
Services
BUILD
SHIP
RUN

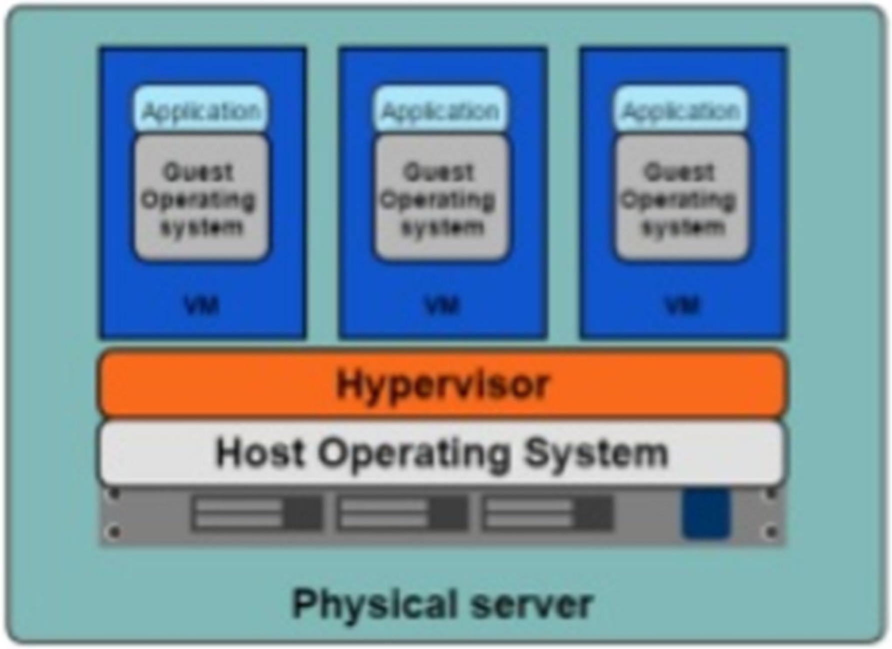
One monolithic application on one physical server



Application deployment limitations:

- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock-in

Hypervisor-based Virtualization



- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)

Hypervisor-based Virtualization

Benefits:

- Better resource pooling
    - One physical machine divided into multiple virtual machines
- Easier to scale
    - VMs in the cloud
    - Rapid elasticity
    - Pay as you go

Microsoft Azure

amazon
web services™

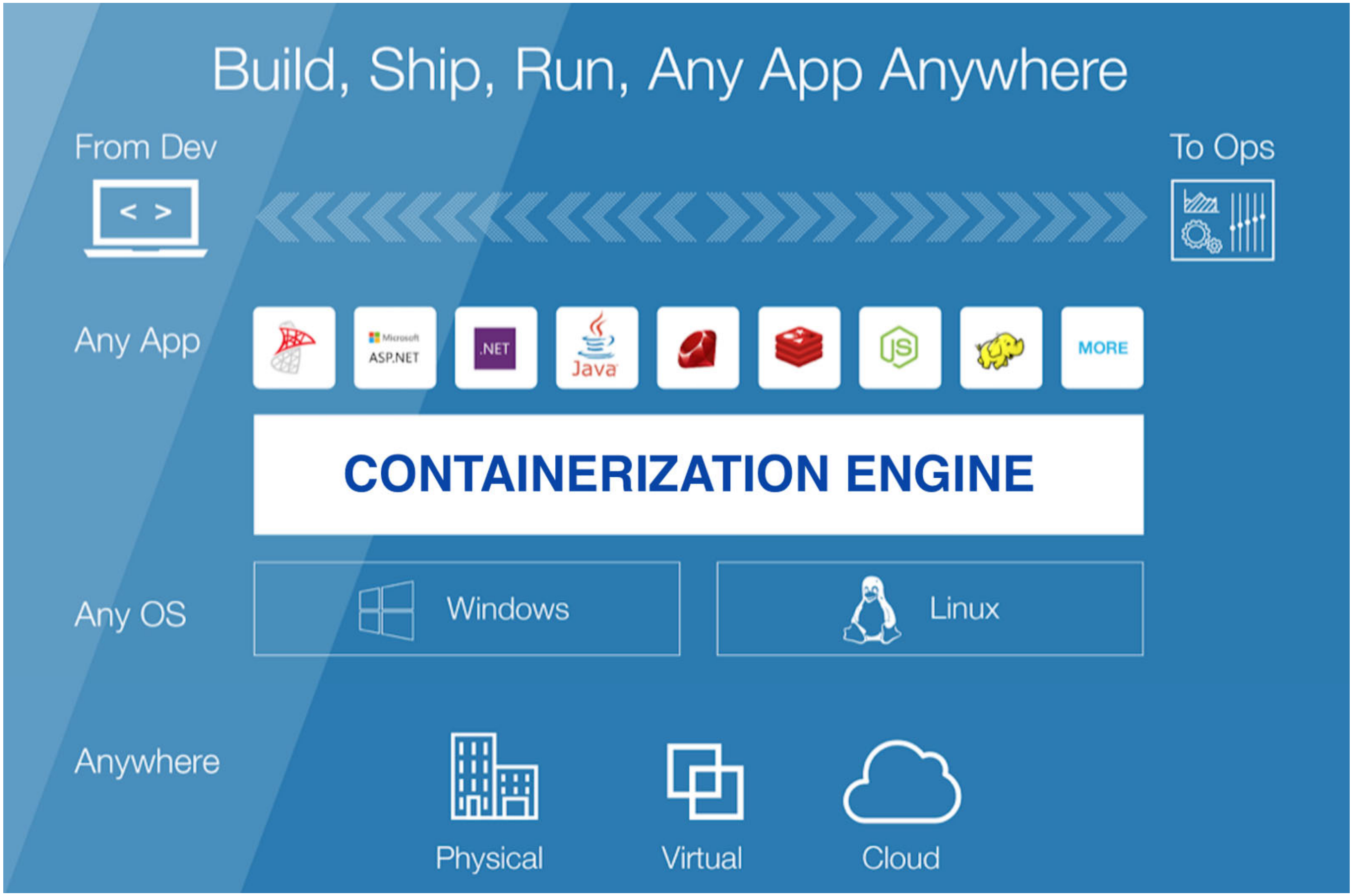
vmware

Hypervisor-based Virtualization

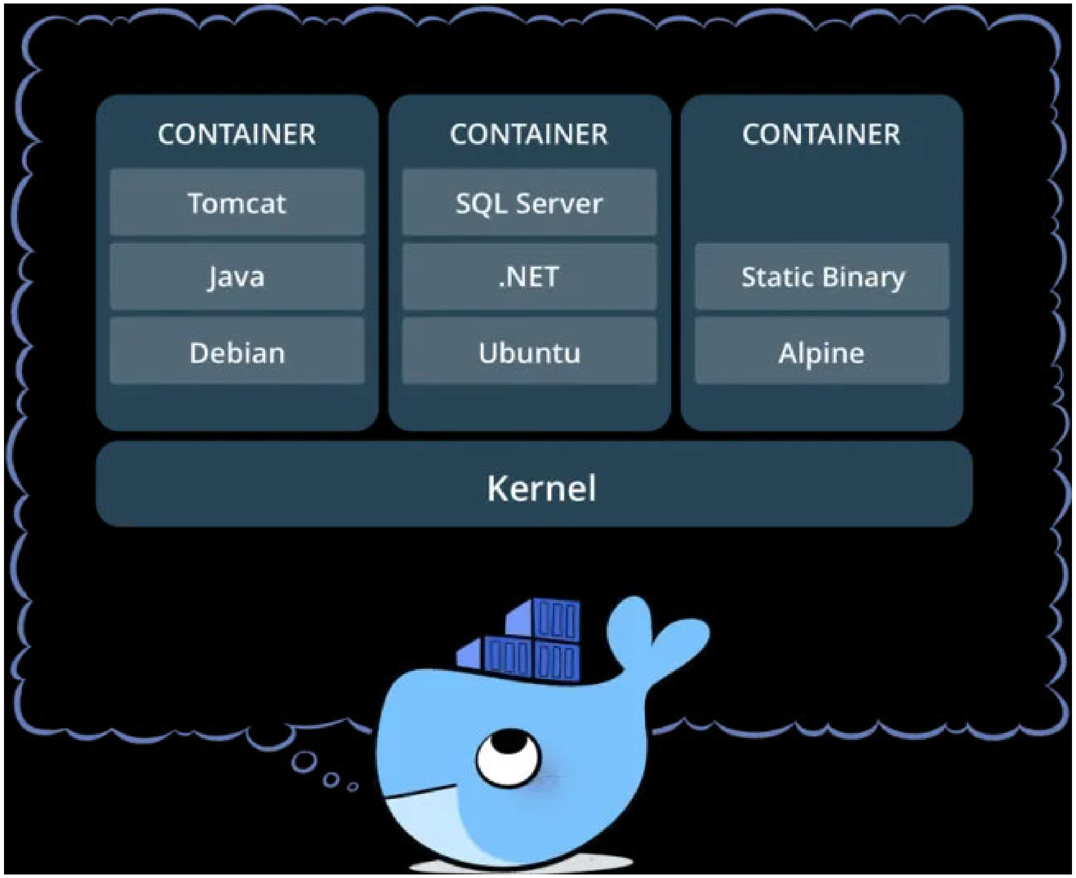
Limitations:

- Each VM still requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

Containerization

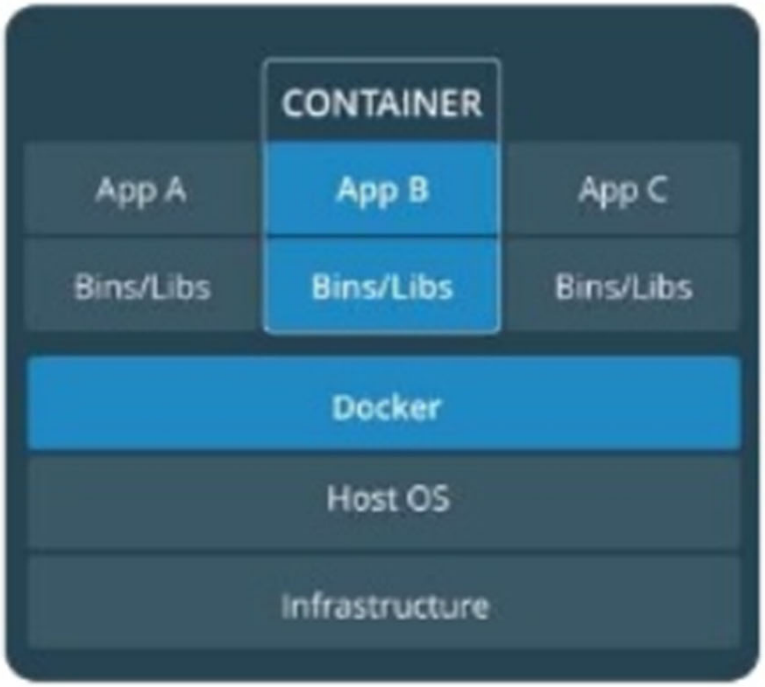


What is a Container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Virtualization of applications instead of hardware
- Runs on top of the core OS (Linux and Windows Server)
- Doesn’t require dedicated CPU, Mem, Network (managed by core OS)

Comparing Containers vs VMs



CONTAINER

App A | App B | App C

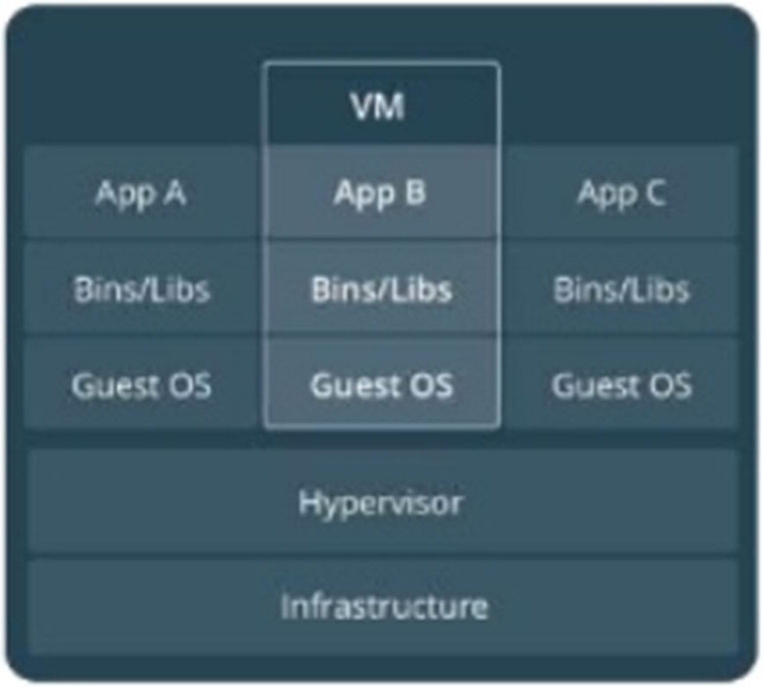
Bins/Libs | Bins/Libs | Bins/Libs

Docker

Host OS

Infrastructure

Containers are an app level construct



VM

App A | App B | App C

Bins/Libs | Bins/Libs | Bins/Libs

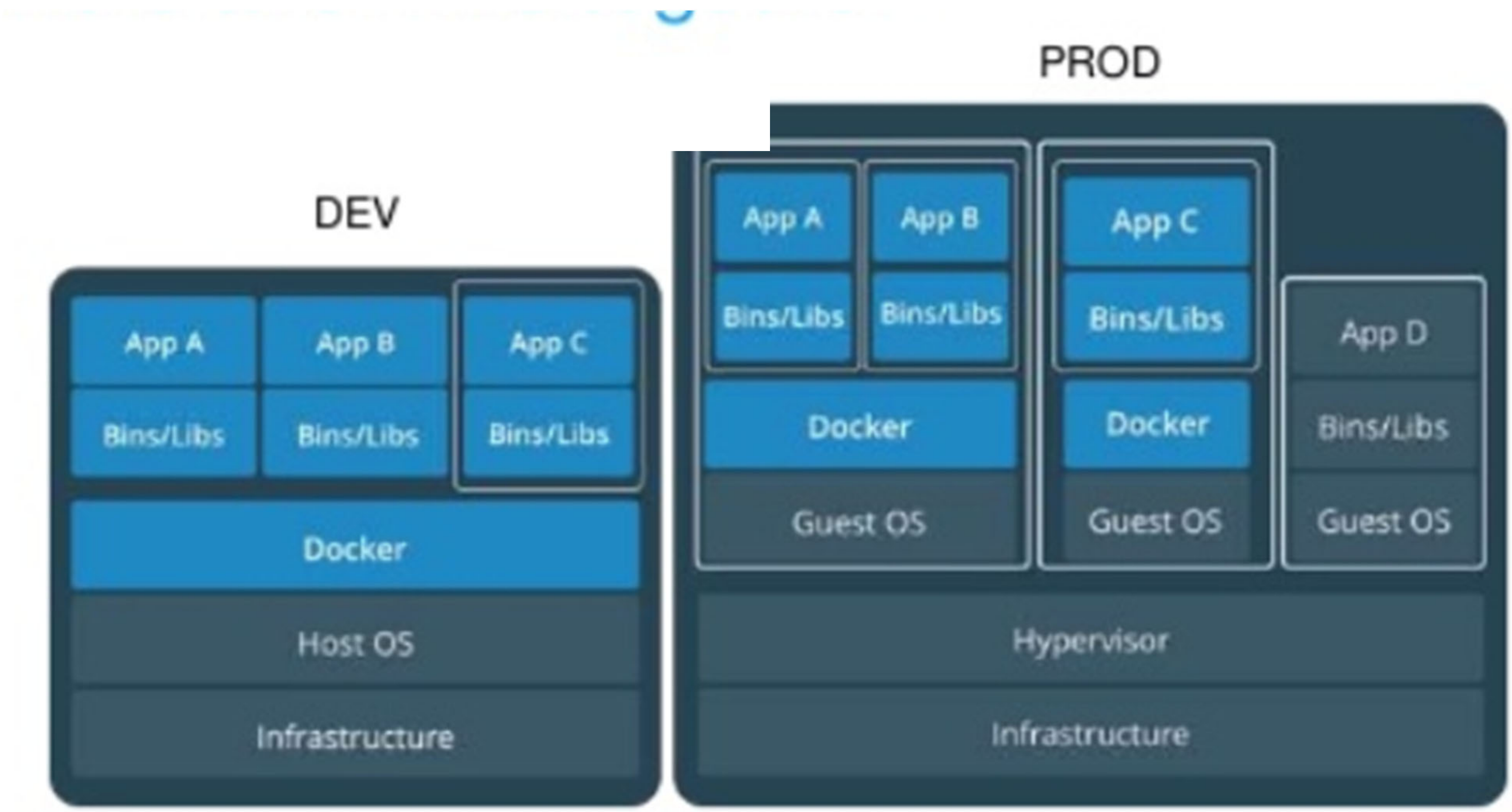
Guest OS | Guest OS | Guest OS

Hypervisor

Infrastructure

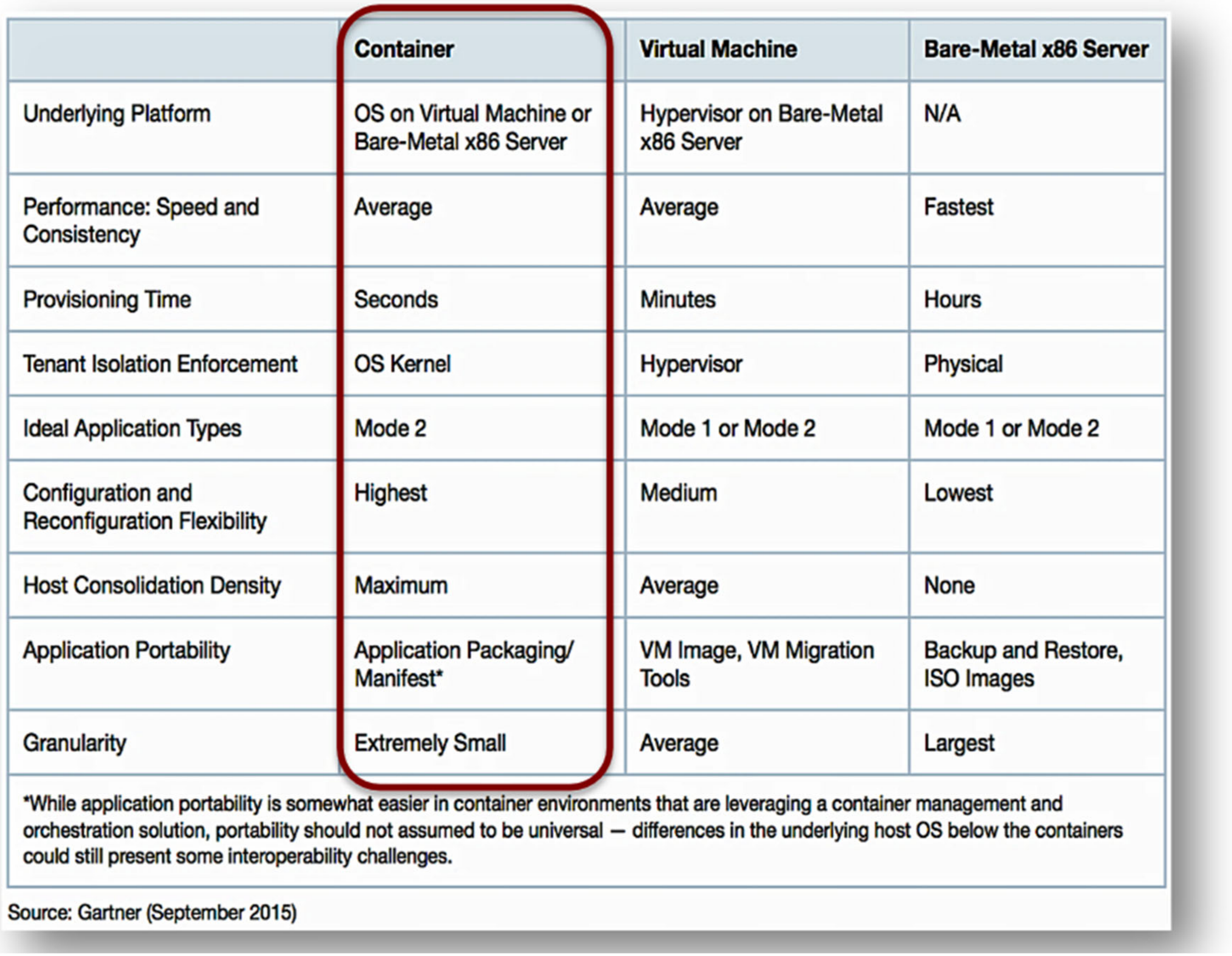
VMs are an infrastructure level construct to turn one machine into many servers

Containers and VMs together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Containers vs VMs vs Bare-metal Servers

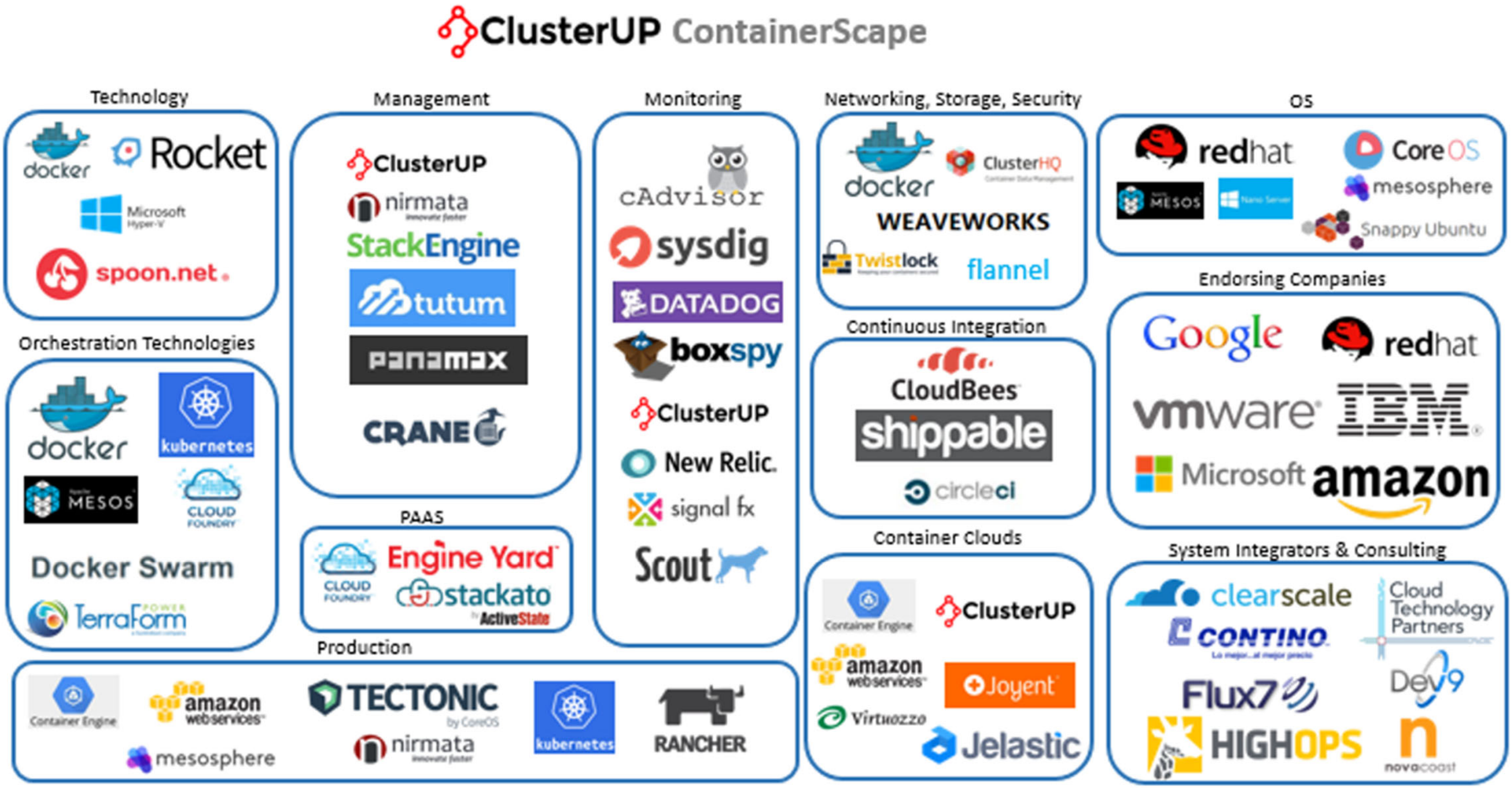


	Container	Virtual Machine	Bare-Metal x86 Server
Underlying Platform	OS on Virtual Machine or Bare-Metal x86 Server	Hypervisor on Bare-Metal x86 Server	N/A
Performance: Speed and Consistency	Average	Average	Fastest
Provisioning Time	Seconds	Minutes	Hours
Tenant Isolation Enforcement	OS Kernel	Hypervisor	Physical
Ideal Application Types	Mode 2	Mode 1 or Mode 2	Mode 1 or Mode 2
Configuration and Reconfiguration Flexibility	Highest	Medium	Lowest
Host Consolidation Density	Maximum	Average	None
Application Portability	Application Packaging/ Manifest*	VM Image, VM Migration Tools	Backup and Restore, ISO Images
Granularity	Extremely Small	Average	Largest

*While application portability is somewhat easier in container environments that are leveraging a container management and orchestration solution, portability should not assumed to be universal — differences in the underlying host OS below the containers could still present some interoperability challenges.

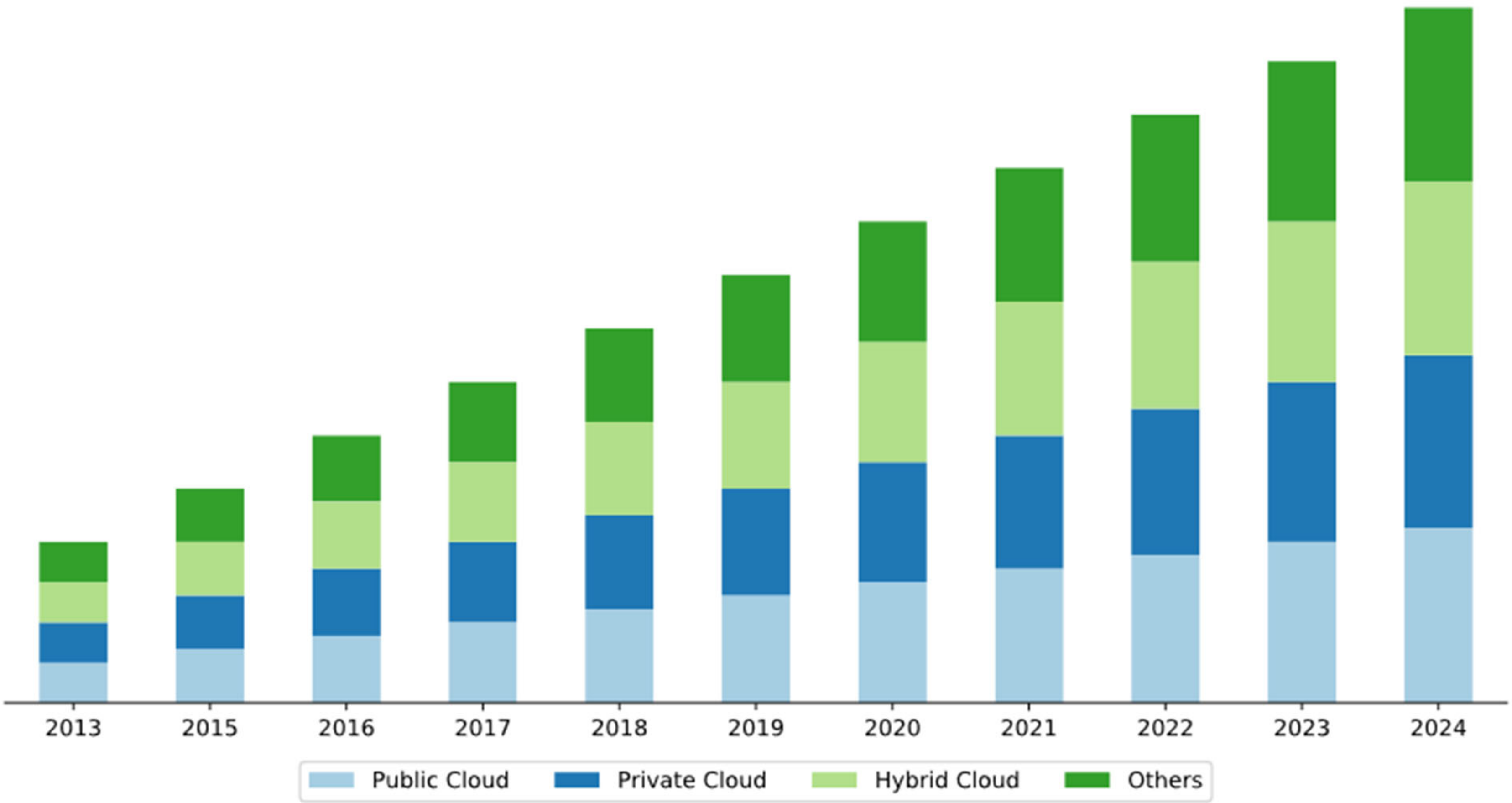
Source: Gartner (September 2015)

Container Business Landscape



* Reproduced from ClusterUP

Container as a Service (CaaS)

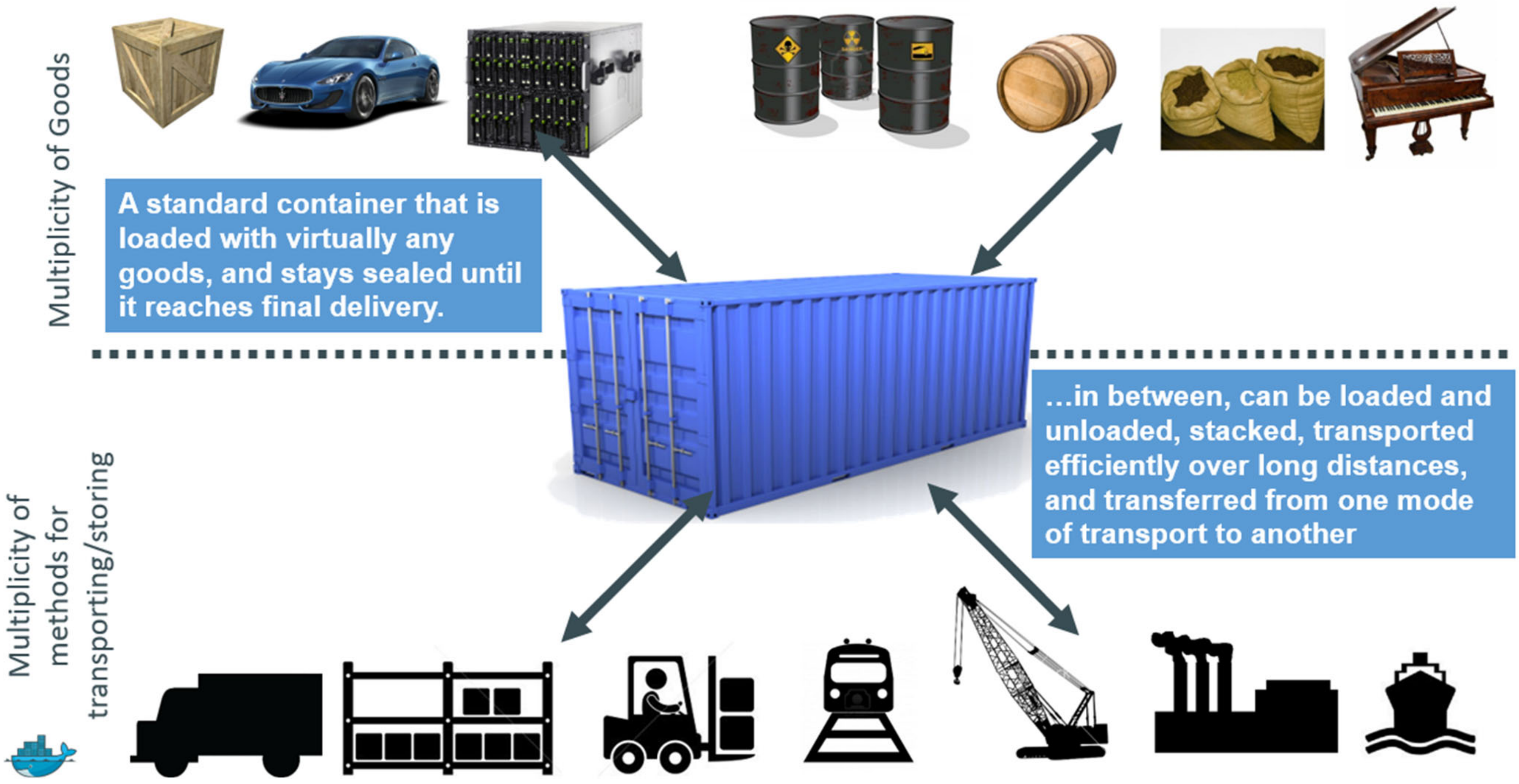


Container as a Service (CaaS) market size, by product, 2013-2024 (USD Million)
www.marketintellica.com

Outline

- Monoliths
- Service Oriented Architectures
- MicroServices
- **Containerization**
- **Docker**
- Orchestration

Intermodal shipping containers



Multiplicity of Goods

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

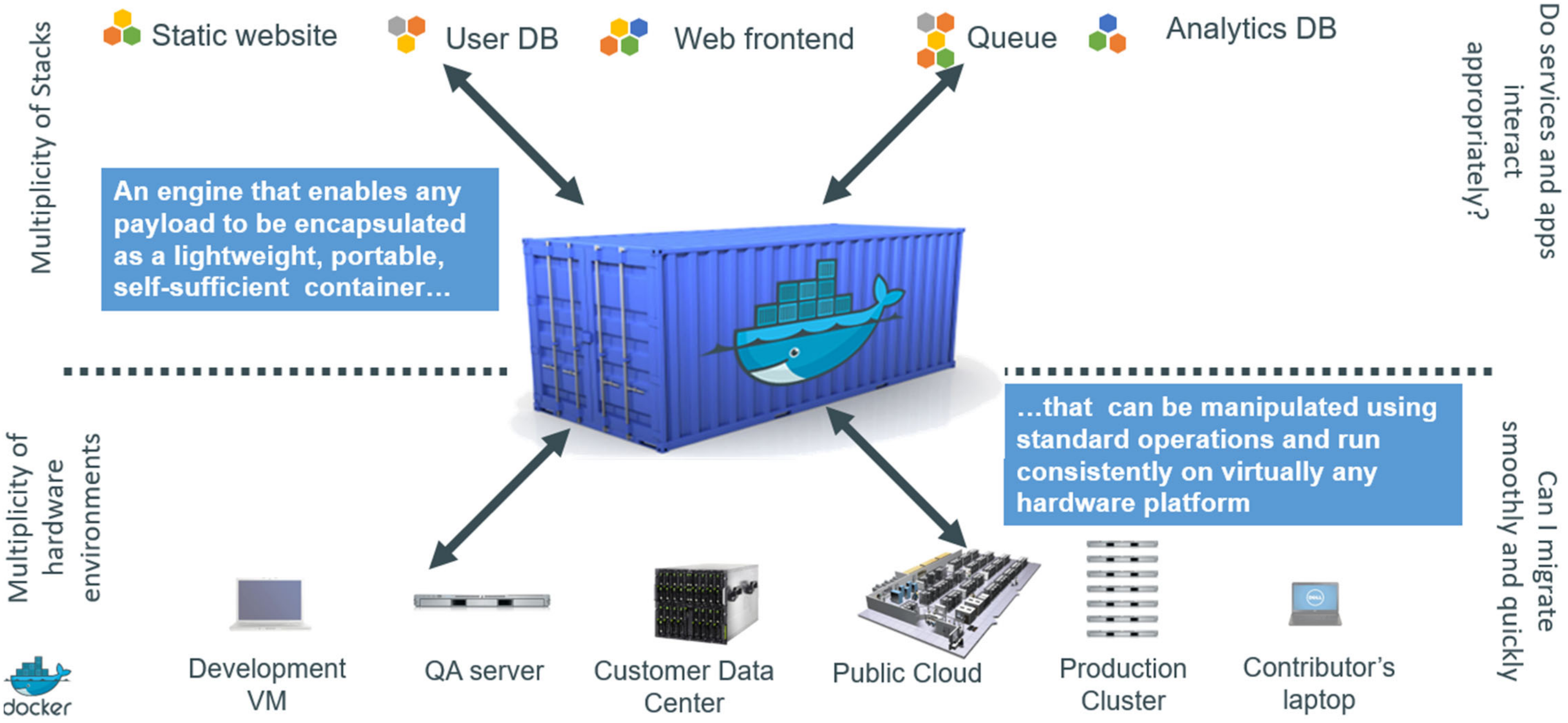
Do I worry about how goods interact (e.g. coffee beans next to spices)

…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)

Docker is a shipping container for code



Multiplicity of Stacks

Static website

User DB

Web frontend

Queue

Analytics DB

Do services and apps interact appropriately?

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

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud

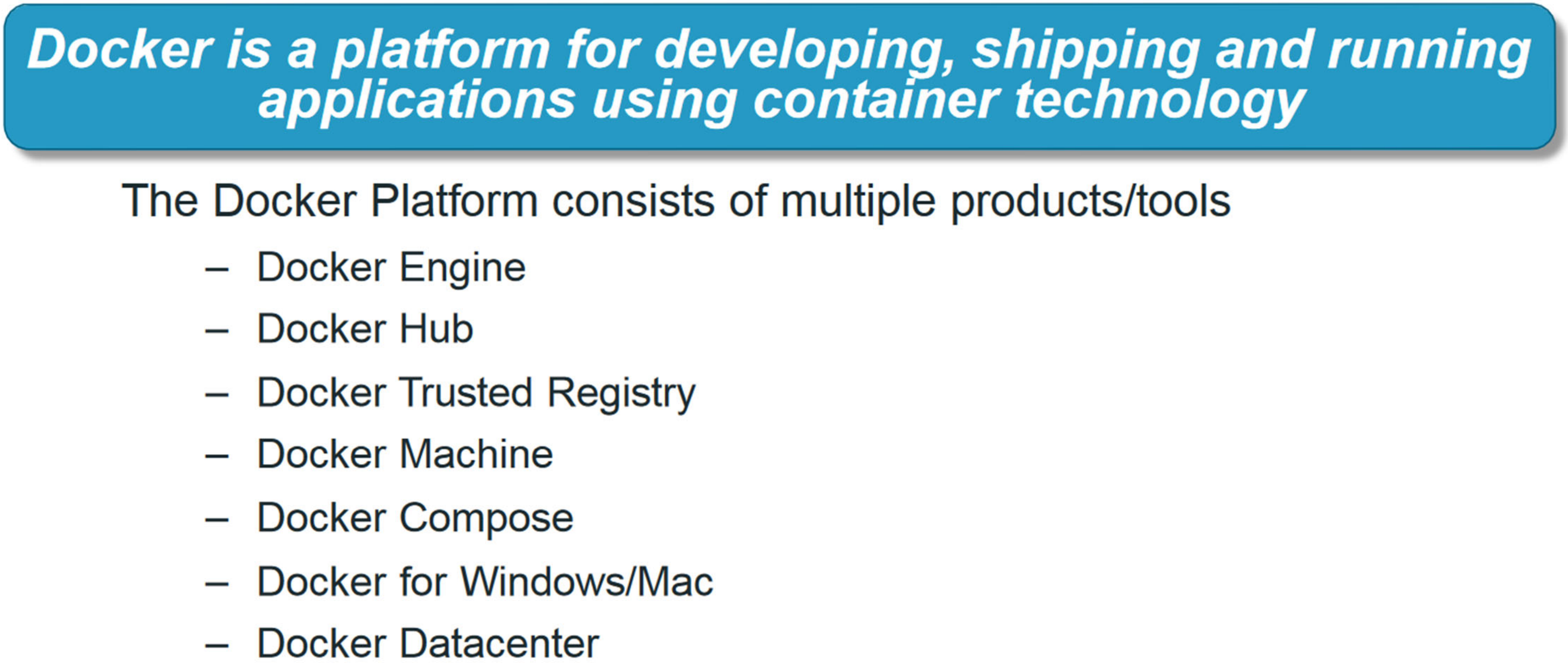
Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly

docker

Docker: Containerization for software

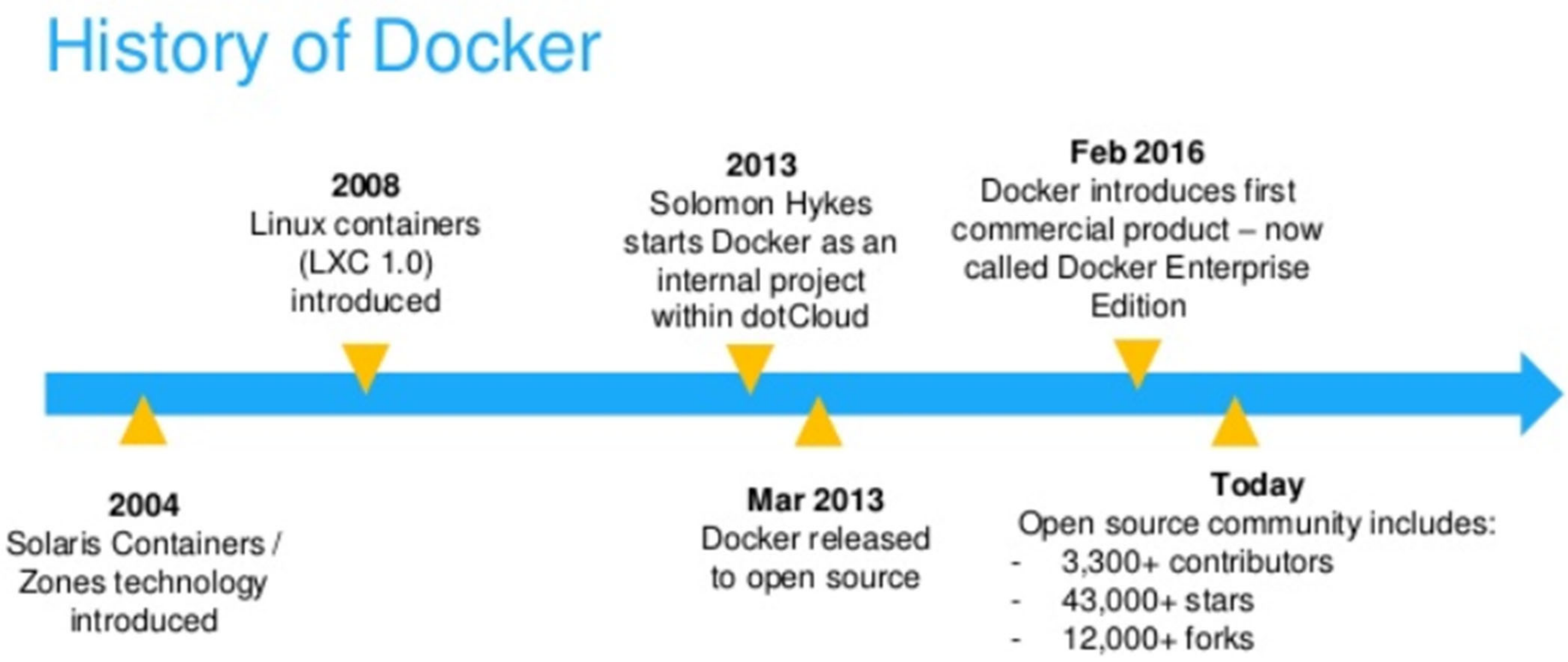


Docker is a platform for developing, shipping and running applications using container technology

The Docker Platform consists of multiple products/tools

- Docker Engine
- Docker Hub
- Docker Trusted Registry
- Docker Machine
- Docker Compose
- Docker for Windows/Mac
- Docker Datacenter

Docker: Evolution



History of Docker

2004
Solaris Containers / Zones technology introduced

2008
Linux containers (LXC 1.0) introduced

2013
Solomon Hykes starts Docker as an internal project within dotCloud

Mar 2013
Docker released to open source

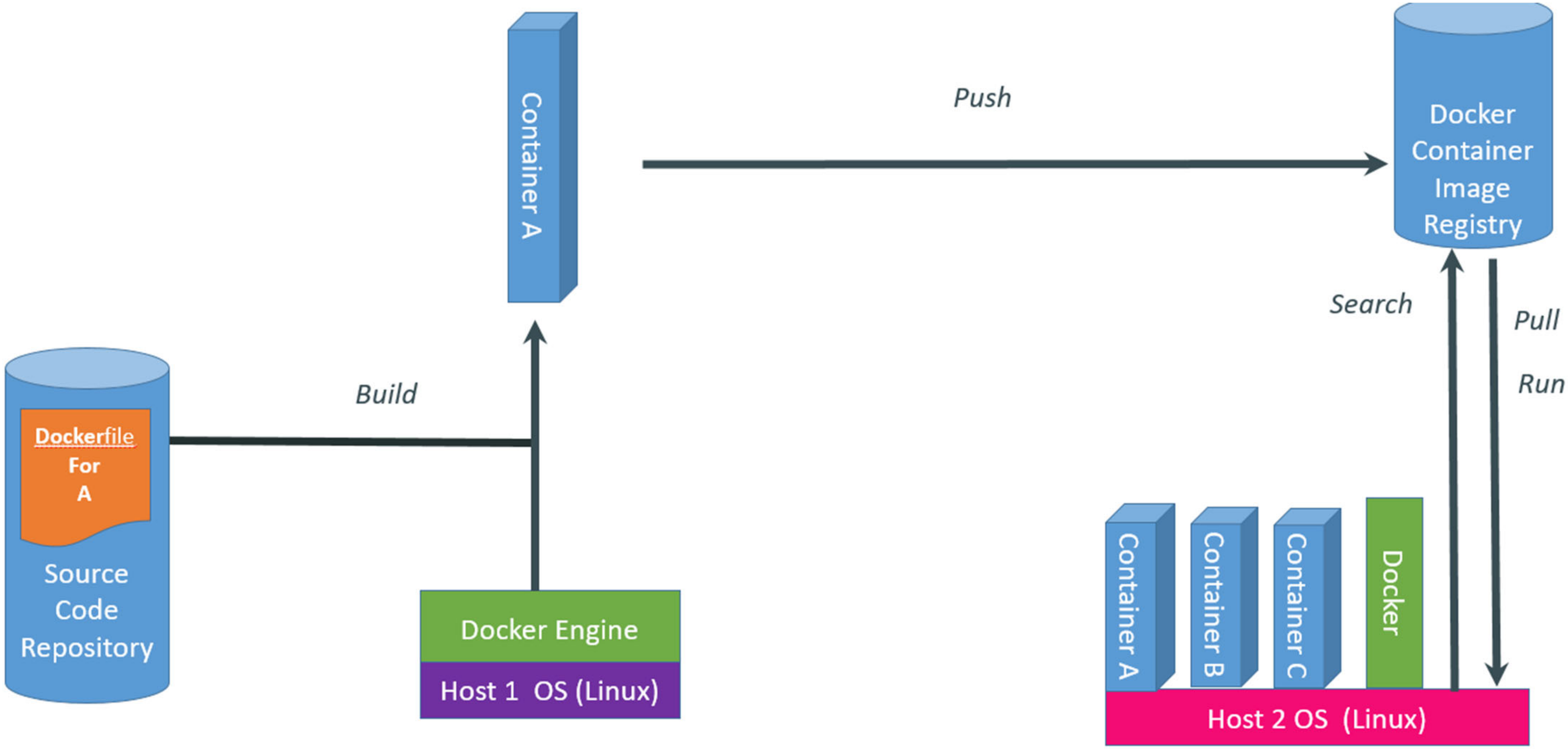
Feb 2016
Docker introduces first commercial product – now called Docker Enterprise Edition

Today
Open source community includes:

- 3,300+ contributors
- 43,000+ stars
- 12,000+ forks

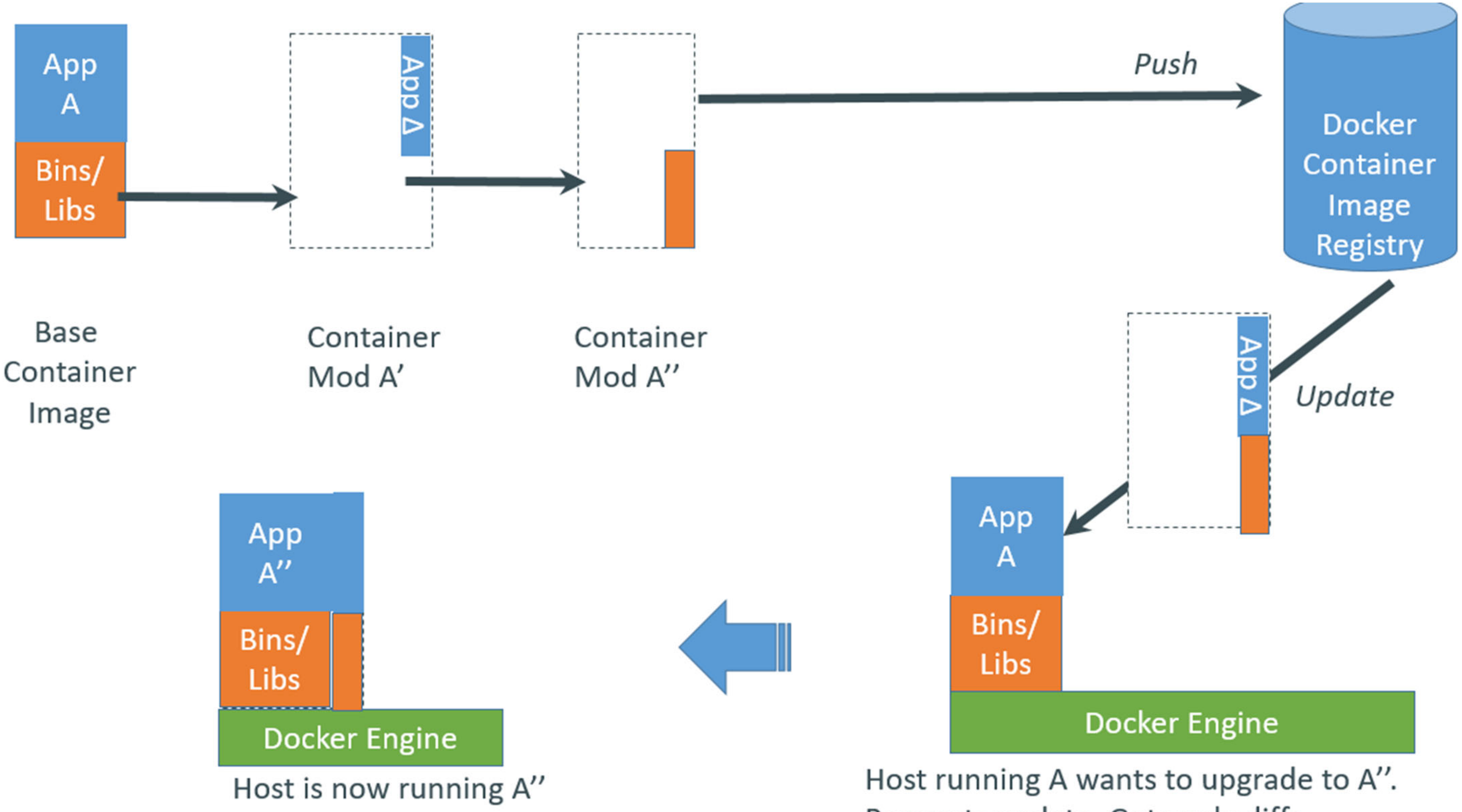
Iceland Liechtenstein Norway grants

What are the basics of the Docker system?



Iceland Liechtenstein Norway grants

Docker: Easy changes and updates



Host is now running A''

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

Docker: Containerization for software

Docker is a platform for developing, shipping and running applications using container technology

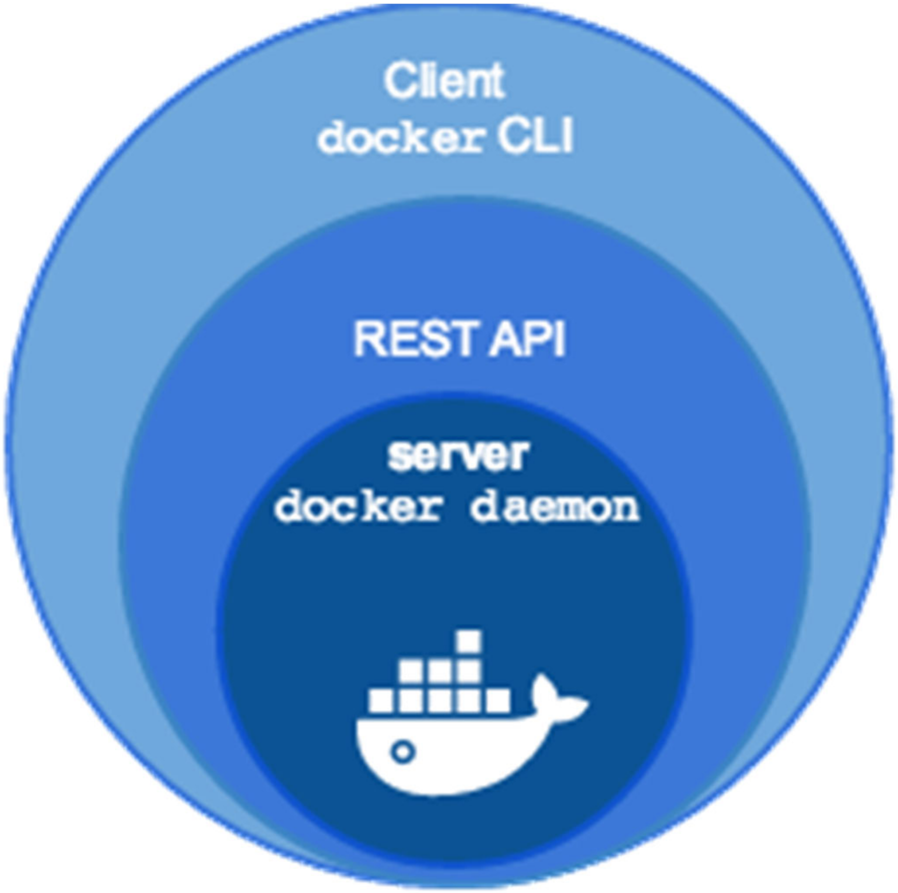
The Docker Platform consists of multiple products/tools

- Docker Engine
- Docker Hub
- Docker Trusted Registry
- Docker Machine
- Docker Compose
- Docker for Windows/Mac
- Docker Datacenter

- Dockerfile
- Docker Networking

Let’s look at some of these more in depth

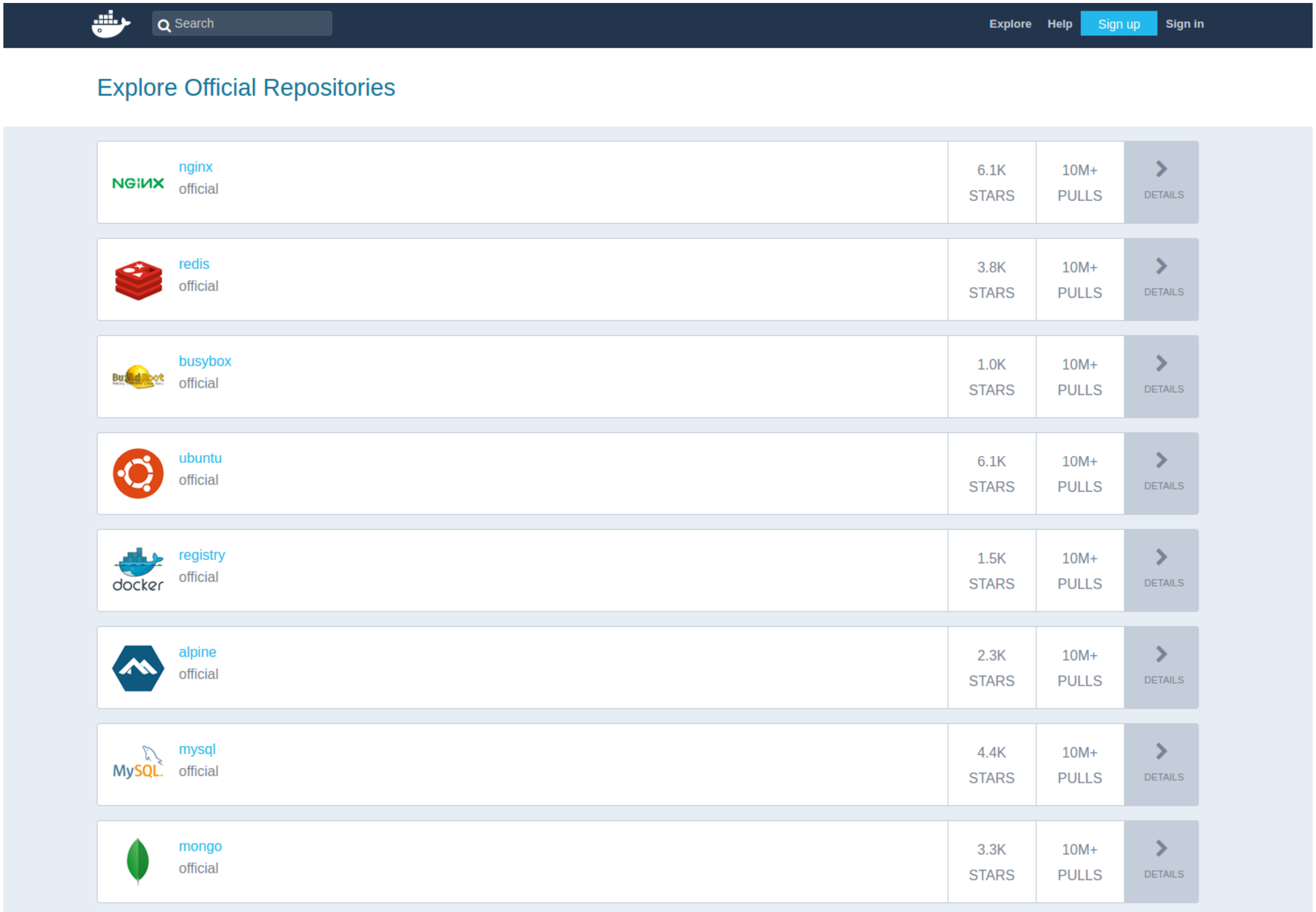
Docker Engine



Docker Engine:

- Docker daemon
- REST API
- CLI interface

Docker Hub



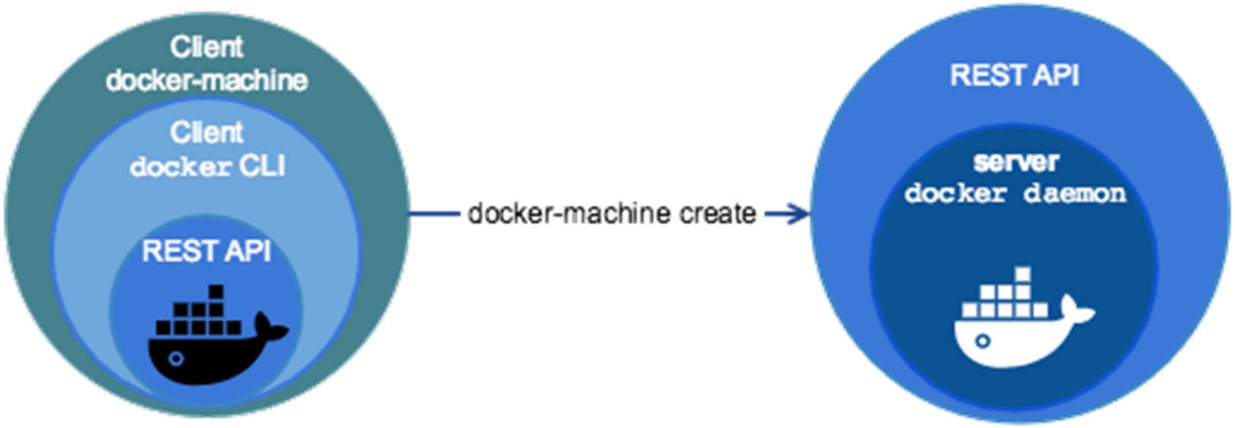
Explore Official Repositories

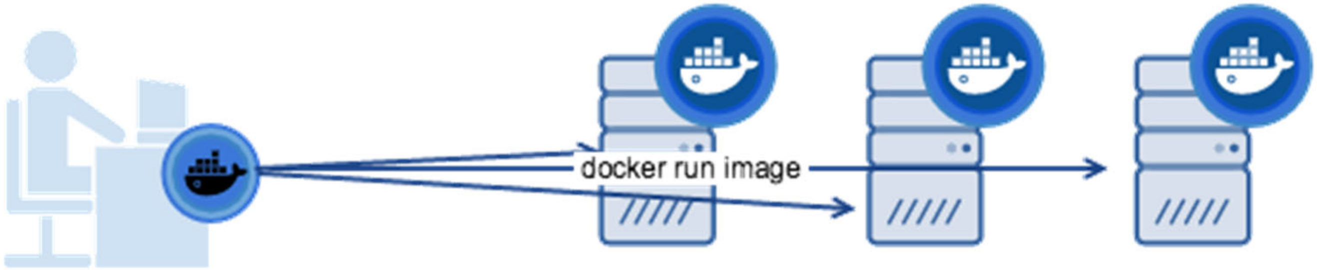
nginx official	6.1K STARS	10M+ PULLS	DETAILS
redis official	3.8K STARS	10M+ PULLS	DETAILS
busybox official	1.0K STARS	10M+ PULLS	DETAILS
ubuntu official	6.1K STARS	10M+ PULLS	DETAILS
registry official	1.5K STARS	10M+ PULLS	DETAILS
alpine official	2.3K STARS	10M+ PULLS	DETAILS
mysql official	4.4K STARS	10M+ PULLS	DETAILS
mongo official	3.3K STARS	10M+ PULLS	DETAILS

https://hub.docker.com/search?q=&type=image

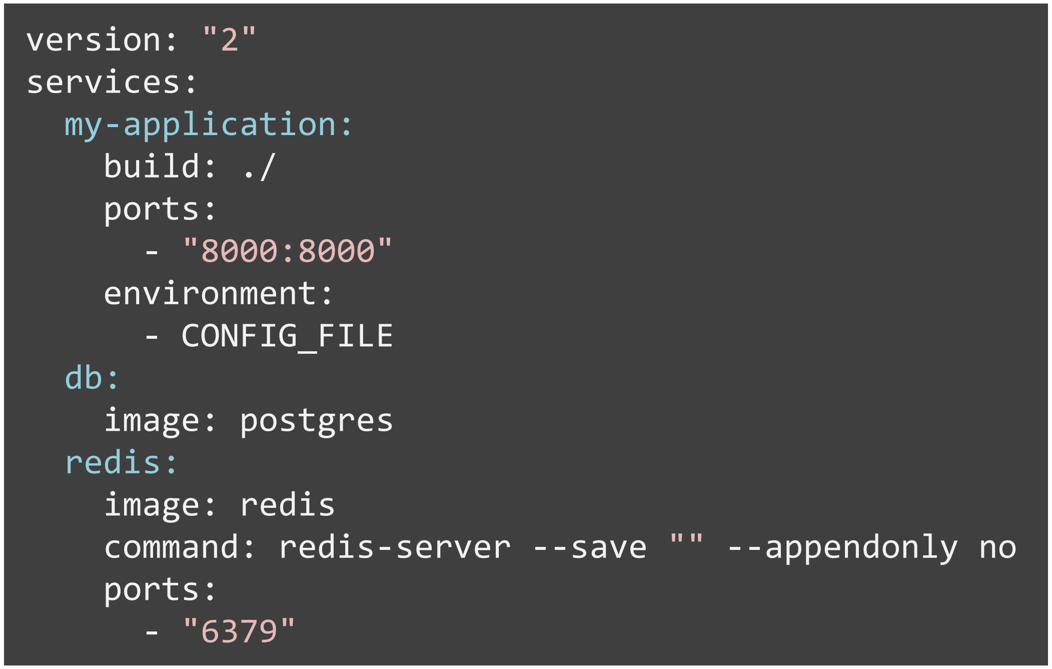
Docker Machine

- A tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands





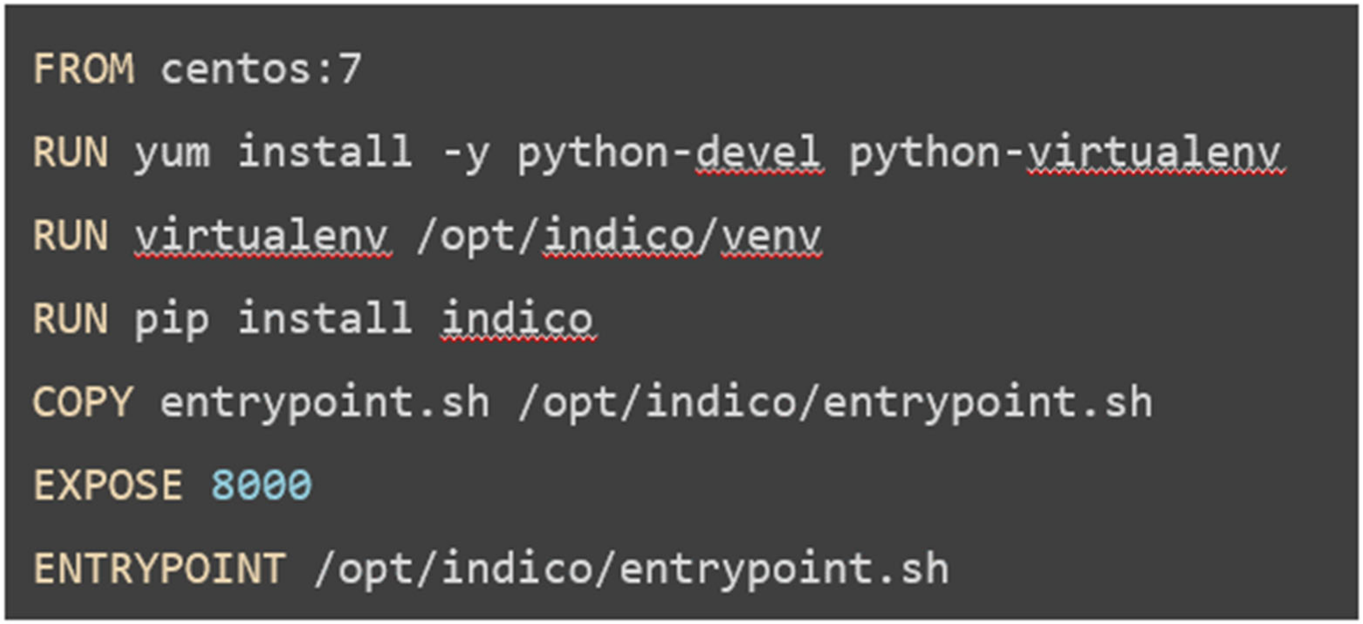
Docker Compose



```yaml
version: "2"
services:
  my-application:
    build: ./
    ports:
      - "8000:8000"
    environment:
      - CONFIG_FILE
  db:
    image: postgres
  redis:
    image: redis
    command: redis-server --save "" --appendonly no
    ports:
      - "6379"
```

Allows to run multi-container Docker applications reading instructions from a docker-compose.yml file

Dockerfile



```
FROM centos:7
RUN yum install -y python-devel python-virtualenv
RUN virtualenv /opt/indico/venv
RUN pip install indico
COPY entrypoint.sh /opt/indico/entrypoint.sh
EXPOSE 8000
ENTRYPOINT /opt/indico/entrypoint.sh
```

- Create images automatically using a build script: «Dockerfile»
- Can be versioned in a version control system like Git or SVN, along with all dependencies
- Docker Hub can automatically build images based on dockerfiles on Github

Docker Networking

Docker Networking supported:

- UDP/TCP port allocation to containers
  - specify *which* public port to redirect. If you don’t specify a public port, Docker will revert to allocating a random public port.
  - Docker uses IPtables/netfilter
- IP allocation to containers
  - Docker uses virtual interfaces, network bridge

Docker Basics

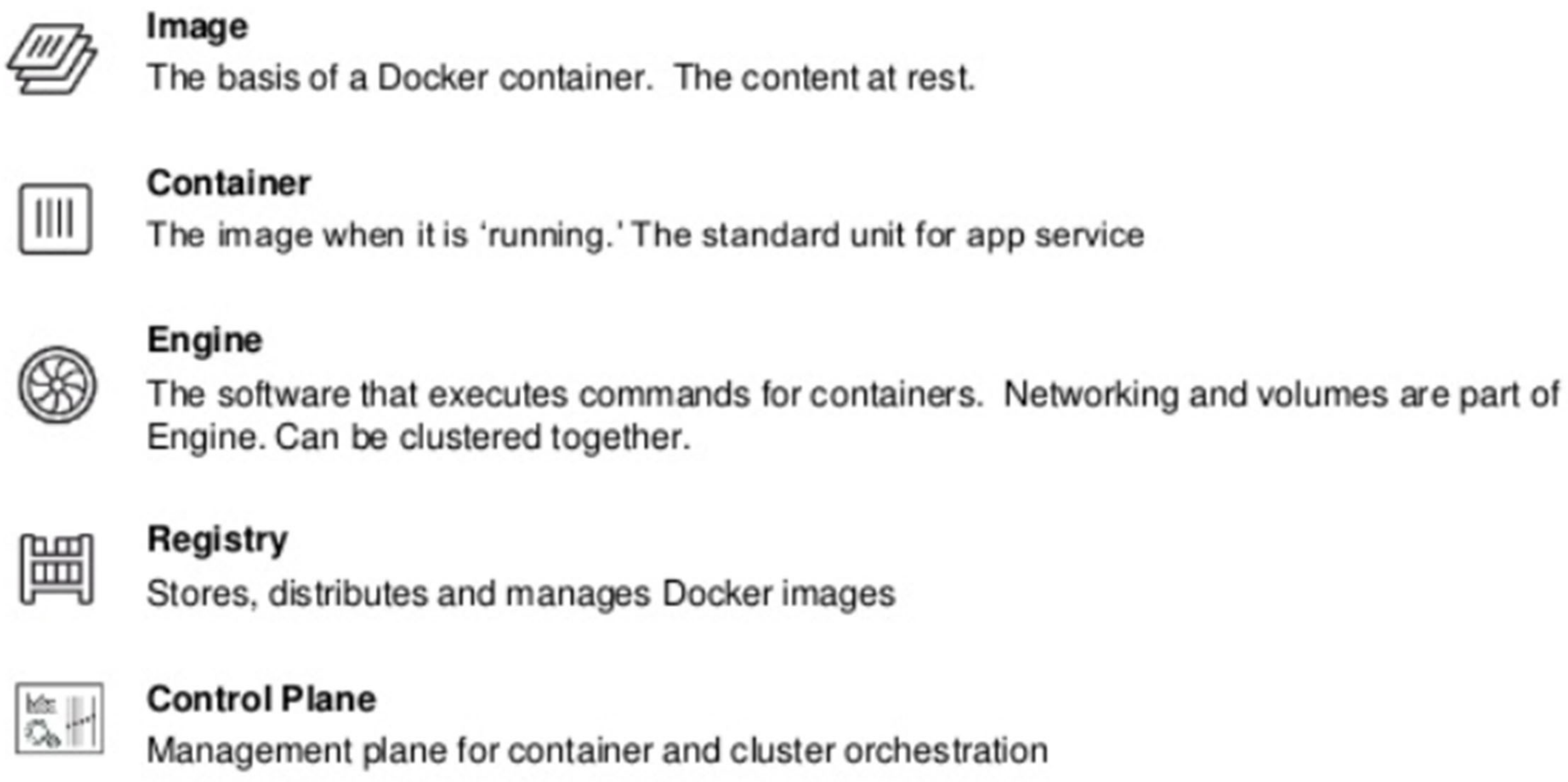


Image
The basis of a Docker container. The content at rest.

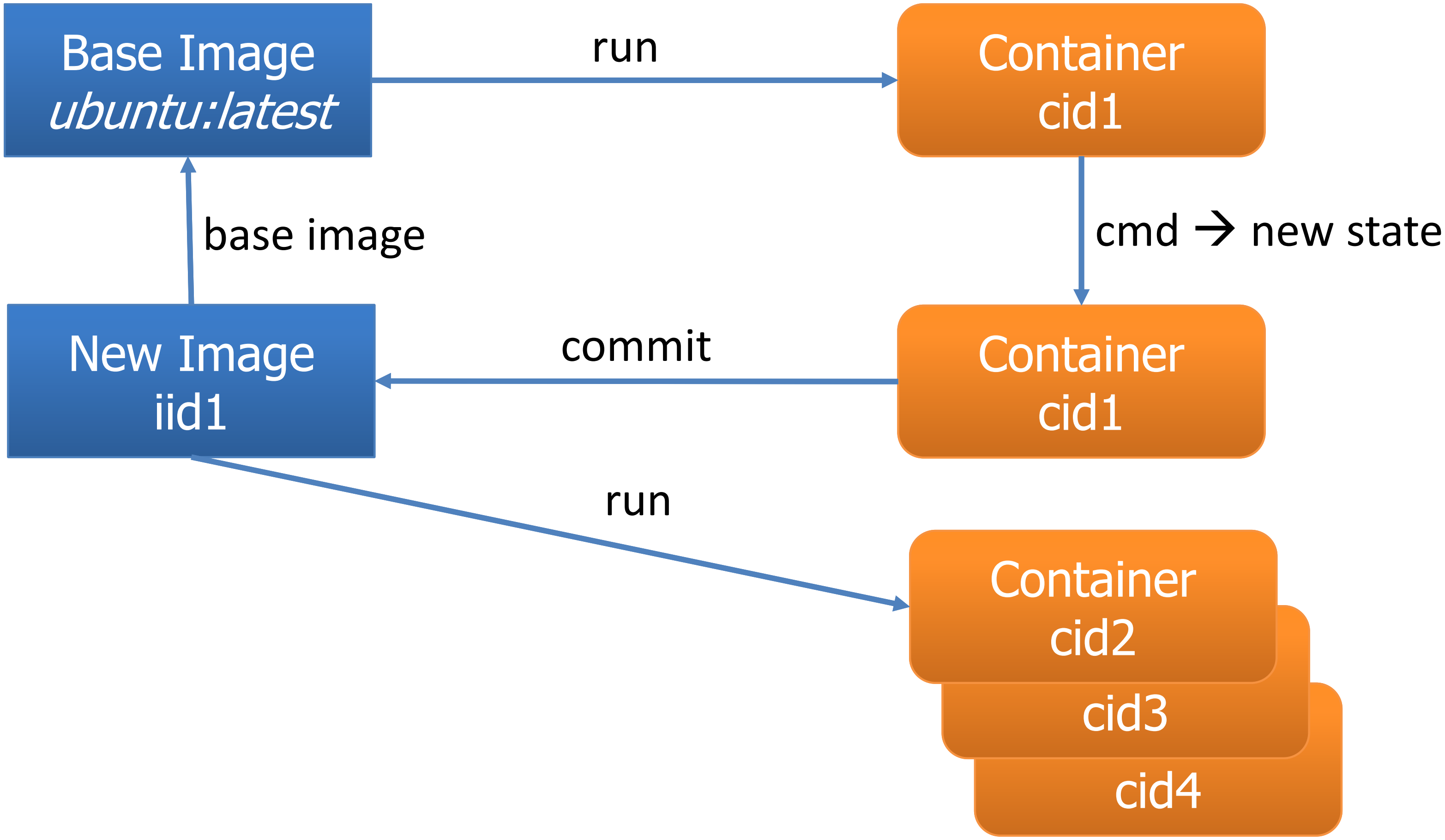
Container
The image when it is 'running.' The standard unit for app service

Engine
The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry
Stores, distributes and manages Docker images

Control Plane
Management plane for container and cluster orchestration

Docker Image vs Container



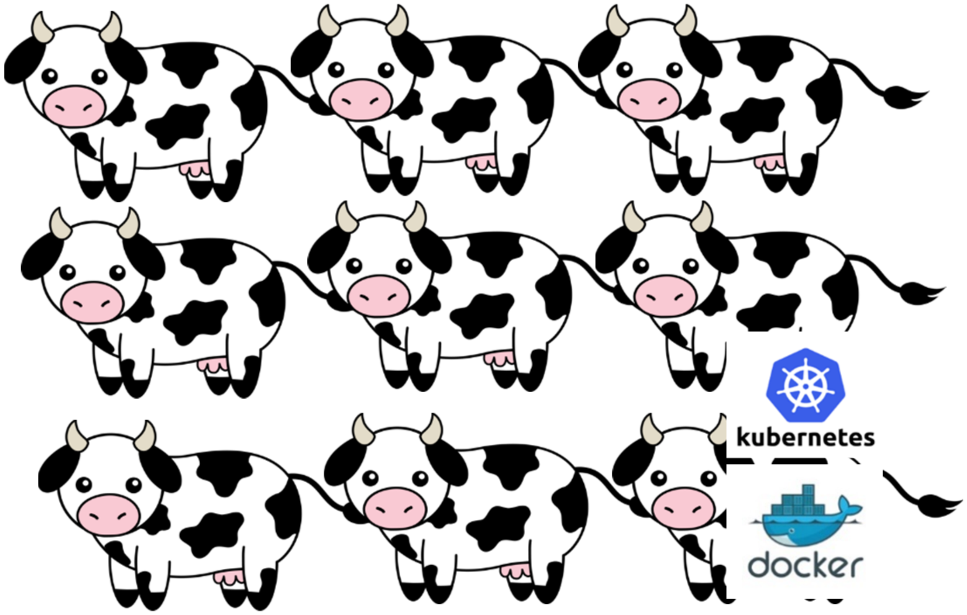
Outline

- Monoliths
- Service Oriented Architectures
- MicroServices
- Containerization
- **Orchestration**

Things Docker can’t do by itself

- monitor running containers
- handle dead containers
- move containers so utilization improves
- autoscale container instances to handle load

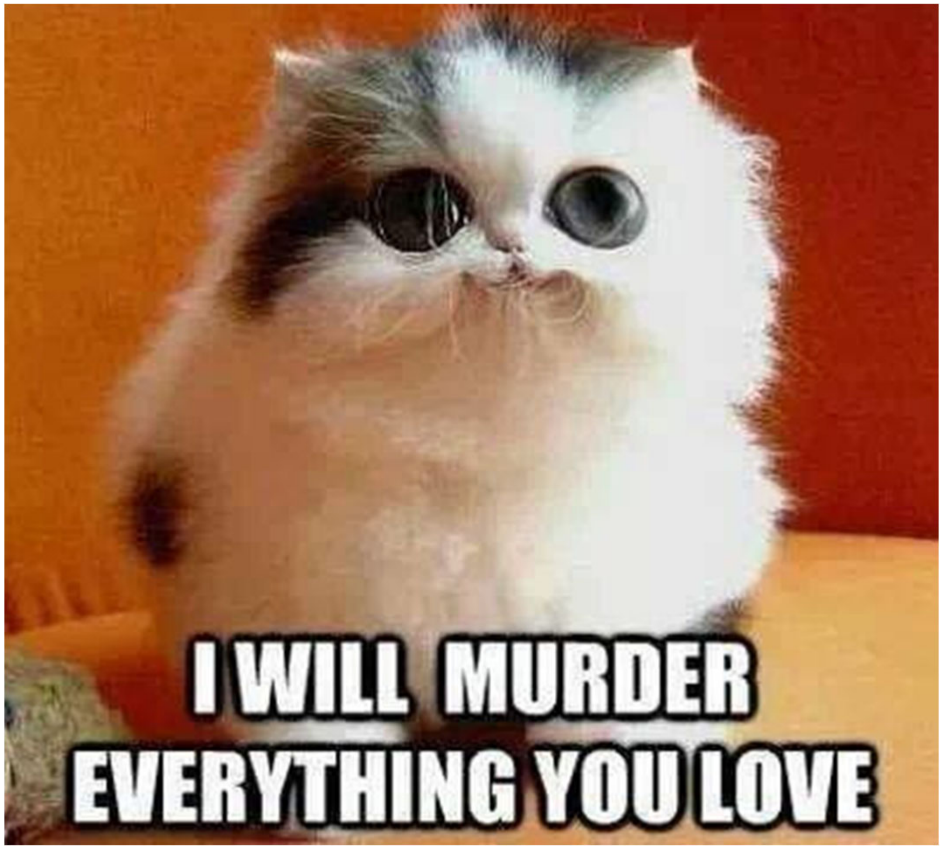
Pets vs Cattle



- Long-lived
- Care for them
- Name them

- Ephemeral
- Brand them with #’s
- Well…vets are expensive

Pets vs Cattle



You can never get away from pets unless:

- You handle the problem of container state
- You need an environment to support cattle

Orchestration is the solution

Outline

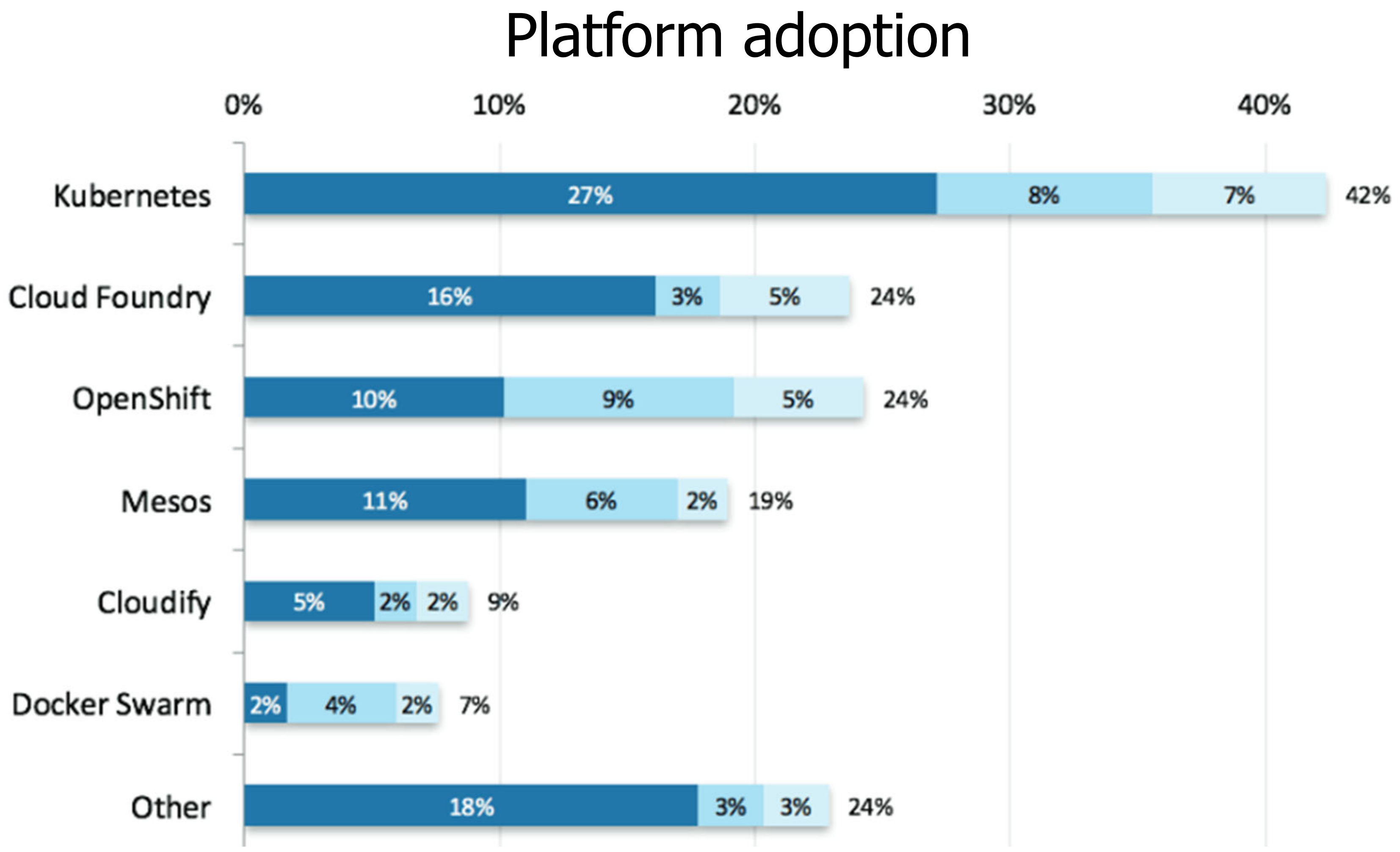
- Monoliths
- Service Oriented Architectures
- MicroServices
- Containerization
- **Orchestration**
- **Kubernetes**

What is Kubernetes?

- Kubernetes - greek word for pilot or helm
- Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications
- Kubernetes started life as a successor to Google’s Borg project

Iceland Liechtenstein Norway grants

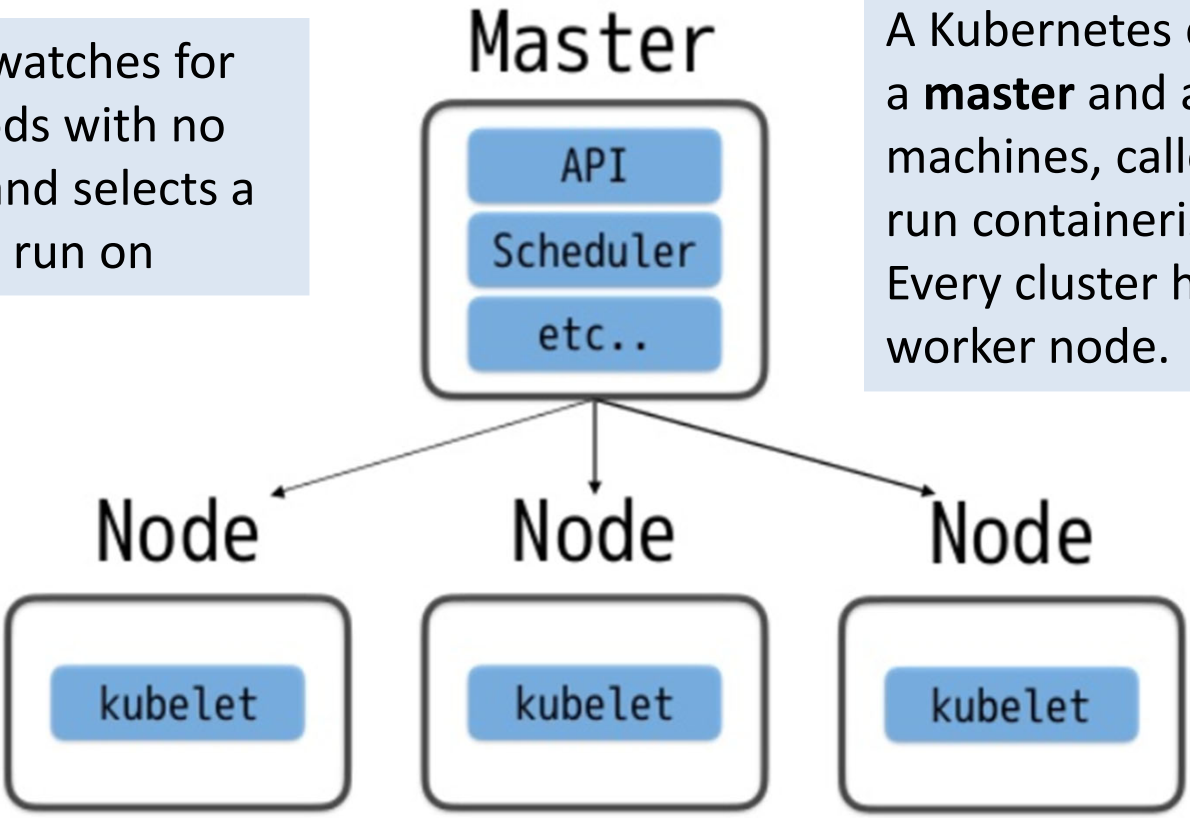
Kubernetes won the orchestration war



Basic Kubernetes concepts

Nodes = hosts running k8s daemons

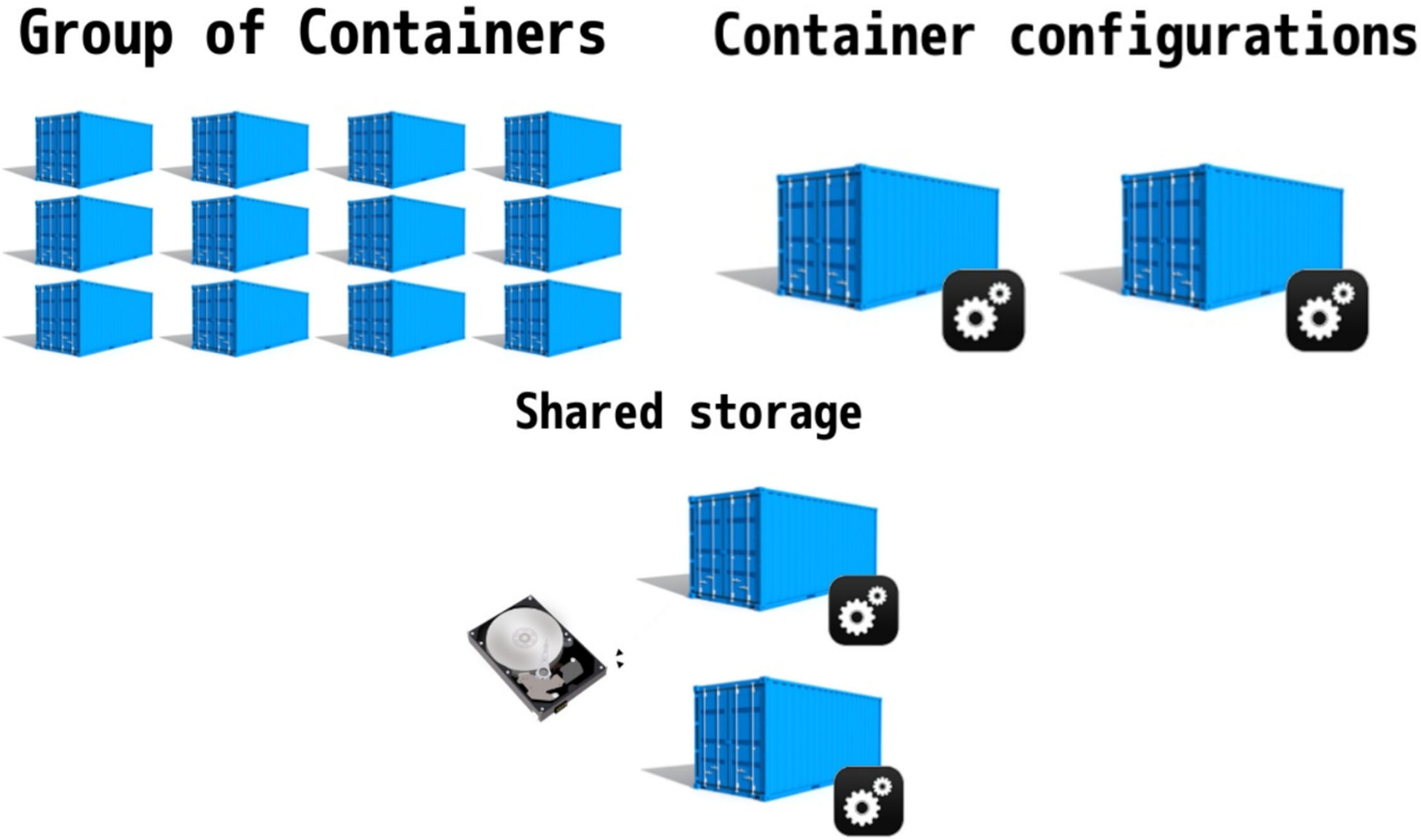
Kube-scheduler watches for newly created Pods with no assigned node , and selects a node for them to run on



A Kubernetes cluster consists of a **master** and a set of worker machines, called **nodes** , that run containerized applications. Every cluster has at least one worker node.

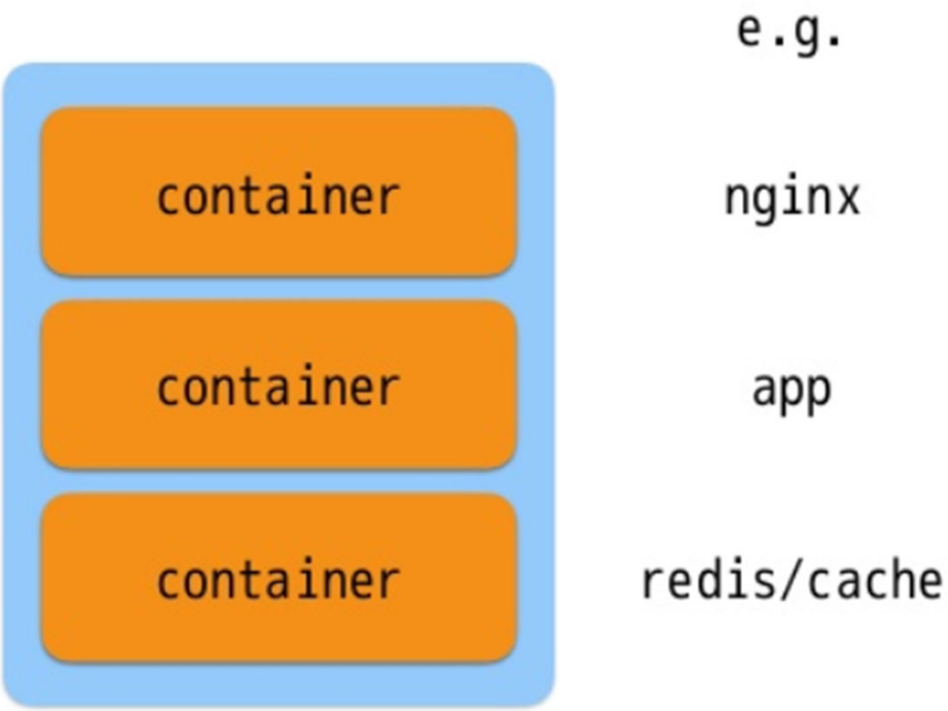
Kubelet is an agent that runs on each node in the cluster. It makes sure that containers are running in a Pod

Basic Kubernetes concepts



Basic Kubernetes concepts

Pod = basic deployment unit in k8s



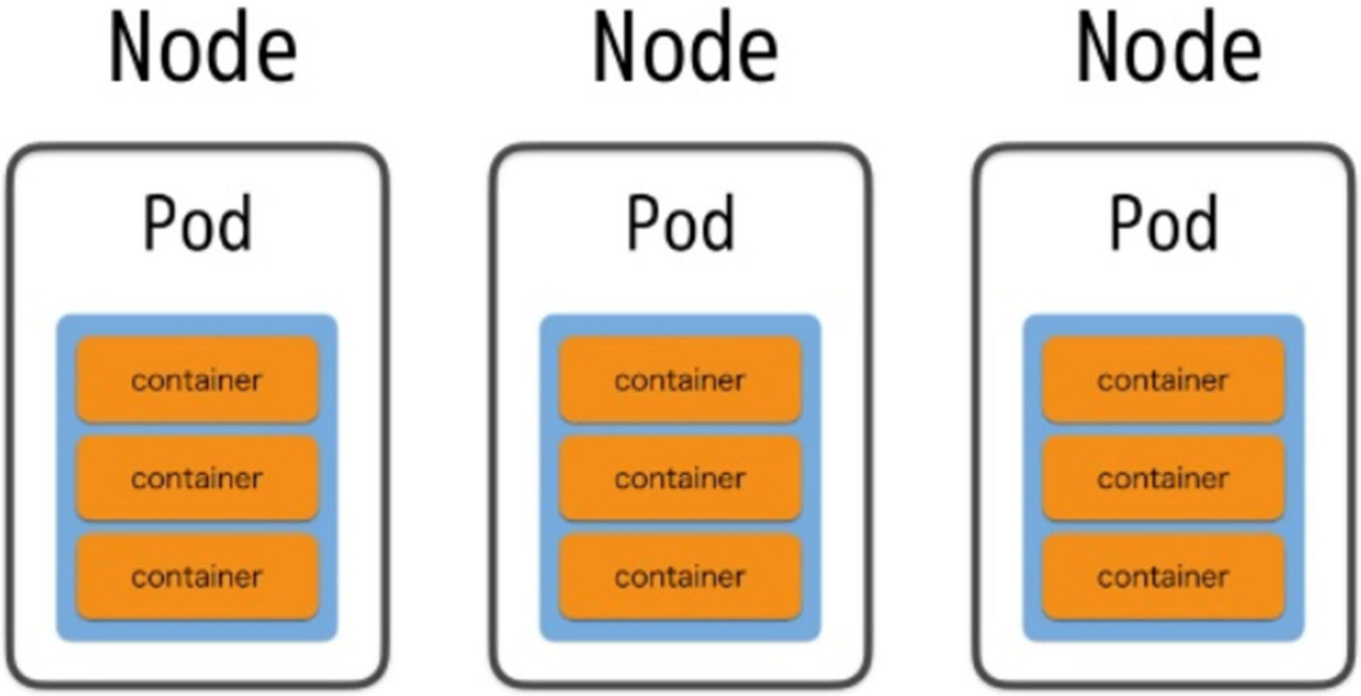
A Pod represents a set of running containers

Basic Kubernetes concepts

Pod = basic deployment unit in k8s

Containers are:

- Scheduled together (“co-scheduled”)
- Guaranteed to be on the same node (“co-located”)



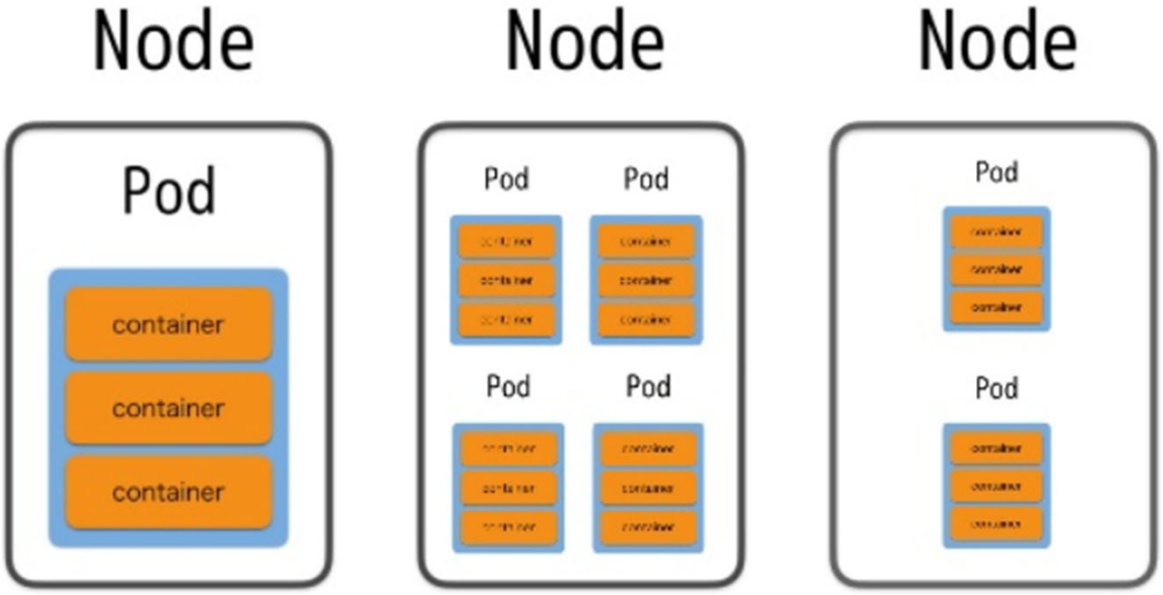
Iceland Liechtenstein Norway grants

Basic Kubernetes concepts

Pod = basic deployment unit in k8s

Containers are:

- Scheduled together (“co-scheduled”)
- Guaranteed to be on the same node (“co-located”)



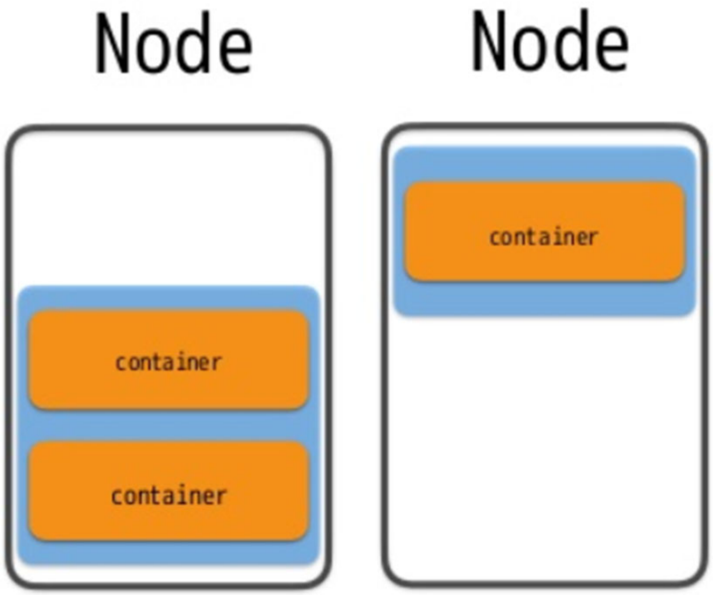
Depends on each node’s resource availability and each pod’s resource requirements

Basic Kubernetes concepts

Pod = basic deployment unit in k8s

Containers are:

- Scheduled together ("co-scheduled")
- Guaranteed to be on the same node ("co-located")



This will never happen !

Basic Kubernetes concepts

Pod = basic deployment unit in k8s

Containers are:

- Scheduled together (“co-scheduled”)
- Guaranteed to be on the same node (“co-located”)

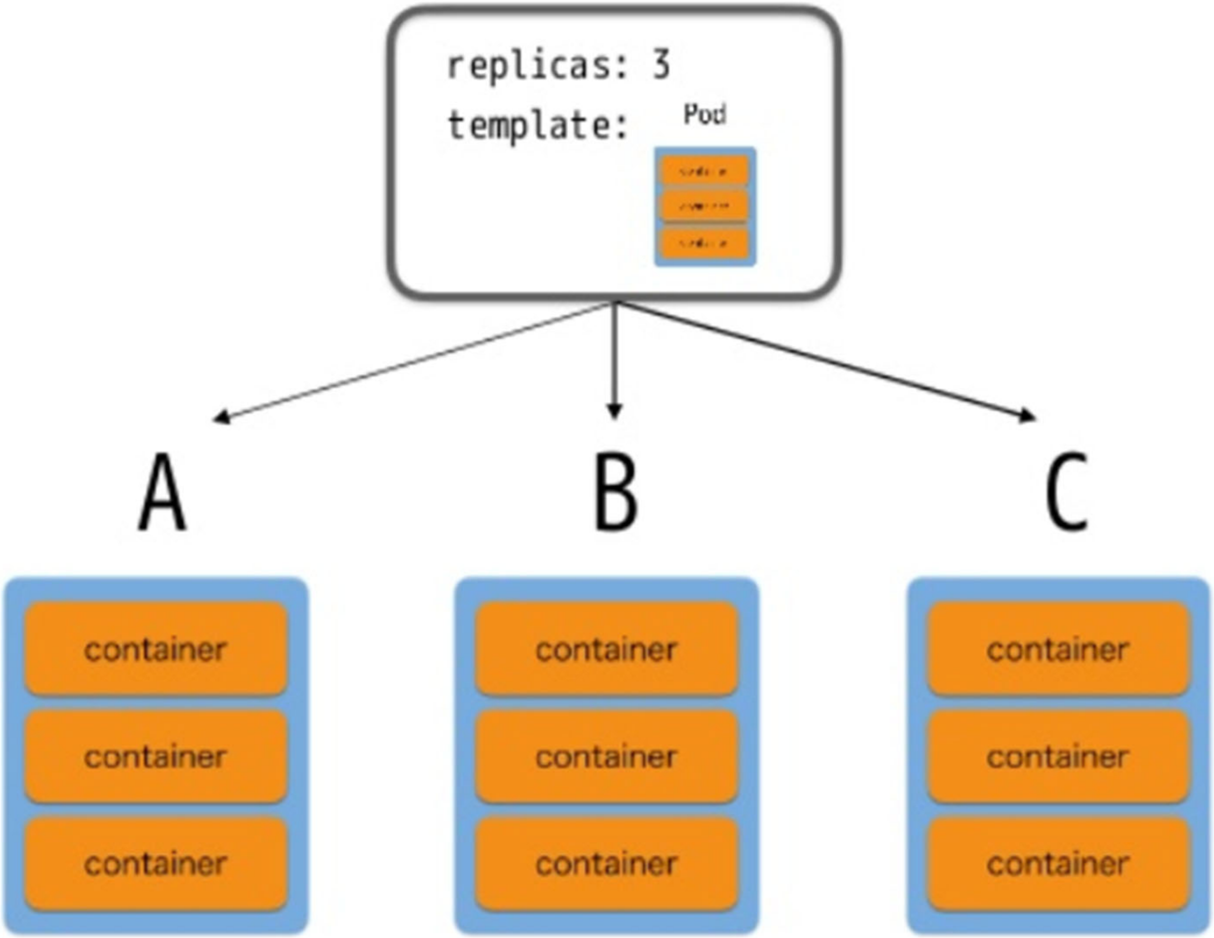
Pods:

- Each pod has its own IP address
- Pods are expected to be stateless

Iceland Liechtenstein Norway grants

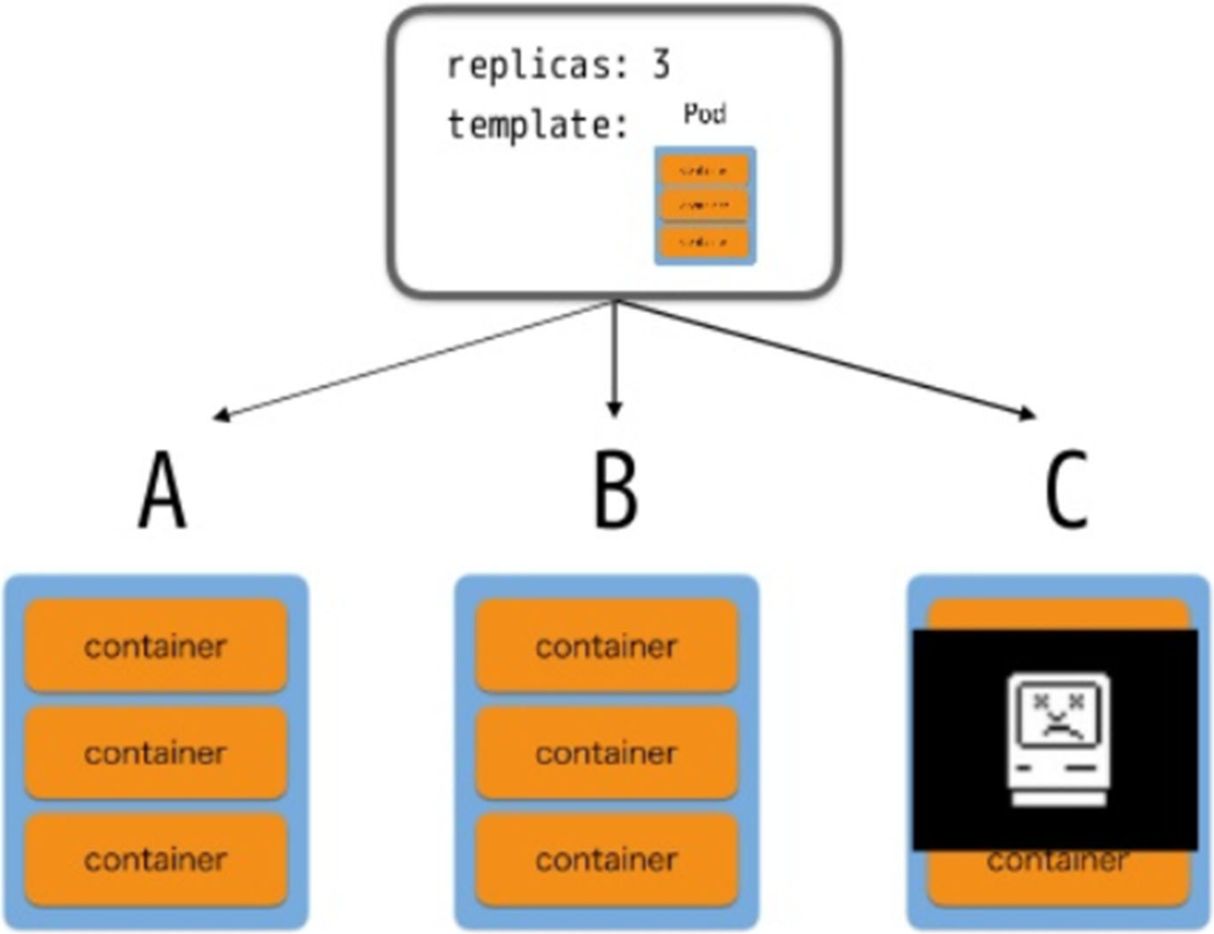
Basic Kubernetes concepts

Replica set = keeps track of Pod replicas



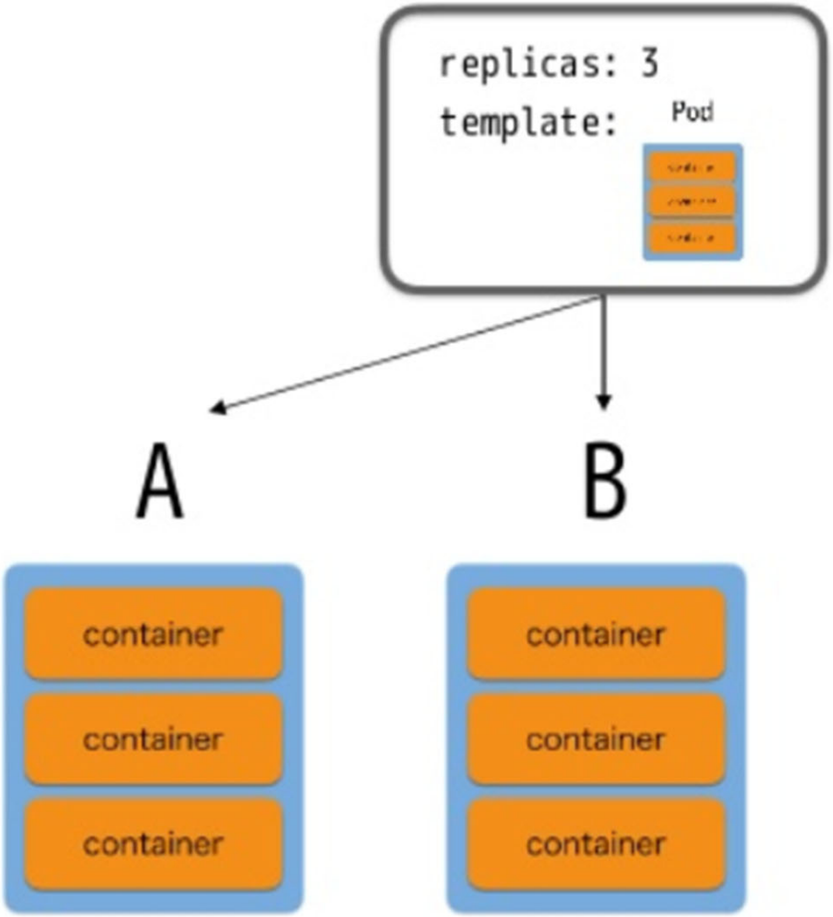
Basic Kubernetes concepts

Replica set = keeps track of Pod replicas



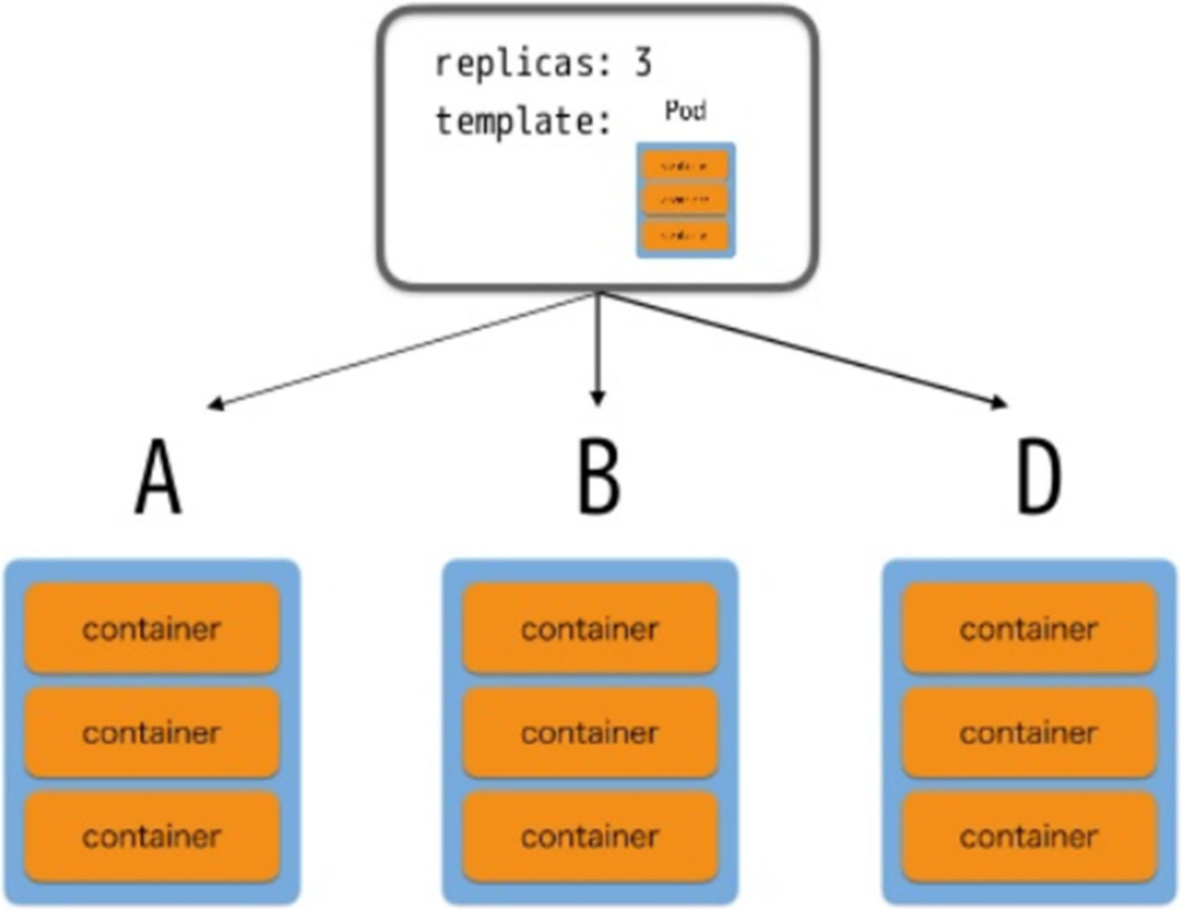
Basic Kubernetes concepts

Replica set = keeps track of Pod replicas



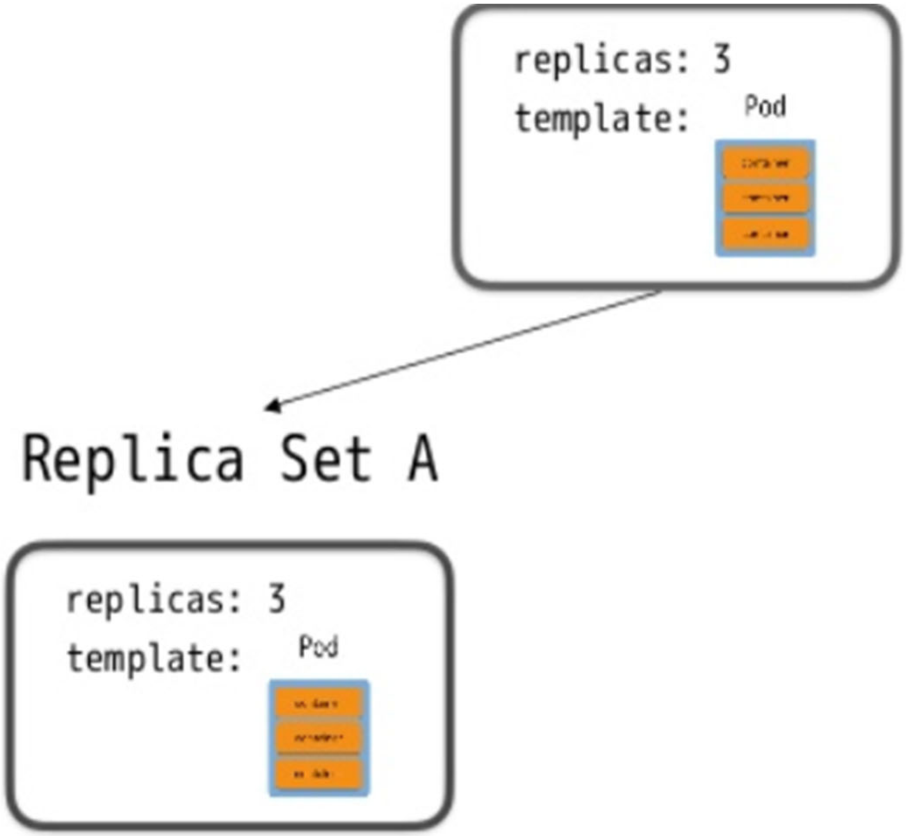
Basic Kubernetes concepts

Replica set = keeps track of Pod replicas



Basic Kubernetes concepts

Deployment = manages Replica Set state transitions



replicas: 3
template: Pod

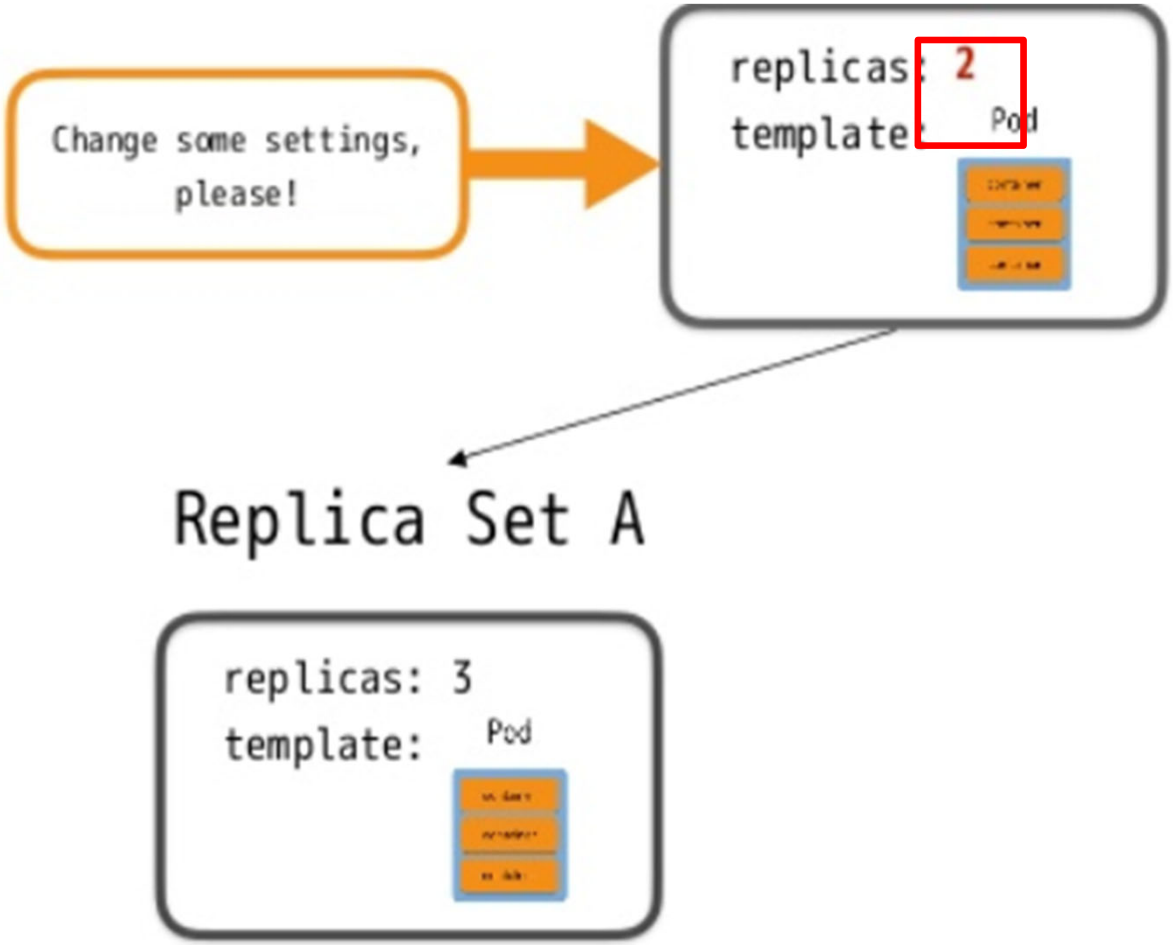
Replica Set A

replicas: 3
template: Pod

A Deployment provides declarative updates for Pods and Replica Sets.

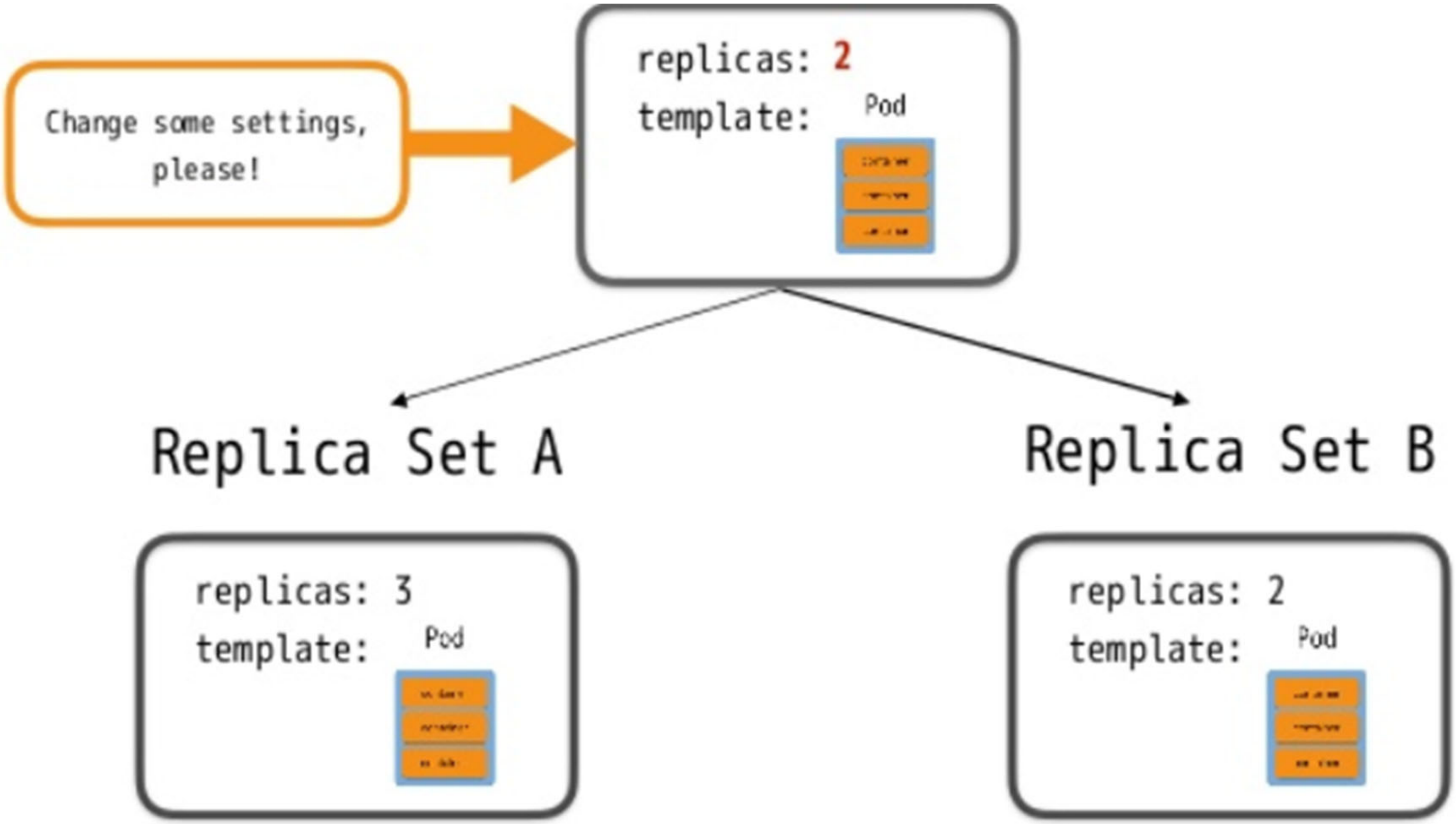
Basic Kubernetes concepts

Deployment = manages Replica Set state transitions



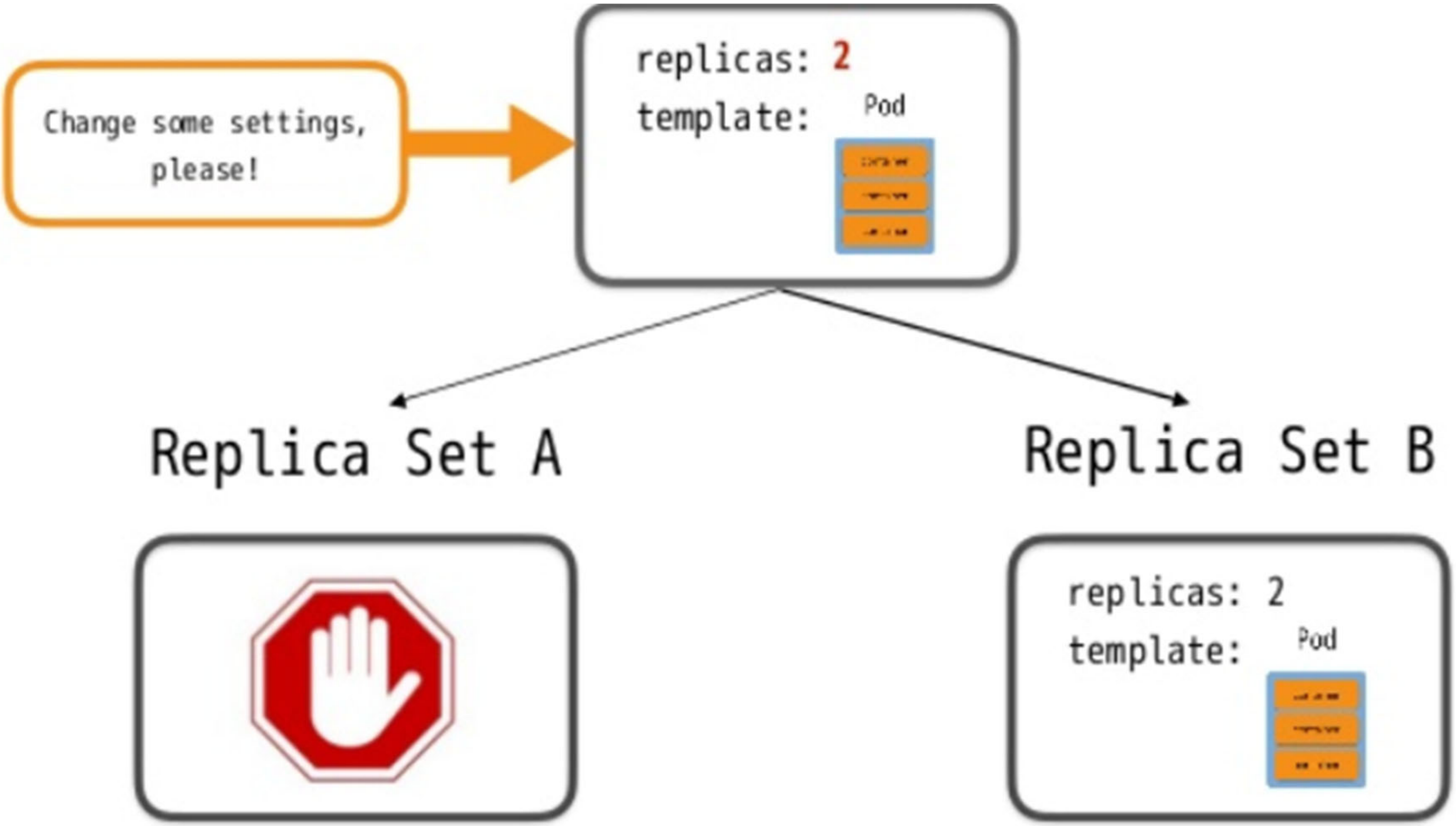
Basic Kubernetes concepts

Deployment = manages Replica Set state transitions



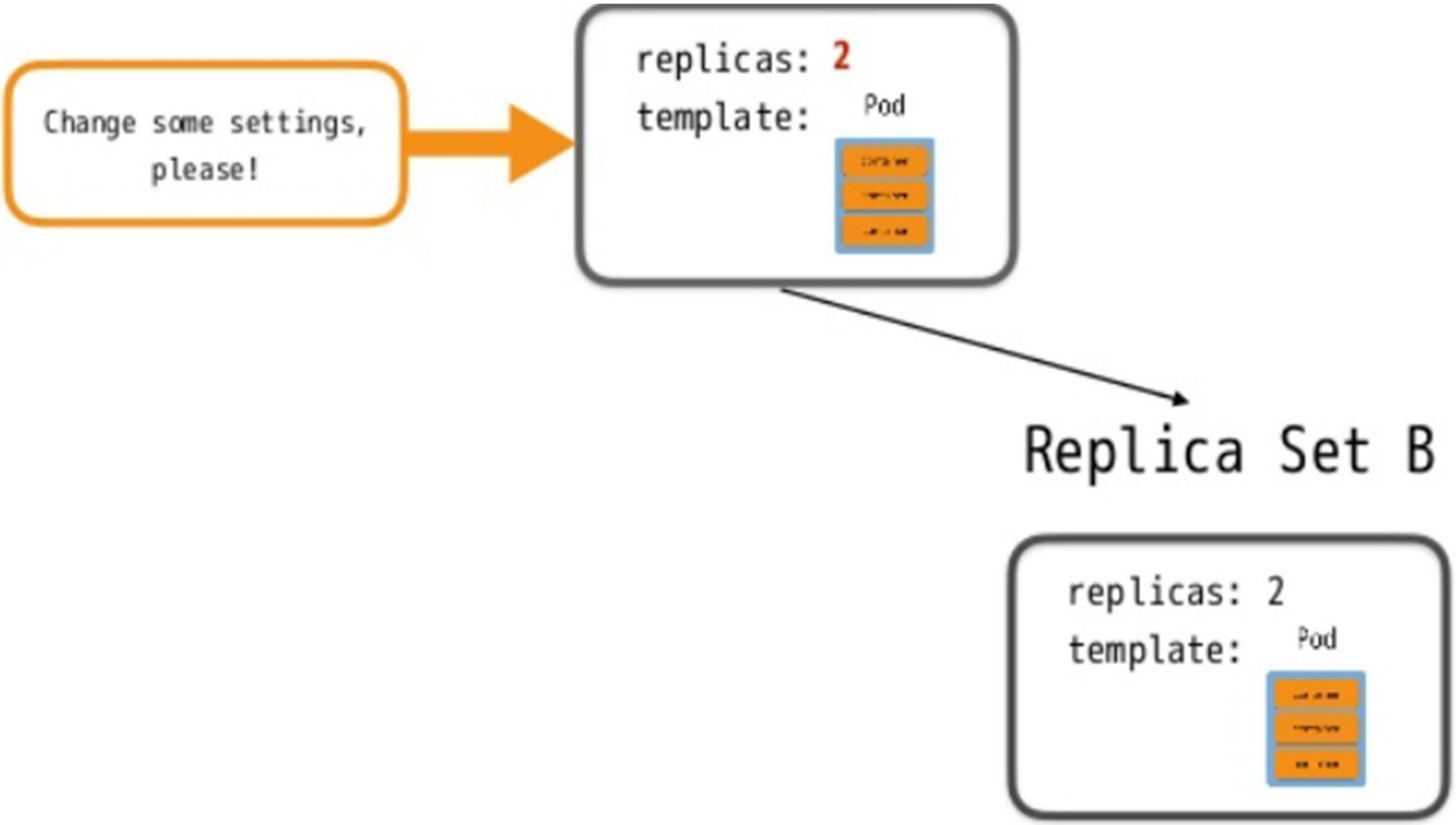
Basic Kubernetes concepts

Deployment = manages Replica Set state transitions



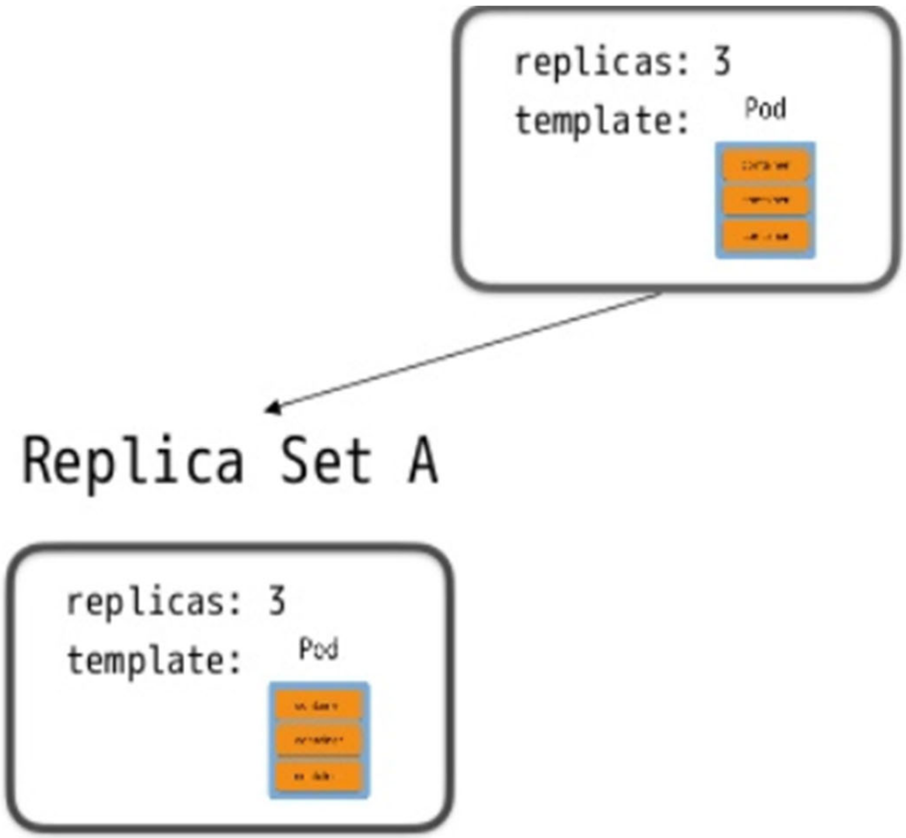
Basic Kubernetes concepts

Deployment = manages Replica Set state transitions



Basic Kubernetes concepts

Deployment = keeps track of state change history



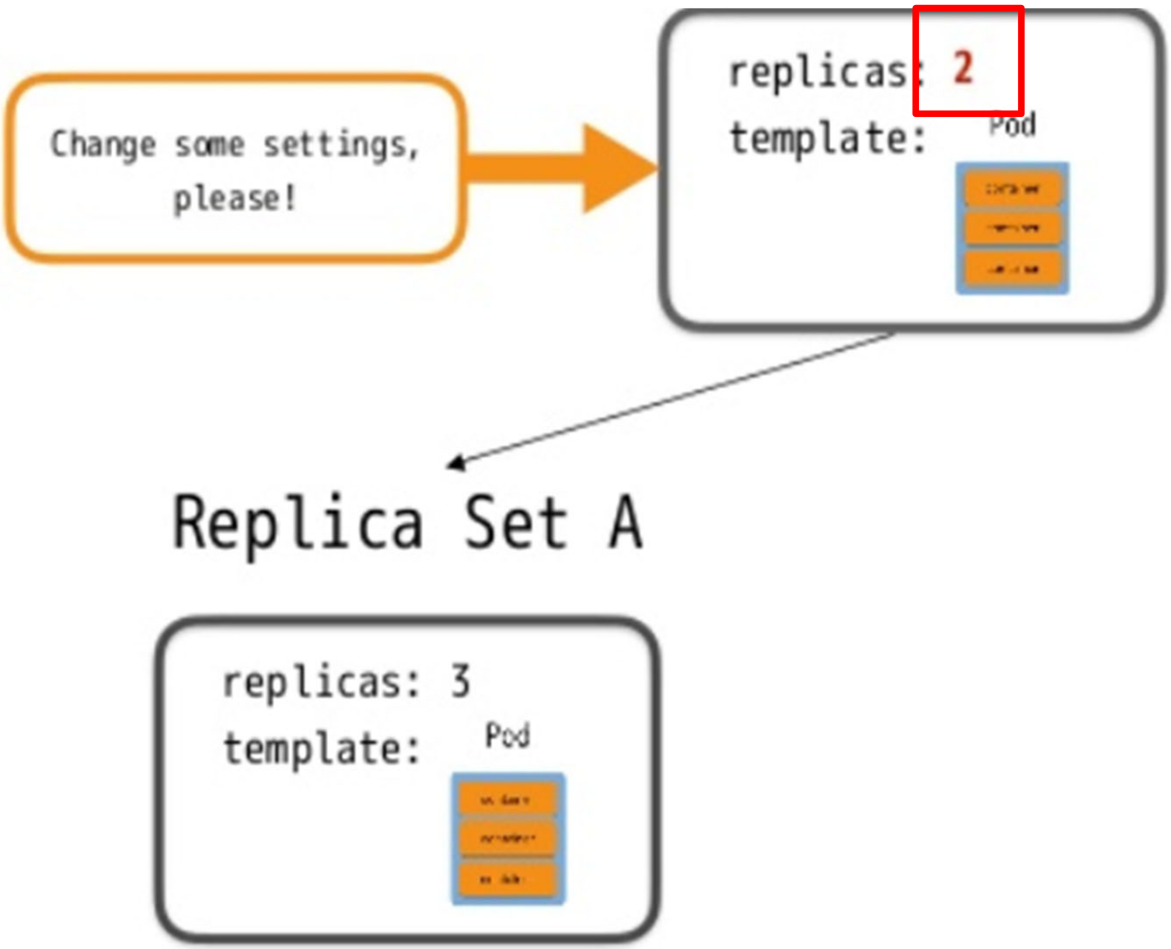
replicas: 3
template: Pod

Replica Set A

replicas: 3
template: Pod

Basic Kubernetes concepts

Deployment = keeps track of state change history



Change some settings, please!

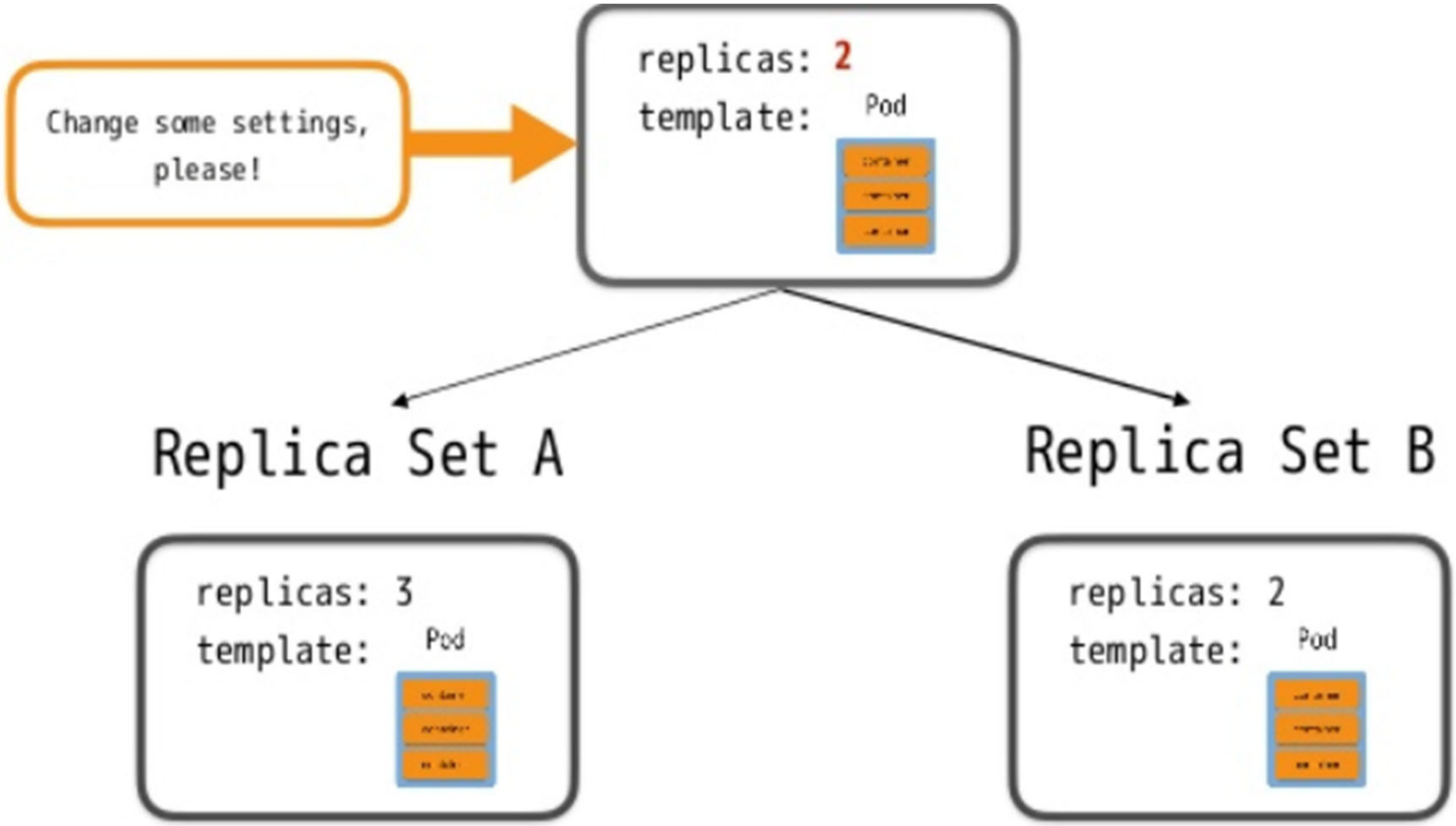
replicas: 2
template: Pod

Replica Set A

replicas: 3
template: Pod

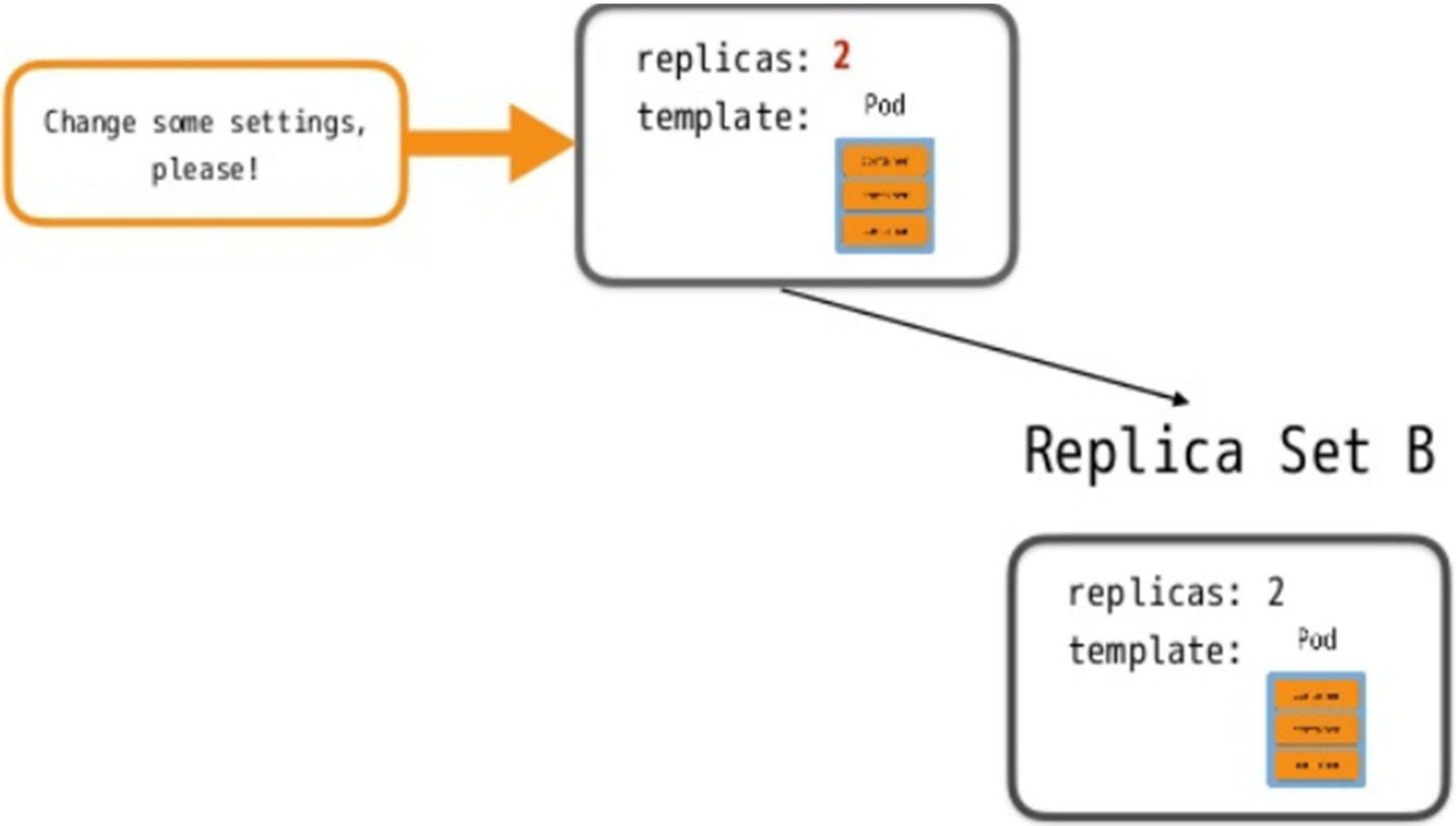
Basic Kubernetes concepts

Deployment = keeps track of state change history



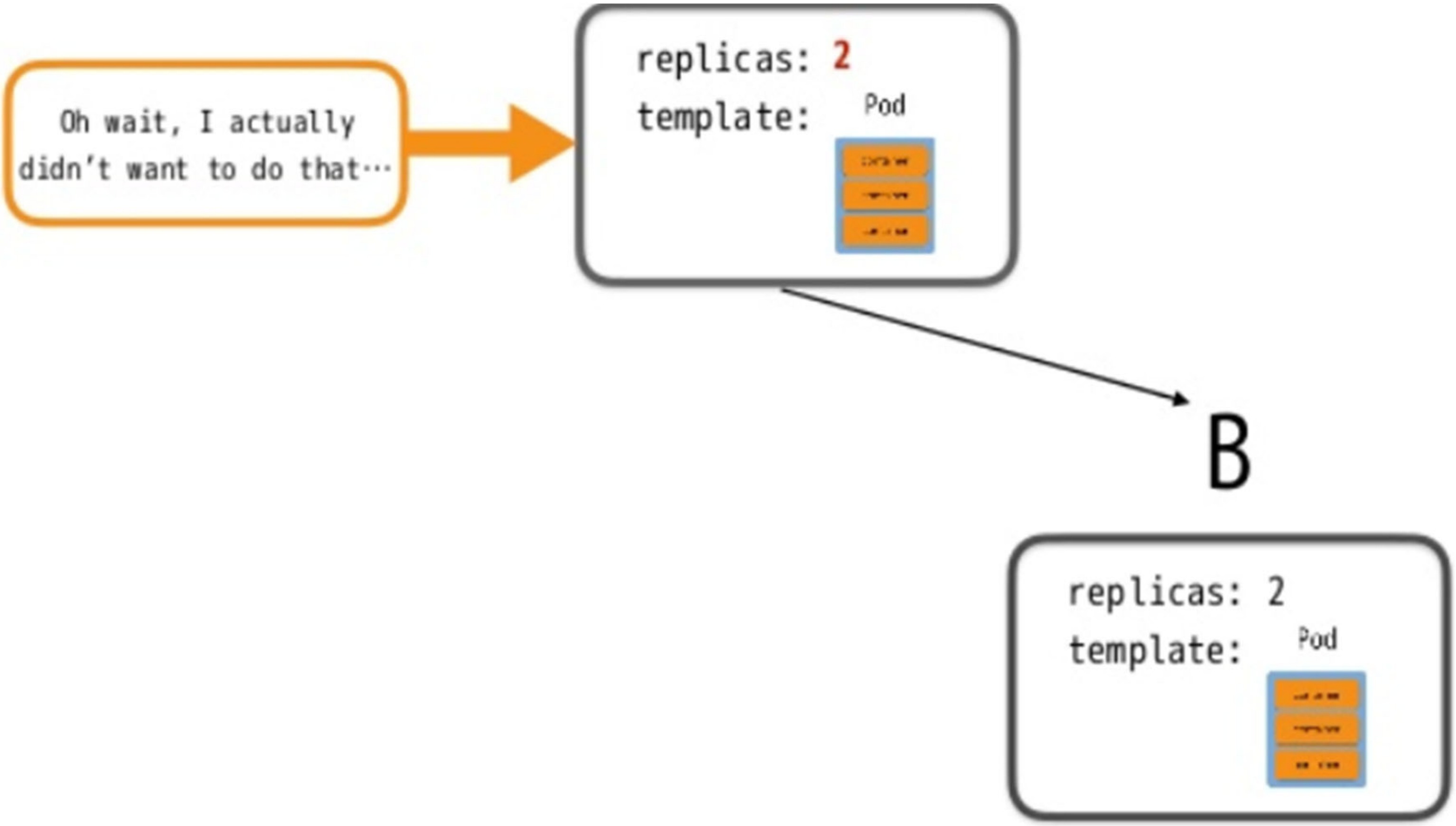
Basic Kubernetes concepts

Deployment = keeps track of state change history



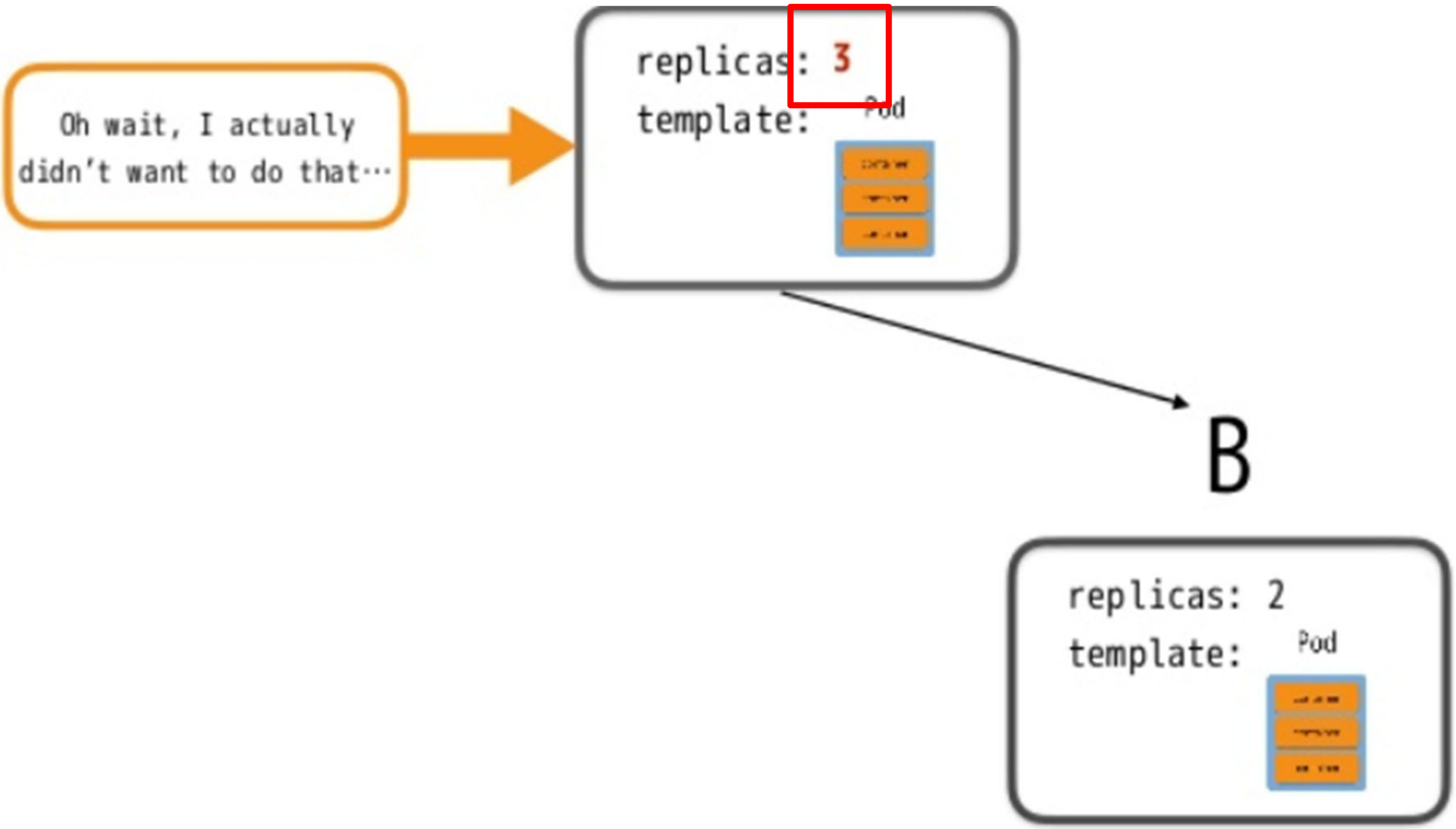
Basic Kubernetes concepts

Deployment = keeps track of state change history



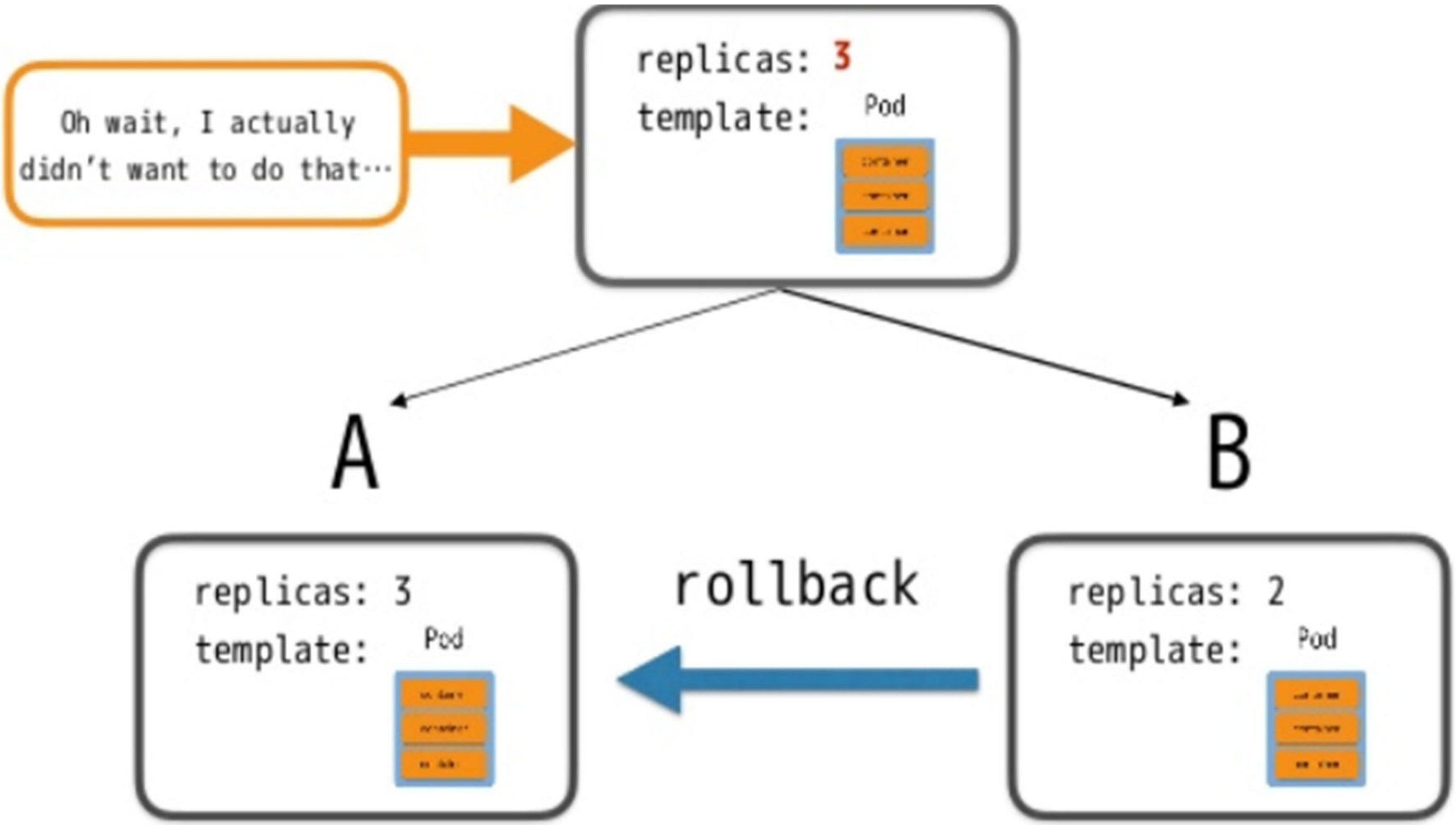
Basic Kubernetes concepts

Deployment = keeps track of state change history



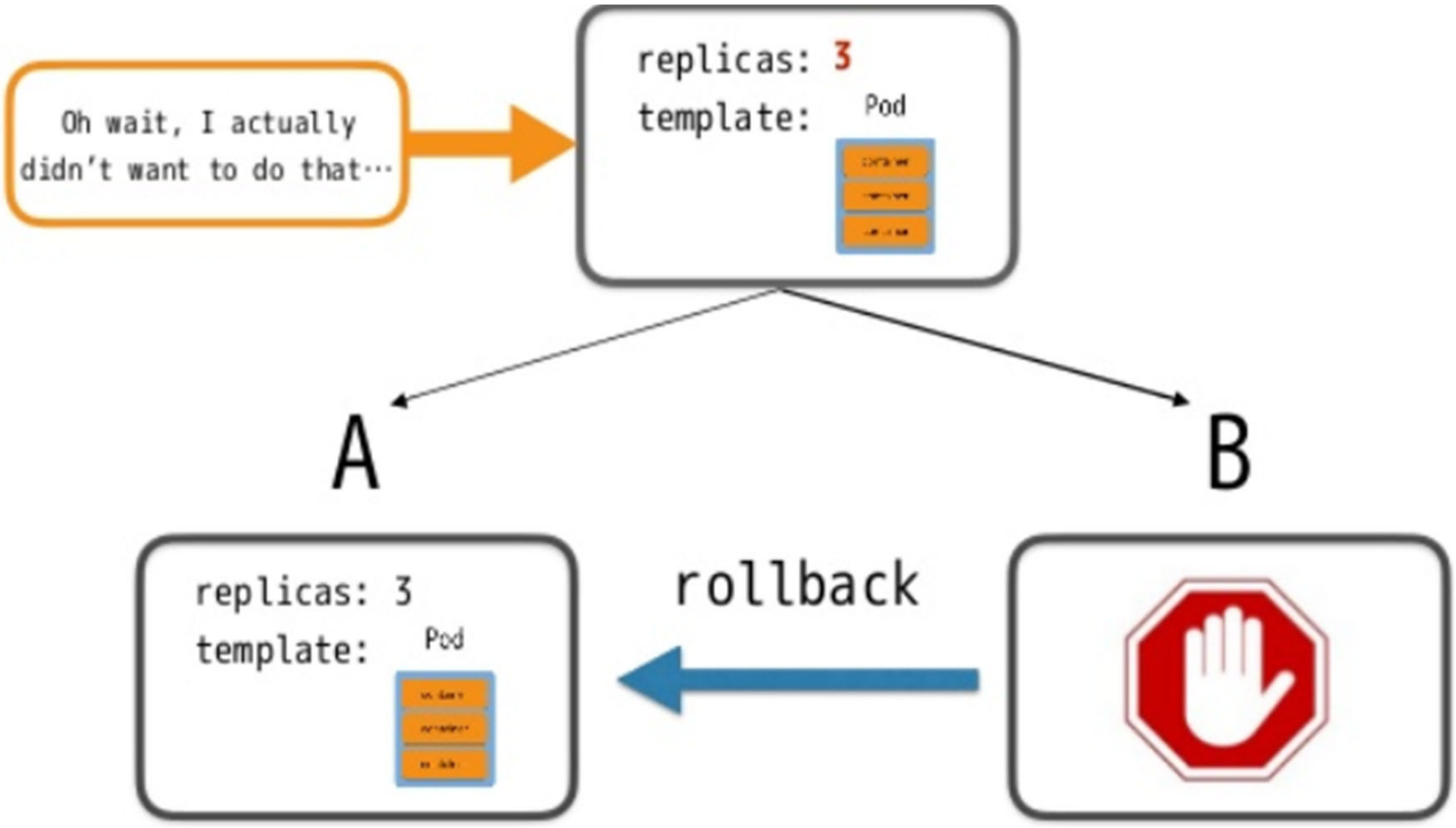
Basic Kubernetes concepts

Deployment = keeps track of state change history



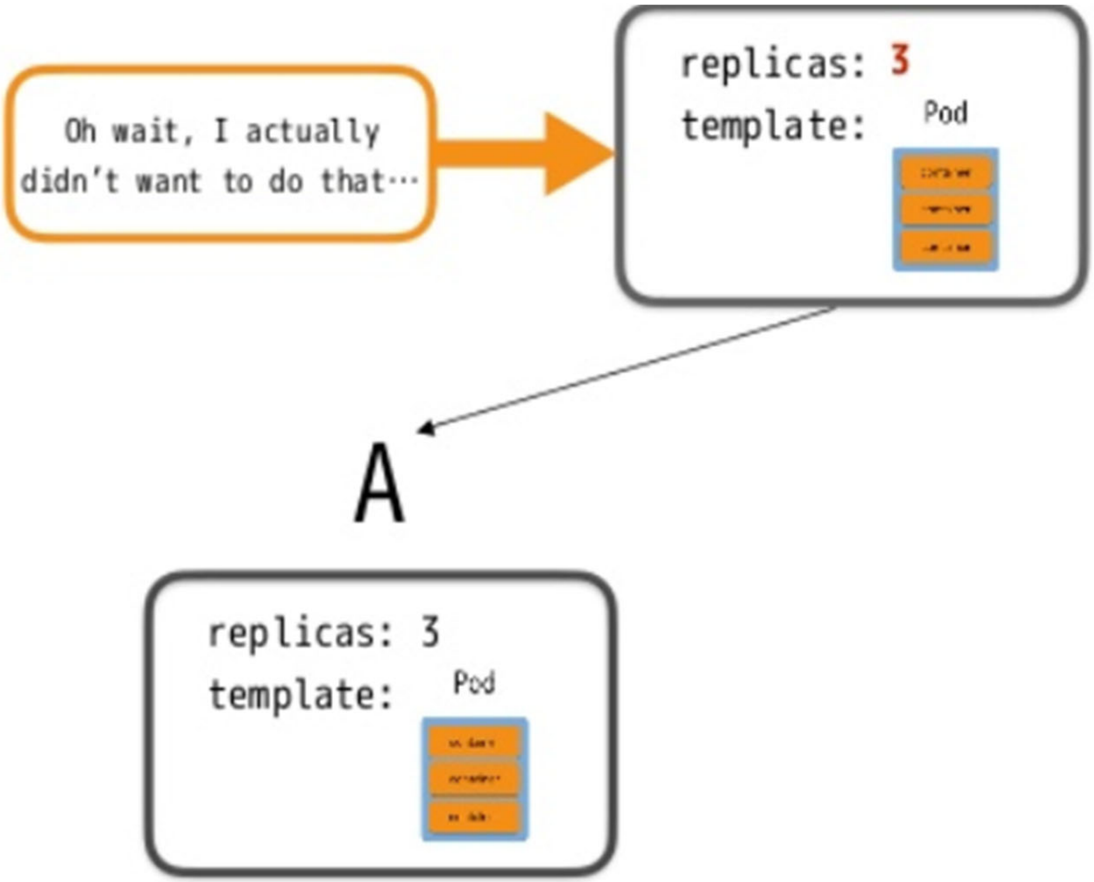
Basic Kubernetes concepts

Deployment = keeps track of state change history



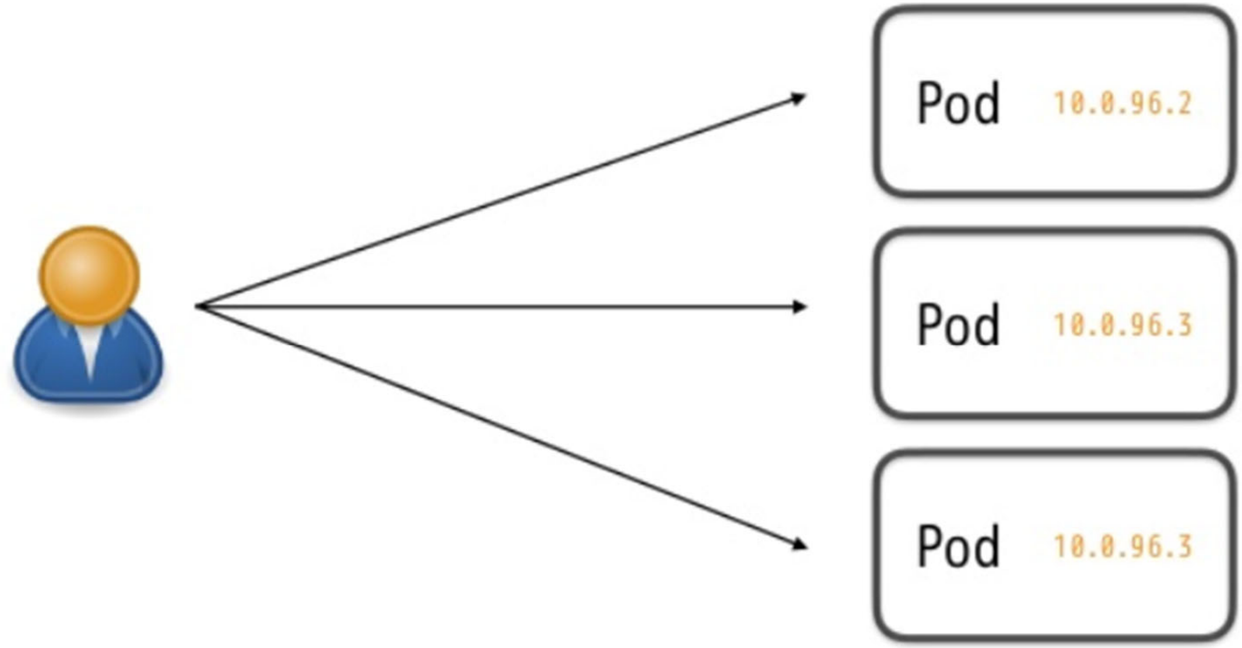
Basic Kubernetes concepts

Deployment = keeps track of state change history



Basic Kubernetes concepts

Services = logical set of pods (and ways to access them)

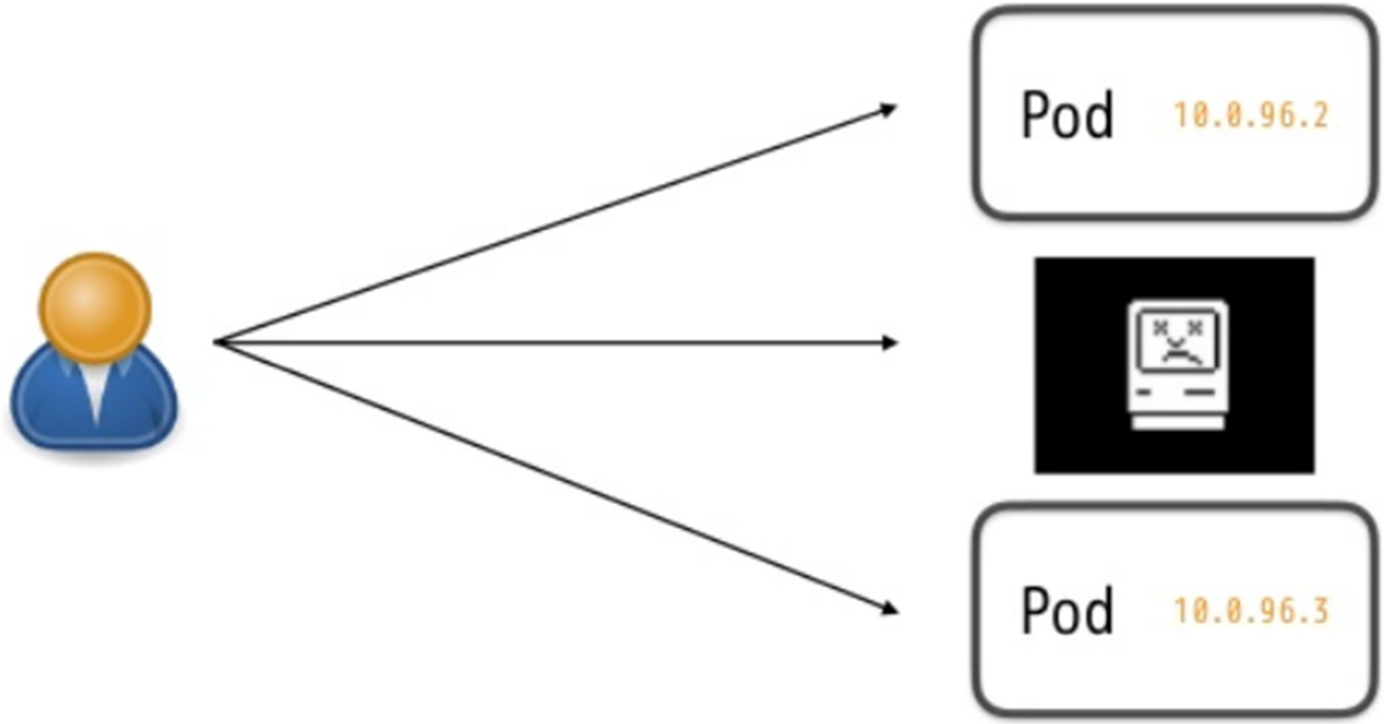


Raw Pod access

Service is an abstract way to expose an application running on a set of Pods as a network service.

Basic Kubernetes concepts

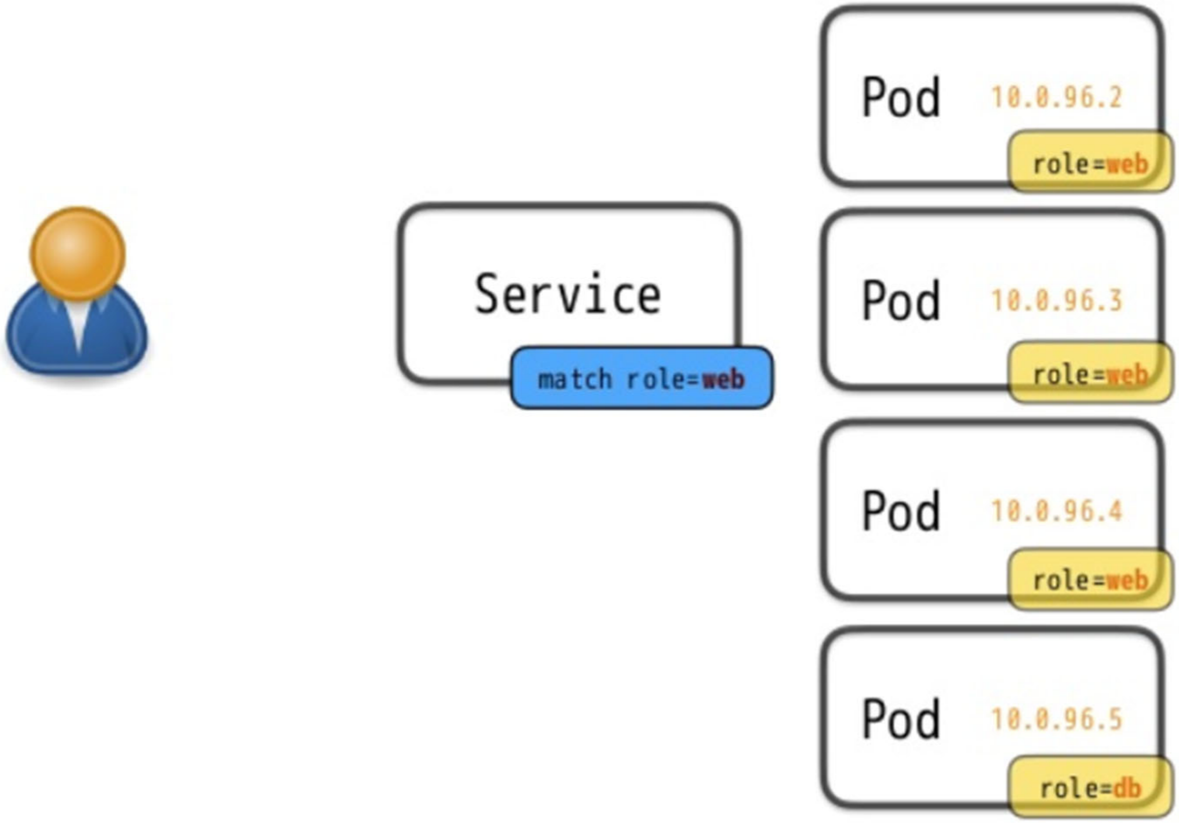
Services = logical set of pods (and ways to access them)



Doesn’t work anymore if a Pod breaks down…

Basic Kubernetes concepts

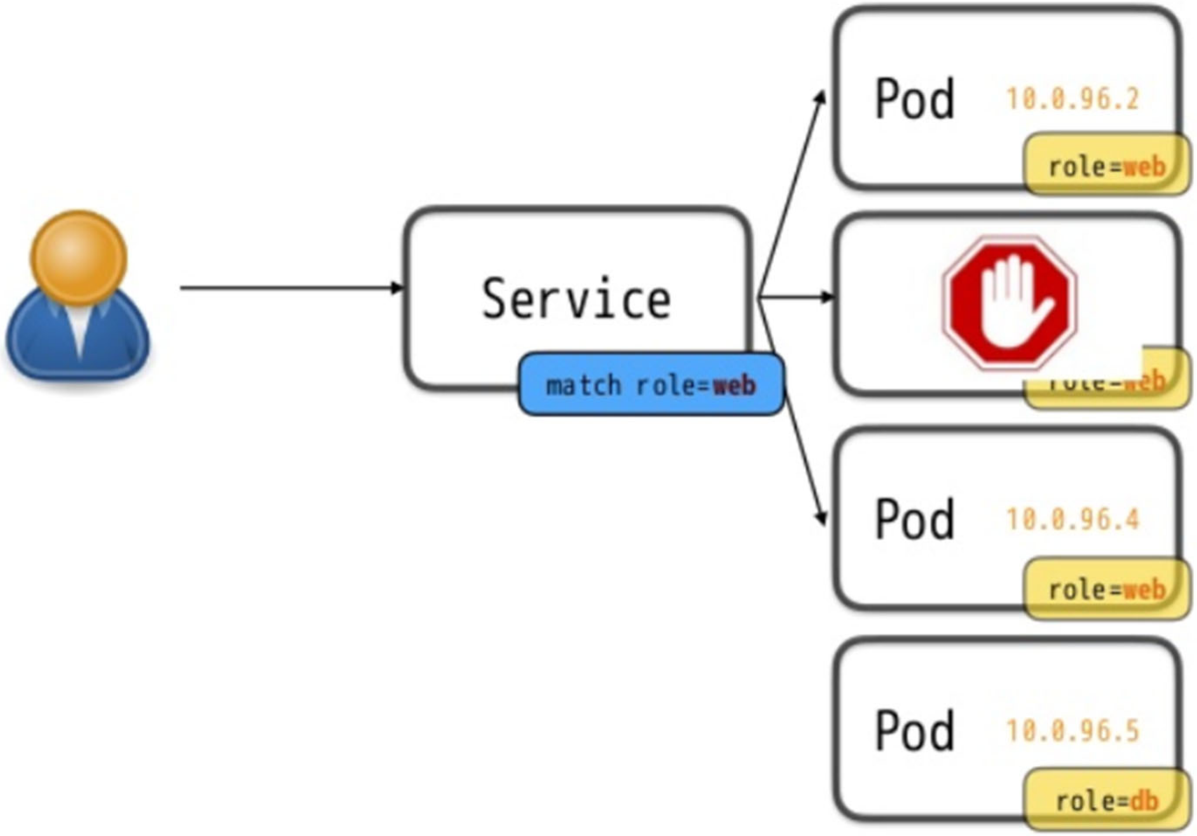
Services = logical set of pods (and ways to acces



Access via Service

Basic Kubernetes concepts

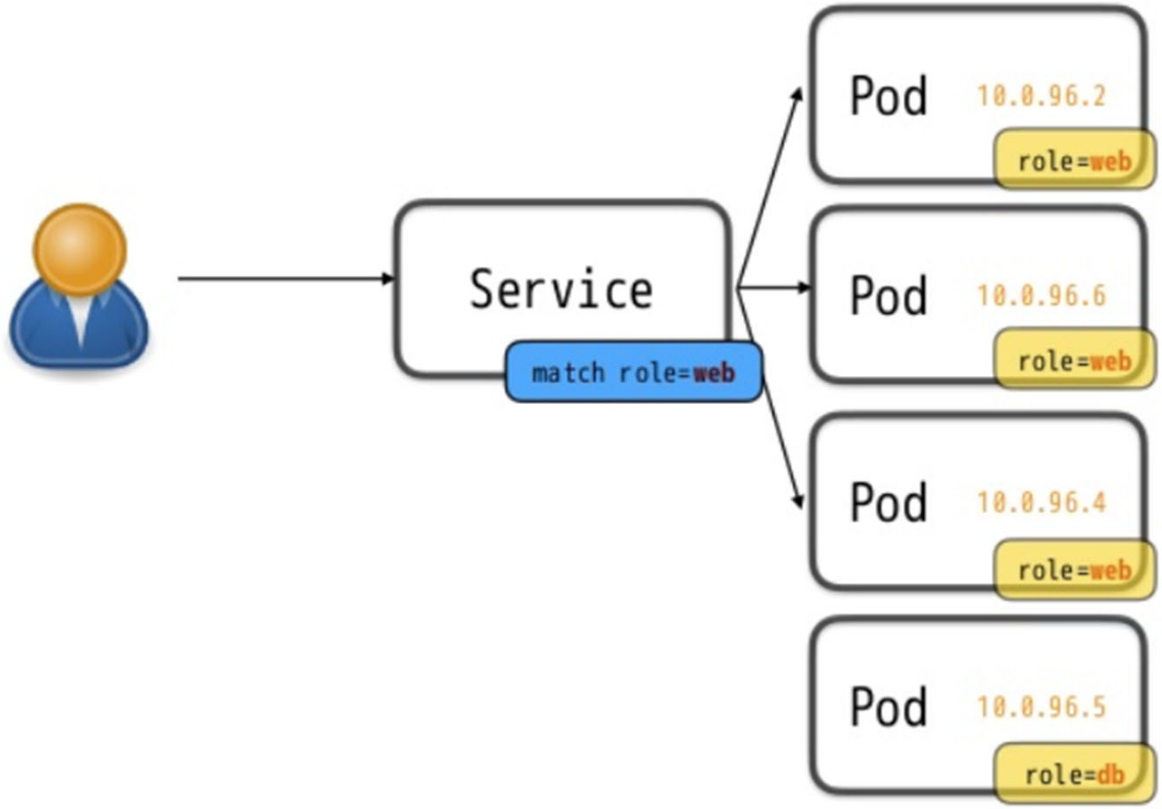
Services = logical set of pods (and ways to access them)



Now if a Pod breaks down…

Basic Kubernetes concepts

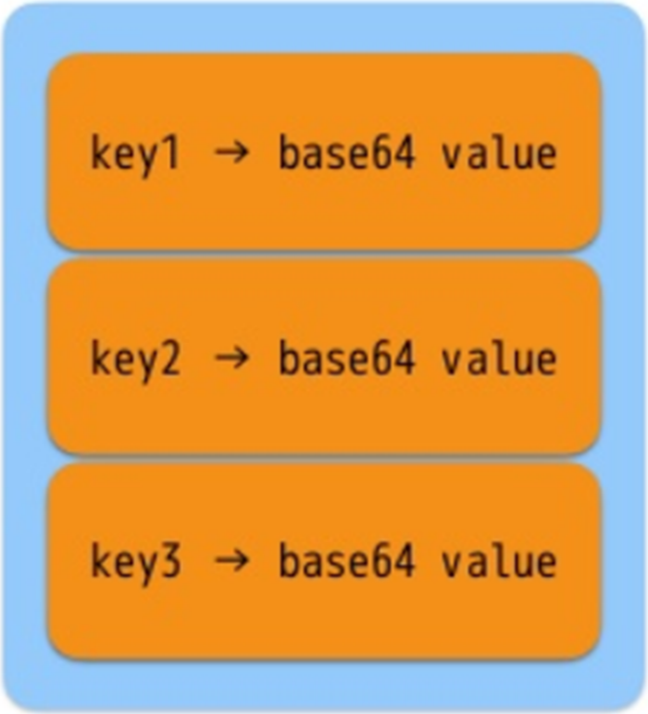
Services = logical set of pods (and ways to access them



Can transparently replace it

Basic Kubernetes concepts

Secrets = store pieces of data in k8s



key1 → base64 value

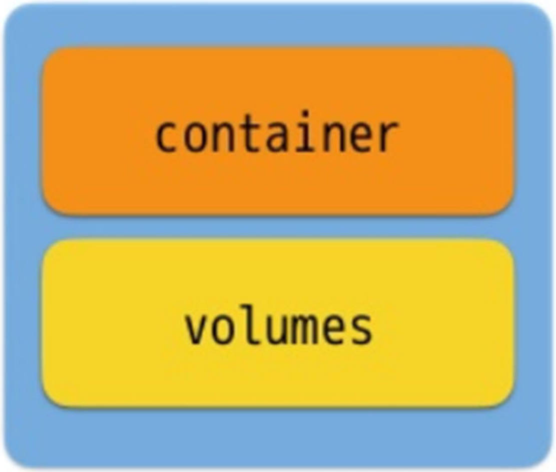
key2 → base64 value

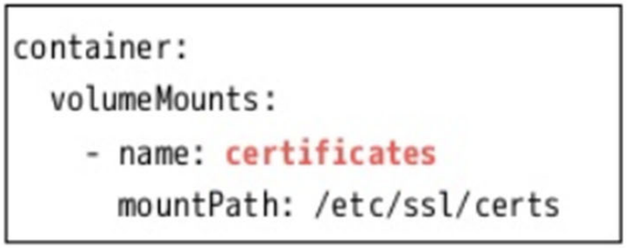
key3 → base64 value

- Kubernetes Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- Storing confidential information in a Secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image.

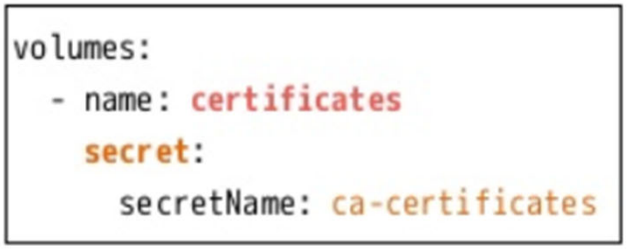
Basic Kubernetes concepts

Secrets = store pieces of data in k8s





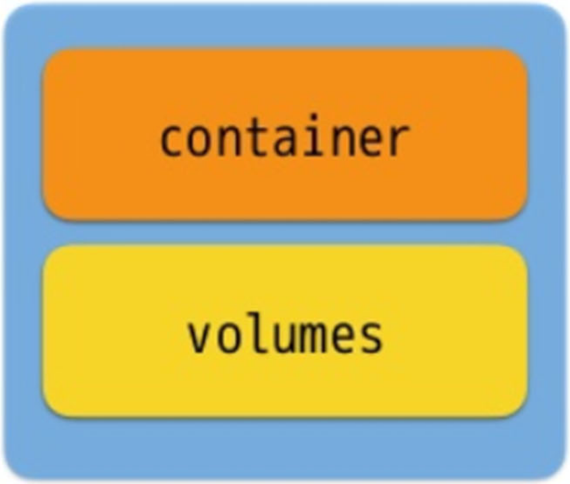
```
container:
  volumeMounts:
    - name: certificates
      mountPath: /etc/ssl/certs
```

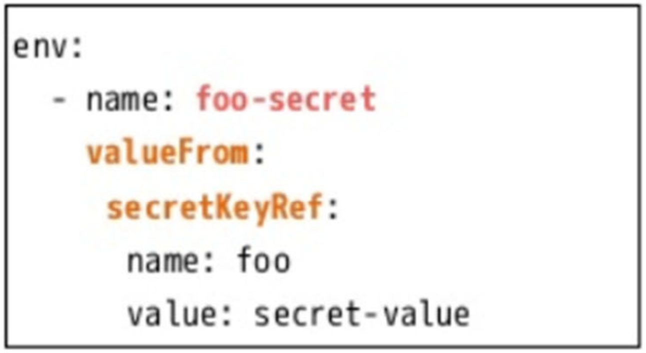


```
volumes:
  - name: certificates
    secret:
      secretName: ca-certificates
```

Basic Kubernetes concepts

Secrets = store pieces of data in k8s



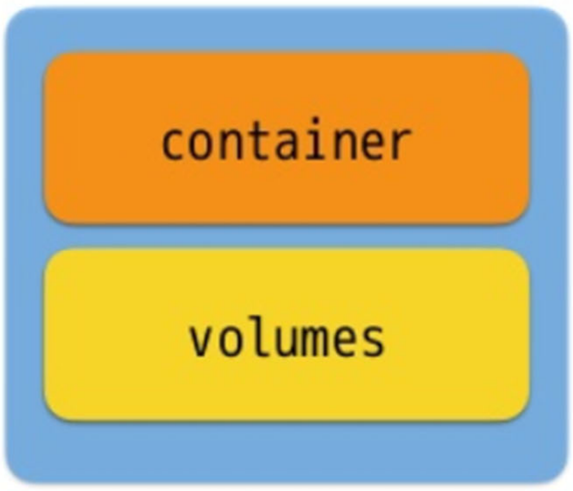


```
env:
  - name: foo-secret
    valueFrom:
     secretKeyRef:
      name: foo
      value: secret-value
```

Using Secrets as environment variables

Basic Kubernetes concepts

ConfigMaps = same as Secrets (unprotected)

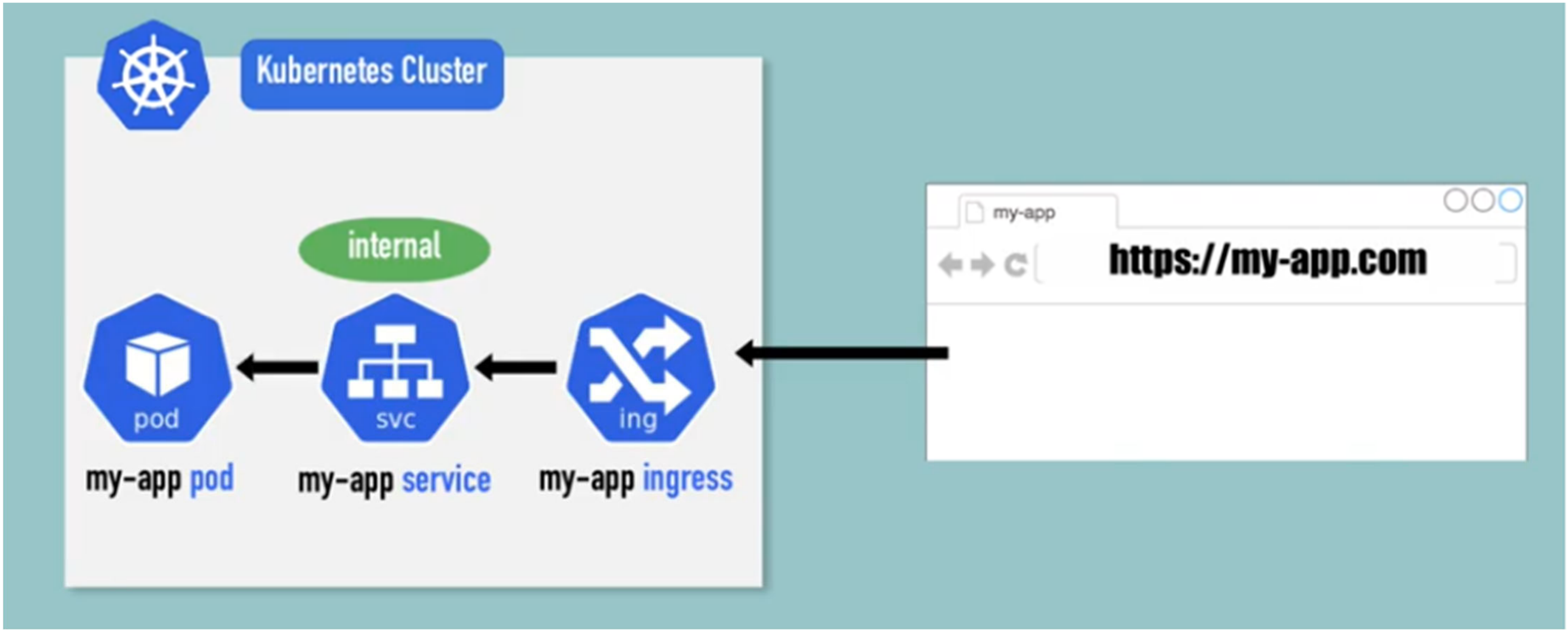


```
env:
  - name: foo-secret
    valueFrom:
       configMapKeyRef:
       name: foo
       key: secret-value
```

- A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume .
- **ConfigMap does not provide secrecy or encryption.**

Basic Kubernetes concepts

Ingress = inbound connections to internal cluster services

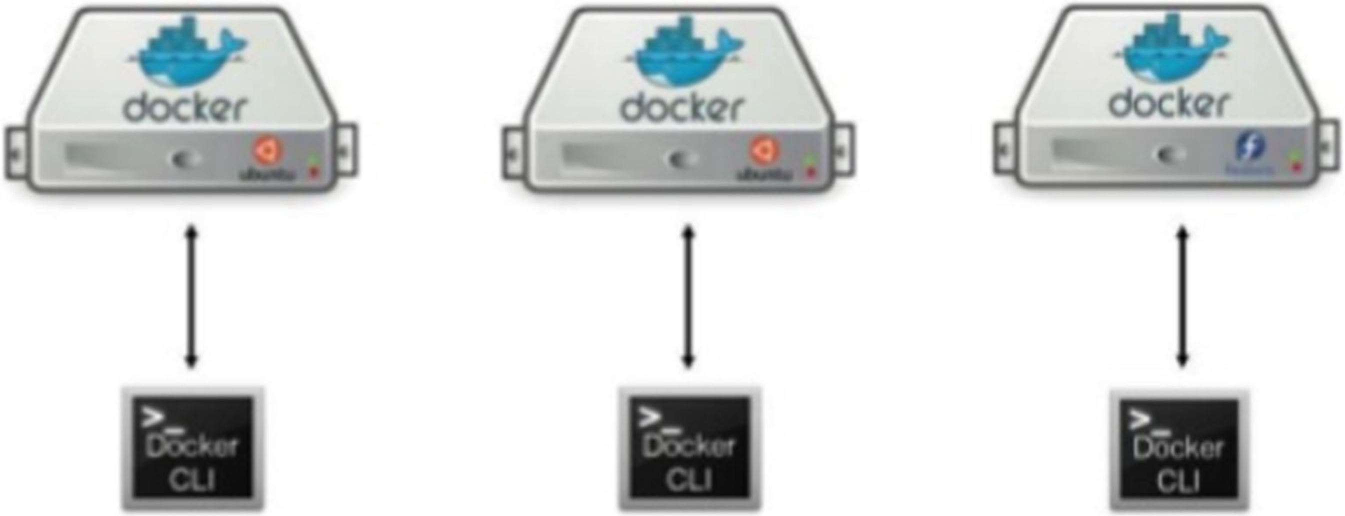


- Ingress is an API object that manages external access to the services in a cluster, typically HTTP
- Ingress may provide load balancing

Outline

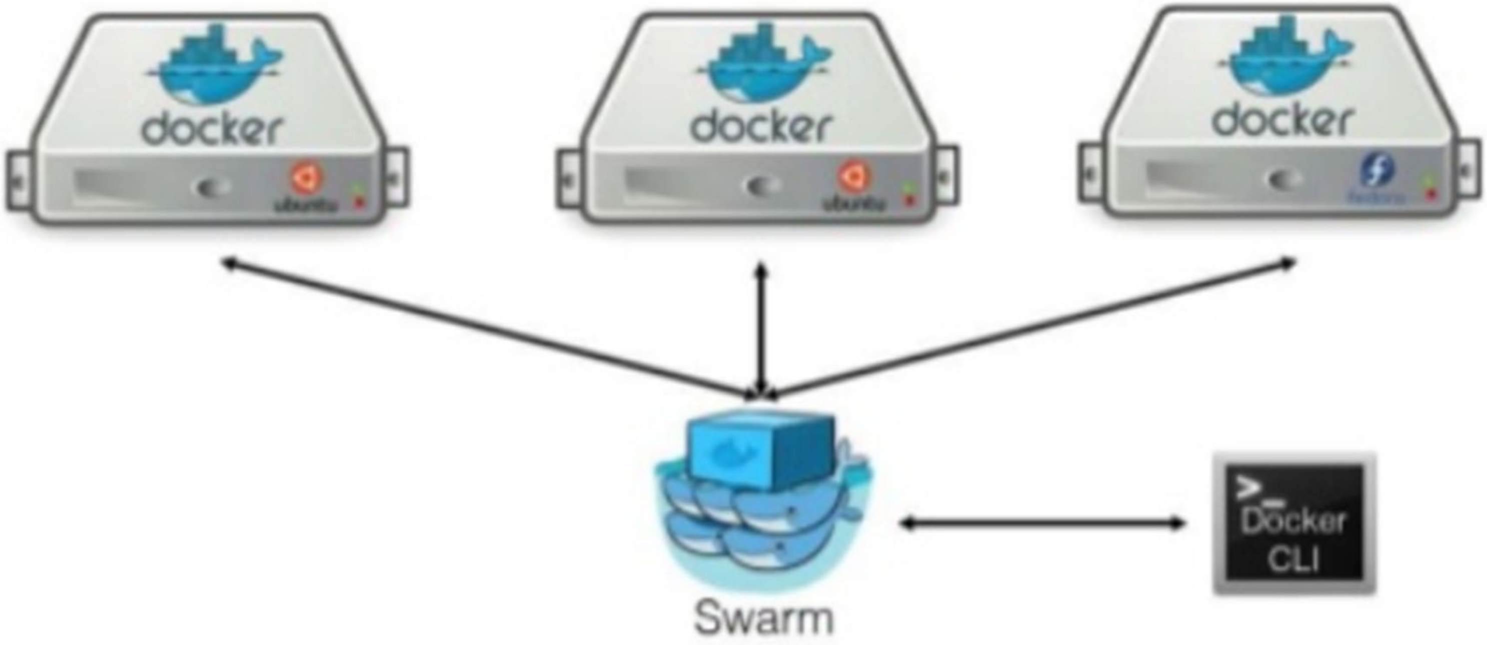
- Monoliths
- Service Oriented Architectures
- MicroServices
- Containerization
- **Orchestration**
- **Docker Swarm**

Traditional Docker Container Deployment

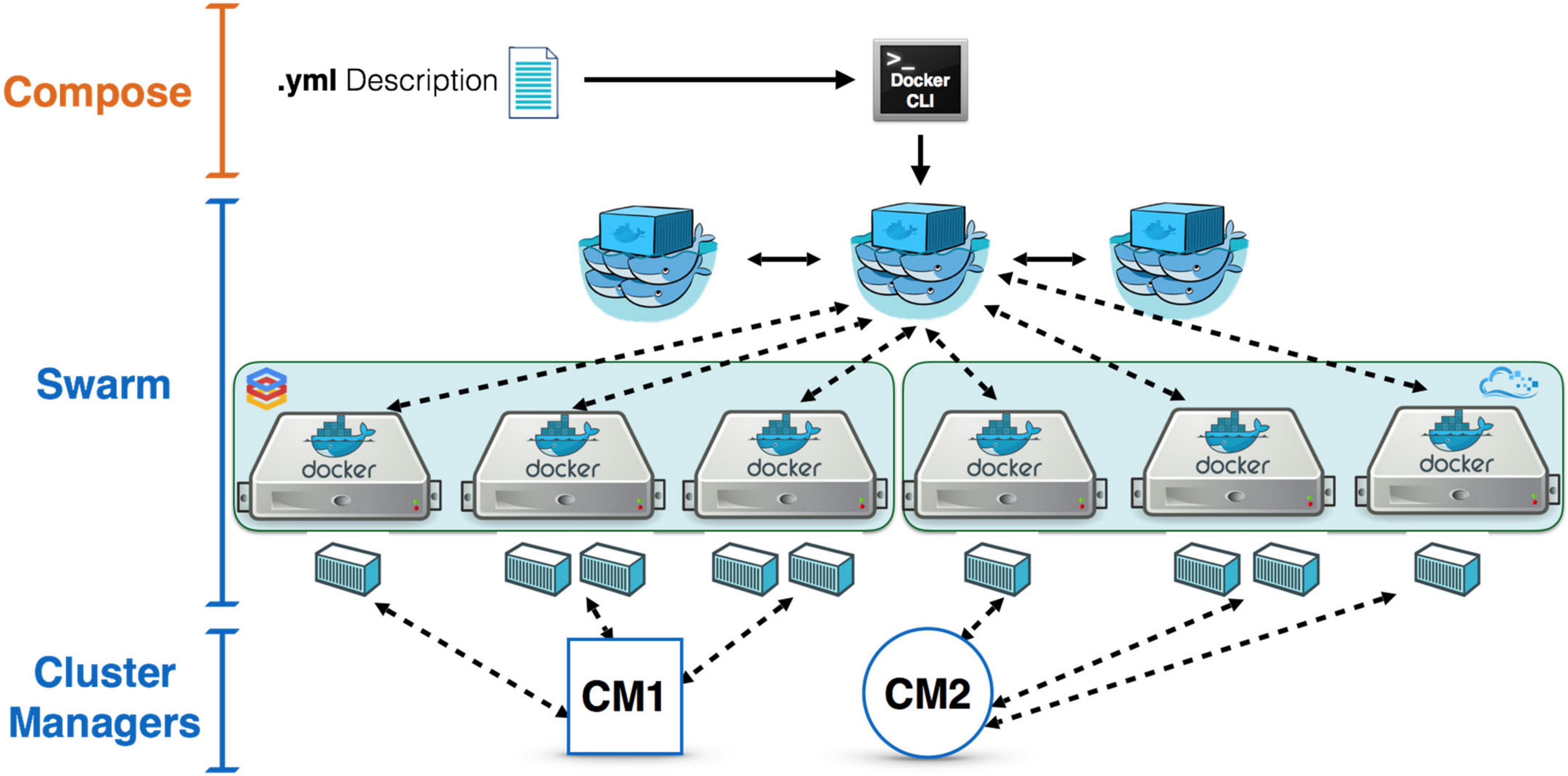


Docker Swarm

- Docker Swarm is a container orchestration platform for Docker containers.
- Swarm turns a pool of Docker hosts into a virtual, single host.

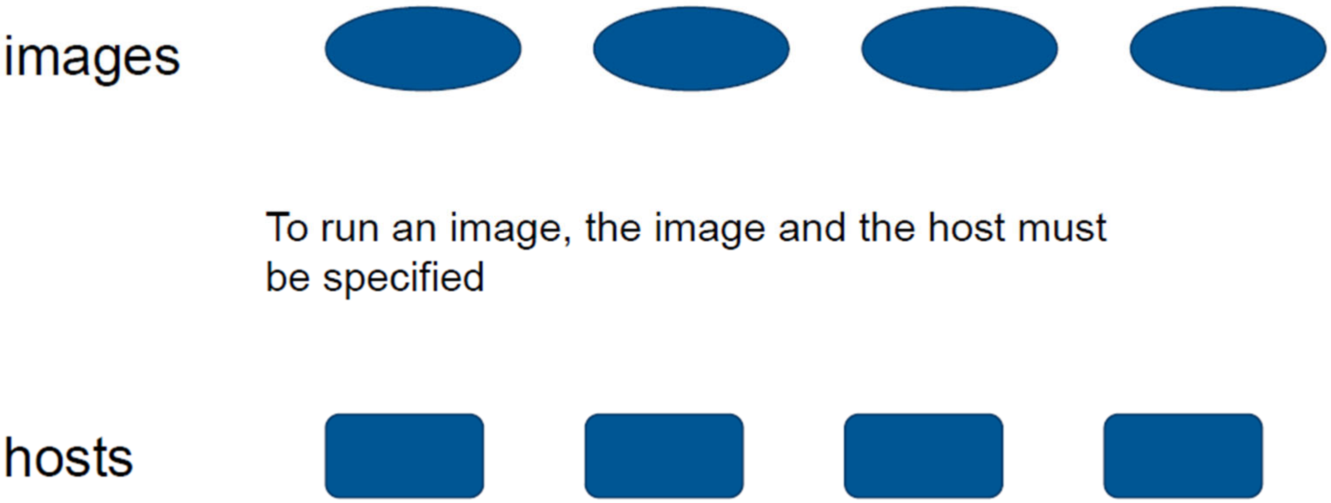


Docker Platform with Swarm



So how does Swarm work?

Allocation of images to hosts



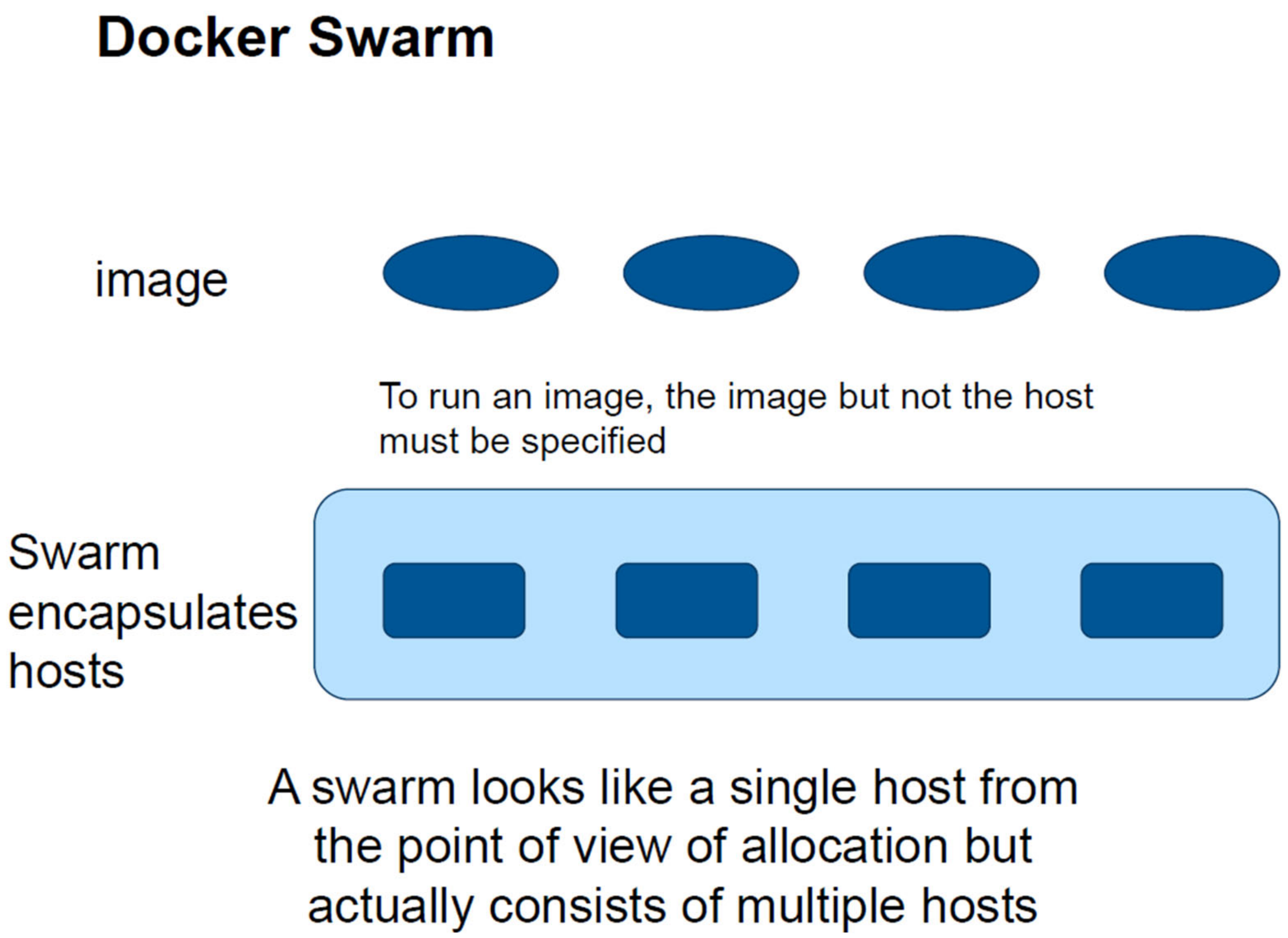
images

To run an image, the image and the host must be specified

hosts

With basic Docker this allocation must be done manually

So how does Swarm work?



Docker Swarm

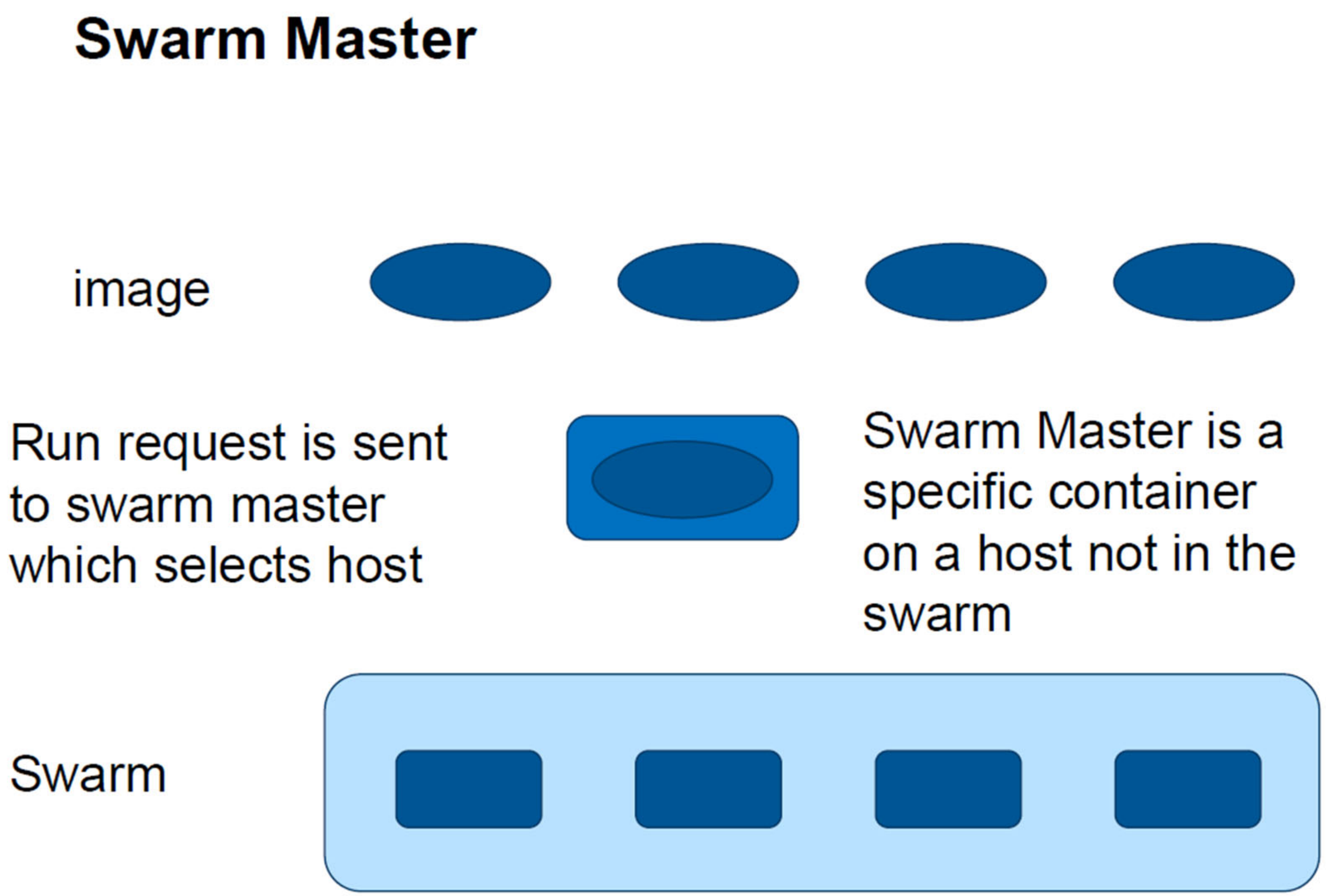
image

To run an image, the image but not the host must be specified

Swarm encapsulates hosts

A swarm looks like a single host from the point of view of allocation but actually consists of multiple hosts

So how does Swarm work?



Swarm Master

image

Run request is sent to swarm master which selects host

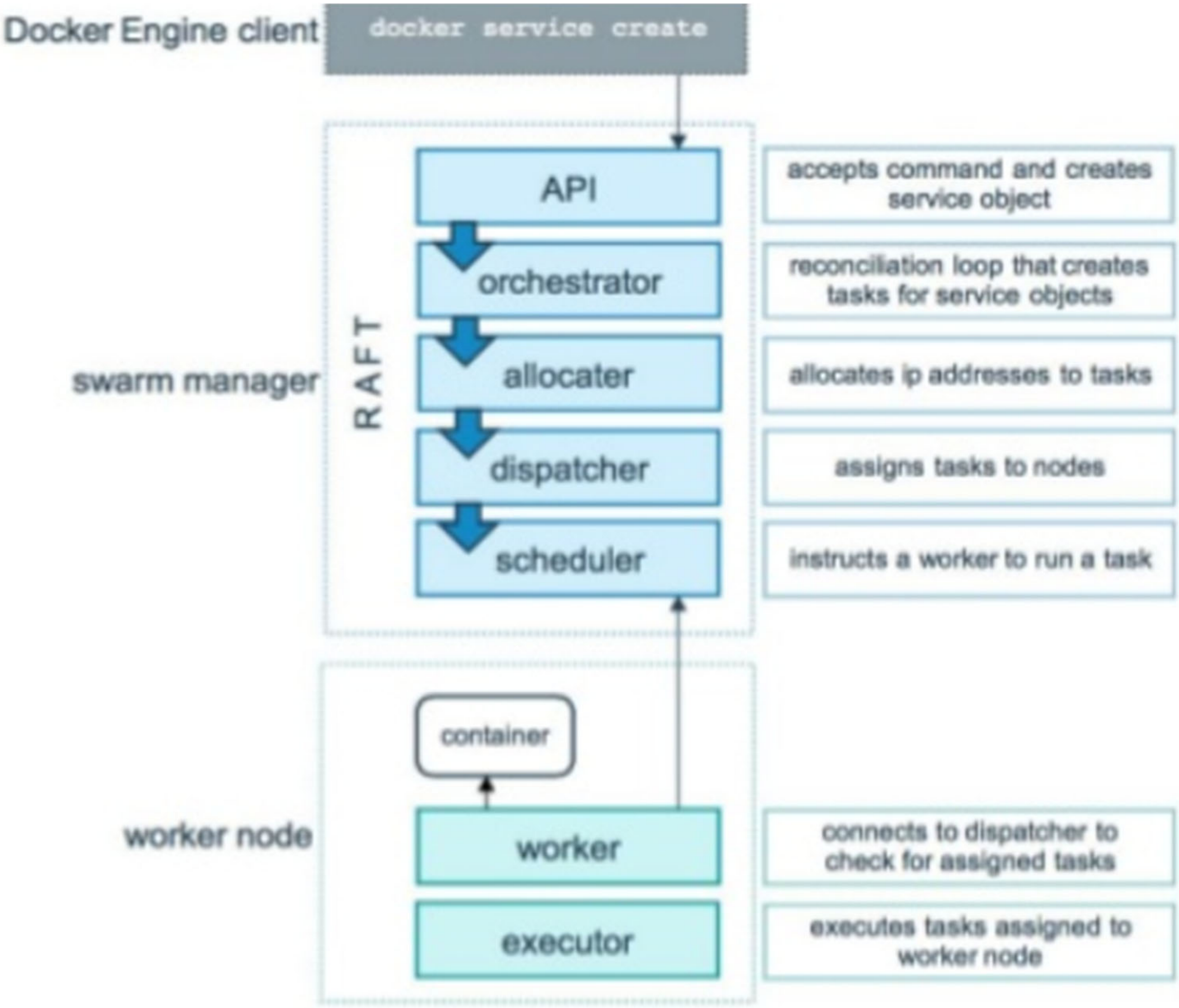
Swarm Master is a specific container on a host not in the swarm

Swarm

Docker Swarm Concepts

- A node can be a manager or a worker
- You can talk to the manager using the Swarm API
- One manager is elected as a leader, the others merely forward requests to it
- Using the API, you can indicate that you want to run a service
- A service is specified by its desired state: which image, how many instances,…
- The leader uses different subsystems to break down services into tasks: orchestrator, scheduler, allocator, dispatcher, …
- A task corresponds to a specific container, assigned to a specific node
- Nodes know which tasks should be running, and will start or stop containers accordingly (through the Docker Engine API)

Docker Swarm Architecture


Docker Engine client
docker service create
swarm manager
RAFT
API
accepts command and creates service object
orchestrator
reconciliation loop that creates tasks for service objects
allocater
allocates ip addresses to tasks
dispatcher
assigns tasks to nodes
scheduler
instructs a worker to run a task
worker node
container
worker
connects to dispatcher to check for assigned tasks
executor
executes tasks assigned to worker node

THANK YOU