



Getting started with Kubernetes

VAIBHAV GUJRAL
CLOUD ARCHITECT | MICROSOFT AZURE MVP

About me



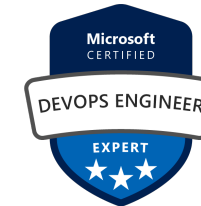
Director, Global Microsoft Cloud CoE at Capgemini

Born and brought up in India and based out of Omaha, NE since 2016






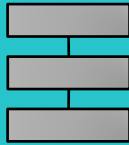






4x Microsoft Azure MVP since 2020

Leader, Omaha Azure User Group(<https://omahaazure.org>)

15+ cloud certifications and counting...

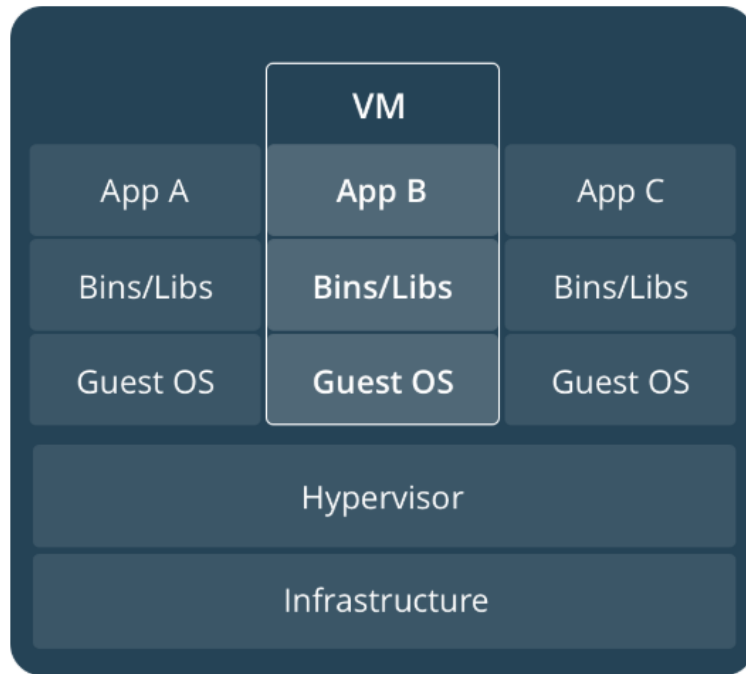


Technology Evolution

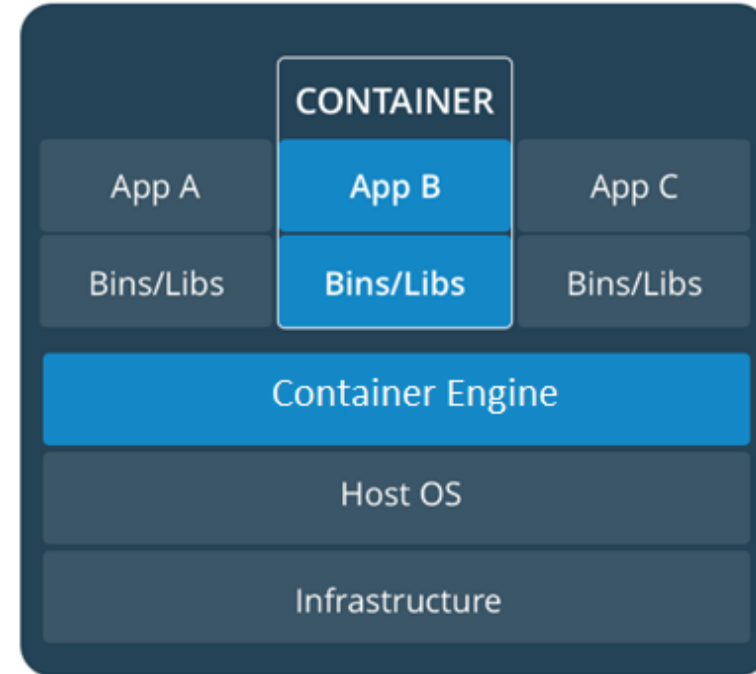
	Development Process	Application Architecture	Deployment Model	Infrastructure
Now	 DevOps	 Microservices	 Containers	 Cloud
2000-2010	 Agile	 N-Tier	 Virtual Machines	 Hosted
1990's and early 2000s	 Waterfall	 Monolithic	 Physical Servers	 Data Center

Virtual Machines vs Containers

VIRTUAL MACHINES



CONTAINERS



Why Containers?

Containers are –

Flexible: Even the most complex applications can be containerized.

Lightweight: Containers leverage and share the host kernel.

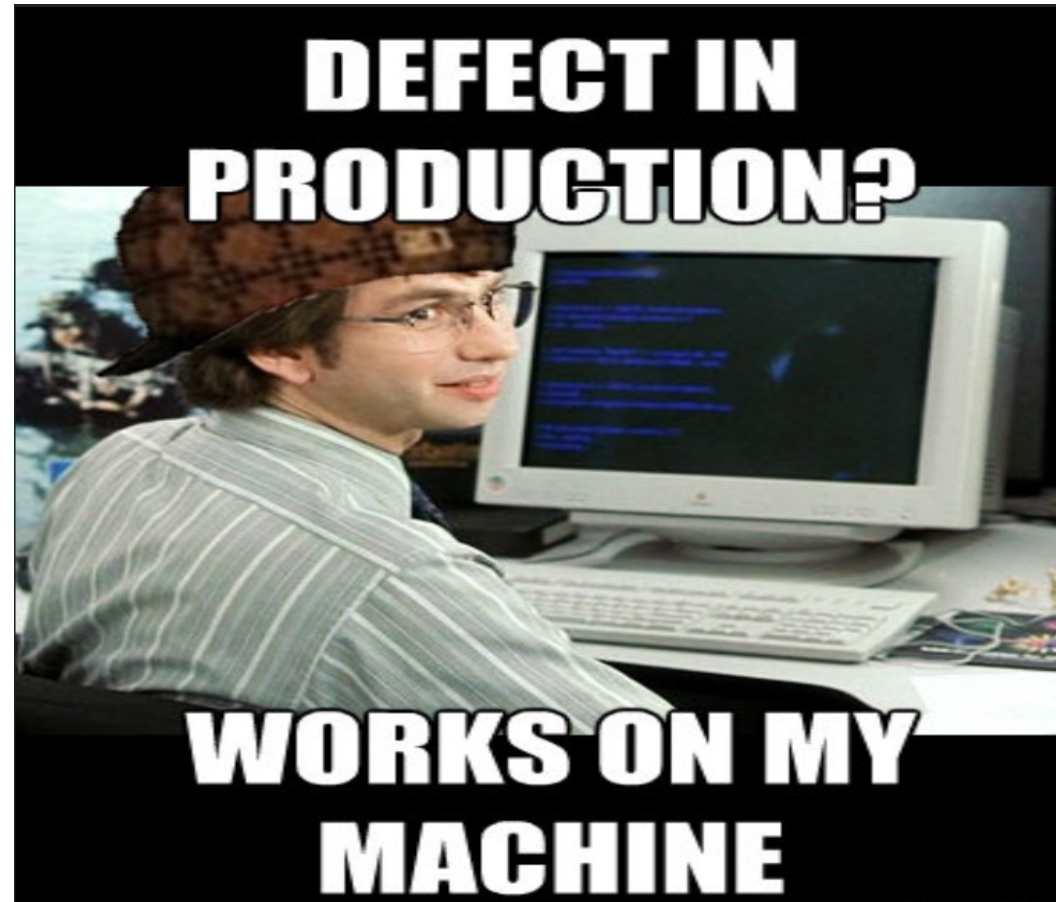
Interchangeable: Deploy updates and upgrades on-the-fly.

Portable: Build locally, deploy to the cloud, and run anywhere.

Scalable: Increase and automatically distribute container replicas.

Stackable: Stack services vertically and on-the-fly.

“It Works on My Machine” syndrome



What is Docker?

Docker is an open-source project for automating the deployments of applications as portable self-sufficient containers that can run on cloud or on-premises

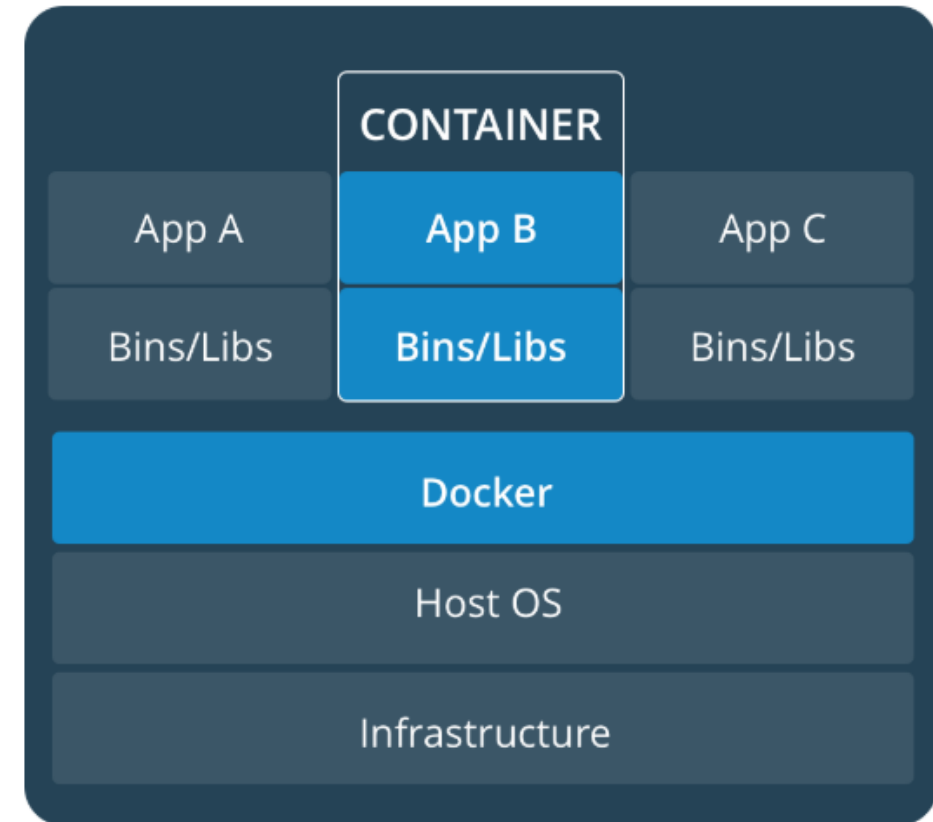
Docker is also the name of the company which promotes and evolves the Docker project

Docker Engine, a software daemon, is the core of Docker

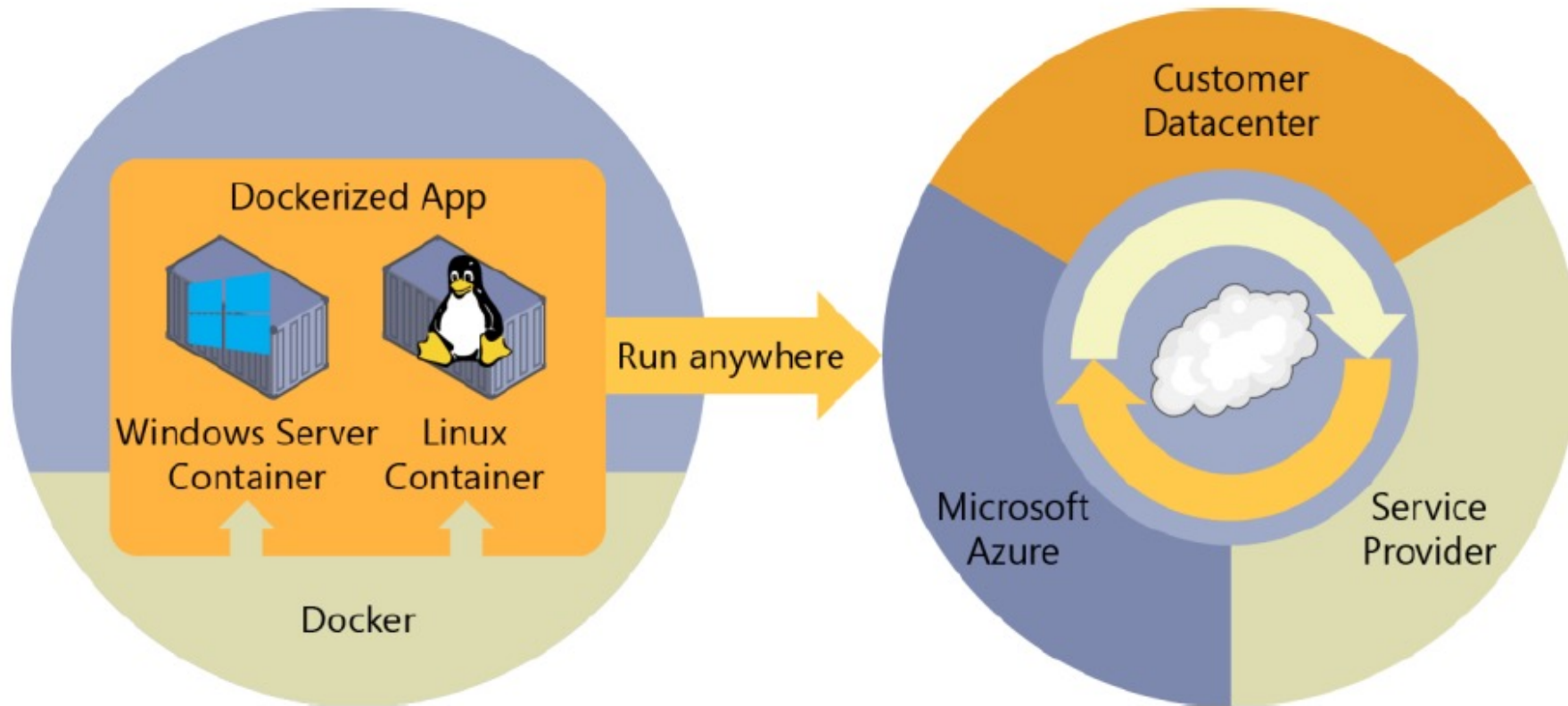
Docker Containers run on top of Docker Engine

Docker Hub is the most widely used container registry

Docker Trusted Registry is the enterprise-grade image storage solution from Docker which can be installed on-premise



Build Once, Run Anywhere



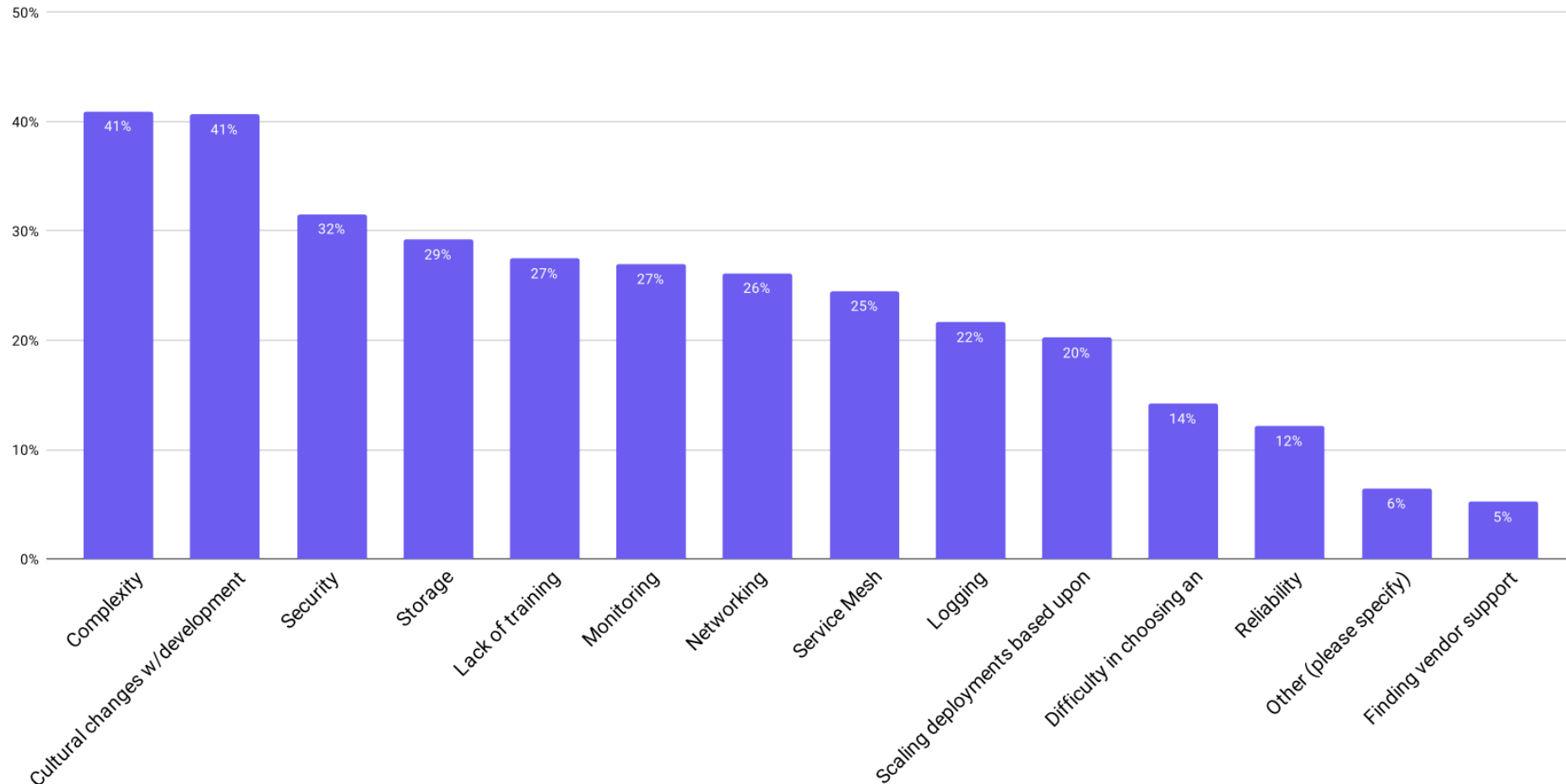
Challenges with Container Management

Complexity rises as the number of containers increases

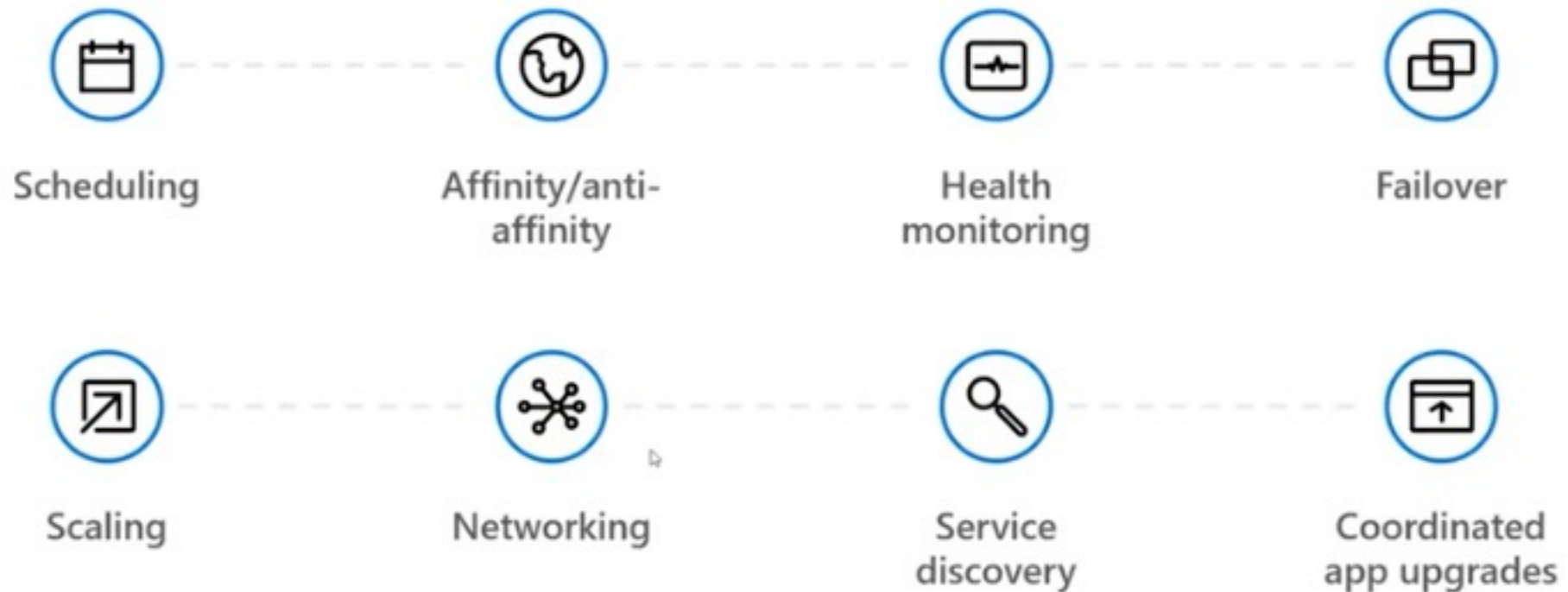
Some challenges -

1. How do you load balance your requests to the containers running your service?
2. How do you heal the containers which go down?
3. How do you manage rollouts and rollbacks?
4. How do you handle configurations and secrets for the containers?

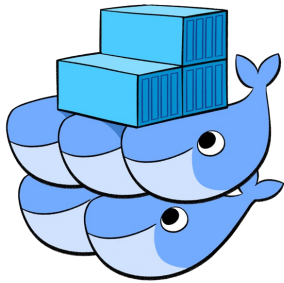
Challenges with Container Management



Elements of Container Orchestration



Container Orchestrators



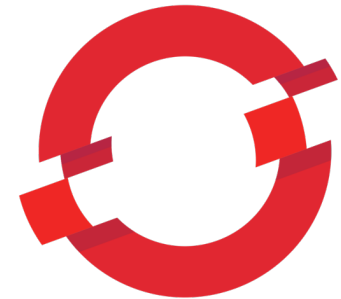
Docker Swarm



Kubernetes



HashiCorp Nomad



RedHat OpenShift

What is Kubernetes?

Kubernetes is a portable, extensible, open-source platform for automating the deployment, scaling, and management of containerized workloads.



*Kubernetes (k(j)u:bər'netɪs)
Greek for “helmsman of a ship”*



Kubernetes History

First announced by Google in 2014

Heavily influenced by Google's **Borg** system.

Original codename for Kubernetes project was Project 7 (a reference to the Star Trek ex-Borg character Seven of Nine)

V1.0 was released on July 21, 2015

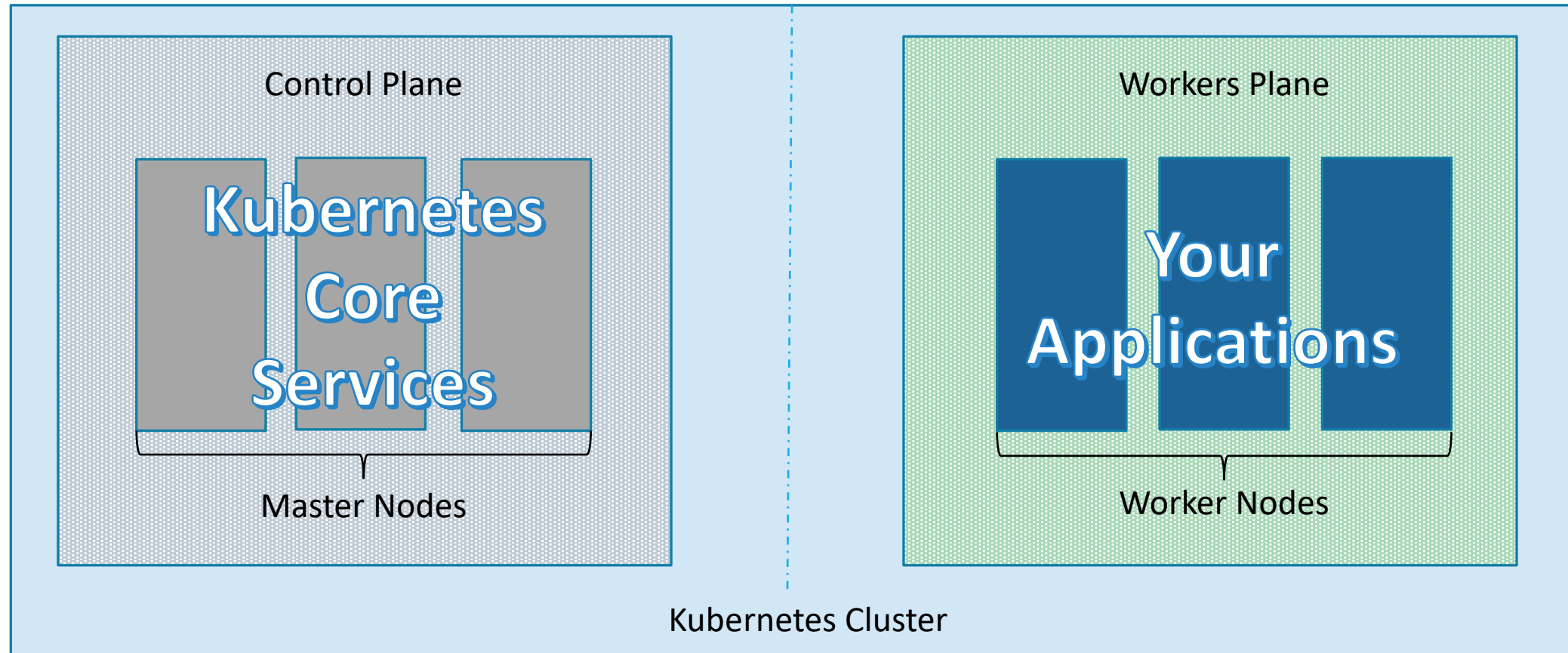
Current version is 1.27... 1.28 releases on August 15th

Originally written in C++, the current system is written in Go language.



Kubernetes Core Concepts

Kubernetes Architecture – Big Picture



Pod

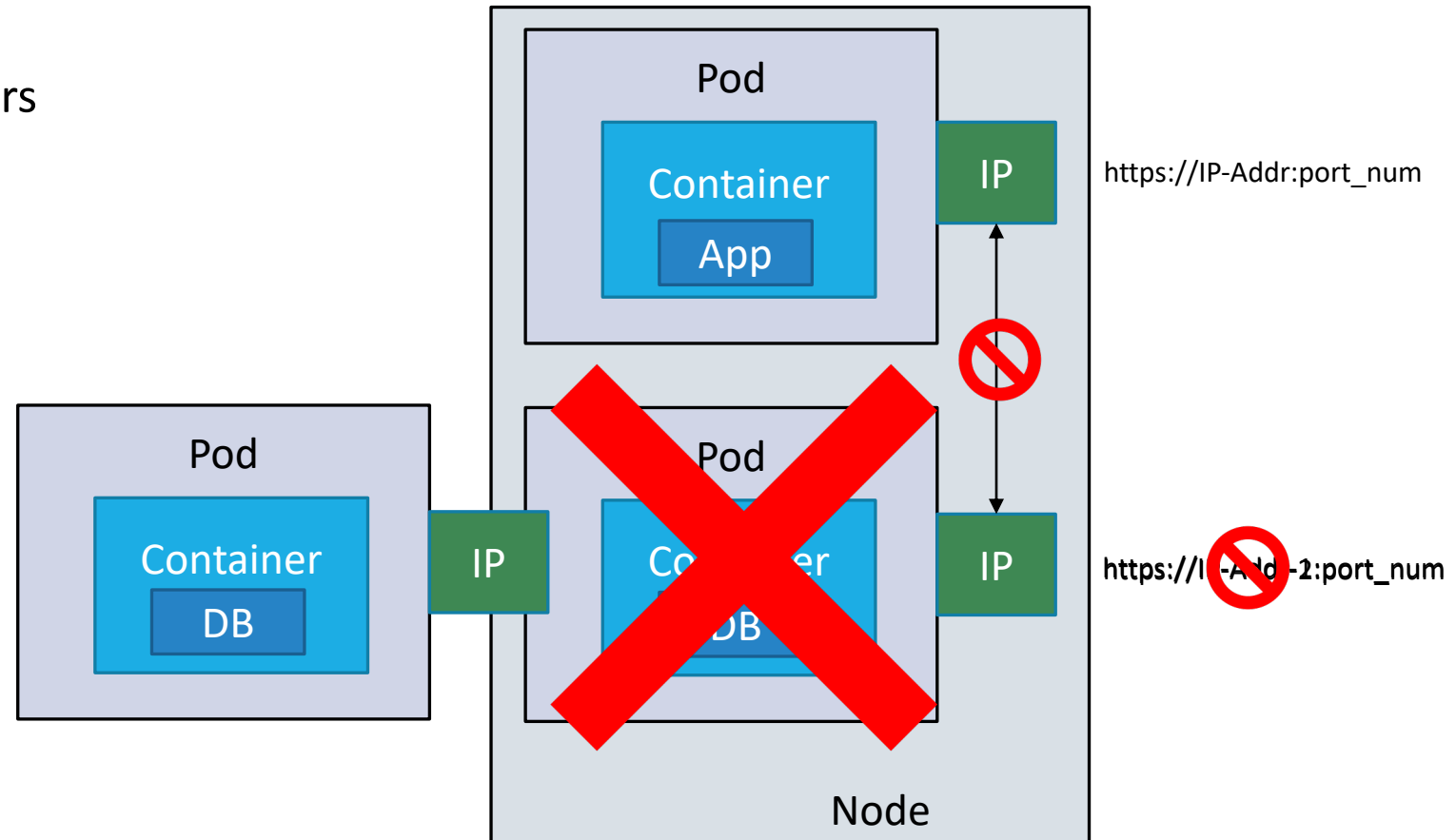
Smallest unit in Kubernetes

Pod is an abstraction over containers

Generally, one application per pod

Each pod gets its own IP address

Pods are ephemeral



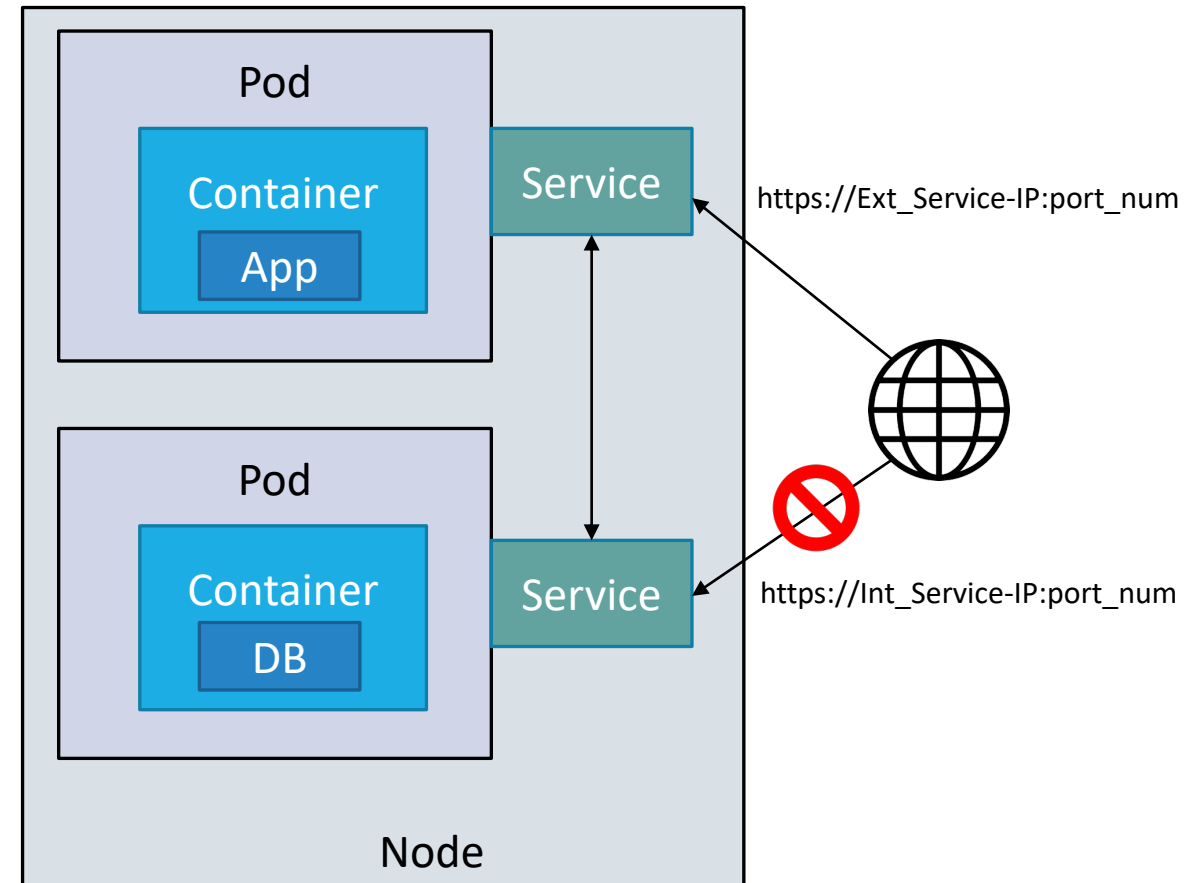
Service

Provides a static IP Address for a set of pods running an application and also acts as a load balancer

Lifecycle of a service is not linked to the lifecycle of a pod

Types of service –

1. Internal Service
2. External Service



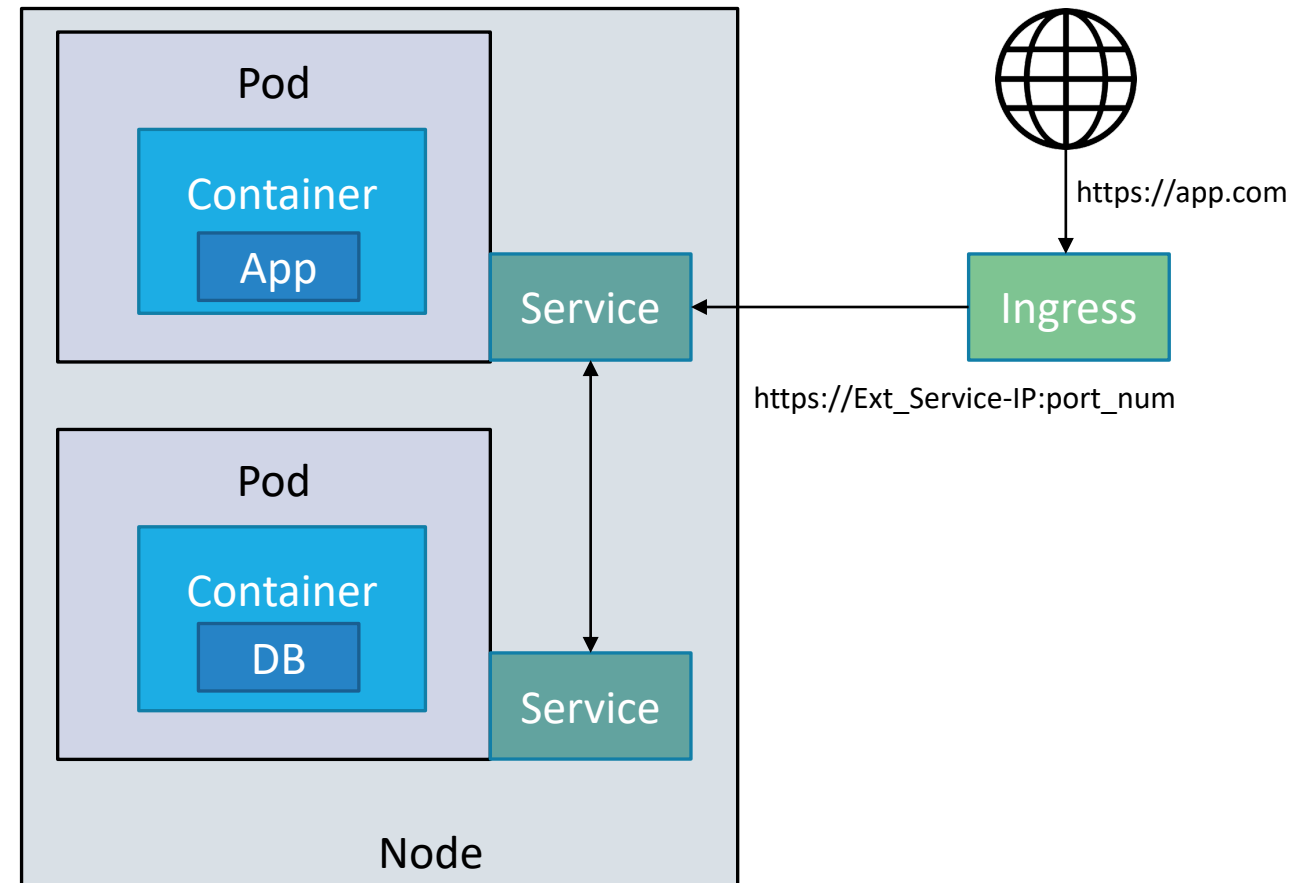
Ingress

Exposes HTTP and HTTPS routes from outside the cluster to services within the cluster

Traffic routing is controlled by rules defined on the Ingress resource.

Ingress –

- gives Services externally-reachable URLs
- load balance traffic
- terminate SSL / TLS
- offer name-based virtual hosting



ConfigMap

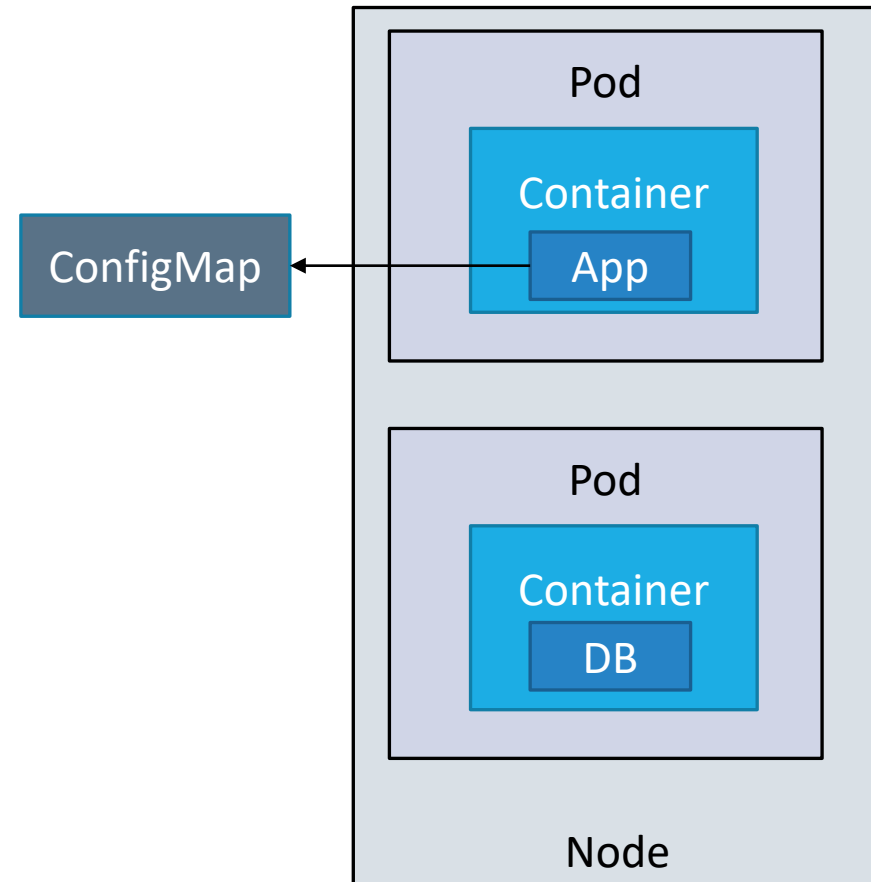
Text-based key-value store to store the external configuration for your application

Set configuration data separately from application code

Use config map for non-confidential data

Pods can consume ConfigMaps as environment variables, command-line arguments or as configuration files in a volume

Do not store credentials or secrets in config map

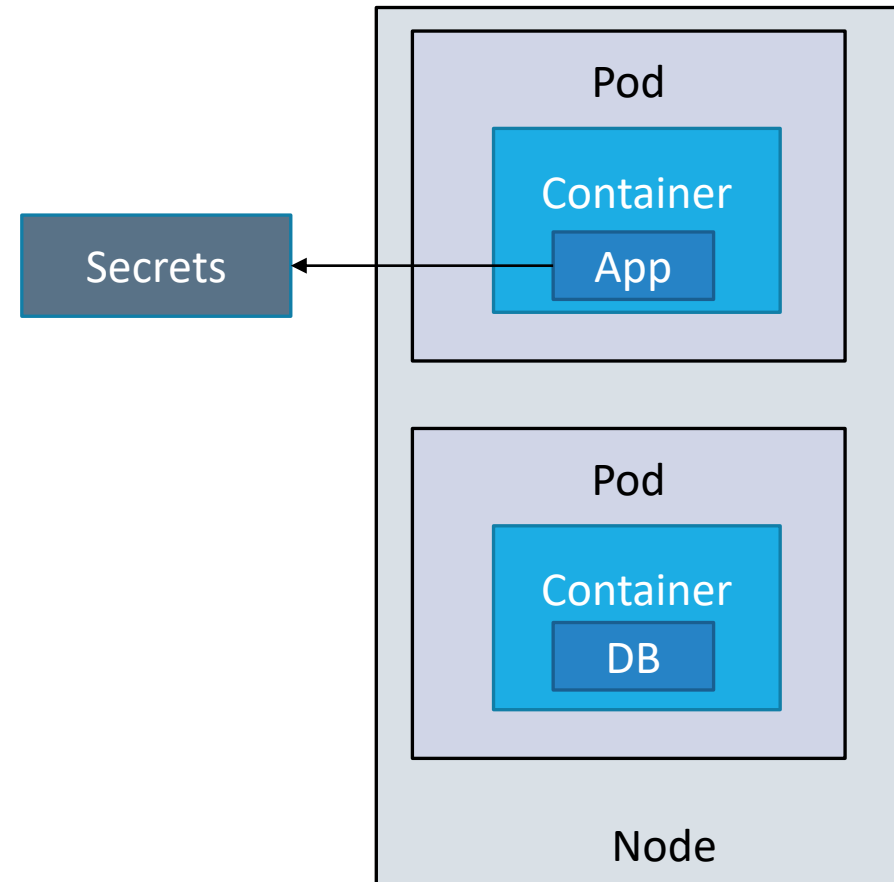


Secrets

Base-64 encoded store to store confidential data

Store secrets, passwords, and certificates in Secrets

Like config maps, secrets can be used as an environment variable or as file in a mounted volume



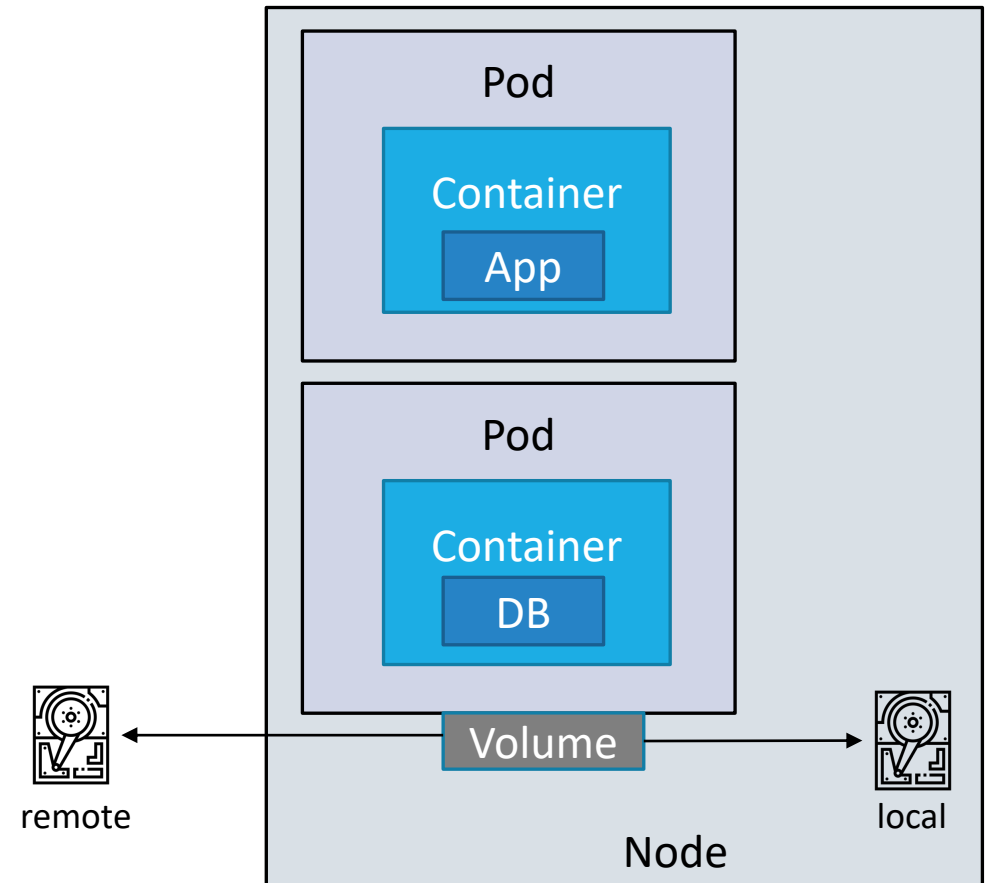
Volumes

Data Storage used to persist data

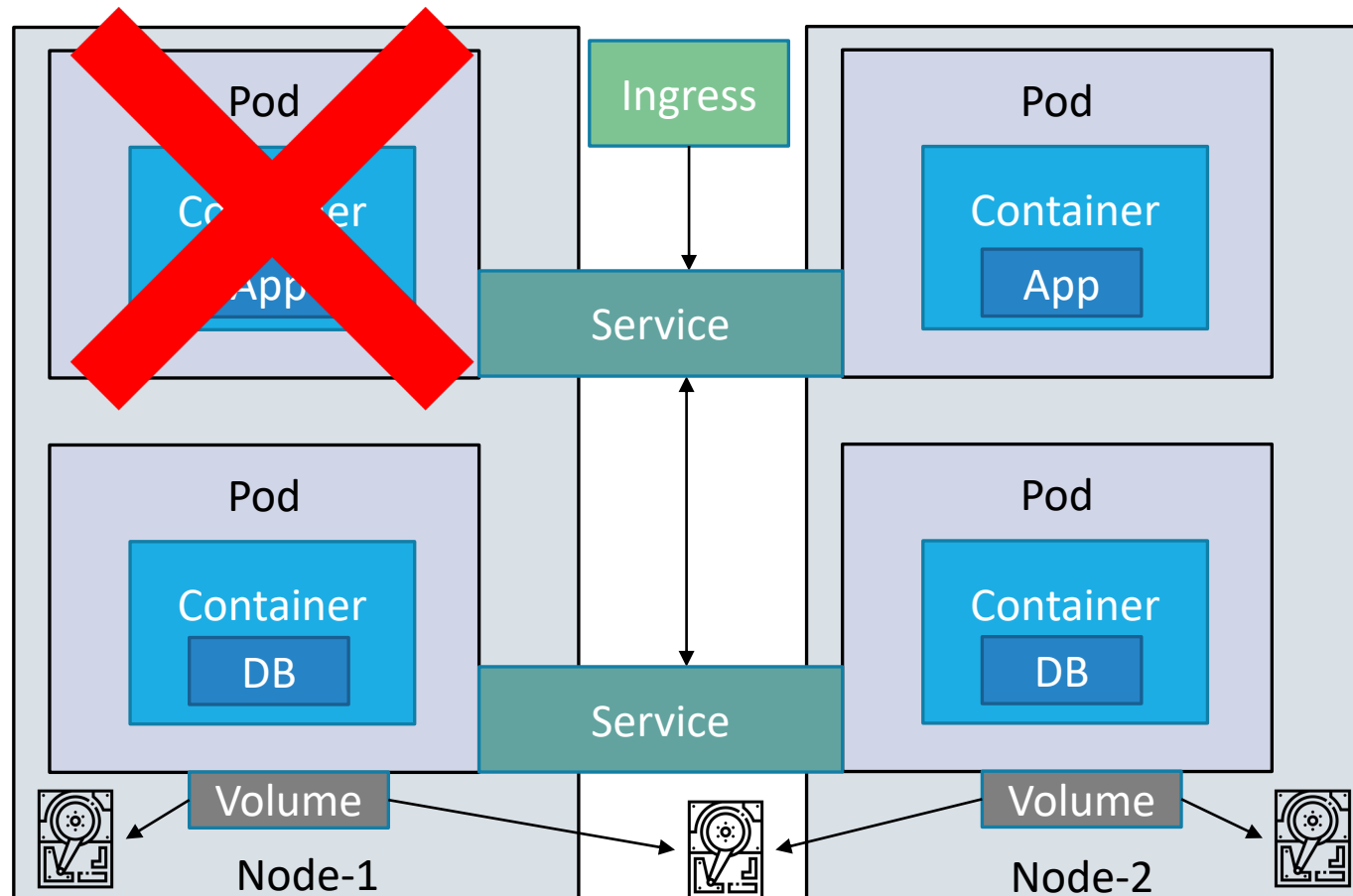
Volumes can exist beyond the lifecycle of a pod

Types of Volumes –

1. Ephemeral Volumes – shares the lifecycle of a pod and defined as pod spec.
2. Persistent Volumes – exists outside of pod and are persisted when a pod is killed. Can be on the node or an external data store like AWS-EBS or Azure Disks

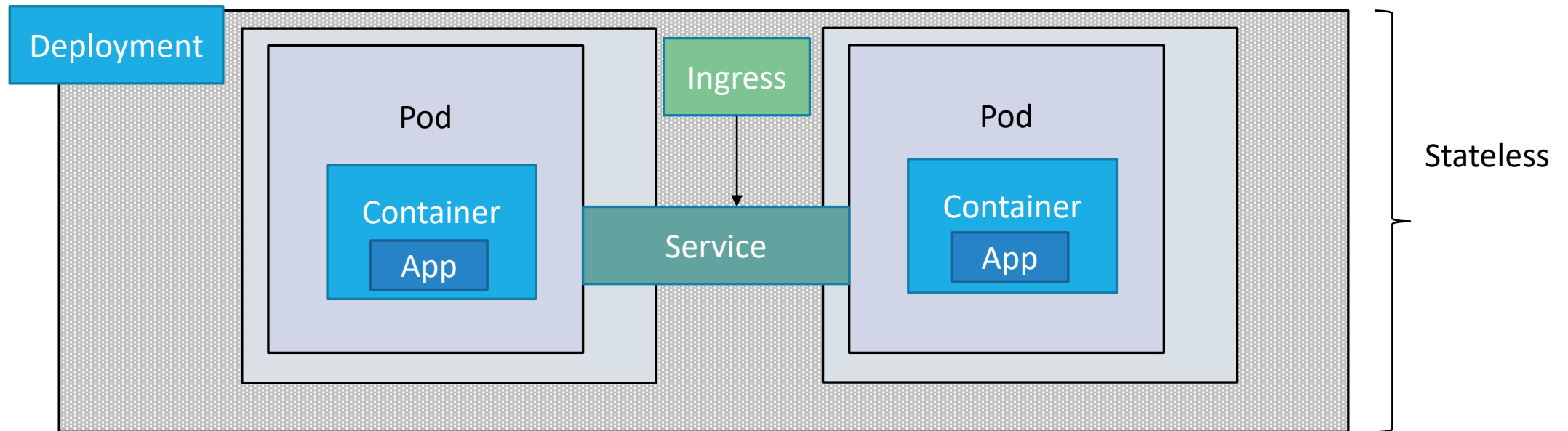


Replica Sets

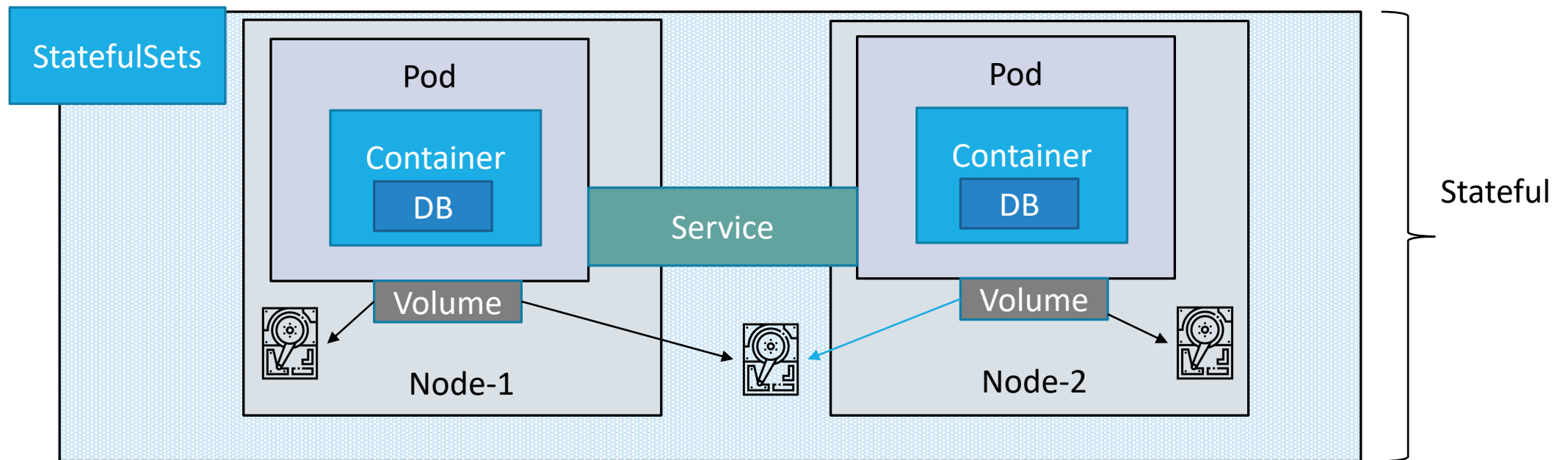


Stateless vs Stateful

Deployments and StatefulSets



Deployments and StatefulSets



Namespaces, Labels and Selectors

namespaces provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

labels are key/value pairs that are used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system. Labels can be attached to objects, such as pods and can be used to organize and to select subsets of objects.

Via a **label selector**, the client/user can identify a set of objects. The label selector is the core grouping primitive in Kubernetes.

annotations lets you attach arbitrary non-identifying metadata to objects

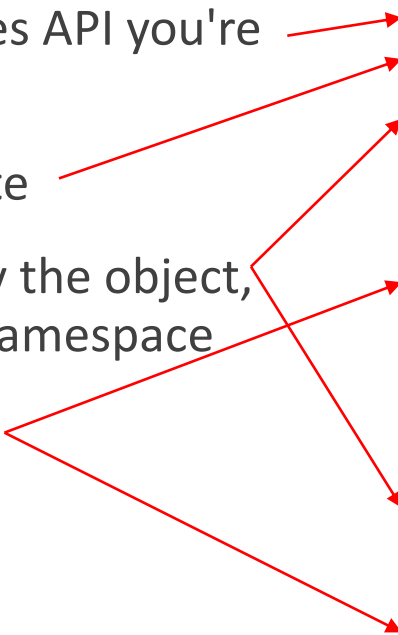
Kubernetes Configuration File

apiVersion - Which version of the Kubernetes API you're using to create this object

kind - What kind of object you want to create

metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace

spec - What state you desire for the object



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

If you're new to YAML, check out <https://yaml.org/>

Kubernetes Core Concepts

Pods are smallest unit in Kubernetes providing an abstraction over containers. Pods are ephemeral and get their own IP Address.

Services provide a persistent IP Address for a set of pods running an application and acts like a load balancer. The lifecycle of a service is not linked to the lifecycle of a pod.

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.

ConfigMaps are text-based key-value stores to store the external configuration for your application.

Secrets are base-64 encoded store for confidential data like passwords and secrets.

Volumes offer data storage for persistent data that needs to exist beyond the lifecycle of a pod.

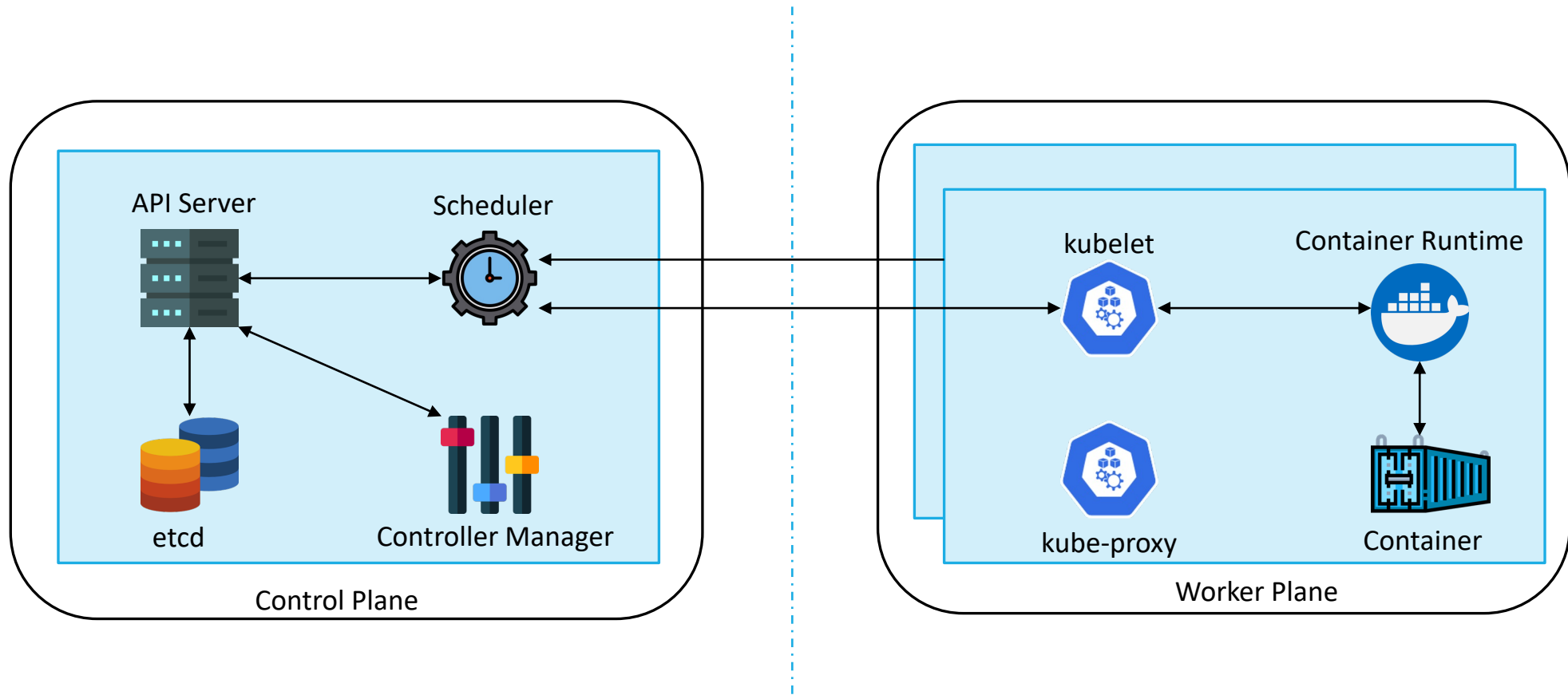
A **ReplicaSet**'s purpose is to maintain a stable set of replica Pods running at any given time.

A **Deployment** provides declarative updates for Pods and ReplicaSets.

A **StatefulSet** is similar to deployment, but it maintains a sticky identity for each of their Pods.

Kubernetes Architecture

Kubernetes Architecture



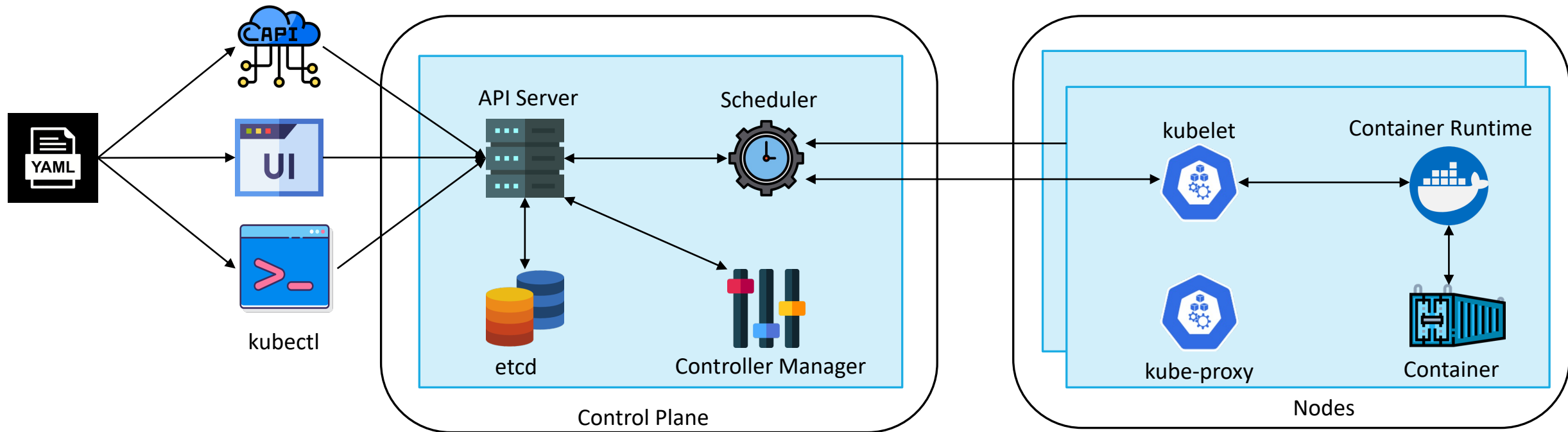
Control Plane Components

Component	Description
kube-apiserver	The API server is how the underlying Kubernetes APIs are exposed. This component provides the interaction for management tools, such as kubectl or the Kubernetes dashboard.
etcd	To maintain the state of your Kubernetes cluster and configuration, the highly available etcd is a key value store within Kubernetes.
kube-scheduler	When you create or scale applications, the Scheduler determines what nodes can run the workload and starts them.
kube-controller-manager	The Controller Manager oversees a number of smaller Controllers that perform actions such as replicating pods and handling node operations.

Node Components

Component	Description
kubelet	The Kubernetes agent that processes the orchestration requests from the control plane and scheduling of running the requested containers.
<i>kube-proxy</i>	Handles virtual networking on each node. The proxy routes network traffic and manages IP addressing for services and pods.
<i>container runtime</i>	<p>Allows containerized applications to run and interact with additional resources, such as the virtual network and storage.</p> <p>Beginning in Kubernetes version 1.20 for Windows node pools, containerd can be used in preview for the container runtime, but Docker is still the default container runtime.</p>

Managing Kubernetes Cluster



kubectl

Command line tool that lets you control Kubernetes clusters

Install steps - <https://kubernetes.io/docs/tasks/tools/>

```
kubectl [command] [TYPE] [NAME] [flags]
```

```
kubectl create deployment nginx --image=nginx
```

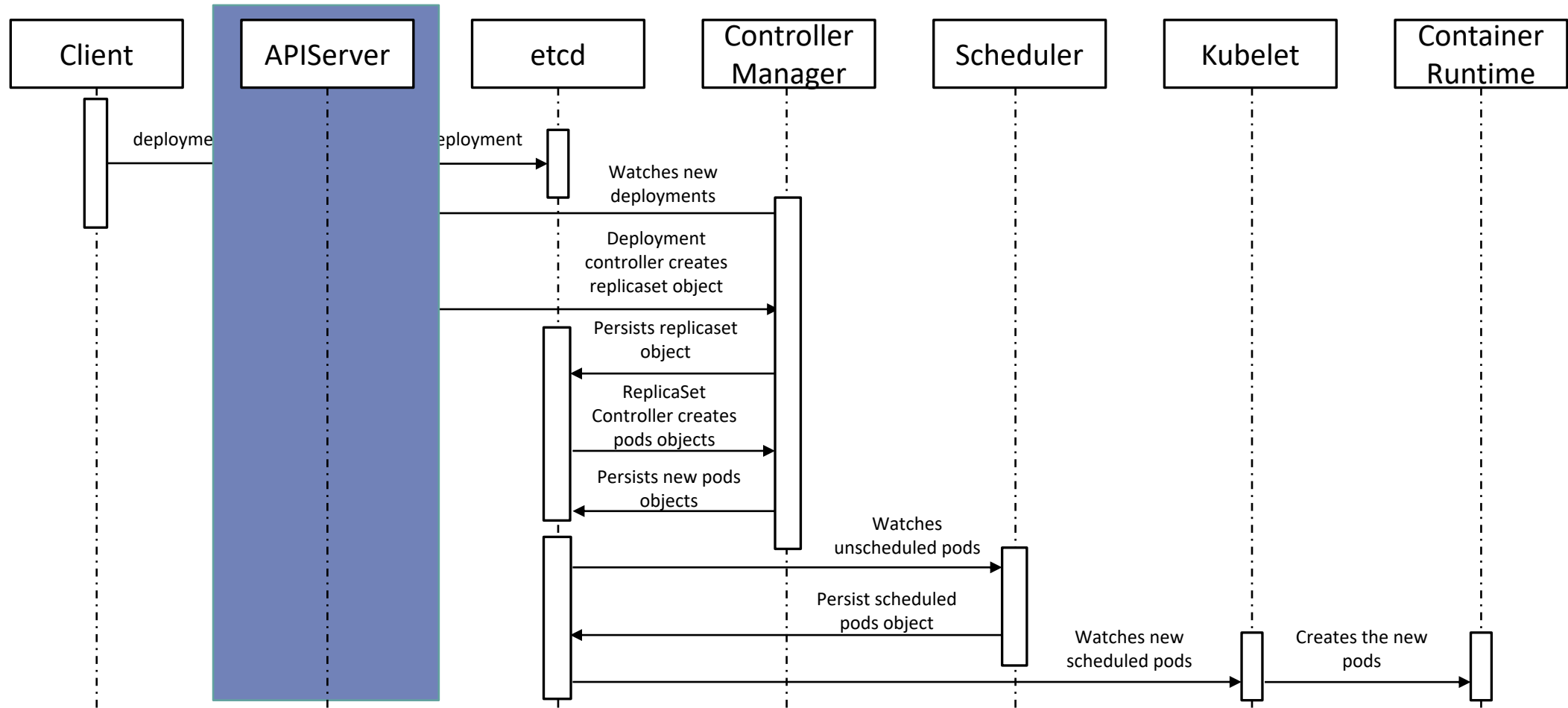
```
kubectl apply -f ./my-deployment.yaml
```

```
kubectl get pods
```

```
kubectl get services
```

<https://kubernetes.io/docs/reference/kubectl/overview/>

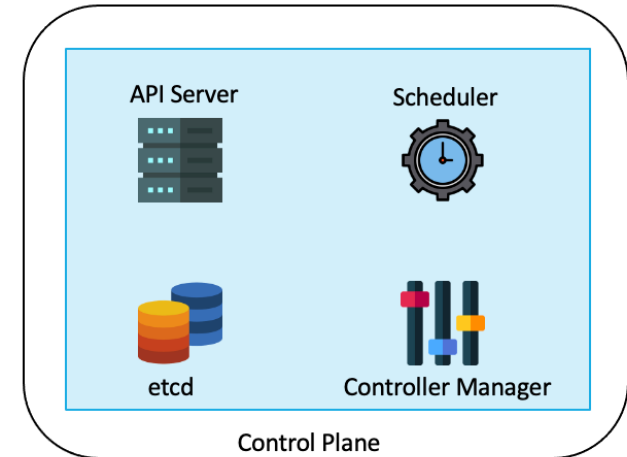
Components Flow in Kubernetes



Kubernetes Components

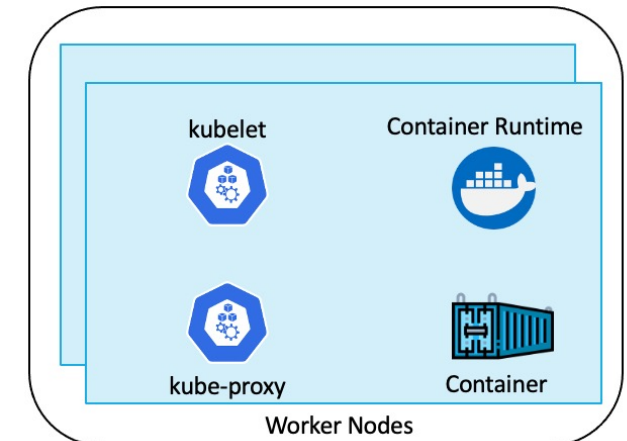
1. Control Plane: manages the agent nodes and the pods in the cluster

- **api-server**: front end of the Kubernetes control plane; exposes Kubernetes API
- **controller-manager**: runs the controller processes
- **scheduler**: tracks newly created pods and selects node to run them on
- **etcd**: stores the state of the cluster (config, running workloads status, etc.)



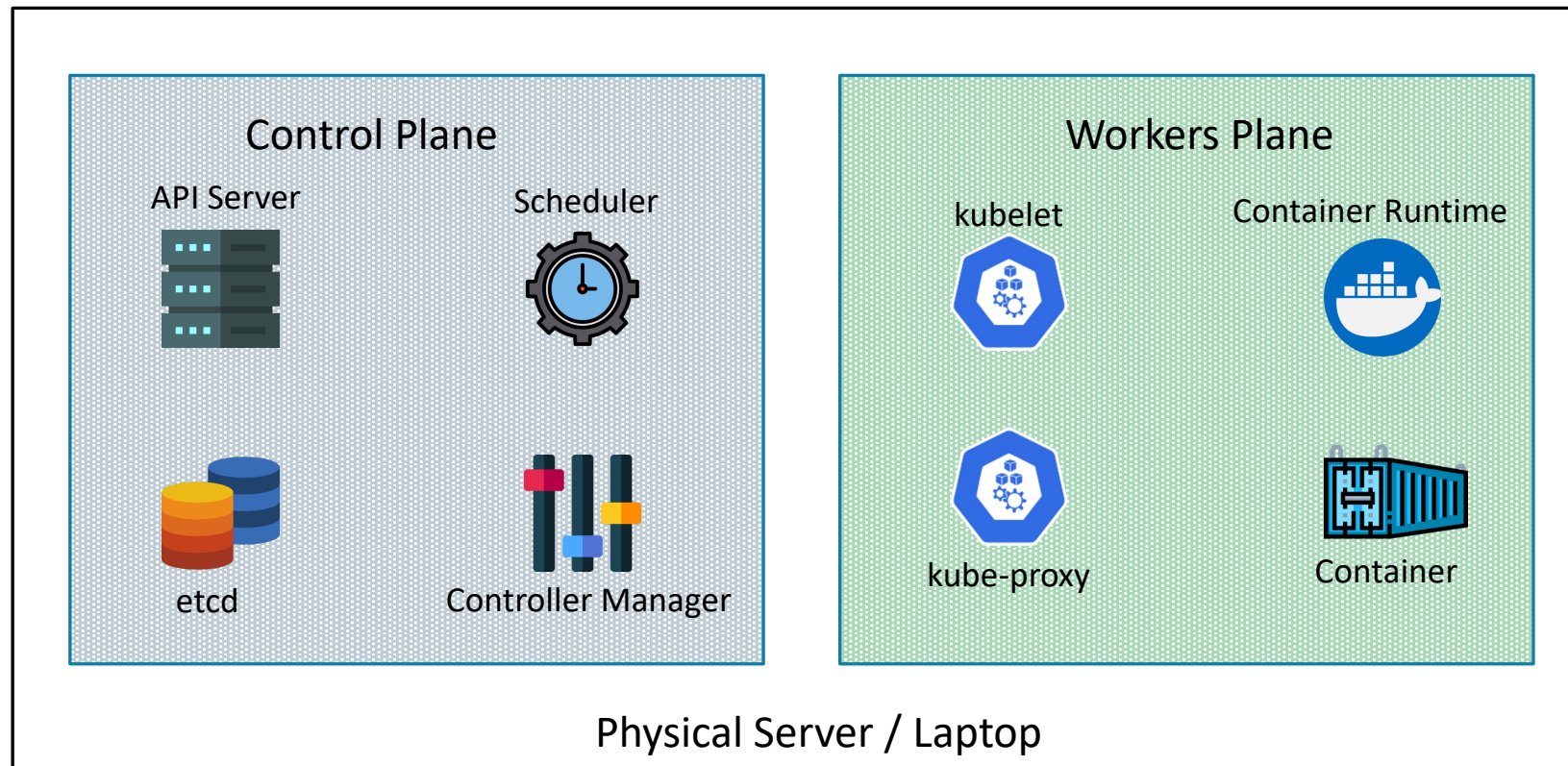
2. Worker nodes: run your application workloads

- **Pods**: a collection of containers co-located on a single machine
- **kube-proxy**: a network proxy that runs on each node in a cluster
- **kubelet**: agent that runs on each node in a cluster; ensures containers are running in a pod
- **Containers**: software responsible for running containers



Local Installation

Local Installation



Kubernetes - the hard way

The screenshot shows the GitHub repository page for `kelseyhightower/kubernetes-the-hard-way`. The repository is public and has 31.8k stars, 10.4k forks, and 967 watchers. The main content area displays a list of files and their commit history, with the `README.md` file selected. The `README.md` file content is visible, showing the title "Kubernetes The Hard Way" and a description of the tutorial. The right sidebar contains sections for "About", "Releases", "Packages", and "Contributors".

Repository Information:

- Repository: `kelseyhightower/kubernetes-the-hard-way` (Public)
- Notifications: 1
- Fork: 10.4k
- Star: 31.8k

File List:

File	Commit Message	Time Ago
<code>deployments</code>	Update to Kubernetes 1.18.6	2 years ago
<code>docs</code>	Update to Kubernetes 1.21.0	15 months ago
<code>.gitignore</code>	Update to Kubernetes 1.15.3	3 years ago
<code>CONTRIBUTING.md</code>	Add brief contribution guide	5 years ago
<code>COPYRIGHT.md</code>	Update to Kubernetes 1.15.3	3 years ago
<code>LICENSE</code>	add LICENSE file	6 years ago
<code>README.md</code>	Update to Kubernetes 1.21.0	15 months ago

README.md Content:

Kubernetes The Hard Way

This tutorial walks you through setting up Kubernetes the hard way. This guide is not for people looking for a fully automated command to bring up a Kubernetes cluster. If that's you then check out [Google Kubernetes Engine](#), or the [Getting Started Guides](#).

Kubernetes The Hard Way is optimized for learning, which means taking the long route to ensure you understand each task required to bootstrap a Kubernetes cluster.

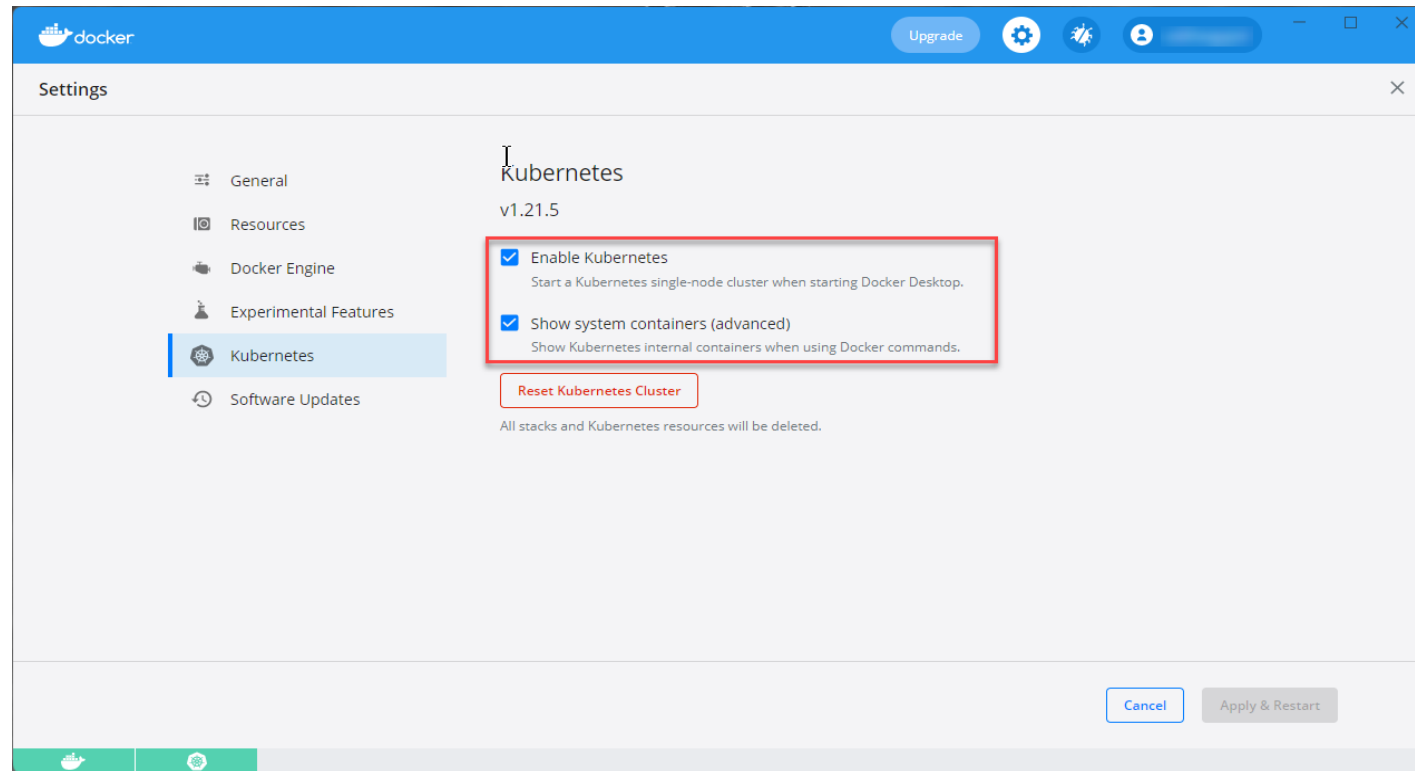
The results of this tutorial should not be viewed as production ready, and may receive limited support from the community, but don't let that stop you from learning!

Right Sidebar:

- About:** Bootstrap Kubernetes the hard way on Google Cloud Platform. No scripts.
- Releases:** 7 tags
- Packages:** No packages published
- Contributors:** 52 contributors (41 additional contributors)

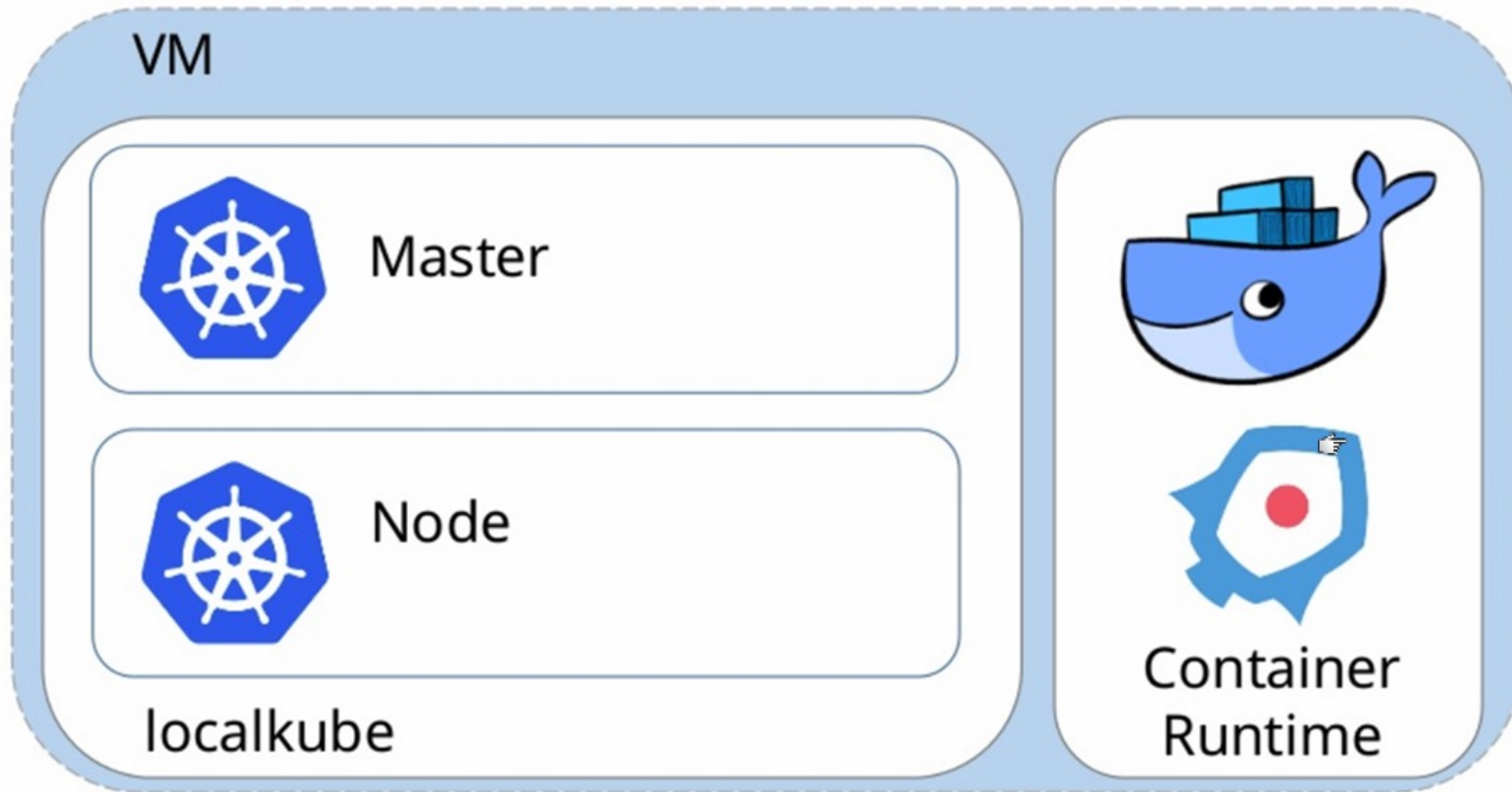
Local Installation – with Docker Desktop

Docker Desktop → Settings → Kubernetes



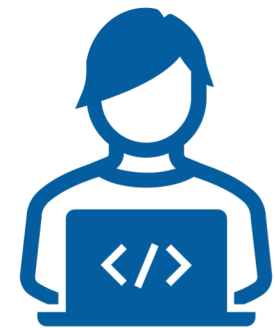
<https://www.docker.com/products/docker-desktop>

Local Installation – with minikube



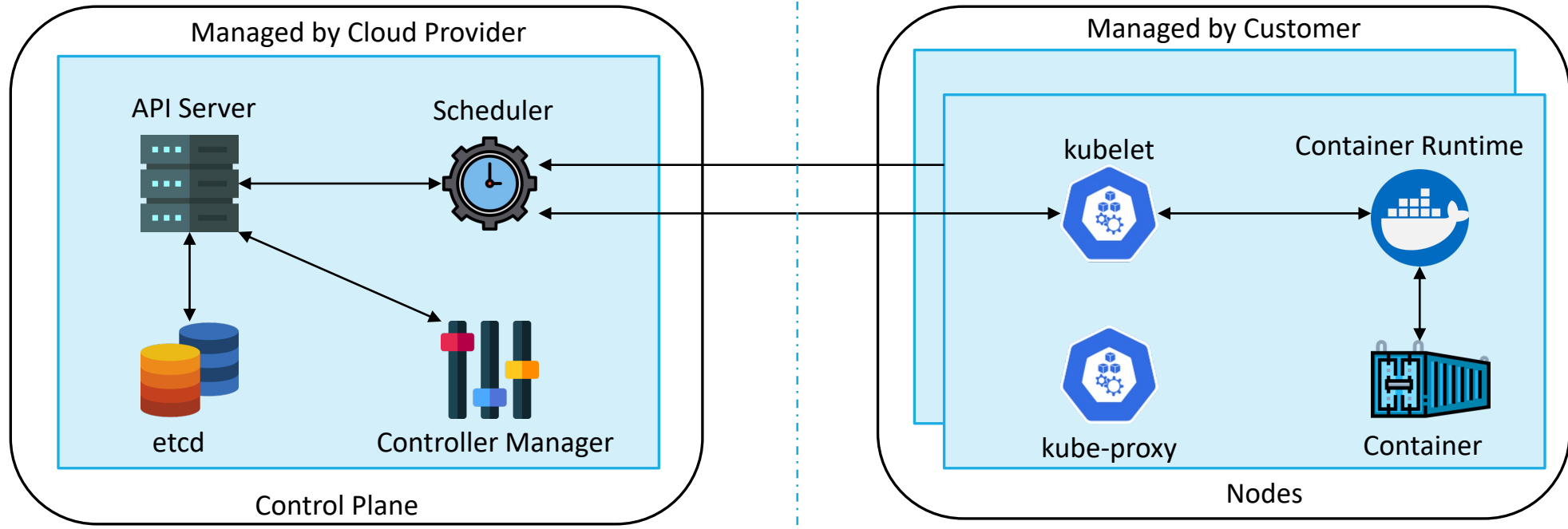
<https://minikube.sigs.k8s.io/docs/>

DEMO



KUBERNETES LOCAL INSTALLATION

Cloud Managed Offerings

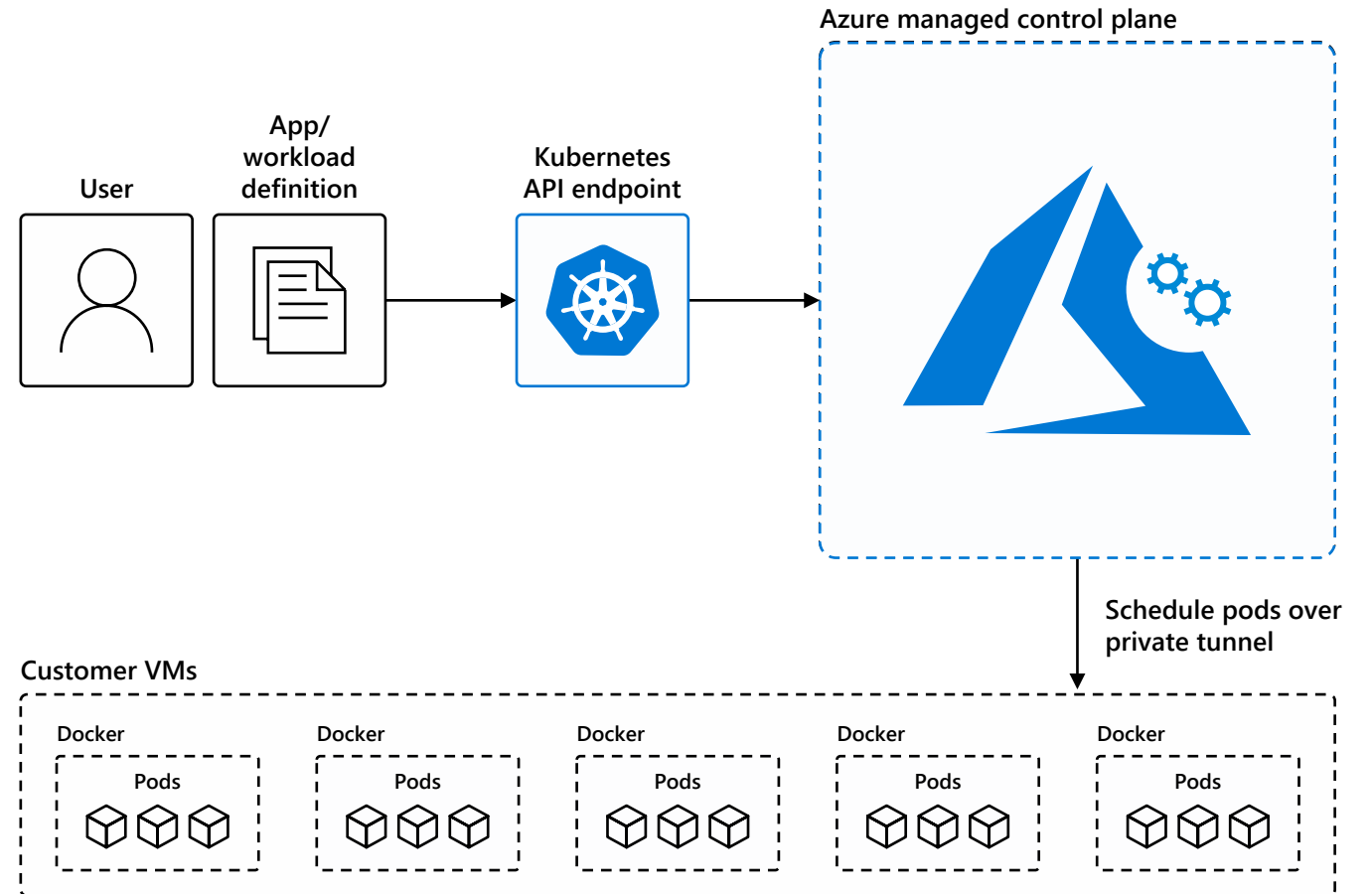


1. Azure Kubernetes Service (AKS)
2. Google Kubernetes Engine (GKE)
3. Amazon Elastic Kubernetes Service (EKS)

Shared Responsibility

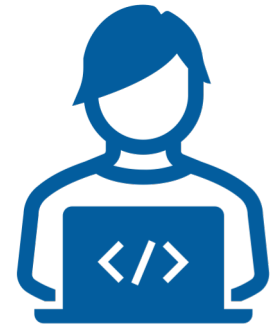
Responsibilities	DIY with Kubernetes	Managed Kubernetes on Azure
Containerization	■	■
Application iteration, debugging	■	■
CI/CD	■	■
Provisioning, upgrades, patches	■	■
Reliability availability	■	■
Scaling	■	■
Monitoring and logging	■	■

■ Customer ■ Microsoft



DEMO

AZURE KUBERNETES SERVICE



Benefits of Kubernetes

Service discovery and load balancing

Scheduling - affinity/anti-affinity

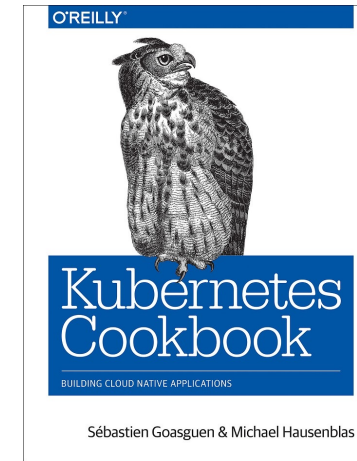
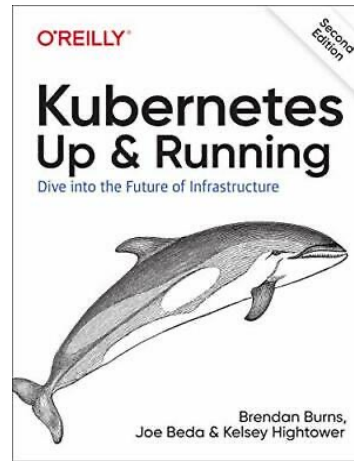
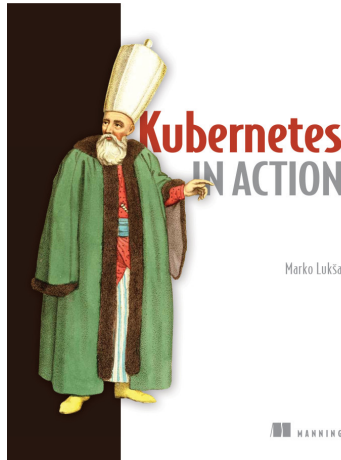
Automated rollouts and rollbacks

Health monitoring and self-healing

Secrets and configuration management

Storage orchestration

Further Learning



1. Kubernetes Documentation - <https://kubernetes.io/docs/home/>
2. Kubernetes Training and Certification - <https://kubernetes.io/training/>
3. Microsoft Learn - <https://docs.microsoft.com/en-us/learn/modules/intro-to-kubernetes/>
4. Kubernetes Slack Channel - <https://slack.kubernetes.io/>

Q&A

Contact Information



 <https://vaibhavgujral.com>

 [@vaibhavgujral_](https://twitter.com/vaibhavgujral_)

 <https://www.linkedin.com/in/vaibhavgujral/>

 <https://www.youtube.com/c/VaibhavGujral>

 vaibhav@vaibhavgujral.com



LinkedIn



Twitter



Email

Slides

**Thank
You!**