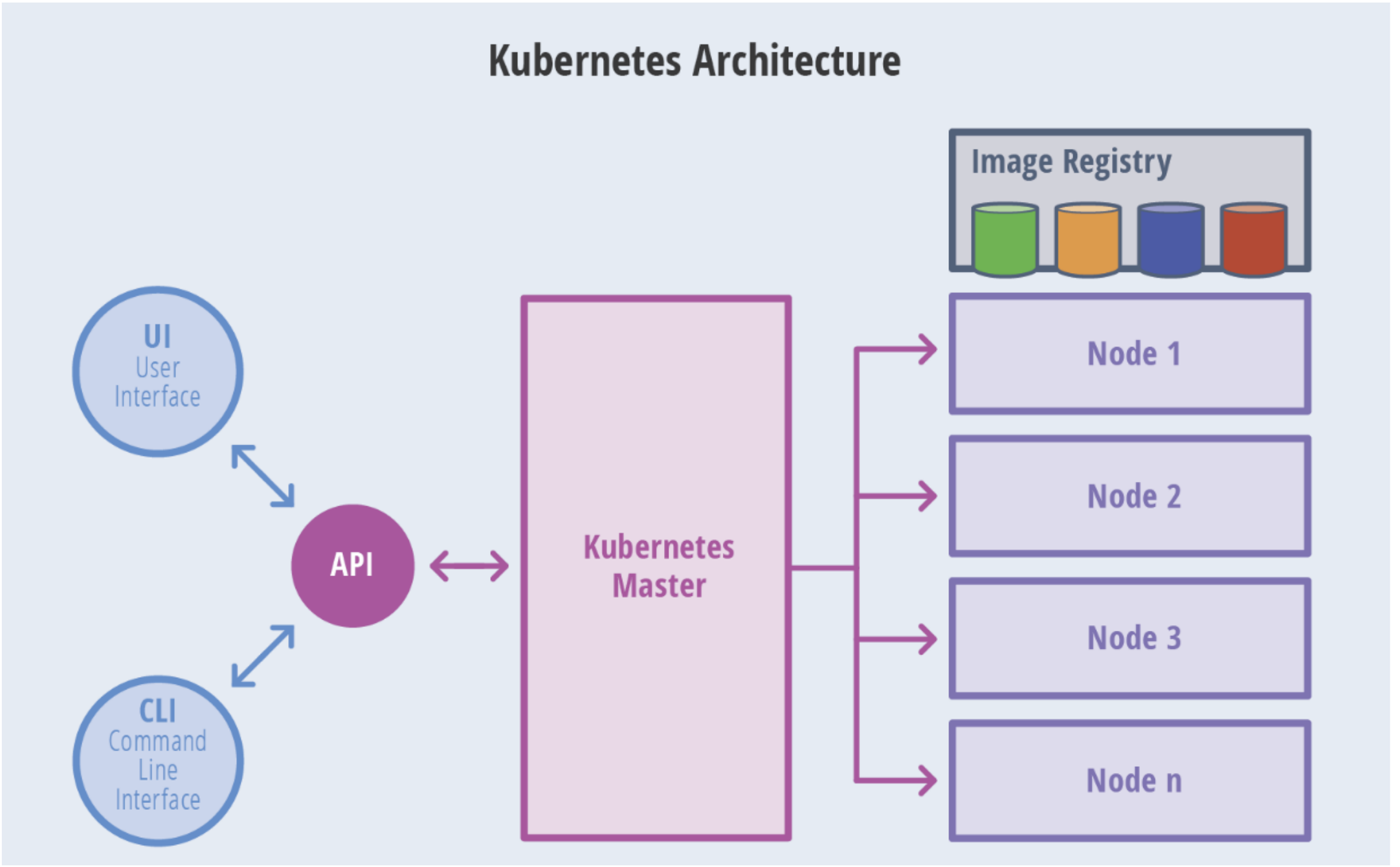

Kubernetes Architecture
Image Registry
UI
User
Interface
API
CLI
Command
Line
Interface
Kubernetes
Master
Node 1
Node 2
Node 3
Node n

Kubernetes

Cheat sheets, Practice Exams and Flash cards 👉 www.exampro.co/kcna

kubernetes

Kubernetes is an **open-source container orchestration system** for automating **deployment, scaling and management** of containers.

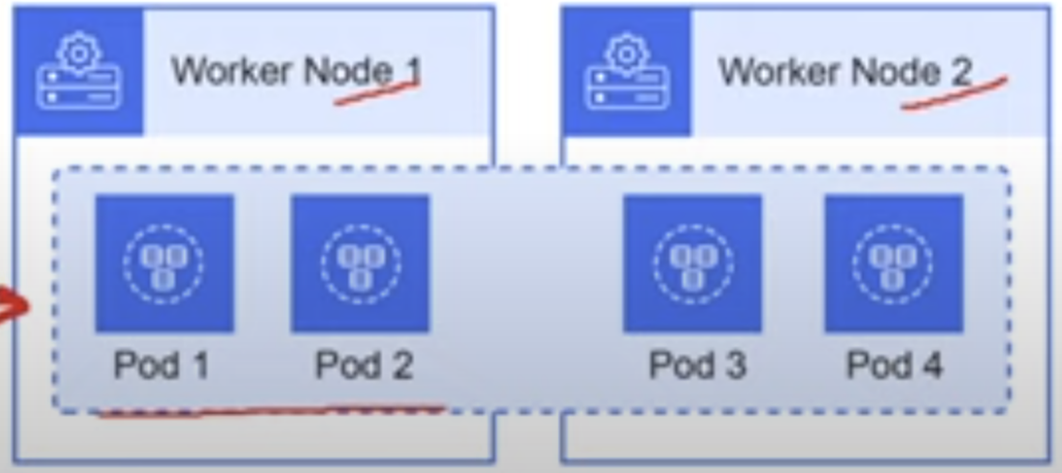
Originally created by Google and now maintained by the **Cloud Native Computing Foundation (CNCF) as a CNCF Project**

Kubernetes is commonly called **K8s**

- The 8 represent the remaining letters "ubernete"

**The advantage of Kubernetes over Docker is the ability to run "container apps" distributed across multiple VMs

A unique component of Kubernetes are **Pods**.
A pod is a group of one more containers with shared storage, network resources and other shared settings.



Kubernetes Components Overview

Cluster
A logical grouping of all components within a cluster.

Namespace
A named logical grouping of Kubernetes components within a cluster.
Used to Isolate different workloads on the same cluster

Node
A virtual machine or underlying server. There are two types of nodes: Control Plane and Worker nodes.
Worker nodes is where your application or workloads run. Control Plane nodes manage worker nodes.

Pod
The smallest unit in K8s. It is an **abstraction over a container.**
Generally defines an application workload

Service
A static IP address and DNS name for a set of pods (persists an address even if a pod dies) and a load balance

A "service" can also mean a container that continuously runs.

Ingress
Translates HTTP/S rules to point to services

Kubernetes Components Overview

Cheat sheets, Practice Exams and Flash cards 👉 www.exampro.co/kcna

API Server
The API Server allows users to interact with K8s components using the KubeCTL or by sending HTTP requests.

Kubelet
Kubelet is an agent installed on all nodes. Kublelet allows users to interact with node via the API Server and KubeCTL

KubeCTL
A command line interface (CLI) that allows users to interact with the cluster and components via the API Server

Cloud Controller Manager
Allows you to link a Cloud Service Provider (CSP) eg. AWS, Azure GCP to leverage cloud services.

Controller Manager
A control loop that watches the state of the cluster and will change the current state back to desired state.

Scheduler
Determines where to place pods on nodes. Places them in a scheduling a queue

Kube Proxy
An application on worker nodes that provides routing and filtering rules for ingress (incoming) traffic to pods.

Network Policy
Acts as a virtual firewall at the namespace-level or pod-level

Kubernetes Components

ConfigMap
allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable. Used to store non-confidential data in key-value pair

Secret
small amount of sensitive data such as a password, a token, or a key

Volumes
mounting storage eg. locally on the node, or remote to cloud storage

StatefulSet
provides guarantees about the ordering and uniqueness of these Pods

- Think of databases where you have to determine read and write order or limit the amount of containers
- StatefulSets are hard, when you can host your db externally from K8s cluster

ReplicaSets
Maintain a stable set of replica pods running at a given time. Can provide a guarantee of availability.

Deployment
Is a blueprint for a pod (think Launch Template)

Manifest Files in Kubernetes

Manifest (not technical description)
A Manifest file is a document that is commonly used for customs to list the contents of cargo, or passengers. Its an itemized list of things.

Manifest File (in the context of Kubernetes)
A Manifest file is a generalized name **for any Kubernetes Configuration File** that define the configuration of various K8s components.

These are all Manifest files with specific purposes:

- Deployment File
- PodSpec File
- Network Policy File

Manifest Files can be written in either:

- YAML
- JSON

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80
```

YAML

```json
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "nginx"
  },
  "spec": {
    "containers": [
      {
        "name": "nginx",
        "image": "nginx:1.14.2",
        "ports": [
          {
            "containerPort": 80
          }
        ]
      }
    ]
  }
}
```

JSON

Manifest Files in Kubernetes

A manifest can contain **multiple** K8s component definitions/configurations.

In YAML you can see the **three hyphens ---** is used to defined multiple components.

kubectl apply command is generally used to deploy manifest files

```
kubectl apply -f resources.yml
```

Resource Configuration file is sometimes used to describe multiple resources in a manifest.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
    port: 80
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  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
```

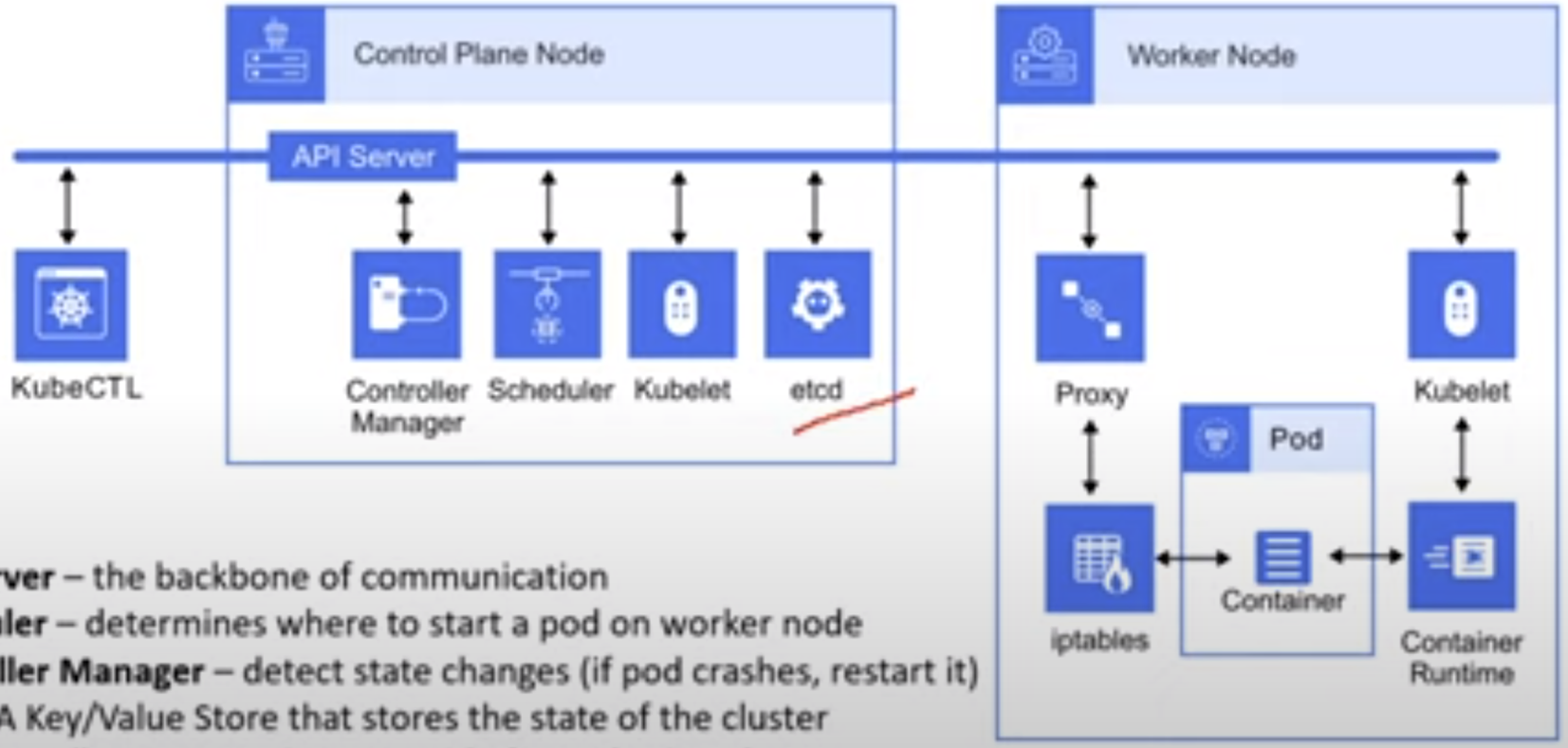
Nodes – Control Plane and Worker Nodes

Control Plane Node *(Formally known as Master Node)*

Manages processes like scheduling, restarting nodes...

Worker Node

Does the work, running your app in pods and containers...



The worker node runs:

- Kubelet
- Kube Proxy
- Container Runtime
- Pods and Containers

- **API Server** – the backbone of communication
- **Scheduler** – determines where to start a pod on worker node
- **Controller Manager** – detect state changes (if pod crashes, restart it)
- **etcd** – A Key/Value Store that stores the state of the cluster
- **Kubelet** – Allows user to interact with the node via KubeCTL

Pods

Pods are the smallest unit in Kubernetes.
Pods abstract away the container layer so you can directly interact with the Kubernetes Layer.

A Pod is intended to run one application in multiple containers

- Database Pod, Job Pod, Frontend App Pod, Backend App Pod
- You can run multiple apps in a pod but those containers will tightly dependent.

Each Pod gets it own private IP address

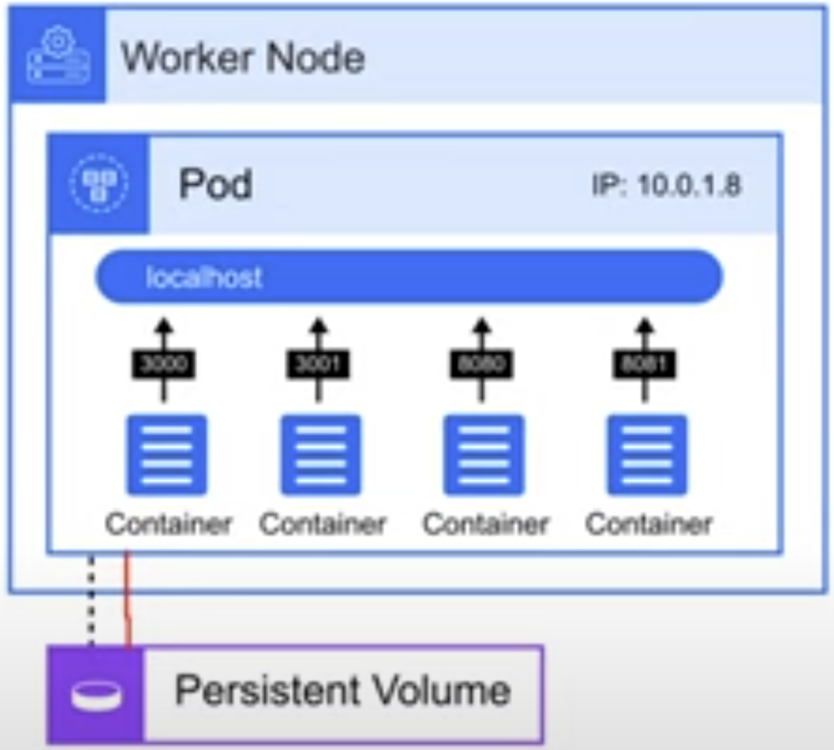
- Containers will run on different ports
- Containers can talk to each other via localhost

Each Pod can have a shared storage volume attached.

- All containers will share the same volume

When the last remaining container dies (maybe crashes) in a pod so does the pod

- When a replacement pod is created, the pod will have an IP address will be assigned.
  - IP addresses are Ephemeral, (temporary) for pods, they don't by default persist.




```
kubectl get pod -o wide
```

Get pods and **show their IP addresses**

API Server

The core of Kubernetes control plane is the API server
The API server exposes an HTTP API that lets end users, different parts of your cluster, and external components communicate with one another.

The Kubernetes API lets you query and manipulate the state of API objects in Kubernetes (for example: Pods, Namespaces, ConfigMaps, and Events).

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is kube-apiserver.
kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances.

You can run several instances of kube-apiserver and balance traffic between those instances.

Everything has to go through the API Server.
You can interact with the API Server in three ways:

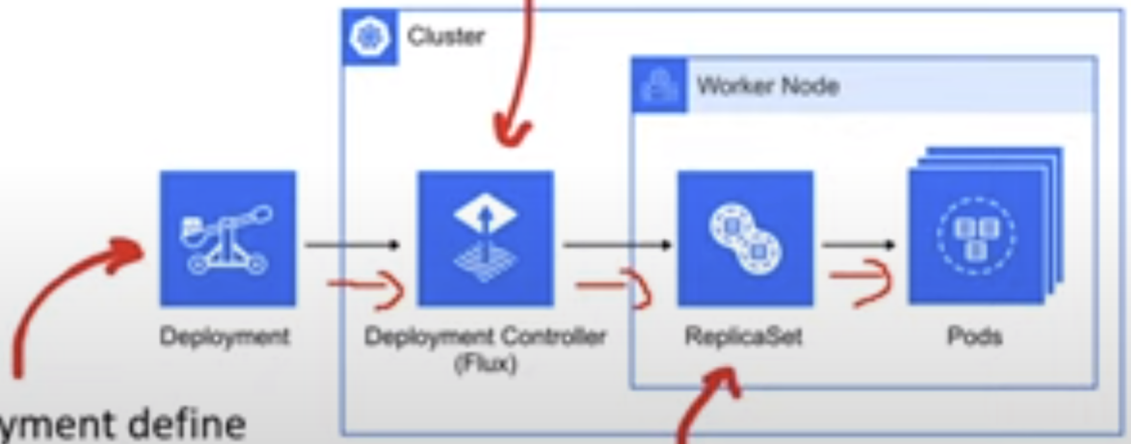
- UI*
- API
- CLI KubeCTL

Deployment

A Deployment provides declarative updates for Pods and ReplicaSets.

A Deployment Controller changes the actual state to the desired state at a controlled rate.

- The default Deployment Controller can be swapped out for other deployments tools eg:
  - Argo CD, Flux, Jenkin X.....



A Deployment define the desired state of ReplicaSets and Pods.

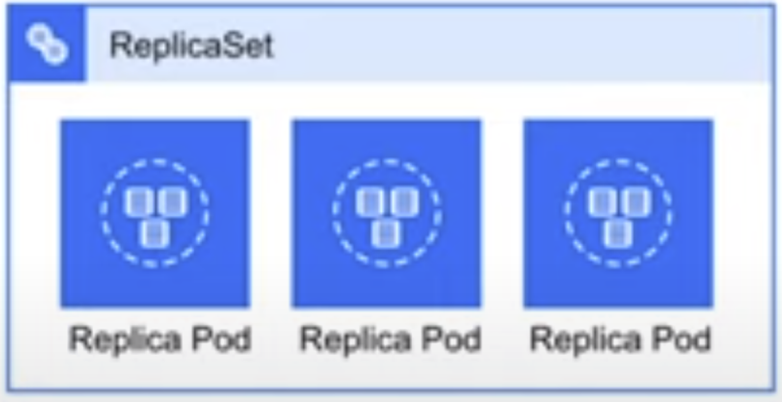
A deployment will create and manage a ReplicaSet.
A ReplicaSet will manage replicas of pod.

```yaml
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
```

Replica Sets

ReplicaSet is a way to **maintain a desired amount** of redundant pods (replicas) to provide a guarantee of availability.

The pod field **metadata.ownerReferences** determines
The link from a pod to a ReplicaSet.




```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

It is not recommended to directly create ReplicaSets
Instead a Deployment can create and manage a ReplicaSet for you.

Horizontal Pod Autoscaler (HPA) can be used to autoscale a ReplicaSet

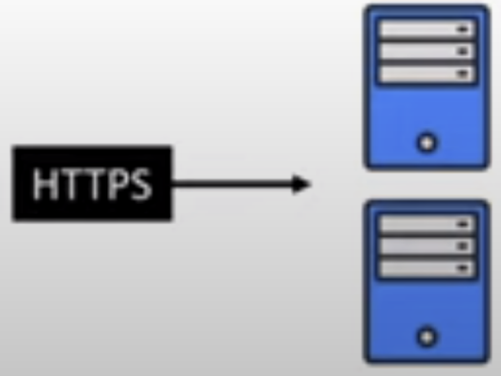
Stateless vs Stateful

Stateless

ReplicaSets

Every request does not care (forgets) about previous or current state

Web-apps that store sessions outside of the web-apps virtual machine (database or cookies) will be stateless web-apps



Send Request to any server, it doesn't matter

Stateful

StatefulSets

Every request relies on state data (remembers)
All databases are stateful

Monolithic web-apps that store session data in memory on a Virtual Machine are Stateful

Remember to send writes only to primary

Stateful Sets

Stateful Sets are used when you need traffic to be sent to a specific pods.

Stateful Set will always have:

- a unique and predictable name and address
- ordinal index number assigned each pod
- a persistent volume attached, with a persistent link from pod to the storage
  - If a pod is rescheduled the original Persistent Volume (PV) will be mounted to ensure data integrity and consistency.
- Stateful Set pods will always start in the same order, and terminate in reverse order

StatefulSets currently require a “headless” service to manage the identities

There is a headless service used to maintain the network identity of the pods, and another service that provides read access to the pods

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labe
  serviceName: "nginx"
  replicas: 3 # by default is 1
  minReadySeconds: 10 # by default is 0
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

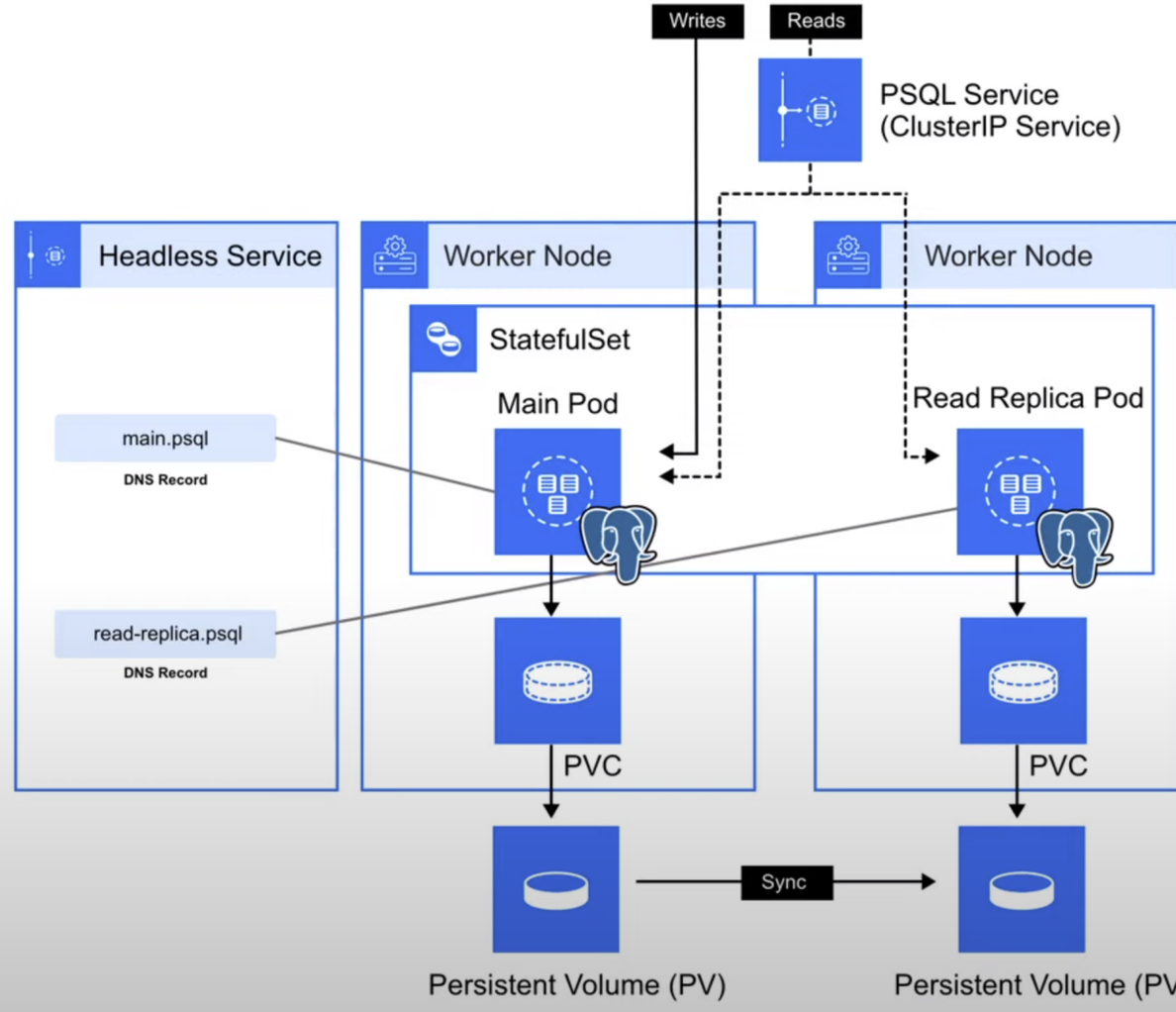
Stateful Sets

DNS Hostname for Writes
Writes will be directed to the Main pod by its DNS Hostname which is identified by a Headless Service

ClusterIP for Reads
For read traffic it can be distributed to all reading pods using a ClusterIP Service

Headless Service
The headless service is a Service with ClusterIP set to none.
It does not provide load balancing
It does not provide a static IP address
A Headless Service is used to identify specific pods by assigning them DNS record.
A Headless Service is required in order for a a StatefulSet to work.

PVC and PV
Each pod has its own volume. A Persistent Volume Claim can dynamically reference a Persistent Volume.



Namespaces

A Namespace is a way to logically isolate resources within a Kubernetes Cluster.
Namespaces can be organized based on project, department or any user-defined grouping.

Kubernetes starts 4 initial namespaces

To **view all** namespaces

```
kubectl get namespace
```

1. default

- The default namespace where all our pods and services run unless a namespace is specified.

2. kube-public

- For resources that are publicly visible and readable.

3. kube-system

- The system namespace that stores objects created by the Kubernetes System.
- Engineers deploying applications are not supposed to touch this namespace.

4. kube-node-lease

- Holds lease objects associated with each node. Used to detect node failures by sending heartbeats

You can **create** your own namespaces

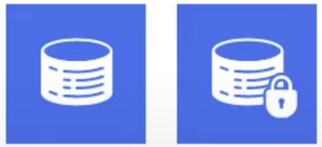
```
kubectl create namespace production
```

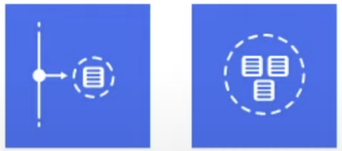
In clusters with a small amount of resources, Namespaces aren't necessary.

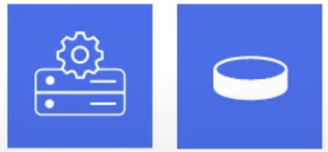
Namespaces

Names of resources need to be unique within a namespace, but not across namespaces.
Namespace-based scoping is applicable only for namespaced objects eg. *Deployments, Services*
But not for cluster-wide objects eg. *StorageClass, Nodes, PersistentVolumes*

Single-Namespace Objects	Multi-Namespace Objects	Cluster-Wide Objects
ConfigMaps and **Secrets** can't be shared across namespaces	A **Service** and **Pods** can belong to multiple namespaces	**Volumes** and **Nodes** cannot be bound to namespace since they are global







You can apply **system quota restrictions** on a namespace to avoid overuse eg Mem, Compute

If you don't provide a namespace for a component it will end up in the default namespace

In-Tree vs Out-of-Tree

In Cloud-Native Projects you'll hear the term "In-Tree" and "Out-of-Tree"

In-Tree:
Plugins, components or functionality that are provided by default and/or reside in the main repository
Think of In-Tree as ***Internal plugins***

Out-of-Tree:
Plugins, components or functionality that must be installed manually, and extends or replaces the default behaviour.
Think of Out-of-Tree as ***External plugins***

This is Andrew's own definition of In-Tree vs Out-of-Tree as the term does not appeared to be standardized.

Graduated

View on the CNCF landscape →

Container Runtime

Coordination & Service Discovery

Service Proxy

Coordination & Service Discovery

Logging

Container Registry

Application Definition & Image Build

Tracing

Scheduling & Orchestration

Service Mesh

Security & Compliance

Monitoring

Cloud Native Storage

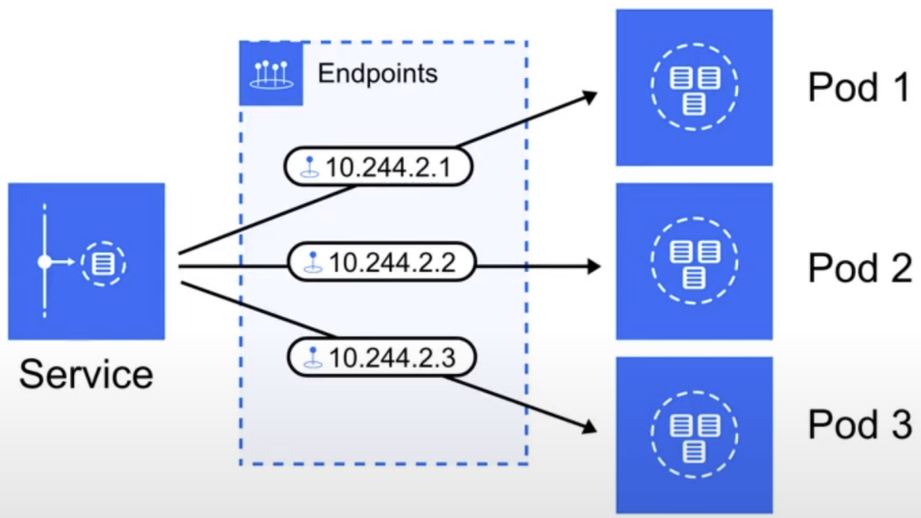
Security & Compliance

Database

Database

Endpoints and Endpoint Slices

Endpoints track the IP Addresses of the pods assigned to a Kubernetes Service.
When a service selector matches a pod label, The pod IP Address is added to the pool of endpoints
Pods expose themselves to services via endpoints.




```
kubectl get endpoints
```

To see a list of endpoints

Endpoint Slices break up Endpoints into smaller manageable segments.
Each Endpoint Slice has a limit of 100 pods.

Jobs and CronJobs

What is a Background Job?

A background job is a one-off task that is used to run a piece of code.
Commonly used to perform maintenance or to trigger a computational workload.
Eg. Back up the database ever x minutes, Delete all users who have not confirmed their email

A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate.

```
kubectl create job hello --image=busybox -- echo "Hello World"
```

A CronJob is a job that execute based on a repeating schedule.

```
kubectl create cronjob hello --image=busybox   --schedule="*/1 * * * *" -- echo "Hello World"
```