THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

Cloud Computing (INFS3208)

Lecture 5: Kubernetes

Lecturer: Dr Heming Du

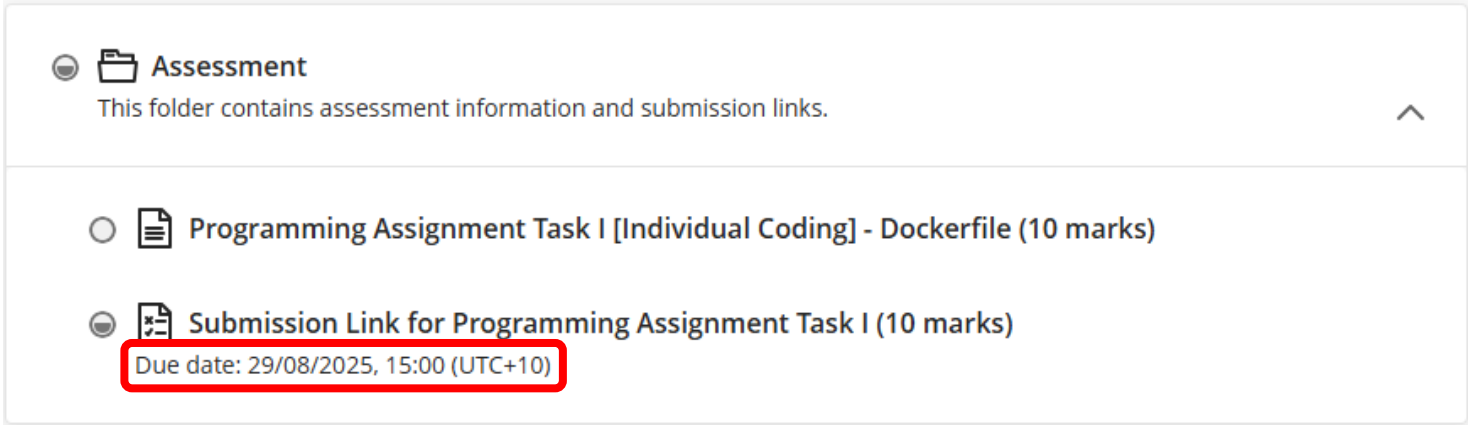
School of Information Technology and Electrical Engineering

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

The reminder of Programming Task I Submission

- Due:
  - **3 p.m. on Friday, 29/08/2025 AEST.**
- Extension application
  - **Must have** a medical certificate
  - by **5 p.m. Wednesday 27 Aug 2025**
  - **Late submission policy**.
- Ed Discussion (more tutors).
- The submission link is available under "Assessment" on the course website



Re-cap

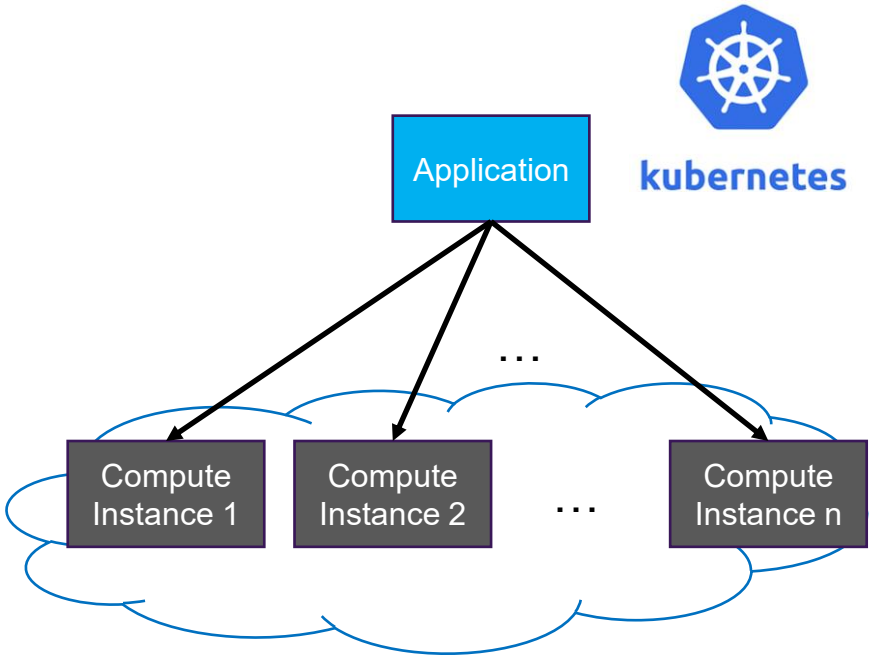
- Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

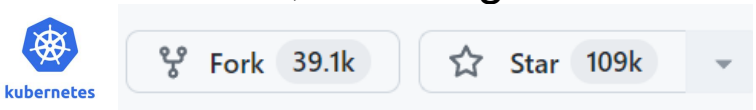
Outline

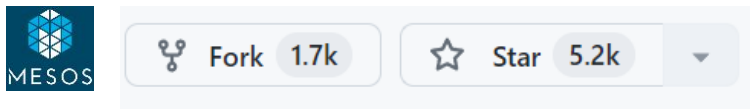
- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

What is Kubernetes?

- Kubernetes (koo-ber-net-eez, commonly stylised as K8s) is an **open-source container-orchestration system** for managing (deploying and scaling) containerised applications across multiple hosts
- K8s provides automating deployment, scaling and management of containerised applications.
- It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.
- It aims to provide a "**platform for automating deployment, scaling, and operations of application containers across clusters of hosts**."
- It works with a range of container tools, including **Docker**.







https://en.wikipedia.org/wiki/Kubernetes
https://kubernetes.io/case-studies/

Why Kubernetes?

- **Scaling & High Availability**:
  - automatically scales applications based on demand without manual intervention.
  - ensures that there are always a specified number of pods running (across multiple nodes), which leads to high availability.
- **Self-healing**:
  - automatically restarts containers that fail, replaces them, and even reschedules containers when nodes die.
  - If a container doesn't respond to user-defined health checks, it's automatically terminated and replaced.
- **Automated Rollouts and Rollbacks:** ensures that updates to applications roll out progressively while monitoring application health to ensure it doesn’t kill all instances simultaneously. If something goes wrong, Kubernetes can roll back the change.
- **Integrated Developer Tools**:
  - There's a broad selection of tools that have been built around the Kubernetes ecosystem, providing enhanced capabilities such as CI/CD, monitoring, logging, and more.
- **More Features**: Automated Service Discovery, Load Balancing, and Storage Orchestration, Secret and Configuration Management, Multi-cloud and Hybrid-cloud Capability (Platform Agnostic), Large Community and Strong Ecosystem, etc.

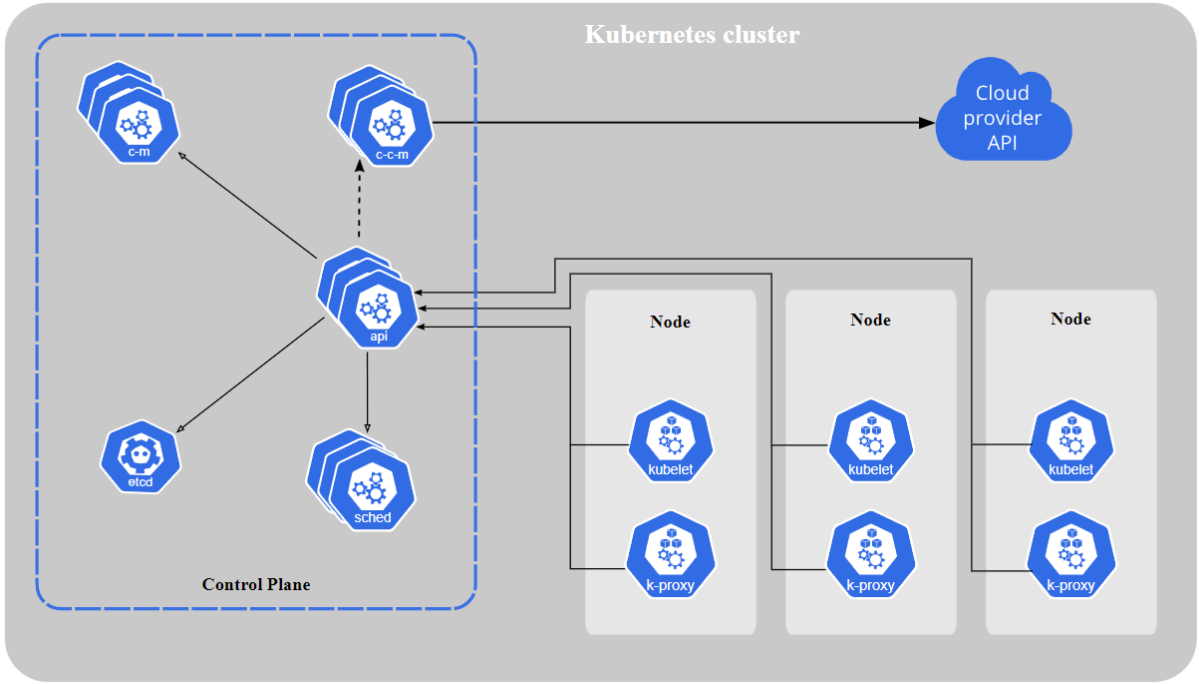
Outline

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Models in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

Kubernetes Architecture

A cluster of machines:

- **Control Plane (Master Node)**
  - Manages the (worker) nodes
    - Schedule (pod),
    - detect or respond to cluster events
  - Provides User Interface
  - FOUR processes need to run in Control Plane:
    - API Server, Scheduler, Controller Manager, Etcd
  - Multi-master high availability (**HA**) is a must-have
- **Worker Nodes (Slave Nodes)**
  - Conduct actual works with multiple pods
  - THREE processes need to run on every node
    - Container runtime, Kubelet, Kube proxy

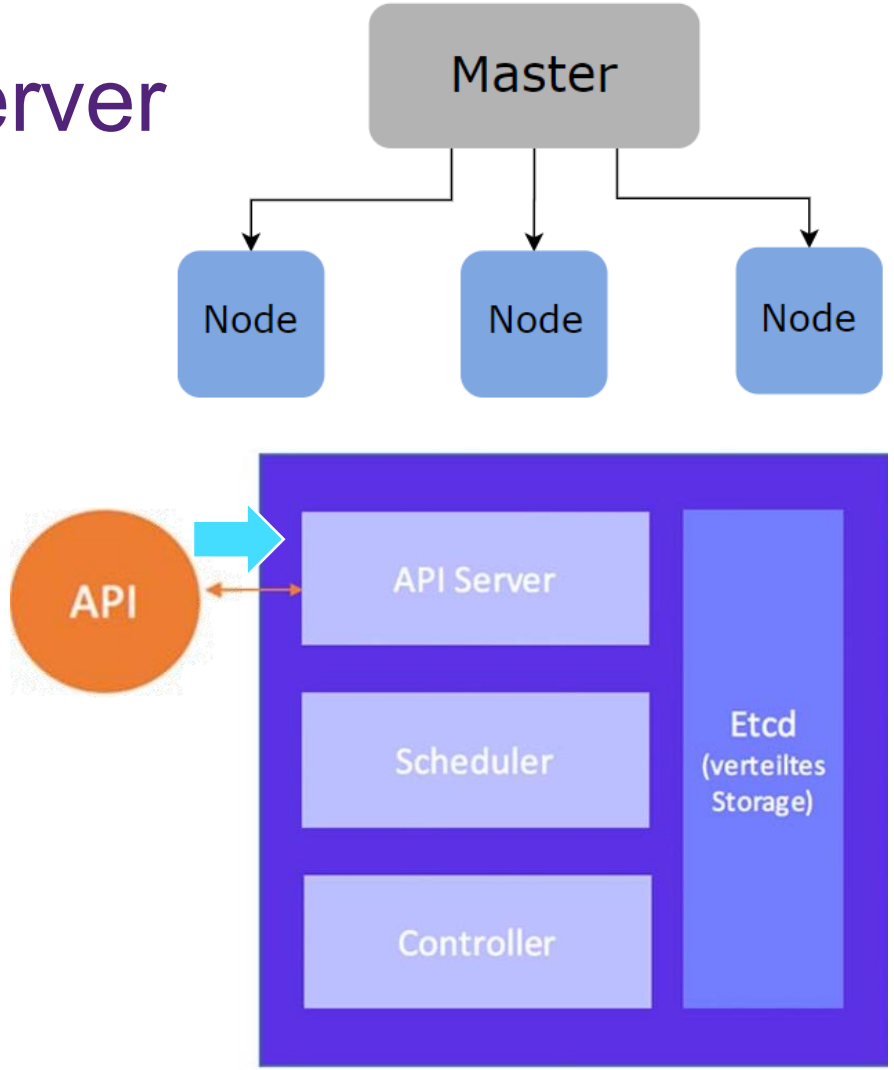


https://kubernetes.io/docs/concepts/overview/components/

Control Plane Components – API Server

API Server

- Central component in Control Plane
  - an interactive interface (cluster gateway) between the cluster and the client (e.g. cmd/UI)
- Updates and queries between the cluster and the client (e.g. kubectl)
  - **Kubectl**: a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes APIs.
- Provides k8s APIs and authentication (i.e. gatekeeper)
- Implementation: ***kube-apiserver*** (horizontally scalable)

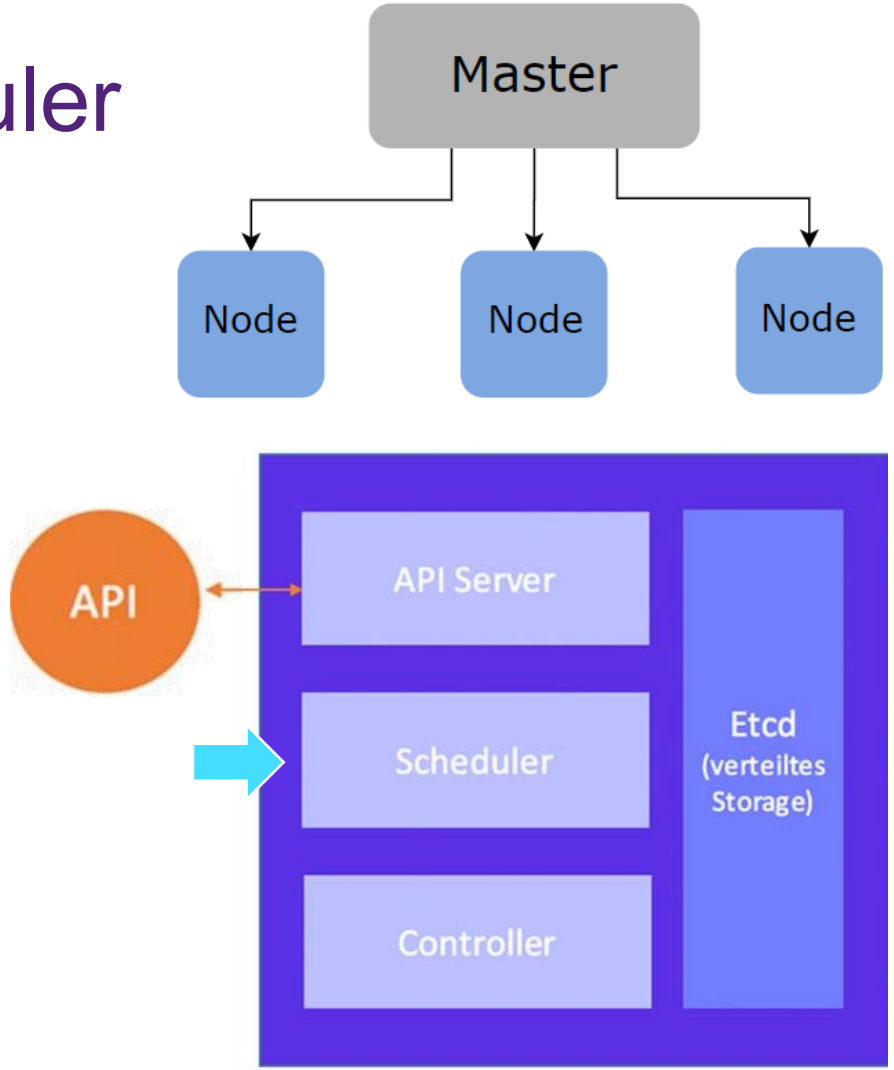


Source:
https://knowitinfo.com/what-is-kubernetes-control-plane/
https://kubernetes.io/docs/concepts/overview/components/

Control Plane Components - Scheduler

Scheduler

- Watch for the unassigned 'task'
- Assign it to a node that has available resources (CPU & memory) matching the requirement
  - Kubelet (on Node) actually executes the scheduled pods.
- Perform predicate checks and rank nodes
- Many factors can be considered when scheduling a task:
  1. Individual and collective resource requirements
  2. Hardware/software/policy constraints
  3. Data locality
  4. Etc.
- Implementation: ***kube-scheduler***



Source:
https://knowitinfo.com/what-is-kubernetes-control-plane/
https://kubernetes.io/docs/concepts/overview/components/

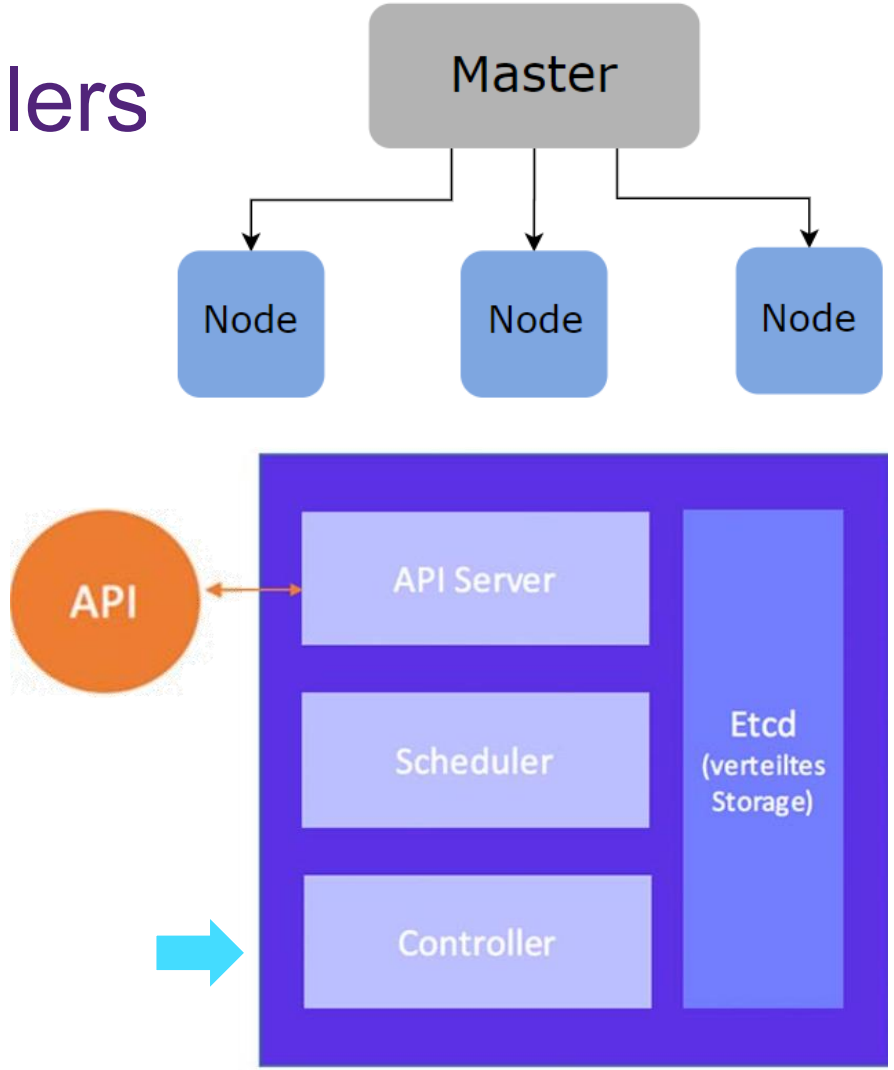
Control Plane Components - Controllers

Controllers

- Node controller - Responsible for noticing and responding when nodes go down.
- Job controller - Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
- Endpoint controller - Populates the Endpoints object (e.g. pod-ip:service-port).
- Namespace controller & Service Account controller & Token controller, etc.

Controller Manager – controller of controllers

- Detects cluster state changes (pods dead, etc)
- Informs Scheduler for re-scheduling
- Implementation: ***kube-controller-manager***

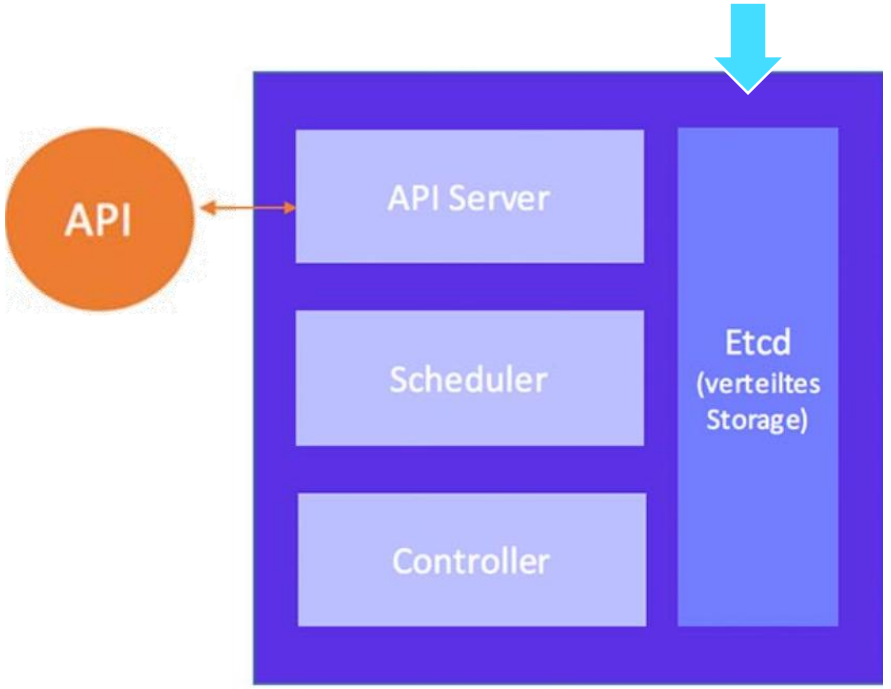


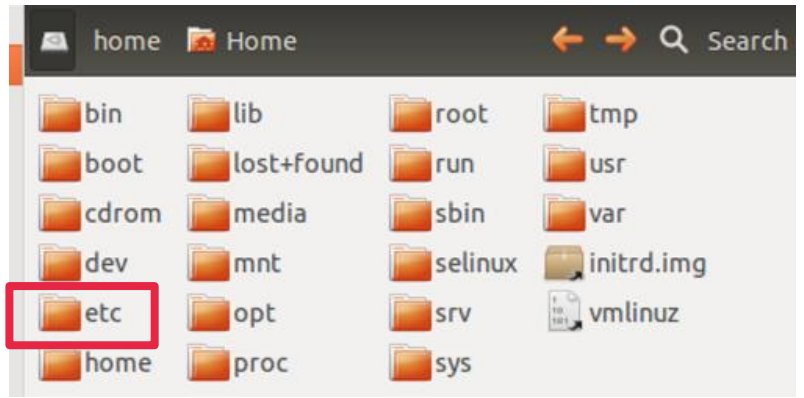
Source:
https://knowitinfo.com/what-is-kubernetes-control-plane/
https://kubernetes.io/docs/concepts/overview/components/

Control Plane Components - Etcd

Etcd

- a distributed key-value (JSON) store database
  - Any information about the cluster
    - e.g. Cluster configuration
  - All state changes (cluster changes)
    - e.g. Cluster resource condition
    - e.g. Actual state and desired state of the system
- Secure
  - automatic Transport Layer Security (TLS)
- Fast and highly-available
  - Over **30,000 requests per second** under heavy load
  - **Raft** consensus algorithm
- Application data is NOT stored in Etcd.





Source:
https://knowitinfo.com/what-is-kubernetes-control-plane/
https://kubernetes.io/docs/concepts/overview/components/
https://etcd.io/docs/v3.4/op-guide/performance/#:~:text=In%20common%20cloud%20environments%2C%20like,per%20second%20under%20heavy%20load.

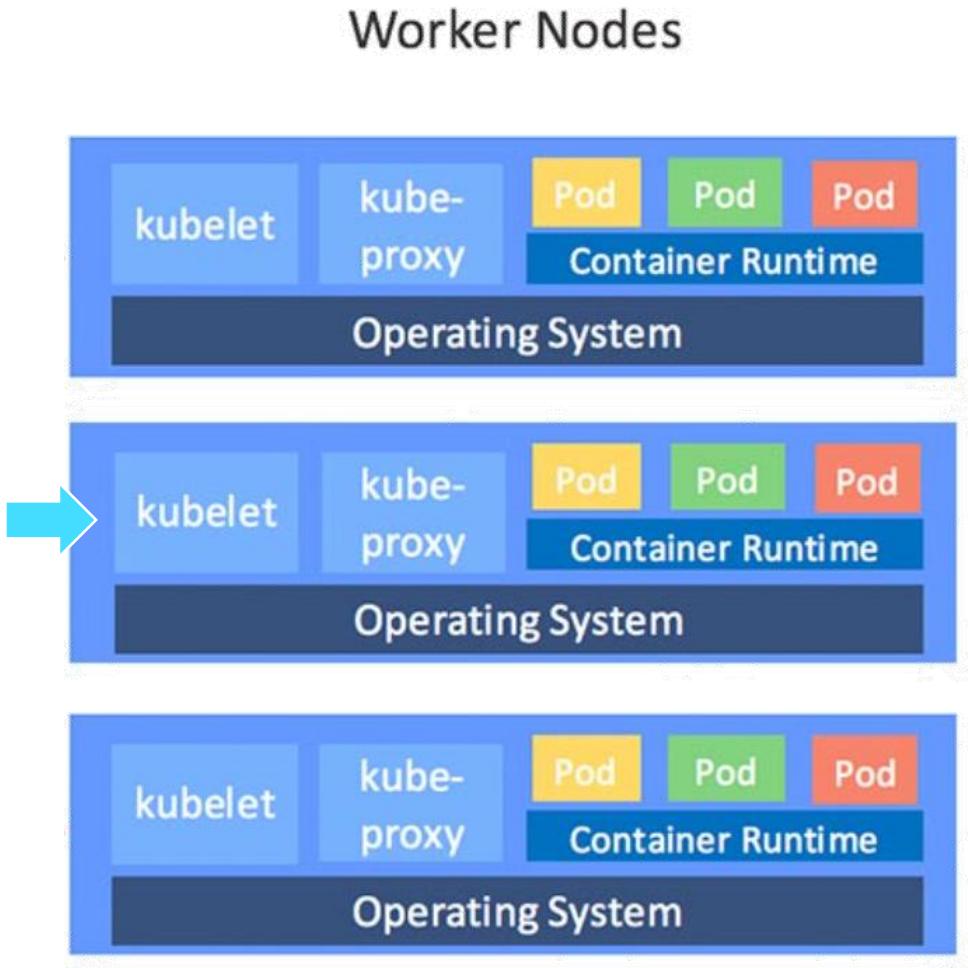
Worker Node Components - Kubelet

Revisited - Worker Nodes (Slave Nodes)

1. Watch the API server for new work assignments
2. Execute new work assignments
3. Report back to the control plane (via the API server)

Kubelet

- **Core agent** - runs on every worker node in the cluster
- Interacts with both the **container** and **node**
  - Registers the node(s) within the cluster
  - Watches the API server,
  - Executes the task - starts the pod with a container inside
  - Maintains the reporting channel and Reports task failure



Source: https://knowitinfo.com/what-is-kubernetes-control-plane/

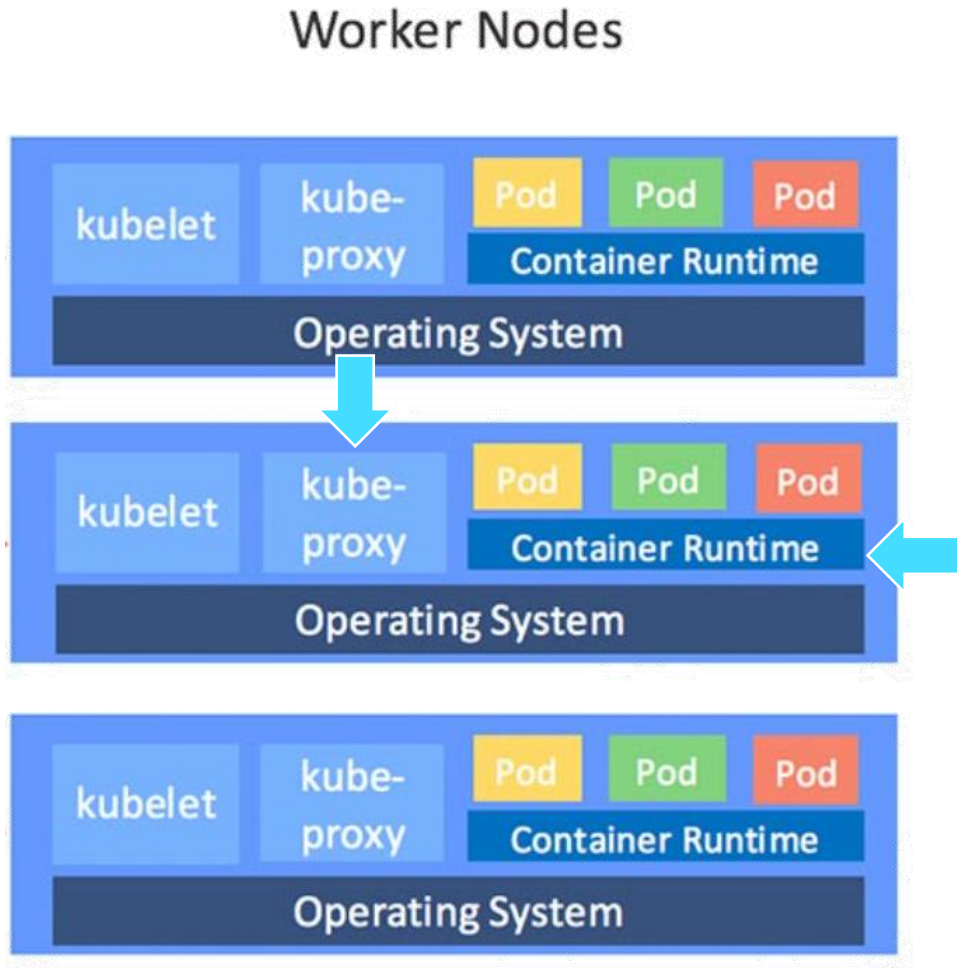
Worker Node Components – CR and K-Proxy

Container Runtime

- Software that is responsible for running containers.
- Perform container-related tasks
- Implementations: e.g. containerd and Kubernetes CRI (Container Runtime Interface)

Kube-proxy

- Network proxy that is responsible for local cluster networking
- Makes sure each node gets its unique IP address
- Maintains network rules on nodes
  - Handle routing and load-balancing (IPTABLES / IPVS)
- Implements parts of K8s Service concept
  - forwards the requests for communication, like a smart balancer

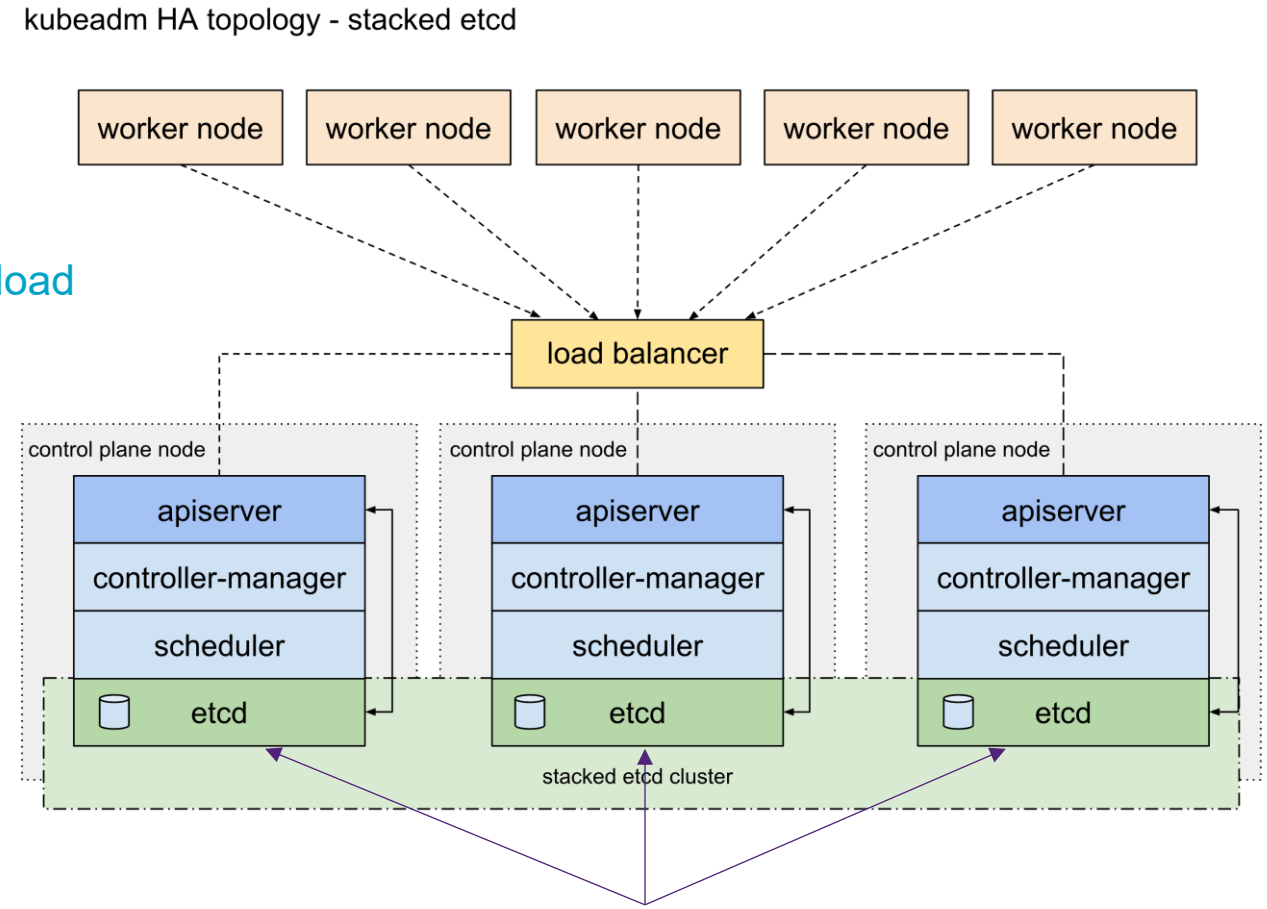


Source: https://knowitinfo.com/what-is-kubernetes-control-plane/

High-Availability for Control Plane (Master Node)

1. With stacked control plane nodes:

- Each CP runs an instance of the kube-apiserver, kube-scheduler, and kube-controller-manager.
- The kube-apiserver is exposed to worker nodes using a load balancer.
- Each CP node creates a local etcd member and it communicates only with the kube-apiserver of this node.
- CPs and etcd members on the same nodes.
  - simpler to set up than an external etcd cluster
  - simpler to manage for replication
  - requires less infrastructure (etcd members and control plane nodes are co-located)
- a stacked cluster runs the risk of failed coupling.
- a minimum of three stacked CP nodes for an HA cluster.



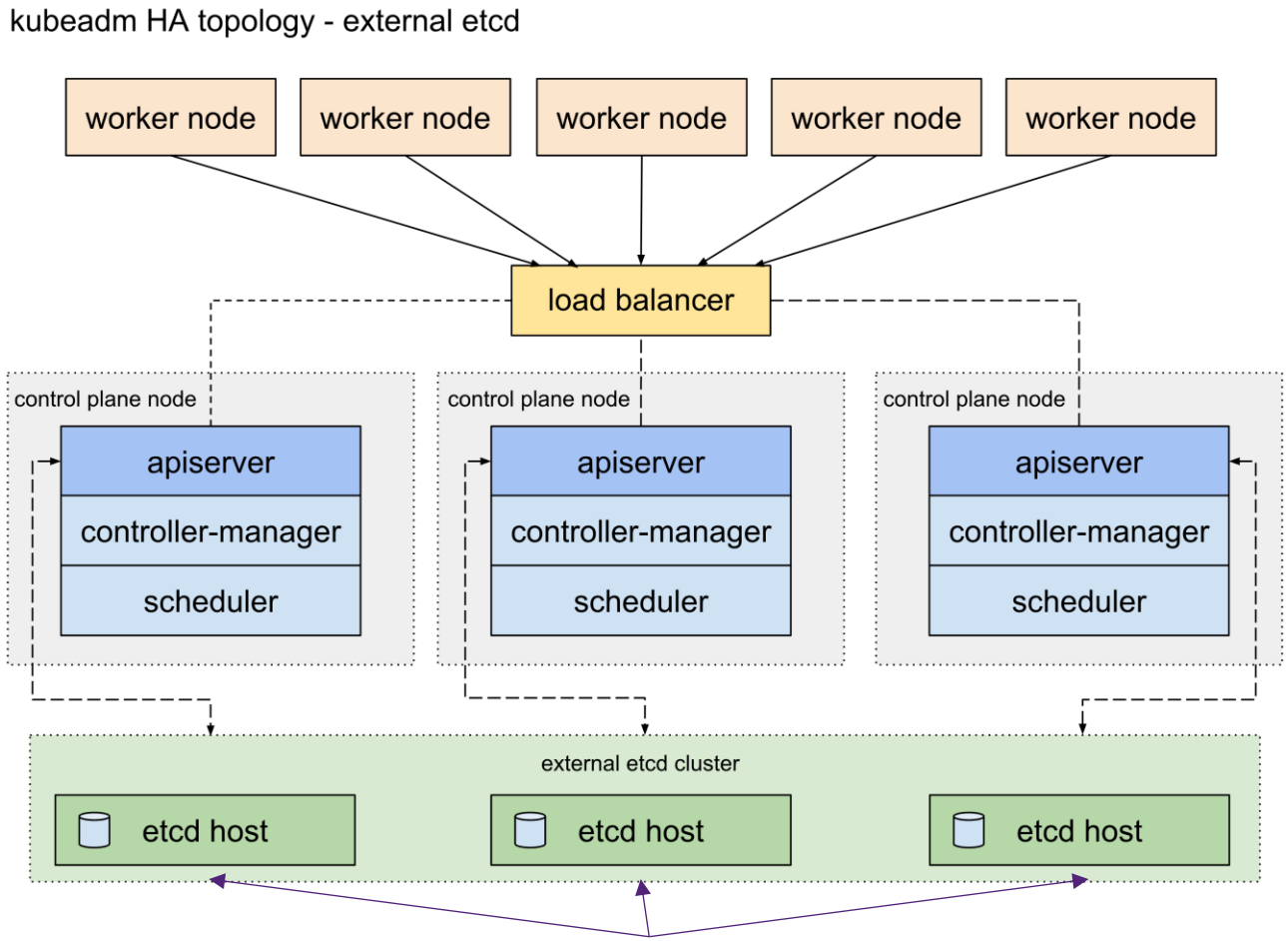
Etcd is stacked on top of the cluster formed by the nodes.

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

Multiple Control Planes (Master Nodes)

2. With an external etcd cluster:

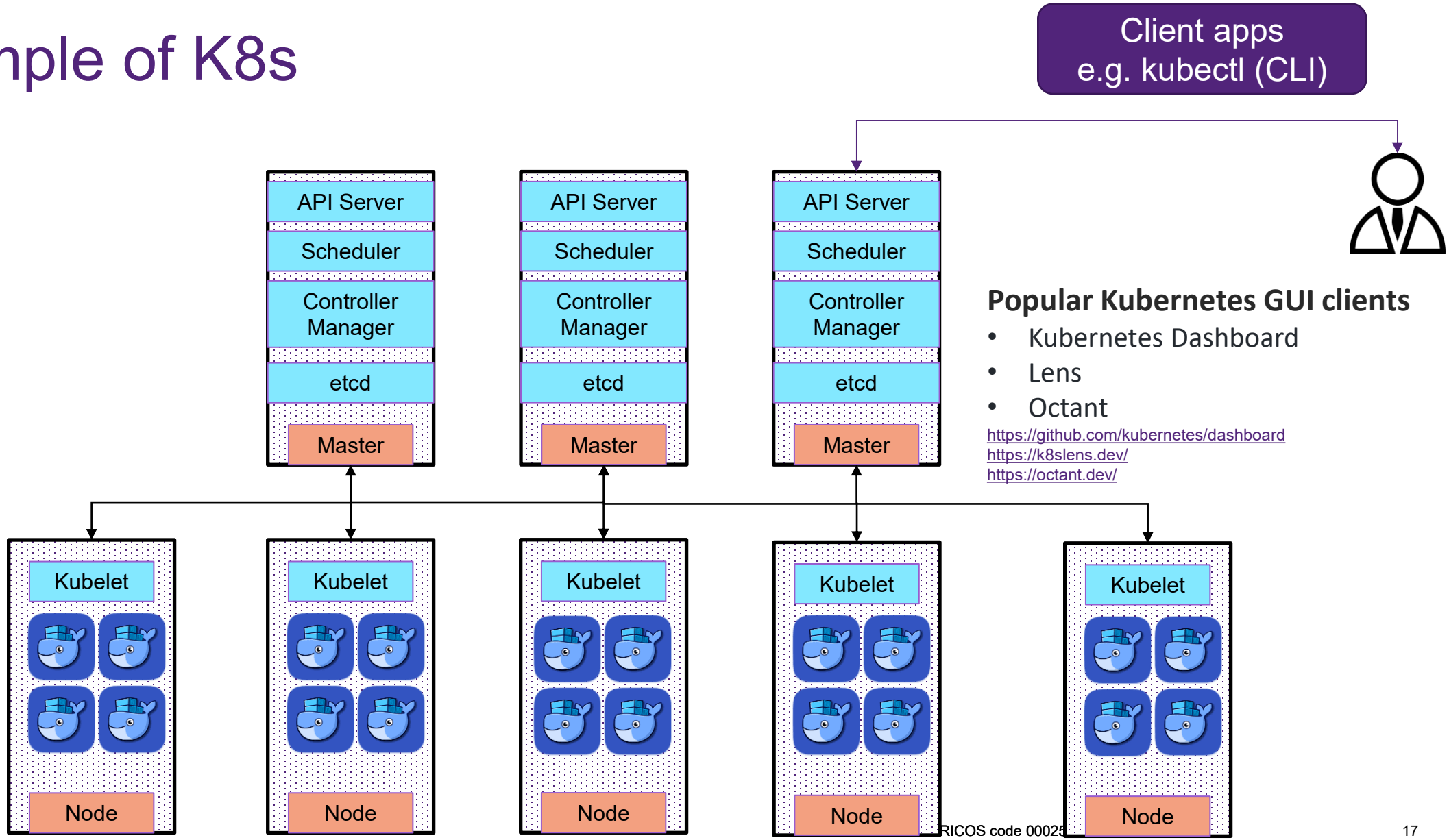
- etcd is an external cluster.
- Like the stacked etcd topology, each CP node runs an instance of the kube-apiserver, kube-scheduler, and kube-controller-manager.
- The kube-apiserver is exposed to worker nodes using a load balancer.
- CP and its etcd member are decoupled:
  - etcd members run on separate hosts,
  - Each etcd host communicates with the kube-apiserver of each CP node.
- The CP's failure has less impact and does not affect the cluster redundancy compared to stacked etcd topology
- But it requires more infrastructure (2x more hosts)



Three extra hosts for the external etcd cluster

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

Example of K8s



Popular Kubernetes GUI clients

- Kubernetes Dashboard
- Lens
- Octant

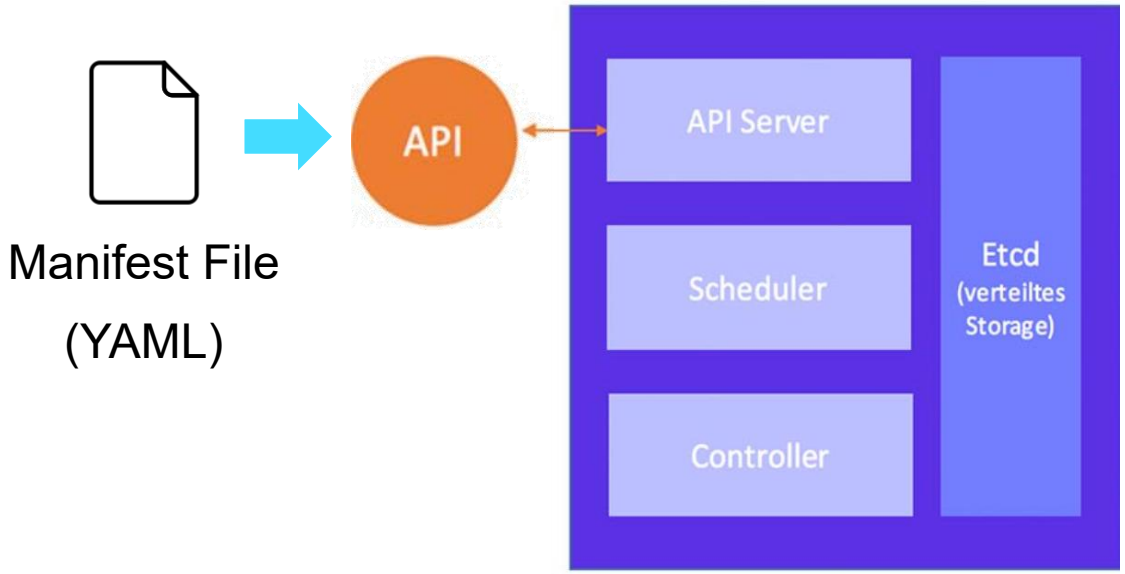
https://github.com/kubernetes/dashboard
https://k8slens.dev/
https://octant.dev/

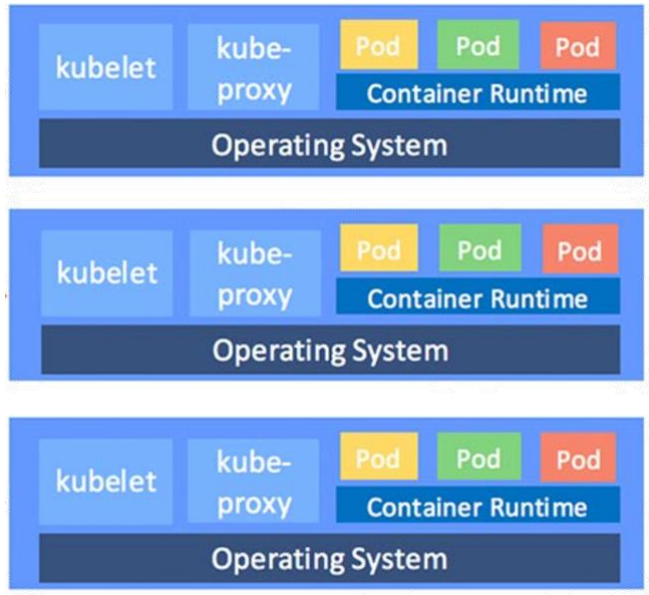
Outline

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

Declarative Model

- The declarative model and the concept of desired state are at the very heart of K8s.
- In K8s, the declarative model works like:



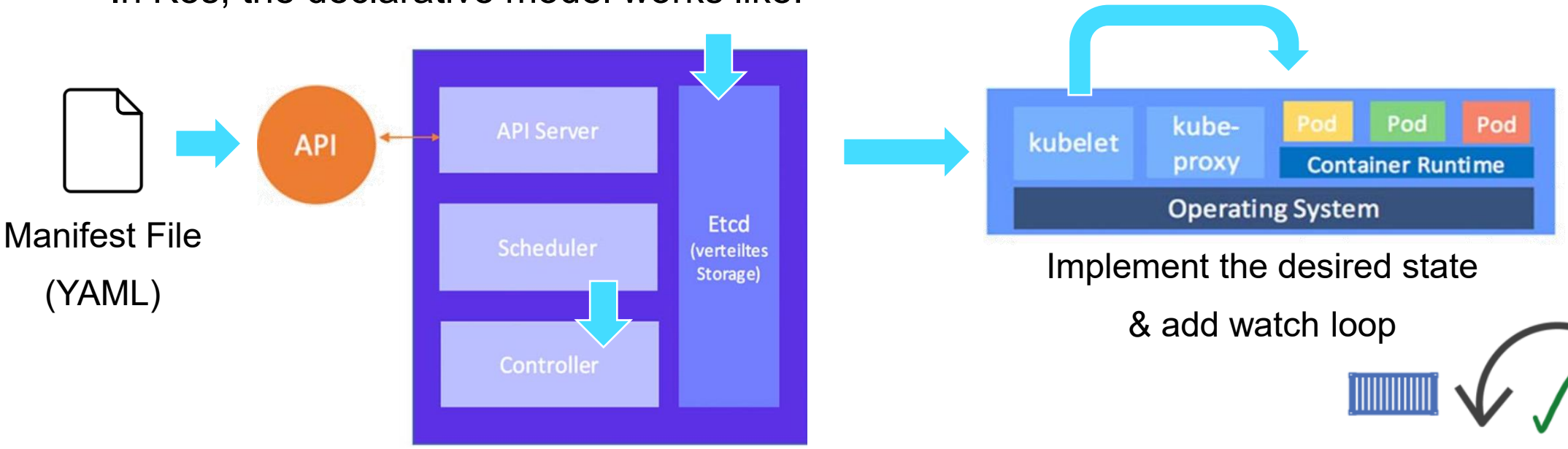


1.Declare the desired state of an application (microservice) in a manifest file
2.POST it to the API server

https://kubernetes.io/docs/concepts/workloads/pods/

Declarative Model

- The declarative model and the concept of desired state are at the very heart of K8s.
- In K8s, the declarative model works like:



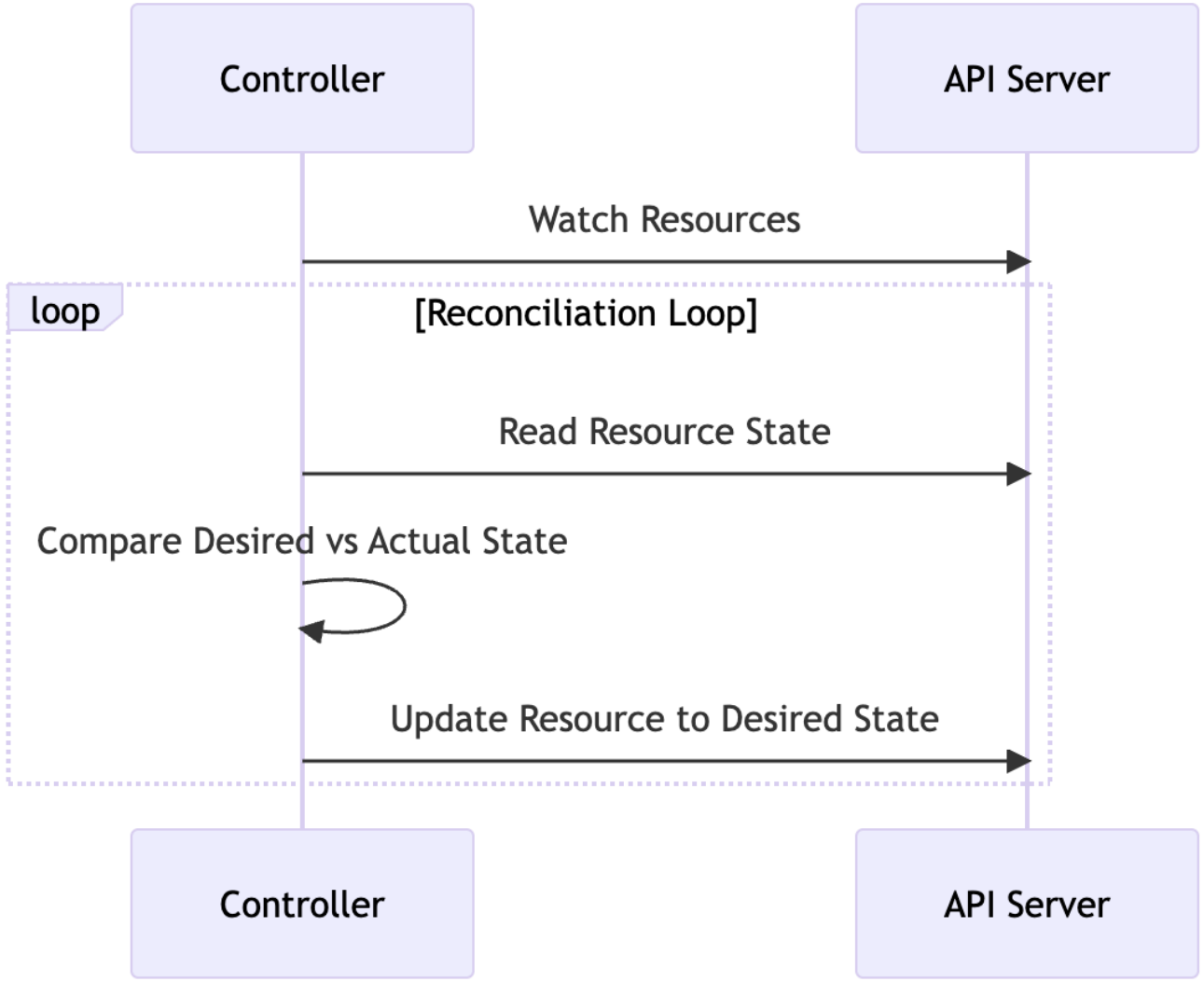
3. Kubernetes stores it in the Etcd as the application's desired state
4. Kubernetes implements the desired state on the cluster
5. Kubernetes implements watch loops -> current state of the application doesn't vary from the desired state

Outline

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

Reconciliation Loop

In Kubernetes, the Reconciliation Loop ensures that the system's actual state always matches the user's desired state. By continuously detecting and correcting differences, it provides self-healing and consistency in a dynamic, unreliable environment.



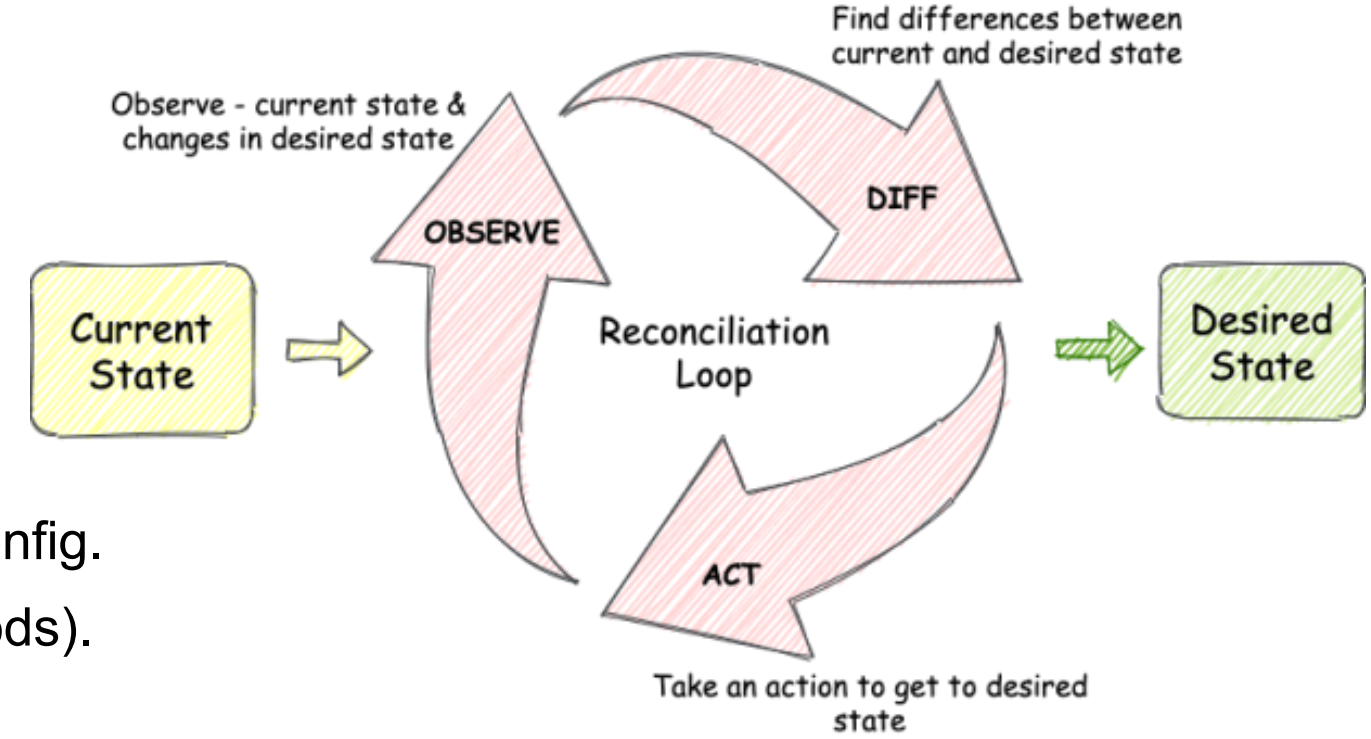
Reconciliation Loop

Goal:

- Make the cluster's current state match the desired state.

How it works:

- Observe – Check what's running now.
- Compare – See if it matches the YAML config.
- Act – Fix differences (e.g., start or stop pods).



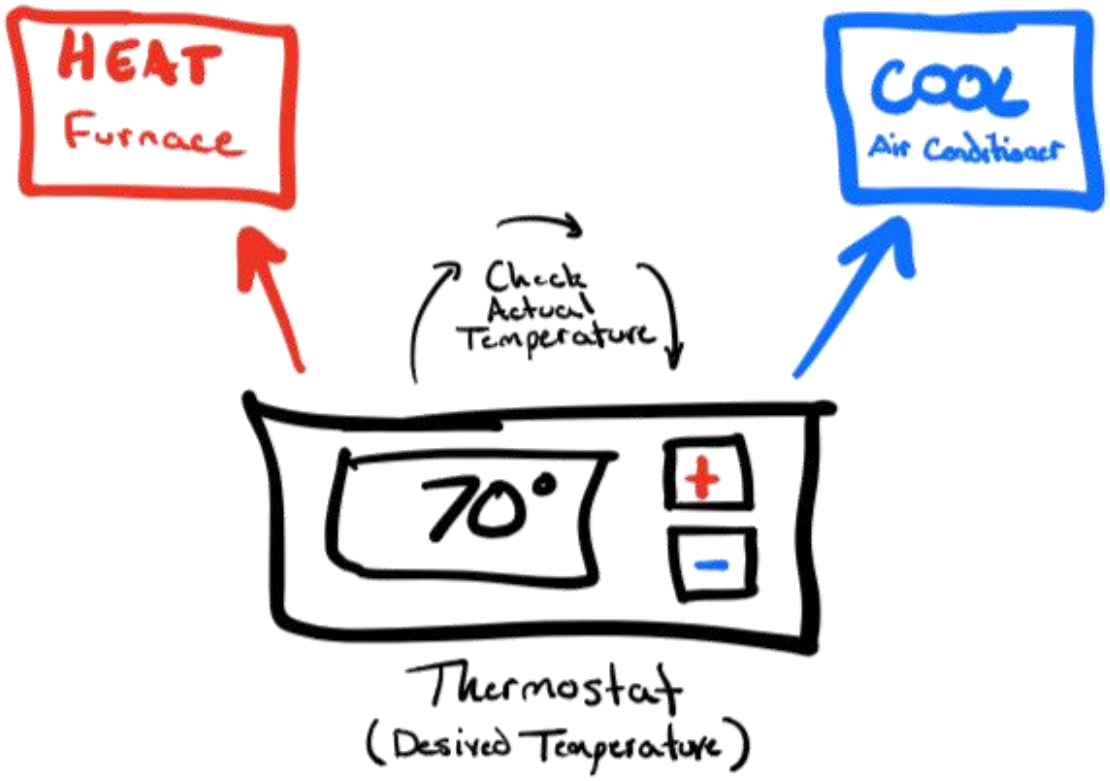
An Intuitive Example: The Thermostat

Desired State: You set the thermostat to 70°C. This is the state you want.

Current State: The thermostat's sensor constantly measures the actual room temperature, let's say it’s 70°C.

The Reconciliation Loop in Action:

- Observe: The thermostat reads the current temperature (60°C).
- Compare: It compares 60°C to your desired temperature of 70°C. They don't match.
- Act: Because the current temperature is too low, the thermostat sends a signal to turn on the heater.



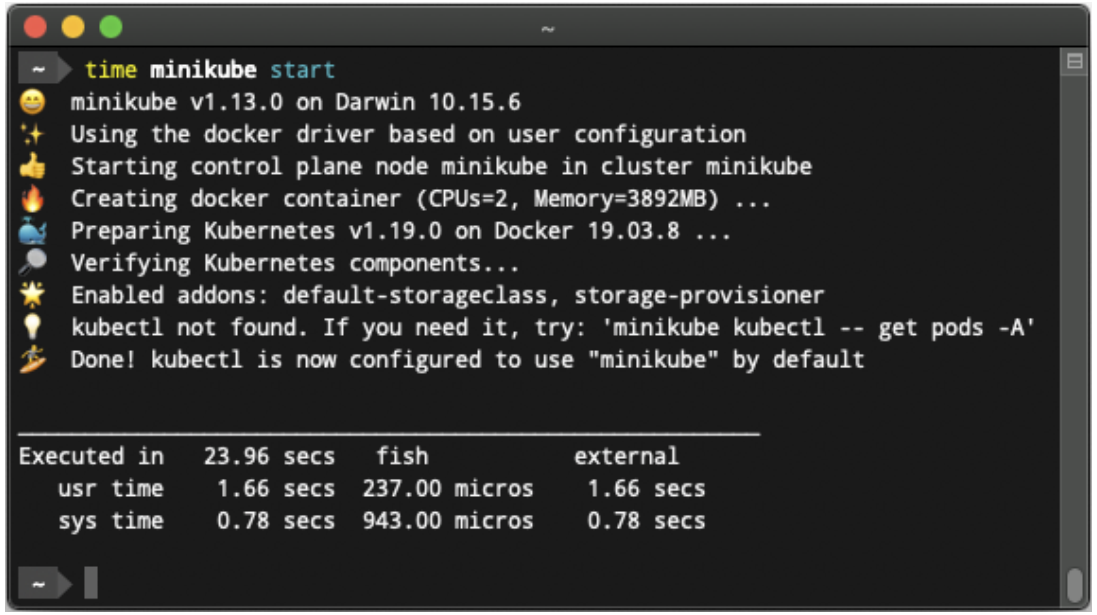
https://downey.io/blog/desired-state-vs-actual-state-in-kubernetes/

Outline

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

Minikube

- minikube quickly sets up a local Kubernetes cluster on
  - macOS, Linux, and Windows.
- Supports the latest Kubernetes release
- Supports multiple container runtimes (CRI-O, containerd, docker)
- Provides advanced features:
  - Load Balancing,
  - Filesystem mounts
  - Network policy and etc.
- Support **add-ons** (extensions of minikube for added functionality)
- Useful for application developers and helpful for new Kubernetes beginners.
- Alternatives to Minikube:
  - Kind, micro-k8s, etc.



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

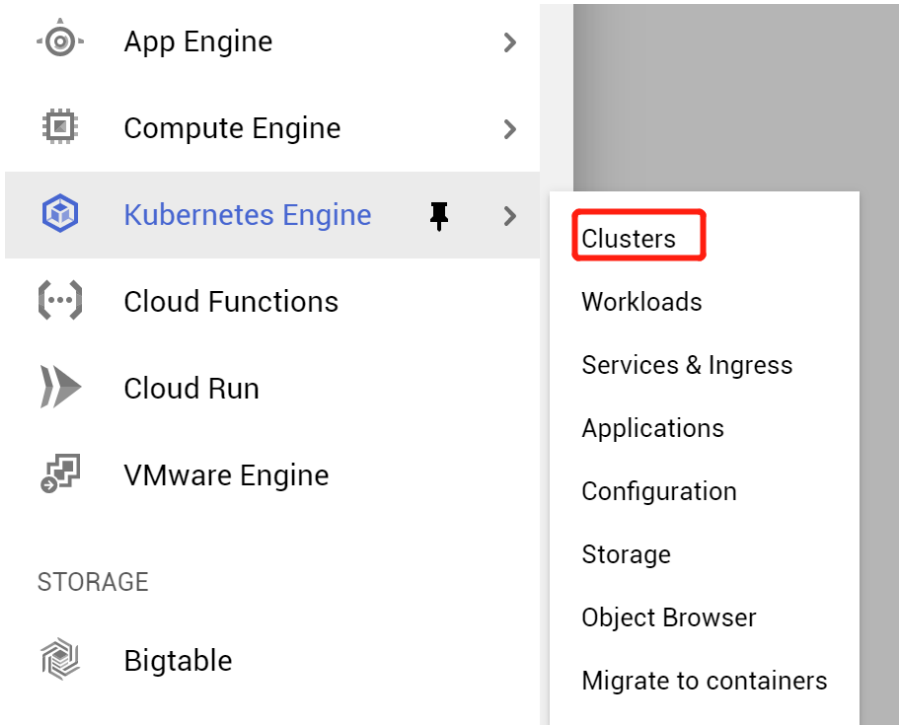
Kubernetes command-line tool - kubectl

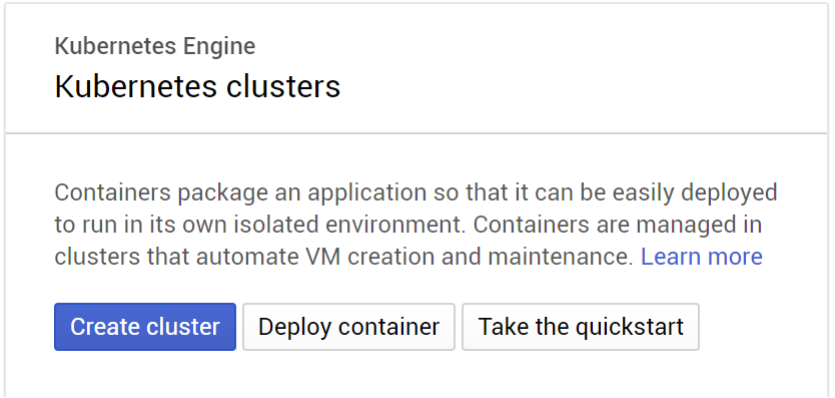
- Run commands against Kubernetes clusters.
  - deploy applications (create/apply pod, deployment, service, pv/pvc),
  - inspect and manage cluster resources (get/delete),
  - view logs (describe)
- kubectl command syntax:
  - kubectl [command] [TYPE] [NAME] [flags]
    - [command]: specify the operation do you want to perform on resources,
      - E.g.: create, get, describe, delete, apply.
    - [TYPE]: specify resource type.
      - E.g.: pods, nodes, services, etc.
    - [NAME]: specify the name of the resource (case-sensitive).
    - [flags]: specify optional flags.
      - E.g.: use “-s” or “--server” flags to specify the address and port of the Kubernetes API server.

Outline

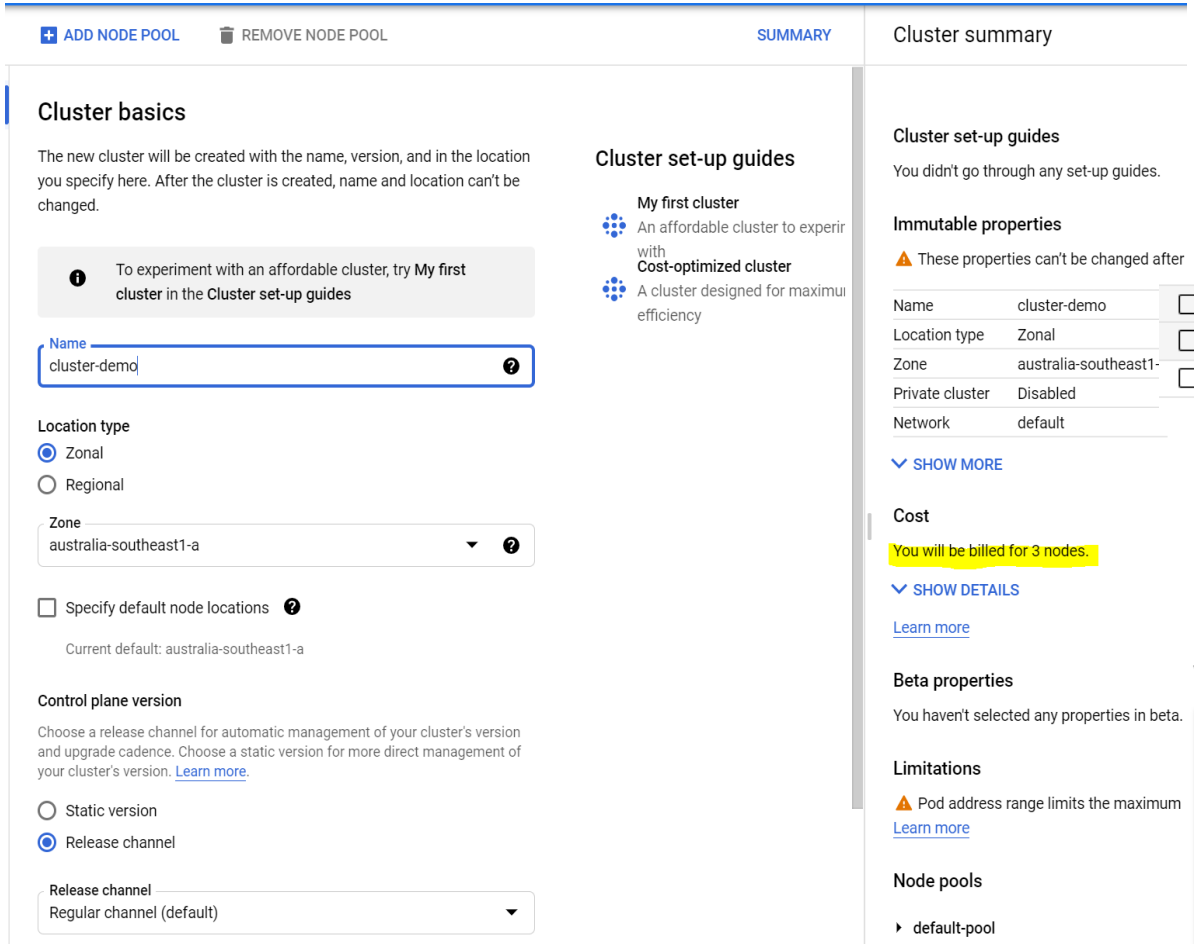
- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

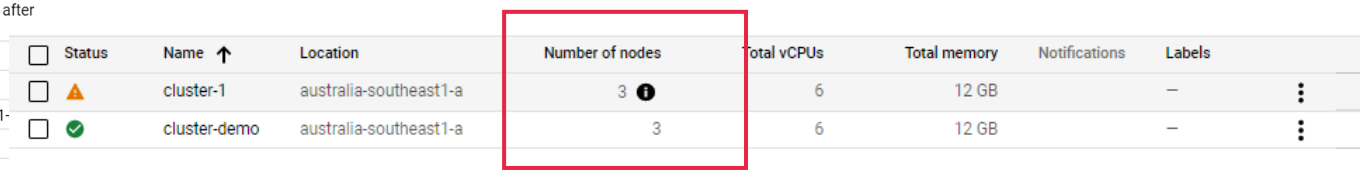
Demo: Google Kubernetes Engine (GKE) – Create a Cluster





Demo: GKE – Create a Cluster





Demo: GKE – Log in using Cloud Shell

```
uqyluo@cloudshell:~ (mythic-dynamo-300704)$ kubectl cluster-info
Kubernetes control plane is running at https://35.197.191.106
GLBCDefaultBackend is running at https://35.197.191.106/api/v1/namespaces/kube-system/services/default-http-backend:http/prox
y
KubeDNS is running at https://35.197.191.106/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://35.197.191.106/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy
```

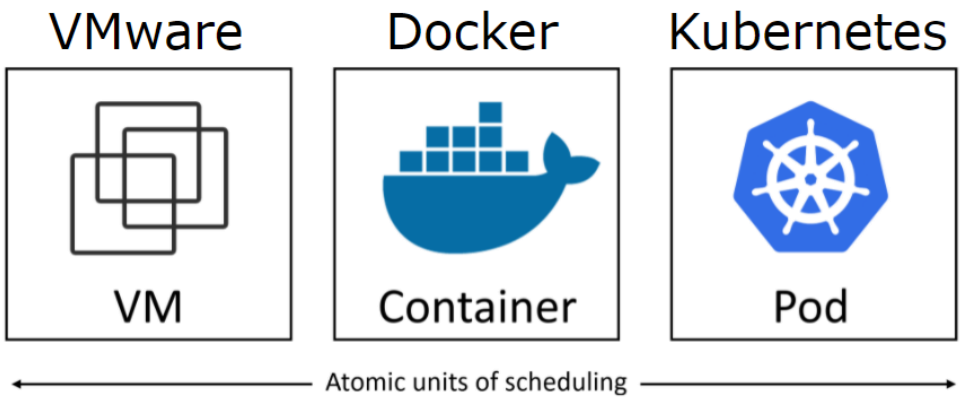
```
uqyluo@cloudshell:~ (mythic-dynamo-300704)$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
gke-cluster-demo-default-pool-368af1f8-6drq   Ready    <none>   8m30s   v1.20.8-gke.900
gke-cluster-demo-default-pool-368af1f8-l9d6   Ready    <none>   8m32s   v1.20.8-gke.900
gke-cluster-demo-default-pool-368af1f8-xh3q   Ready    <none>   8m32s   v1.20.8-gke.900
```

Outline

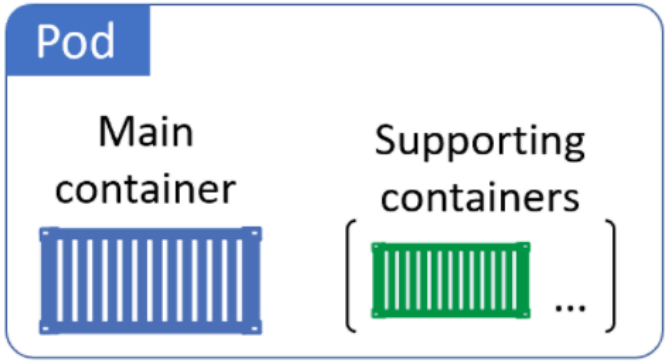
- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Models in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

What is a Pod?

- Pod - The atomic (smallest) unit of scheduling in K8s
- Pod – The abstraction over a container(s)
  - Containers must **always** run inside of Pods



- Simplest model: one container per Pod; Advanced use cases that run multiple containers inside a single Pod

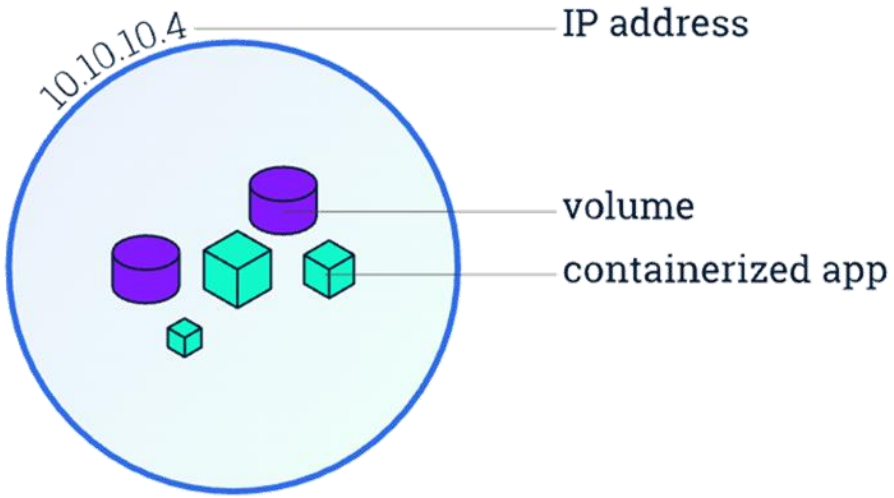


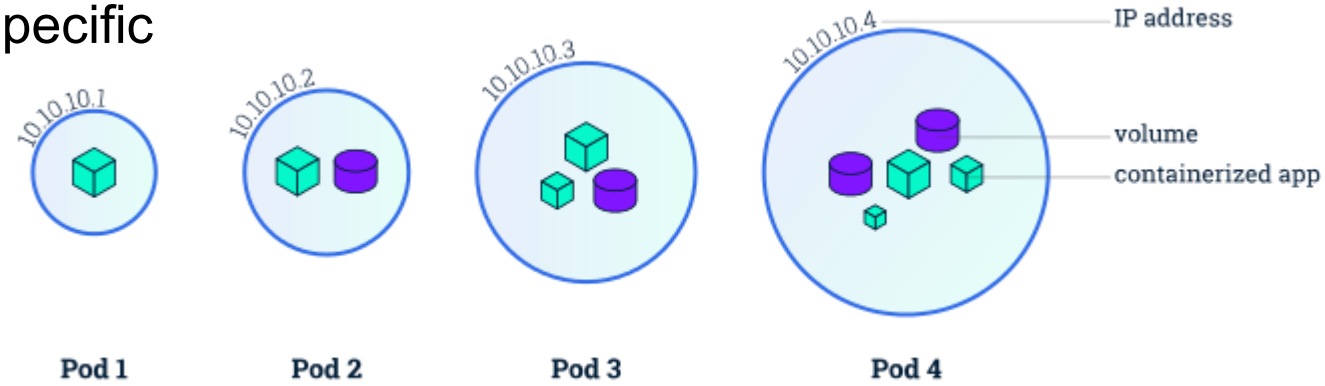
- How do isolated containers communicate?

https://kubernetes.io/docs/concepts/workloads/pods/

What is a Pod?

- Each pod gets its own IP address when starts
- Multiple containers in a Pod share the same Pod environment
  - Networking, as a unique cluster IP address
  - Shared storage, as Volumes
  - Information about how to run each container,
    - (e.g. the container image version or specific ports to use)



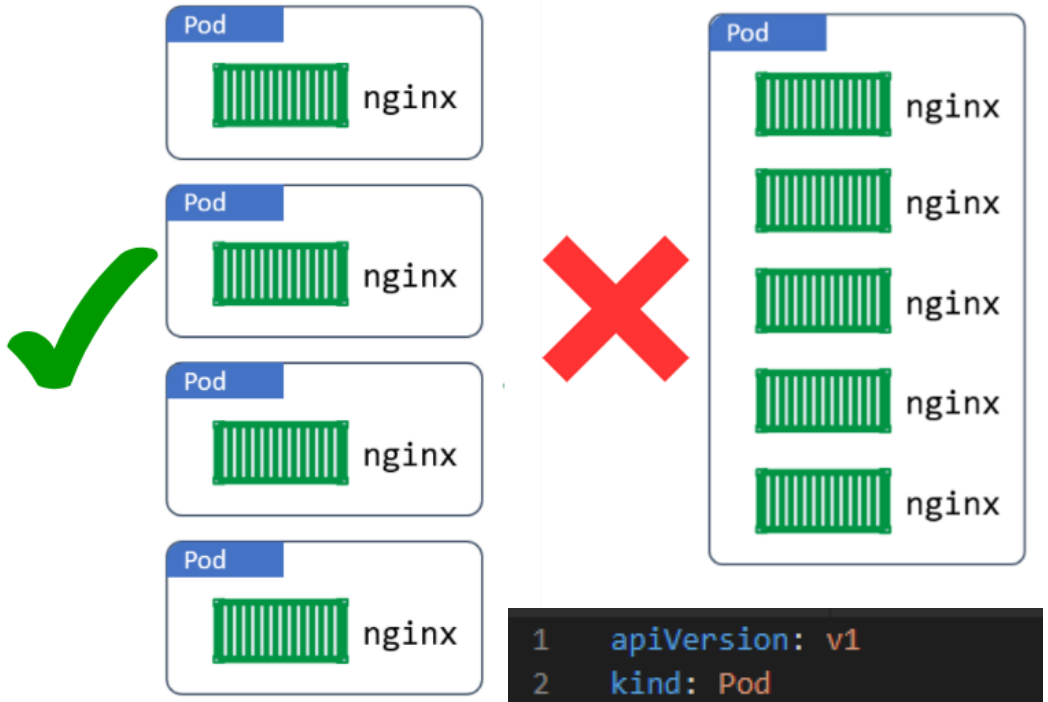


https://kubernetes.io/docs/concepts/workloads/pods/

Pods are ephemeral (lasting for a very short time)!!

What is a Pod?

- Scaling: Pods are minimum unit of scaling
- Deployment: A single Pod can only be scheduled to a single node
- Lifecycle: New Pod is associated with a new ID and IP address




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

For an application to run on a Kubernetes cluster…

1. Being packaged as a container
2. Being wrapped in a **Pod**
3. Being deployed via a **declarative manifest file**

https://kubernetes.io/docs/concepts/workloads/pods/

ReplicaSet

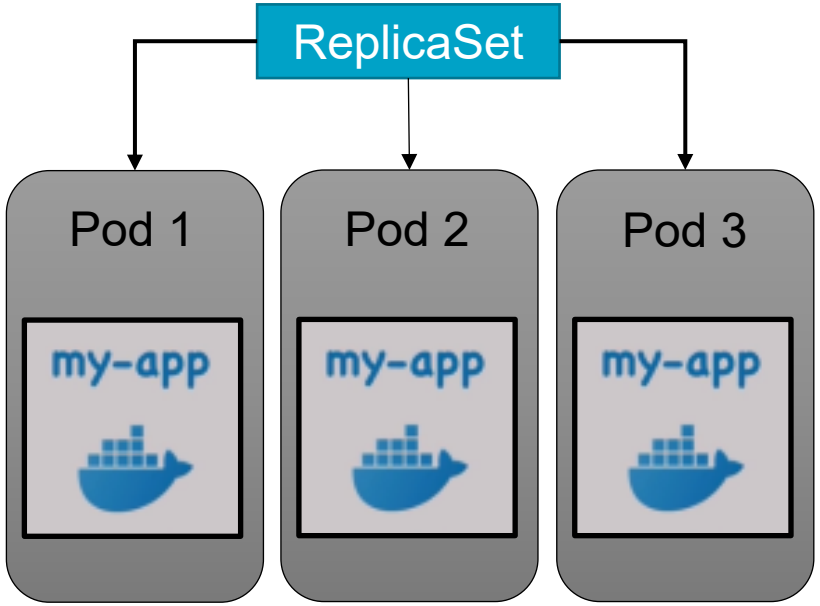
A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.

Field definitions:

- #replicas - indicating how many Pods it should be maintaining;
- a selector - specifying how to identify Pods it can acquire;
- a pod template - specifying the data of new Pods.

Dynamically creates and deletes Pods as needed to reach the desired number

- According to desired stats in the manifest file – yaml file.
- Run kubectl create/apply - f
  - E.g. kubectl apply -f repset.yaml




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

Demo: Deploy Replicasets

Replicaset YAML file

- apiVersion, kind, metadata
- spec:
  - replicas: desired number of replicas
  - selector: select which pods should be included
  - template: required field, same schema as a Pod specification

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

Demo: Deploy Replicasets

Run “kubectl describe rs cc-demo-replicaset”

```
Name:         cc-demo-replicaset
Namespace:    default
Selector:     tier=frontend
Labels:       app=guestbook
              tier=frontend
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  tier=frontend
  Containers:
   php-redis:
    Image:        gcr.io/google_samples/gb-frontend:v3
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                   Message
  ----    ------            ----   ----                   -------
  Normal  SuccessfulCreate  4m28s  replicaset-controller  Created pod: cc-demo-replicaset-hb5xz
  Normal  SuccessfulCreate  4m28s  replicaset-controller  Created pod: cc-demo-replicaset-kpv6r
  Normal  SuccessfulCreate  4m28s  replicaset-controller  Created pod: cc-demo-replicaset-q9mjl
```

Demo: Deploy Replicasets – Self-healing

- Destroy a Pod in the cluster as a simulation of failure

```
POD_NAME=$(kubectl get pods -o name \
    | tail -1)
kubectl delete $POD_NAME
```

```
(base)  ✘ sen@Sens-M1Pro  ~  POD_NAME=$(kubectl get pods -o name | tail -1)
(base)  sen@Sens-M1Pro  ~  kubectl delete $POD_NAME
pod "cc-demo-replicaset-sqnf5" deleted
(base)  sen@Sens-M1Pro  ~  kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
cc-demo-replicaset-2d42j   1/1     Running   0          2m1s
cc-demo-replicaset-6pkrp   1/1     Running   0          35s
cc-demo-replicaset-v48xb   1/1     Running   0          6s
```

Demo: Deploy Replicasets – Scaling

```
(base)  sen@Sens-M1Pro  ~  kubectl get rs
NAME                   DESIRED   CURRENT   READY   AGE
cc-demo-replicaset   3         3         3       11m
(base)  sen@Sens-M1Pro  ~  kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
cc-demo-replicaset-2d42j   1/1     Running   0          4m20s
cc-demo-replicaset-6pkrp   1/1     Running   0          2m54s
cc-demo-replicaset-v48xb   1/1     Running   0          2m25s
```

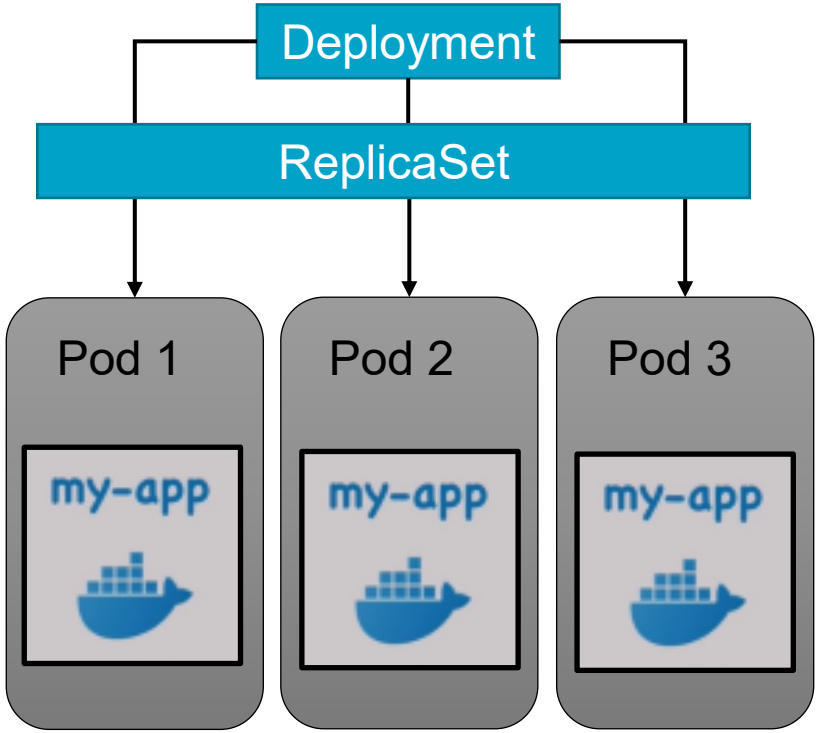
```
(base)  sen@Sens-M1Pro  ~  kubectl scale --replicas=5 rs cc-demo-replicaset
replicaset.apps/cc-demo-replicaset scaled
(base)  sen@Sens-M1Pro  ~  kubectl get rs
NAME                   DESIRED   CURRENT   READY   AGE
cc-demo-replicaset   5         5         5       12m
(base)  sen@Sens-M1Pro  ~  kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
cc-demo-replicaset-2d42j   1/1     Running   0          5m26s
cc-demo-replicaset-6pkrp   1/1     Running   0          4m
cc-demo-replicaset-82hr5   1/1     Running   0          14s
cc-demo-replicaset-jcxcj   1/1     Running   0          14s
cc-demo-replicaset-v48xb   1/1     Running   0          3m31s
```

Deployment

A Deployment provides declarative updates for Pods and ReplicaSets.

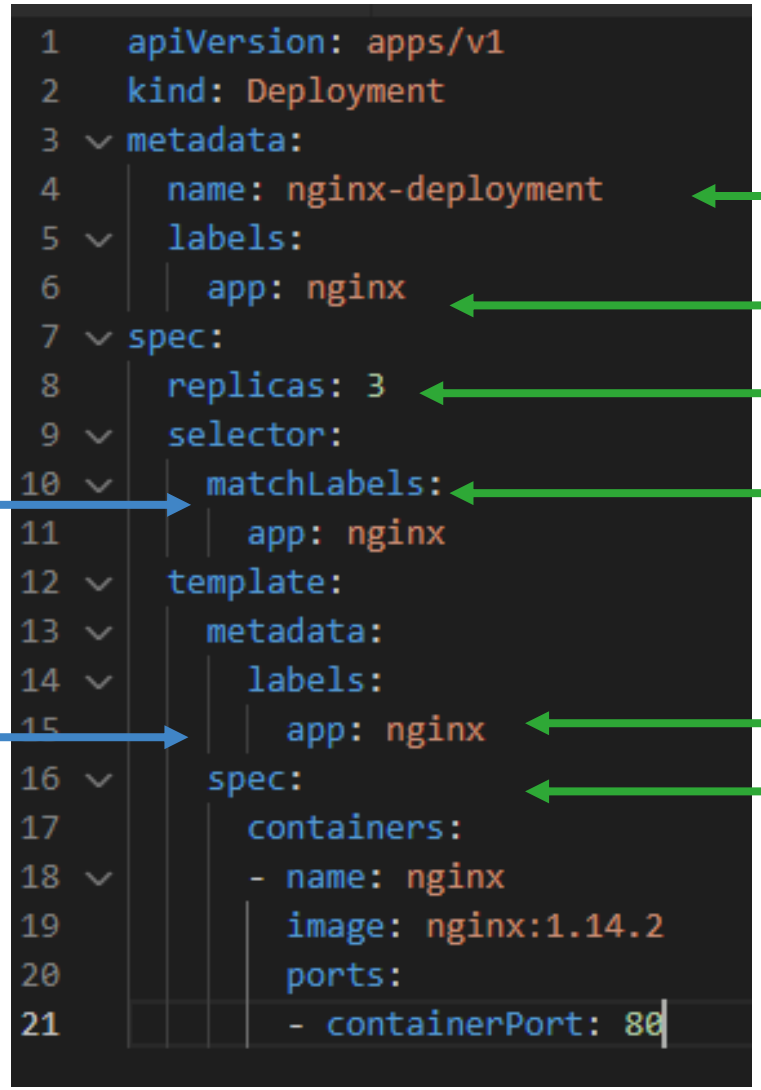
- Blueprint for pods (including replica number of the pod)
- You create deployments rather than pods (in practice no directly manipulating pods)
- ReplicaSet is to maintain a stable set of replica Pods (identical) running at any given time.
- Deployment can manage ReplicaSet (you may never need to manipulate ReplicaSet):
  - Use a Deployment directly and define your application in it.
- Deployment can support for rolling updates

  allowing you to update a PodTemplateSpec and have the changes gradually rolled out in a controlled manner. It also supports rollbacks to previous versions.



Replica=3 for the deployment

Deployment



```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
```

Name of the deployment

Labels of the deployment

Replicated number of the pods

Selector of the target pods who has the label **"app: nginx"**

Container template definition:

- labels: **"app: nginx"** – all pods created here could have different names, but they are sharing the same label. They are selected by the selector above.
- spec in template specifies container info in each pod
  - image
  - ports
  - etc.

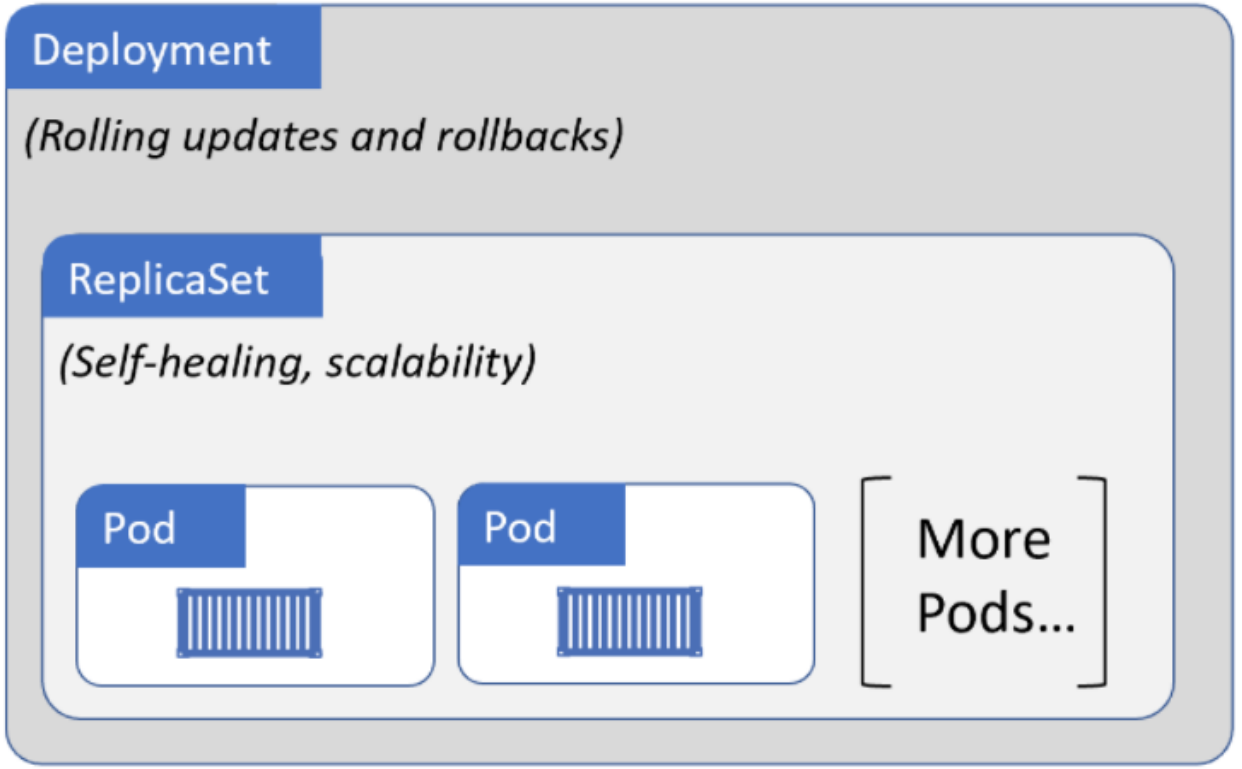
Deployments

Deployment controller

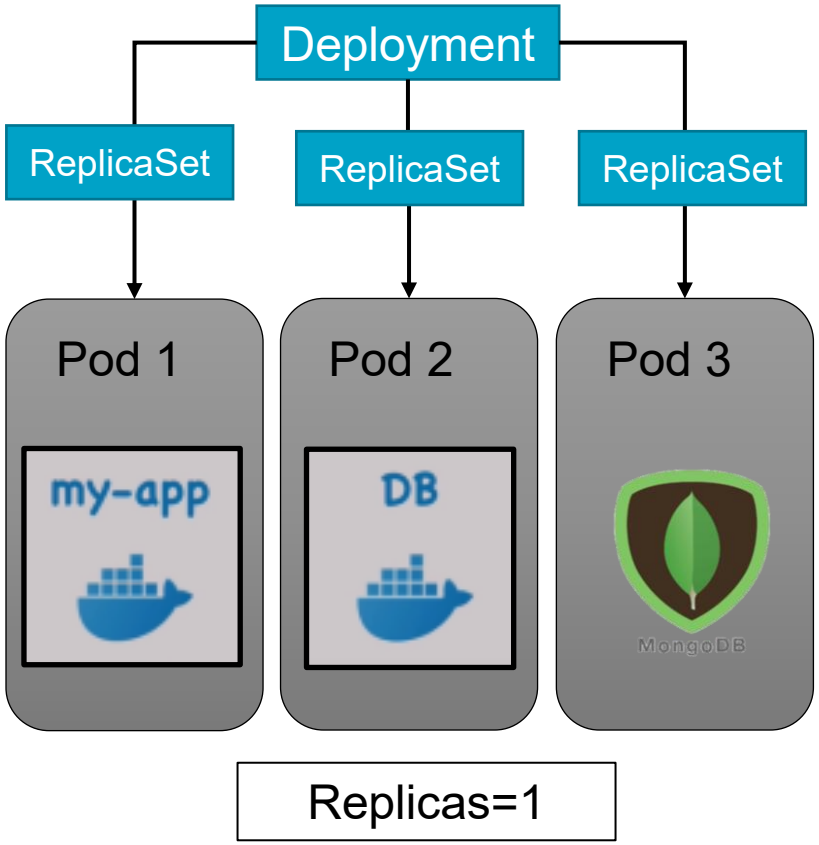
- Deployment: object type in K8s API
- Use Replicasets provide self-healing and scaling

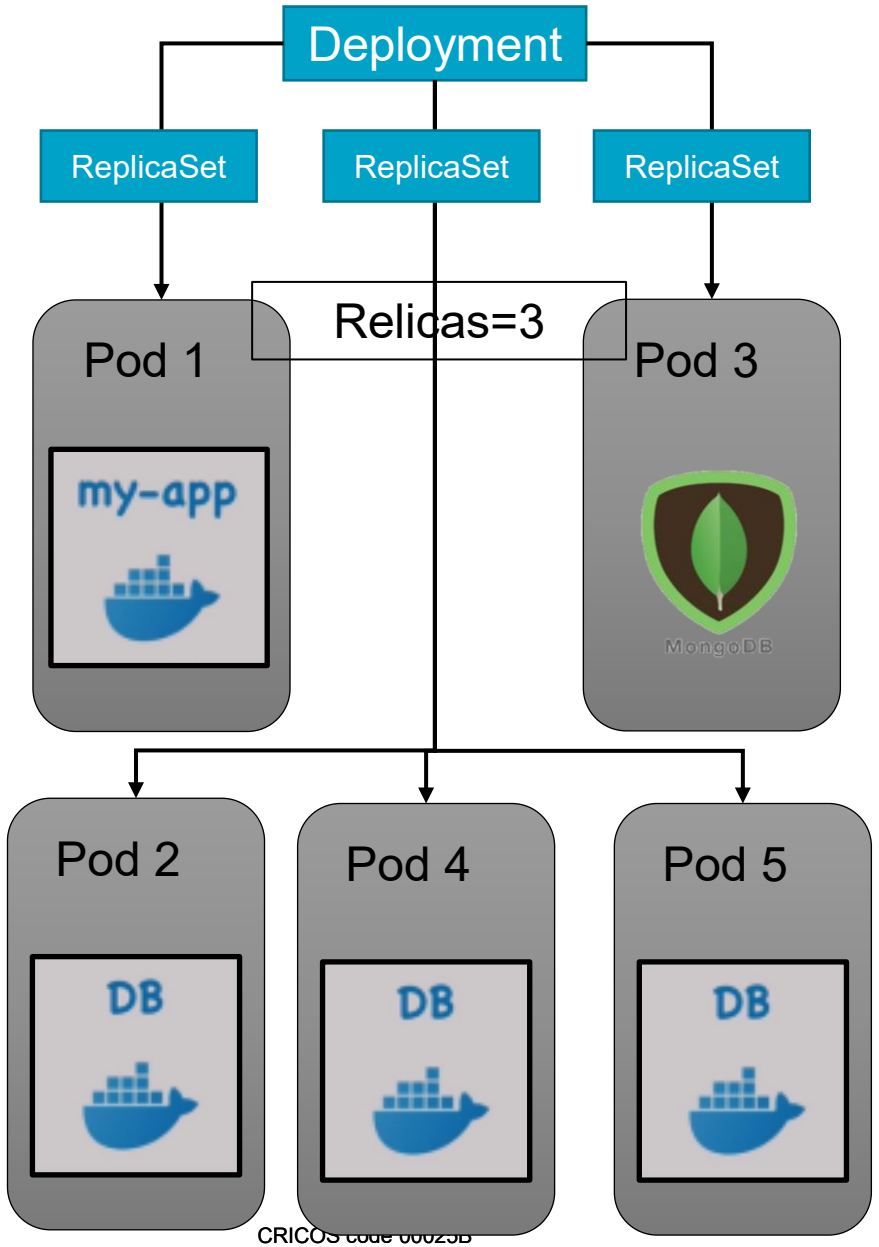
Replicasets controller

- Ensure the specified number of replicas of a service are always running

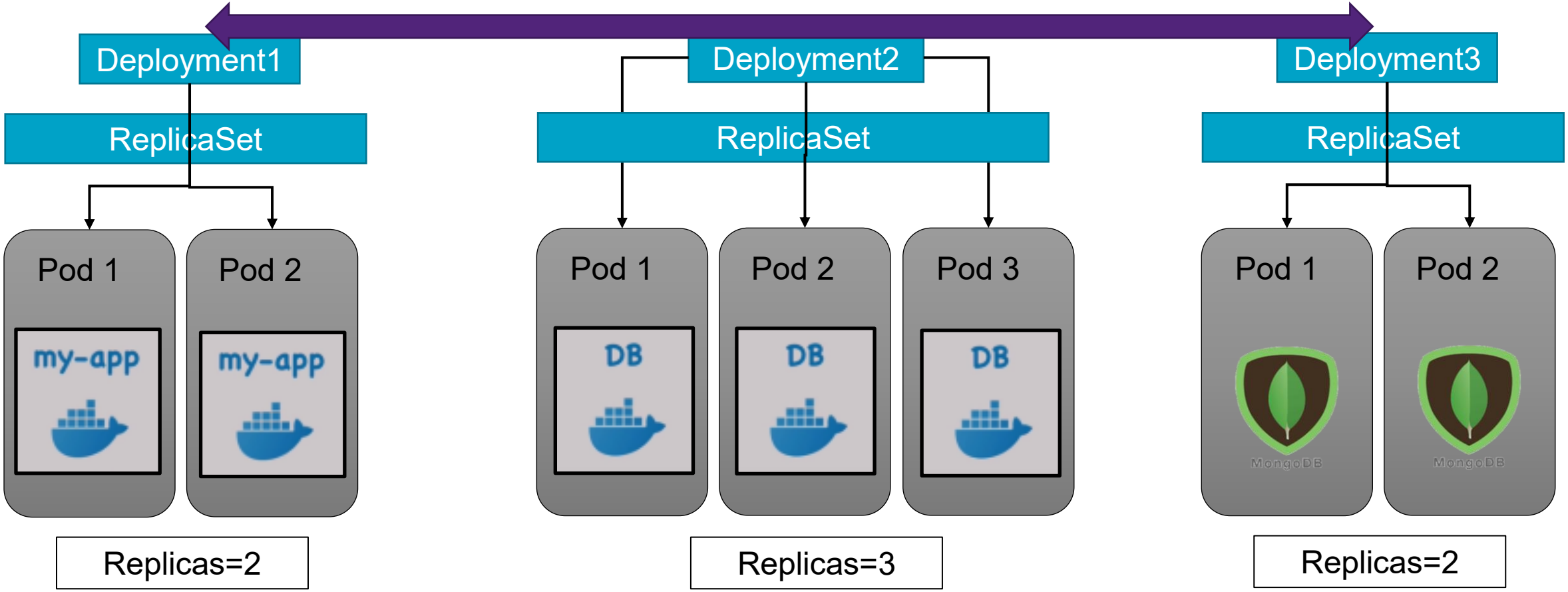


Deployment – Discussion 1





Deployment – Discussion 2

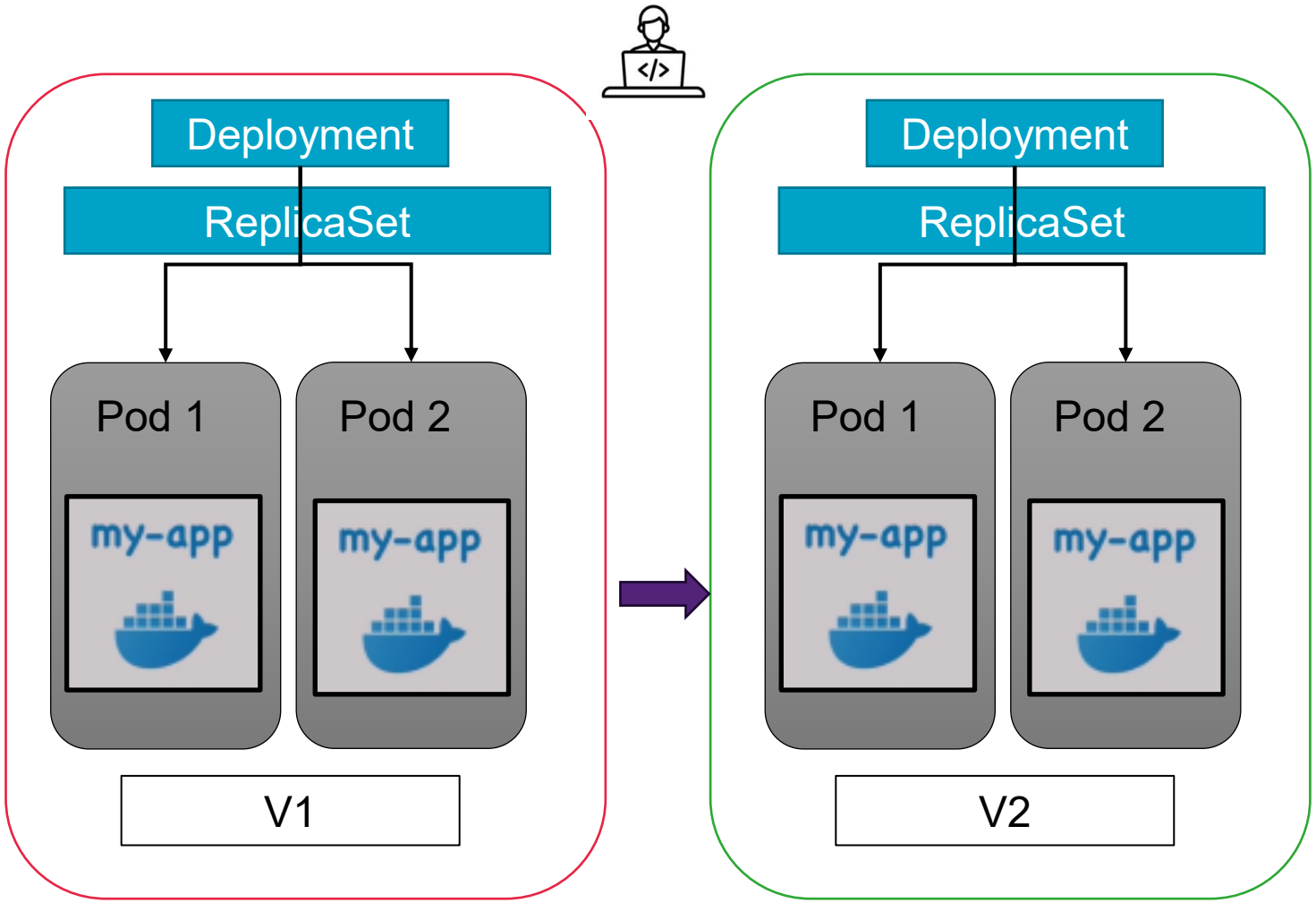


Note: The deployment is meant to be a microservice. For example: Deployment 1 could be a frontend microservice that has two replicas, while Deployment 2 could be a three-replicated DB microservice.

Deployment – Discussion 3

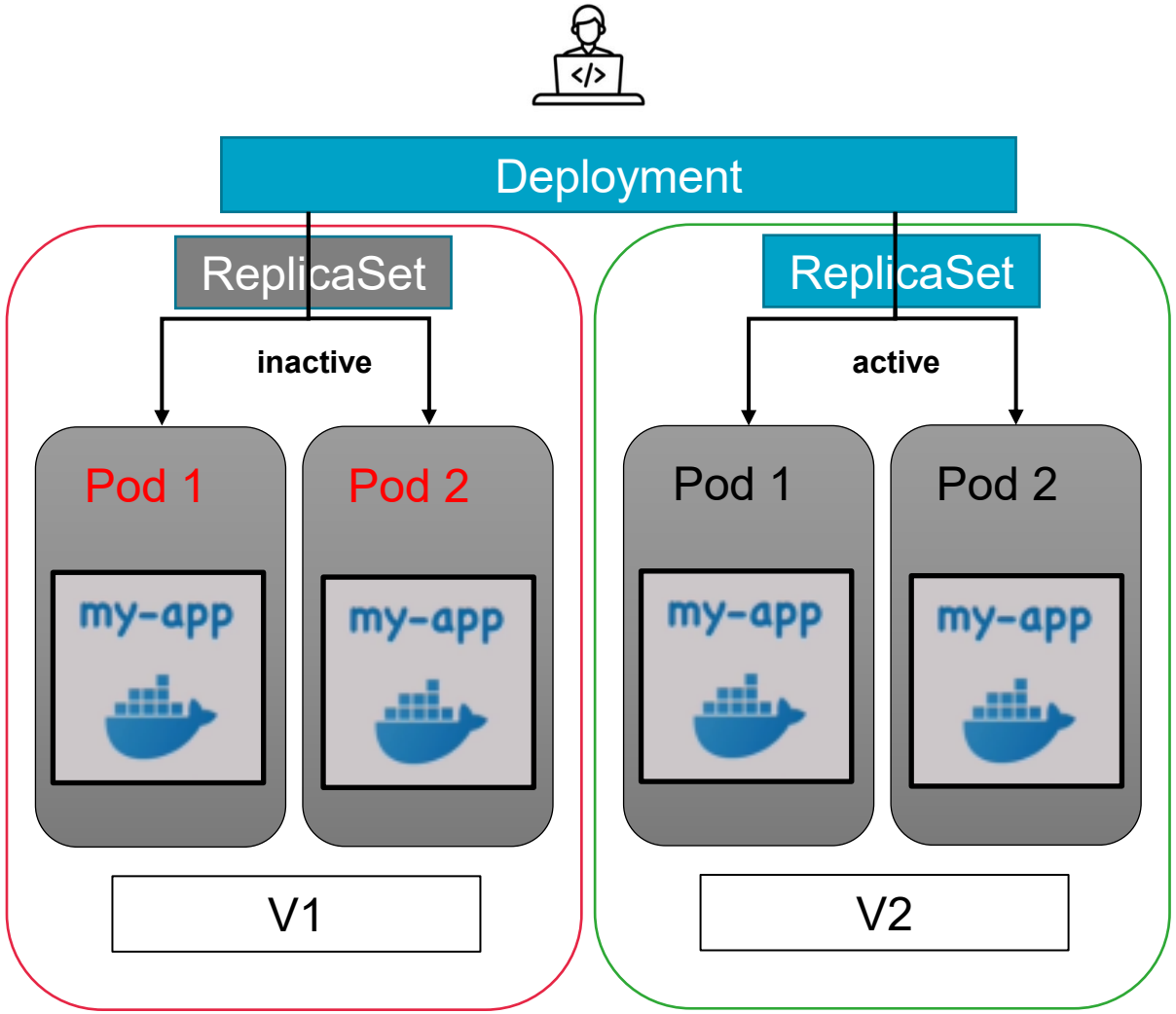
When deploying a newer version of a specific (micro)service: v2 -> v1 (in production), as a DevOps Engineer, you want to have the following features:

1. **Rolling update**
   the pods v1 to pods v2 in batches (the entire deployment won't go down);
2. **Maintain the same number of replicas**
   2 x pods_v1 -> 2 x pods_v2
3. **Rollback can be conducted** when v2 is faulty (bugs, errors, etc.)
   2 x pods_v2 -> 2 x pods_v1



Deployment – Discussion 4

- A Deployment can manage **multiple ReplicaSets**, but there is typically **only one active ReplicaSet**.
- The other **inactive ReplicaSets** are kept around with a reduced replica count (usually zero) **for the sake of rollback and revision history**.
- During a rolling update:
  - The new **ReplicaSet** (which matches the updated pod template in the Deployment) is scaled up.
  - The old **ReplicaSet** is scaled down (zero).



Demo: Deployments (1/4)

Deployment YAML file

- "**cc-demo-deployment**" is created, indicated by the **.metadata.name** field, which will be the basis for the **ReplicaSets** and **Pods.**
- creates a **ReplicaSet** that creates **THREE** replicated **Pods** (.spec.replicas).
- The **.spec.selector** defines how the created **ReplicaSet** finds which Pods to manage: selecting a label that is defined in the Pod template (**app: nginx**).
- The **template** field contains the following sub-fields:
  - **.metadata.labels** field: **app:nginx** using the .
  - **.template.spec** field: the Pods run one container (nginx:1.14.2) and the port of the container is 80.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cc-demo-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
```

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Demo: Deployments (2/4)

```
(base)  sen@Sens-M1Pro  ~  kubectl create -f deploy.yml
deployment.apps/nginx-deployment created
(base)  sen@Sens-M1Pro  ~  kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   0/3     3            0           12s
(base)  sen@Sens-M1Pro  ~  kubectl rollout status deployment/nginx-deployment
deployment "nginx-deployment" successfully rolled out
(base)  sen@Sens-M1Pro  ~  kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           52s
(base)  sen@Sens-M1Pro  ~  kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
cc-demo-replicaset            5         5         5       163m
nginx-deployment-6595874d85   3         3         3       61s
```

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Demo: Deployments (3/4)

```
(base)  sen@Sens-M1Pro  ~  k get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   5/5     5            5           115s
(base)  sen@Sens-M1Pro  ~  k set image deployment/nginx-deployment nginx=nginx:1.16.1
deployment.apps/nginx-deployment image updated
(base)  sen@Sens-M1Pro  ~  k rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 4 out of 5 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 4 of 5 updated replicas are available...
deployment "nginx-deployment" successfully rolled out
(base)  sen@Sens-M1Pro  ~  k get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   5/5     5            5           3m49s
(base)  sen@Sens-M1Pro  ~  k get rs
NAME                          DESIRED   CURRENT   READY   AGE
cc-demo-replicaset            5         5         5       3h14m
nginx-deployment-6595874d85   0         0         0       3m53s
nginx-deployment-66b957f9d    5         5         5       53s
```

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Demo: Deployments (4/4)

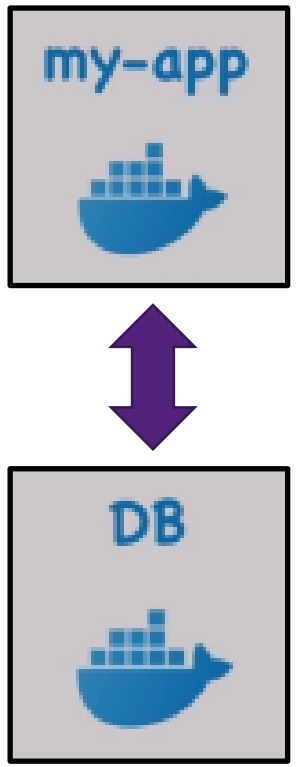
```
(base)  sen@Sens-M1Pro  ~  k describe deployment/nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Mon, 28 Aug 2023 23:22:50 +1000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=nginx
Replicas:               5 desired | 5 updated | 5 total | 5 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.16.1
    Port:         80/TCP
    Host Port:    0/TCP
```

Run “kubectl rollout history deployment/nginx-deployment” to check out the revisions of the Deployment
Run “kubectl rollout undo deployment/nginx-deployment --to-revision=2” to rollback to a specific version

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

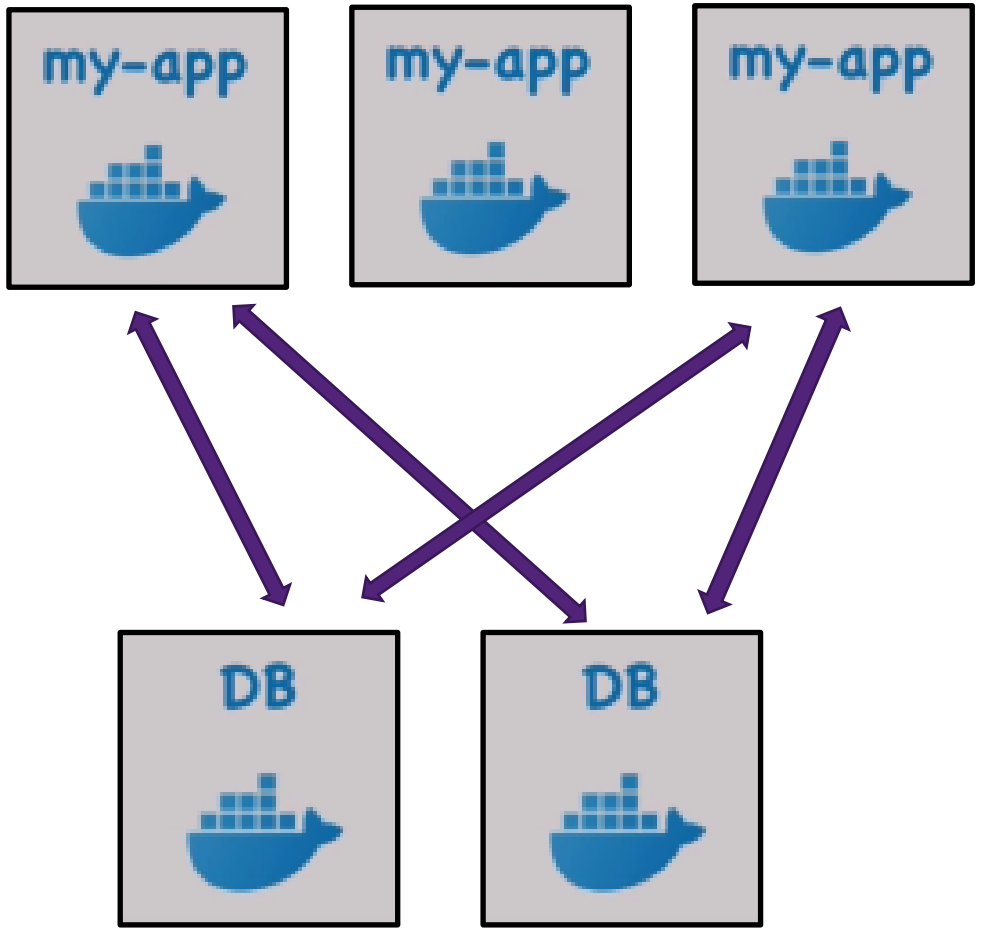
Running Applications on Pods

Pod: **10.0.0.1**



Pod: **10.0.10.1**

Pod 1: **10.0.0.1** Pod 2: **10.0.0.4** Pod 3: **10.0.0.2**



Pod 4: **10.0.10.1** Pod 5: **10.0.10.3**

Service and Ingress

Motivation

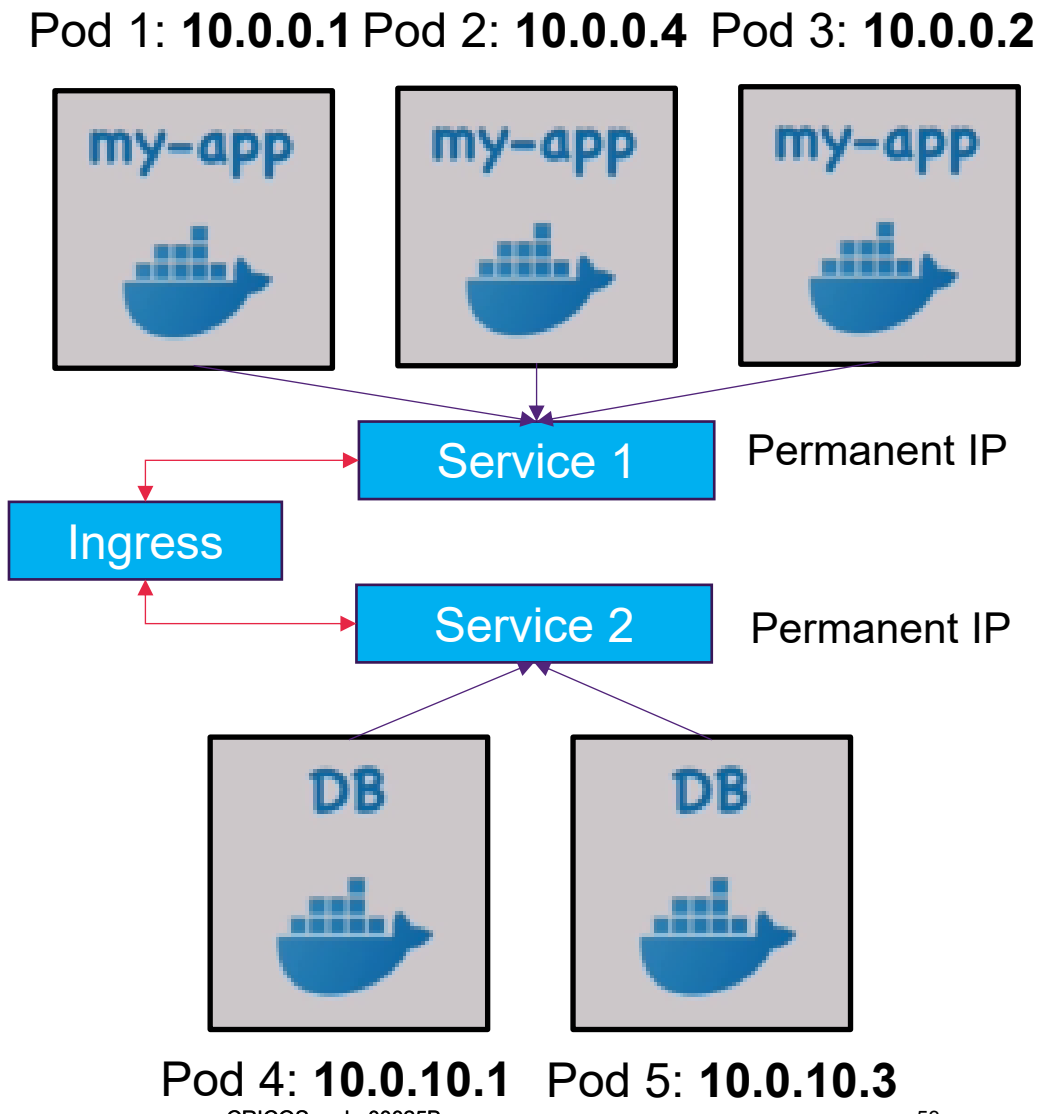
- There are no communication path
- Pods are unreliable
- New Pod gets a new address

Services

- Abstraction way to expose an app running on a set of pods as a networking services
- Provide reliable networking for a set of Pods
- Load-balancing requests across abstracted pods

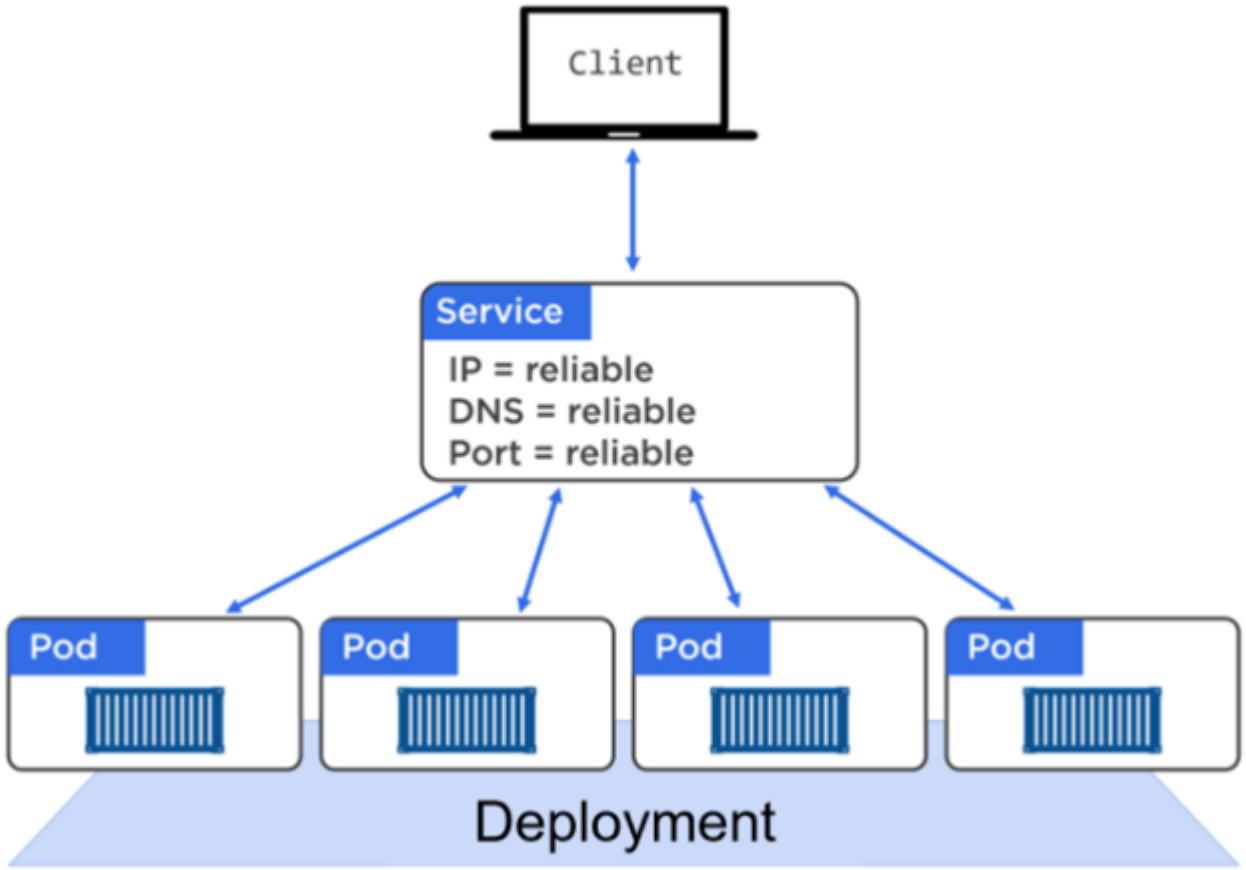
Ingress (DNS)

- manages external access to the services in a cluster, typically HTTP.



Service

- REST object in the API that you define in a manifest and POST to the API server
- Every Service gets its own stable IP address, its own stable DNS name and its own stable port.
- Leverage labels to dynamically select the Pods they will send traffic to.



Service

Create services by exposing ports

```
kubectl expose deployment cc-demo-deployment \
    --name=cc-demo-service \
    --target-port=28017 \
    --type=NodePort
```

- Specified -> expose a deployment object
- Name it as ‘cc-demo-service’
- The port to expose (28017 for Mongodb)

- ClusterIP (default): expose the port only inside the cluster
- NodePort: expose the target port on every node to the outside
- LoadBalancer: only useful when combined with cloud provider’s load balancer

Service

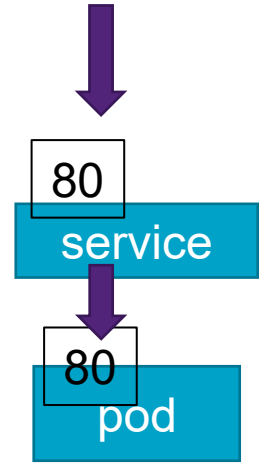
Create services with YAML file (kubectl apply -f service.yaml)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy
spec:
  containers:
  - name: nginx
    image: nginx:stable
    ports:
      - containerPort: 80
        name: http-web-svc
```

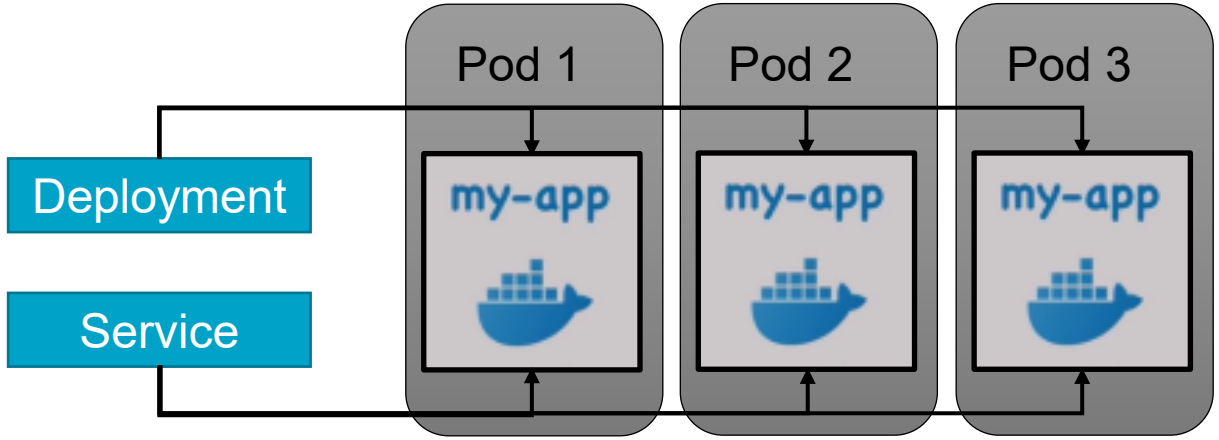
Pod.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app.kubernetes.io/name: proxy
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: http-web-svc
```

Service.yaml



Deployment vs Service



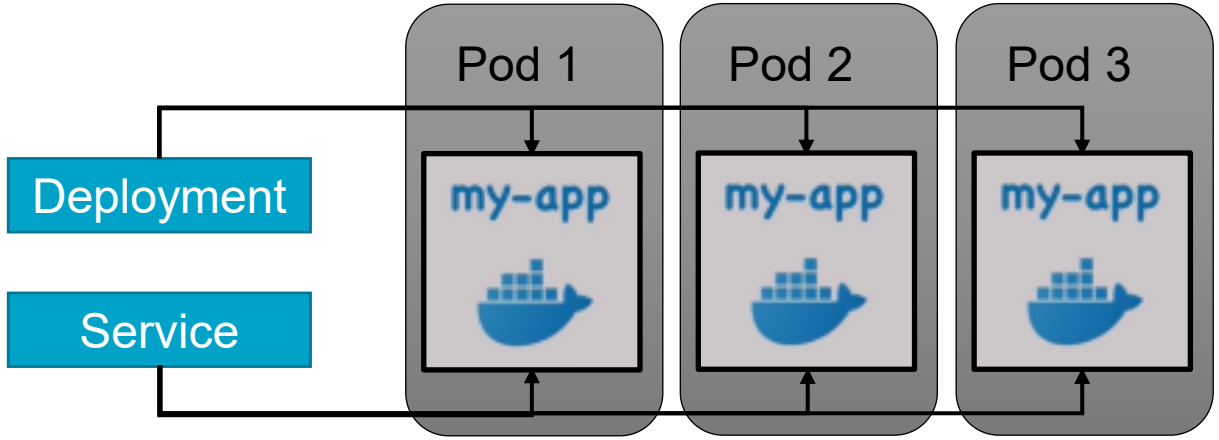
Both **Deployment** and **Service** are abstraction layers of Pods.

What are the differences between them?

Deployment vs Service

	Design Role	Functionality	Usage
Deployment	Pod abstraction **Scalable** (up or down) Blueprint (template)	• Manages the **desired state** for your Pods and **ReplicaSets**. • Allows you to perform rolling updates and rollbacks to your application.	• Ensures a certain number of instances of your application are running; to update your application, or to rollback an update. • Deployment does **NOT** deal directly with **networking or exposing** your application.
Service	Pod abstraction Load-balancer (internal & external)	• Provides a **stable endpoint** (IP address and port) to communicate with a set of Pods, typically exposed by a Deployment. • Abstracts away the **ephemeral nature** of Pods, which might be created and destroyed frequently.	• Needs **a persistent way** to access your application, irrespective of the changes in the Pods (like creation, deletion, scaling). • Service deals with networking and can expose your application **internally** (within the cluster) or **externally** (to the internet or outside world).

Deployment vs Service



Both **Deployment** and **Service** are abstraction layers of Pods.

What are the differences between them?

Simply speaking:

- a **Deployment** ensures your desired application state and its updates,
- a **Service** ensures your application remains accessible regardless of the dynamic changes in the underlying pods.
- They often work hand-in-hand: a **Deployment** maintains the application, and a **Service** provides stable access to it.

ConfigMap & Secret

There is information required when Pods are communicating:

- Non-confidential, such as the configuration of the Database;
- Confidential data, such as the password of database users

ConfigMap

- Used to store non-confidential data in key-value pairs.
- Consumed (by Pods) as environment variables, command-line arguments, or as configuration files in a volume.

Secret

- Similar to ConfigMap, but is used to store secret data (user name, password, tokens, keys, etc)
- Independently created (less risk of being exposed)
- base64 encoded (NOT encryption: binary -> ASCII).

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

Configmap.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
data:
  mysql-root-pwd: cm9vdHB3ZA==
```

Secret.yaml

Data Storage - Volumes

Data Storage for the app in the container (Pods):

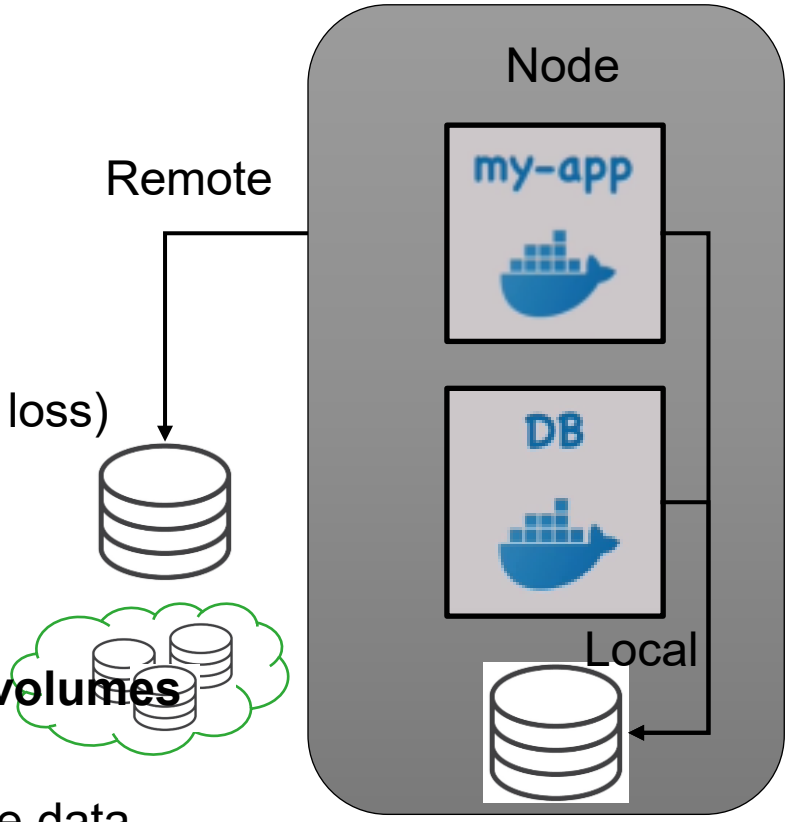
- In files (plain text, serialisation, encryption)
- In database files

Storing data directly in a container or a pod could have a loss of files (container/pod crashes) - **ephemeral**

Container/Pod is restarted with a clean state - no previous state (DB state loss)

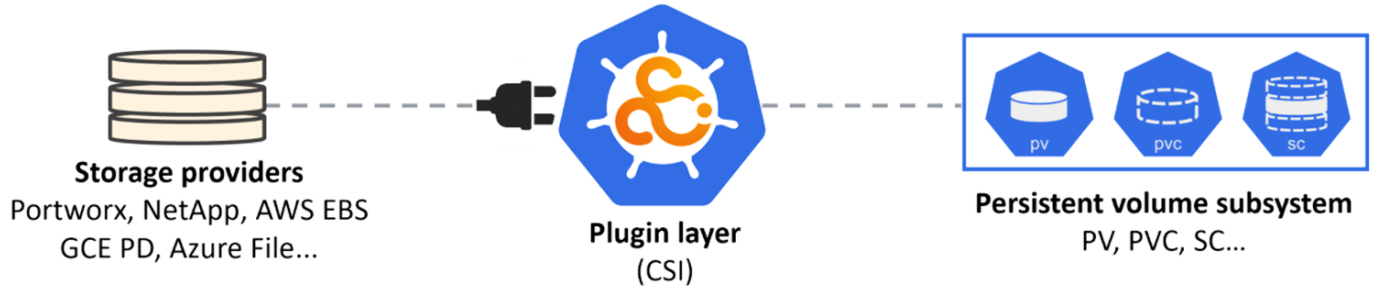
"Volumes" is to attach physical storage facility to persist the data

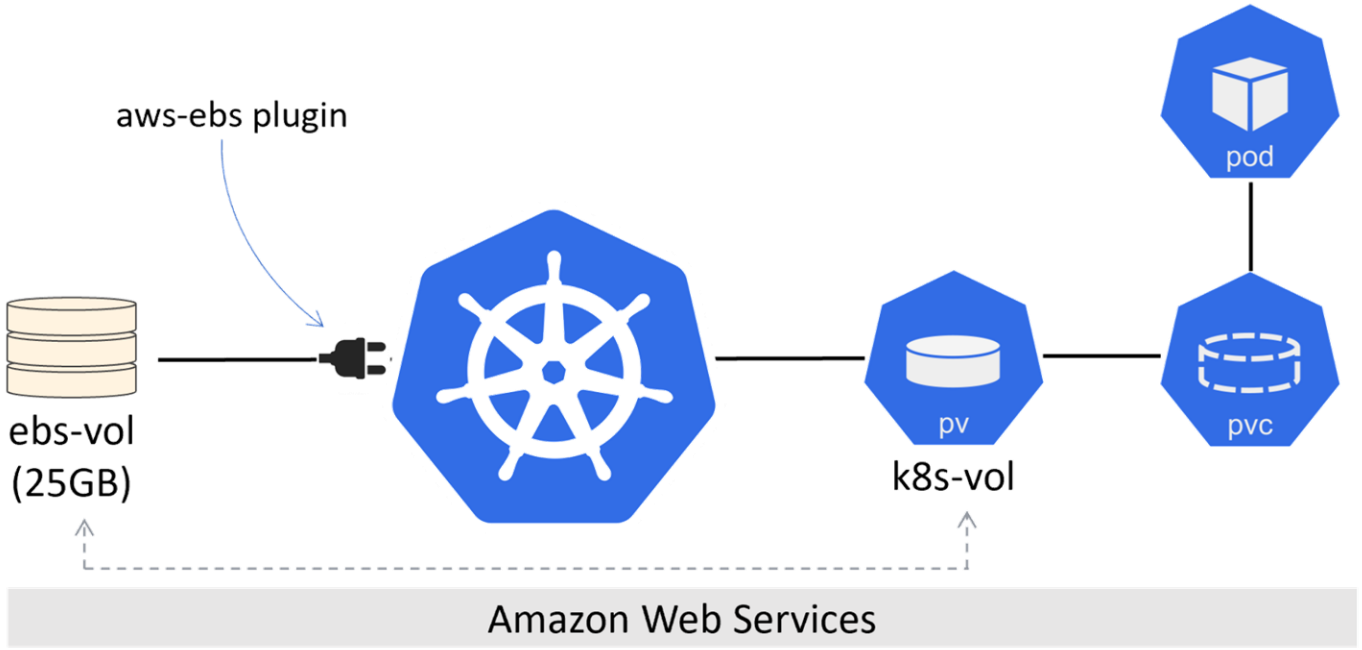
- Local storage (SSD or HDD) on the local machine (node) – **emptyDir & hostPath**
- Remote storage (on other Nodes in K8s or outside of the K8s) – **cloud volumes & NFS** (e.g. **awsElasticBlockStore** or **gcePersistentDisk**)
- K8s DOES NOT manage data persistence!! – mount a drive to read/write data
- K8s administrator should be responsible for data backup and security.



Kubernetes Storage

- Persistent Volume Subsystem:

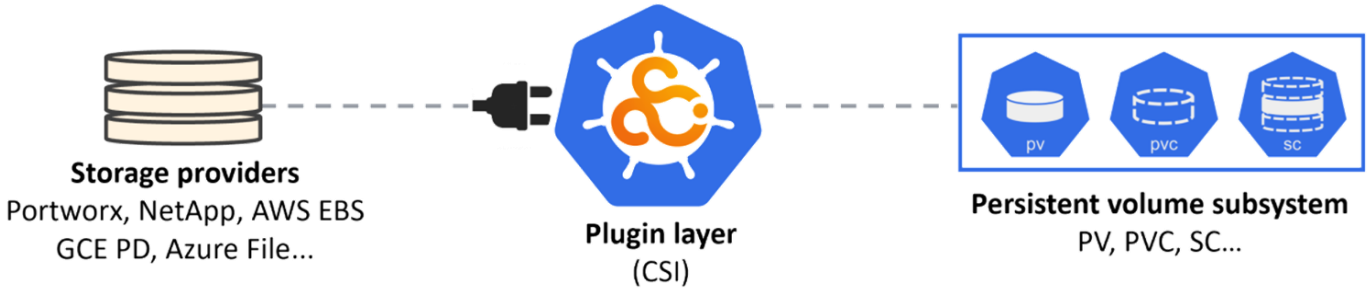


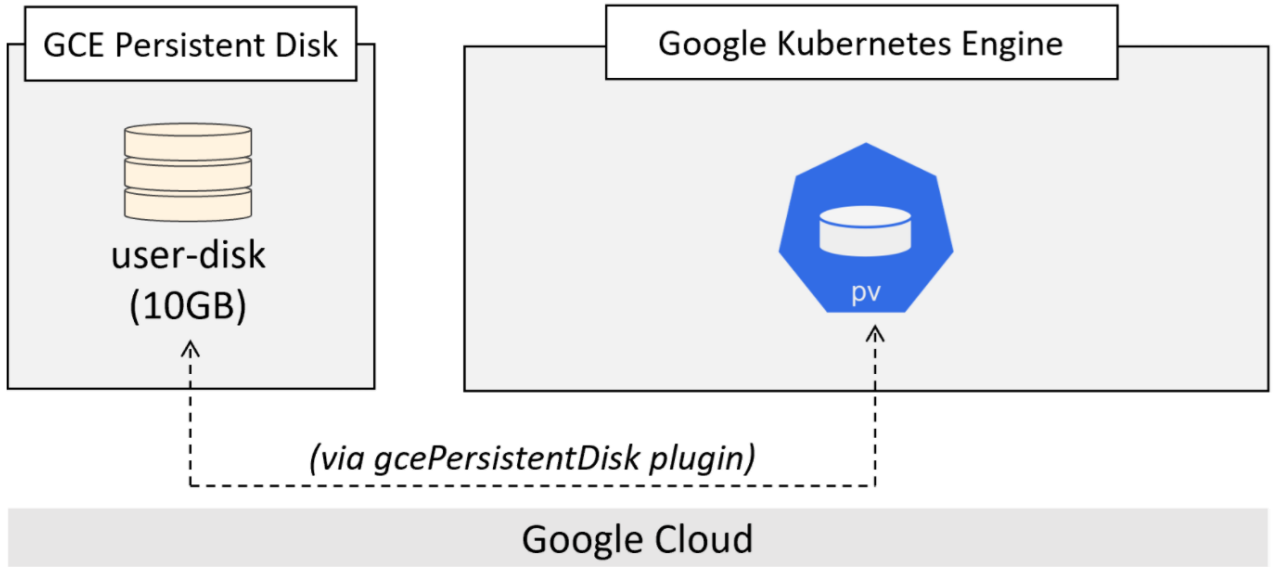


- Persistent Volume (PV): how you map external storage onto the cluster; resource (**admin/ops**)
- Persistent Volume Claim (PVC): a request for storage by a user; claim checks to the resource (**user/dev**)

https://kubernetes.io/docs/concepts/storage/persistent-volumes/
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes.html

Kubernetes Storage

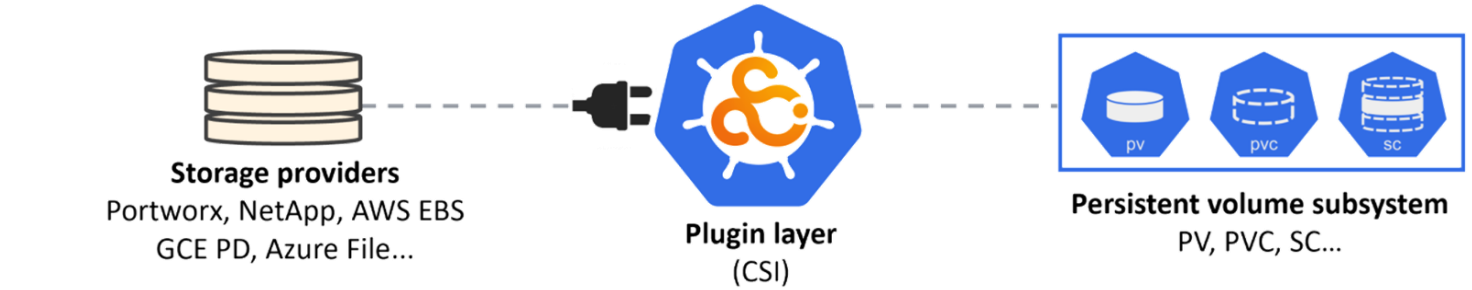


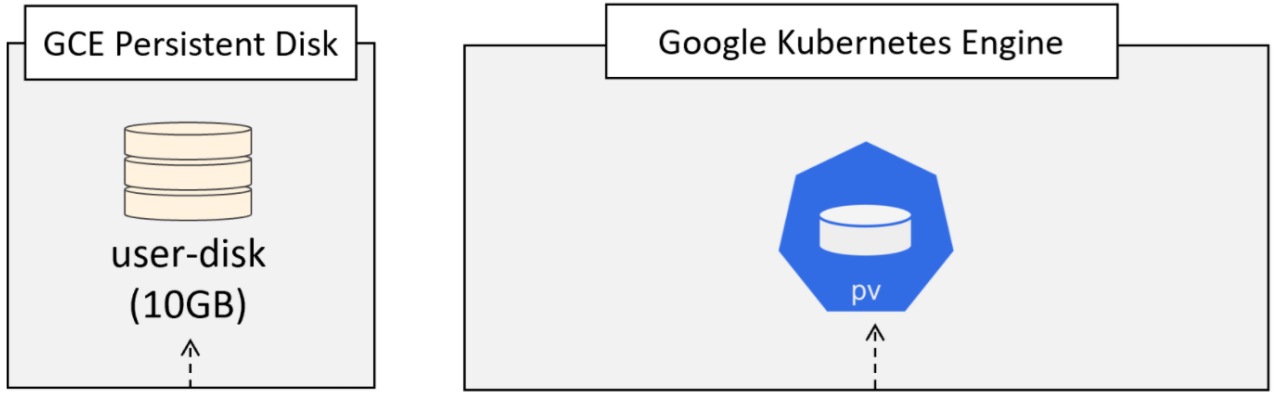


1. Create the PV.
2. Create the PVC.
3. Define the volume.
4. Mount it into a container.

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Kubernetes Storage





- **ReadWriteOnce (RWO)**
- **ReadWriteMany (RWM)**
- **ReadOnlyMany (ROM)**

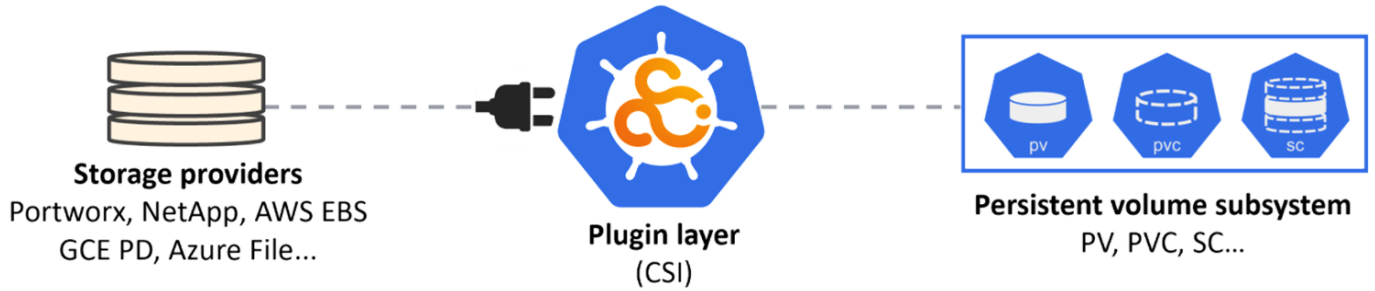
```
$ kubectl apply -f gke-pv.yml
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: test
  capacity:
    storage: 10Gi
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: uber-disk
```

gke-pv.yml

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Kubernetes Storage



Persistent Volume Claim: like a ticket that grants the Pod to the PV

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: test
  capacity:
    storage: 10Gi
  ...
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc1
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: test
  resources:
    requests:
      storage: 10Gi
...
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volpod
spec:
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc1
  containers:
  - name: ubuntu-ctr
    image: ubuntu:latest
    command:
    - /bin/bash
    - "-c"
    - "sleep 60m"
    volumeMounts:
    - mountPath: /data
      name: data
```

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

StatefulSet

Some applications have states, such as Database systems (MySQL, MongoDB, Redis, etc)

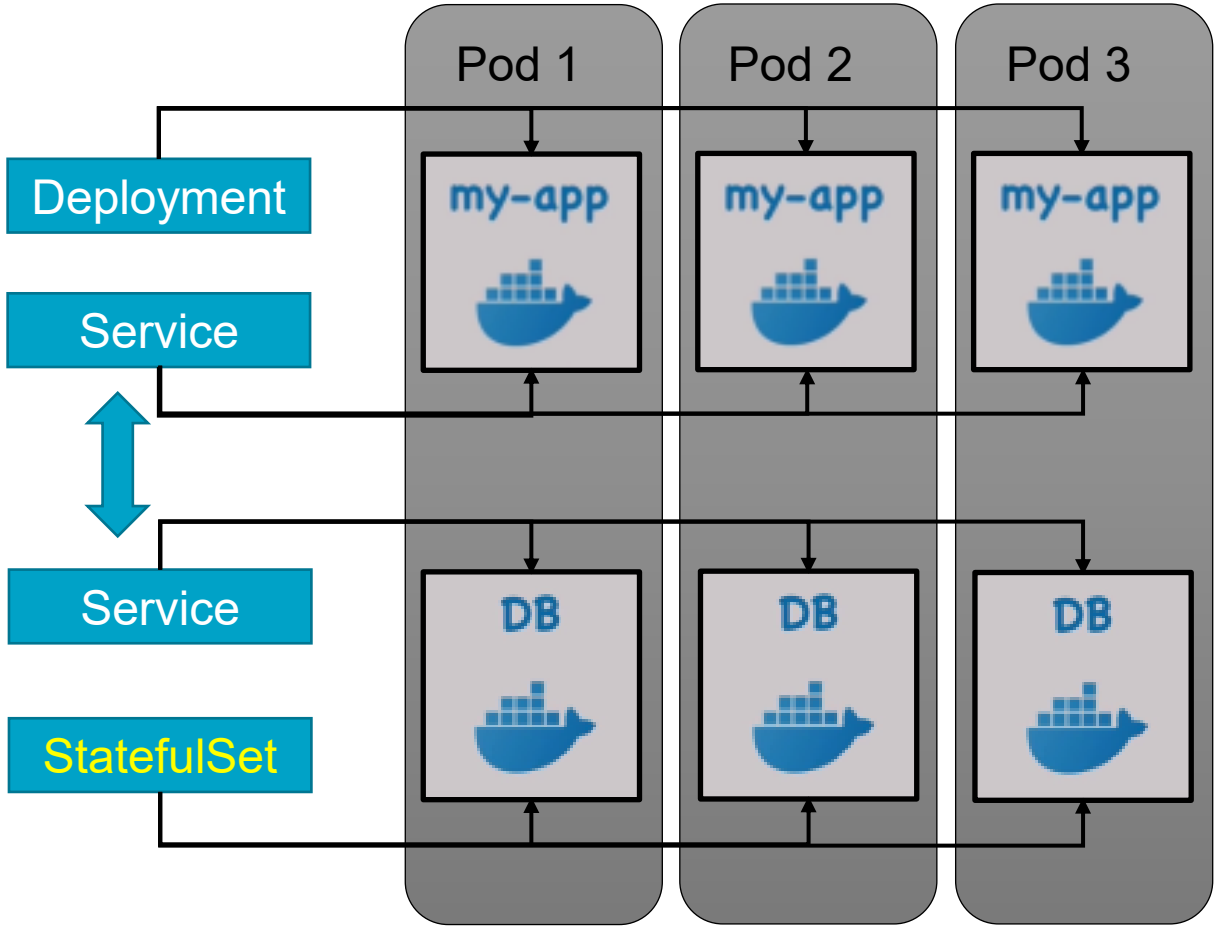
- States guarantee the data consistency.
- Stateful applications cannot be simply scaled up/down by Deployment

StatefulSet is to manage stateful applications.

- deploys and scales a set of Pods (DBs)
- provides guarantees about the ordering and uniqueness of these Pods.

Replicating DBs in K8s via StatefulSet is always difficult

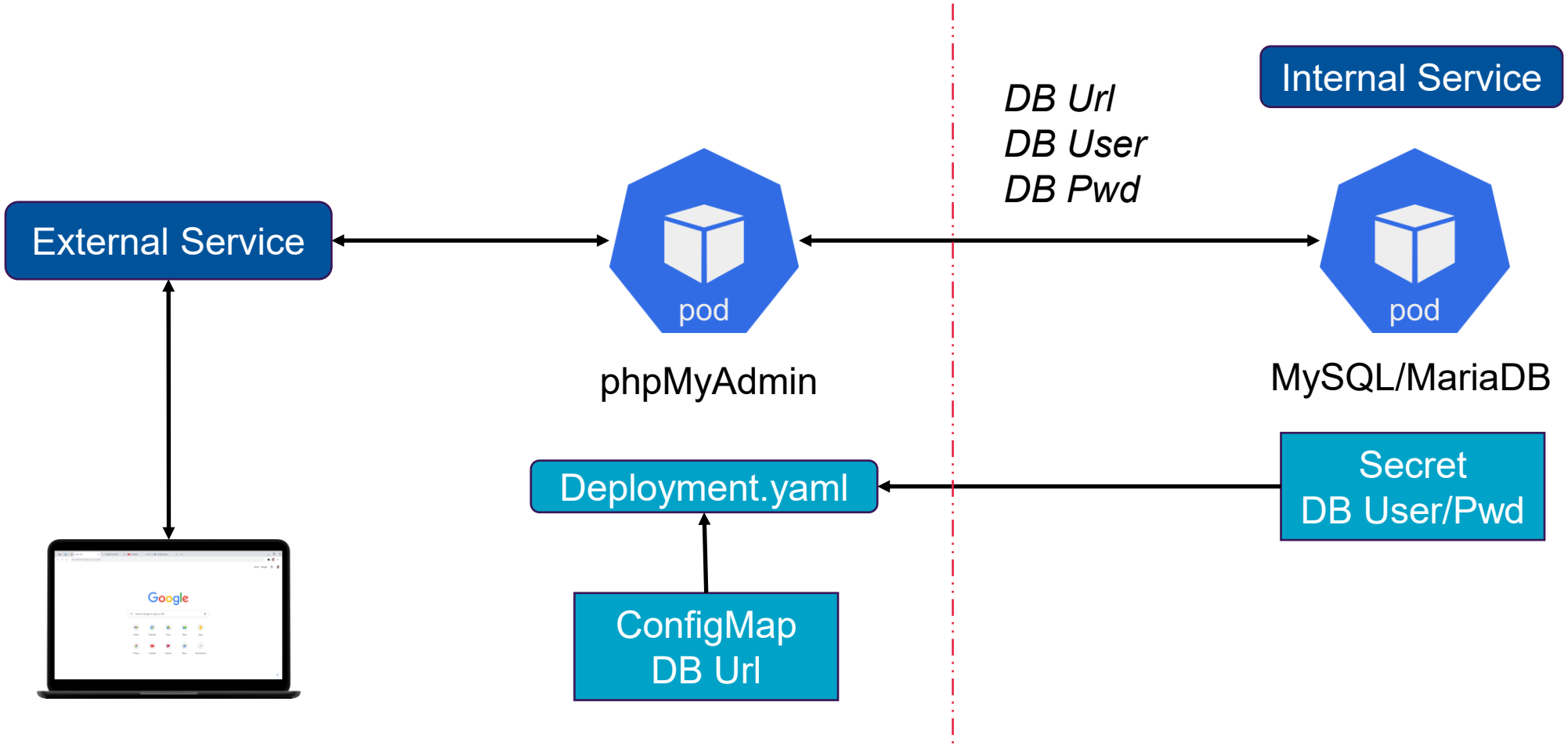
- DBs are often hosted outside of K8s cluster



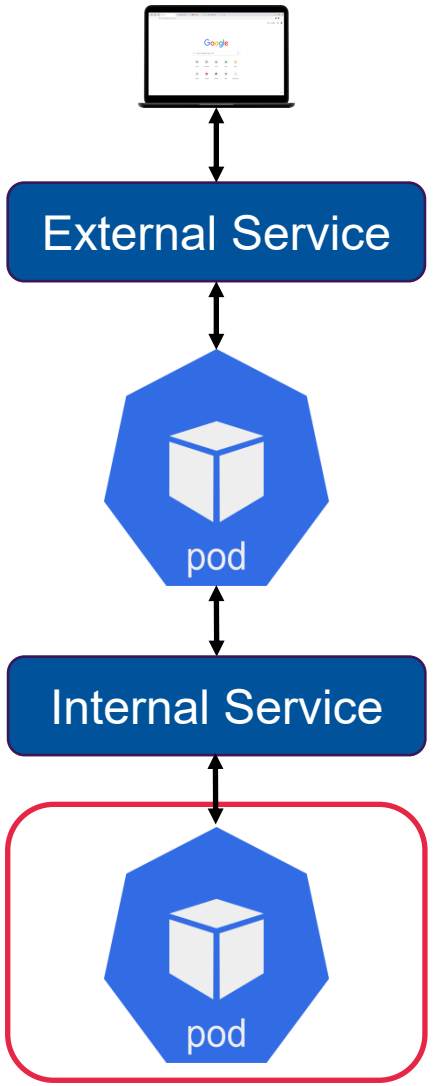
Outline

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- Demo of GKE - Cluster creation
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes
- A K8s deployment example - Mysql & phpMyAdmin

Deploying MySQL and phpMyAdmin on K8s - Framework



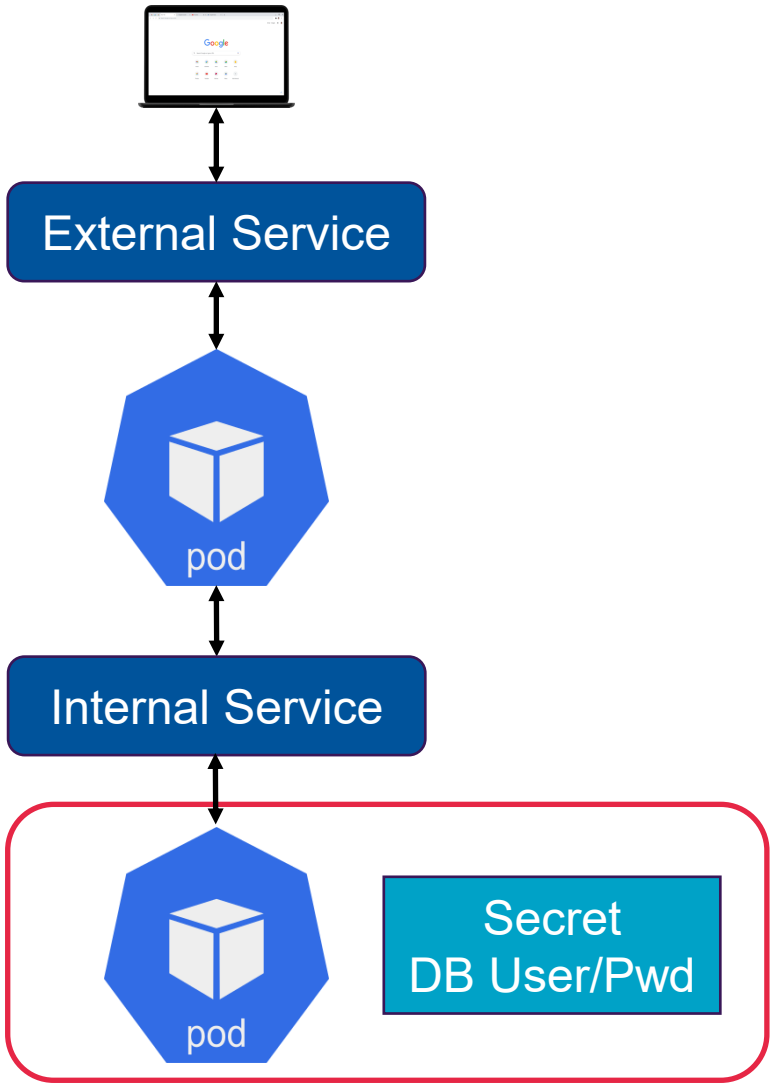
MySQL Pod Setup



mysql_deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql-container
        image: mysql
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: mysql-root-pwd
```

MySQL Secret




```
uqteaching@vm-global:~/lect6_demos$ echo -n 'mysql_root_pwd' | base64
bXlzcWxfcm9vdF9wd2Q=
```

```yaml
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: mysql-secret
5  type: Opaque
6  data:
7    mysql-root-pwd: bXlzcWxfcm9vdF9wYXNzd29yZA==
```

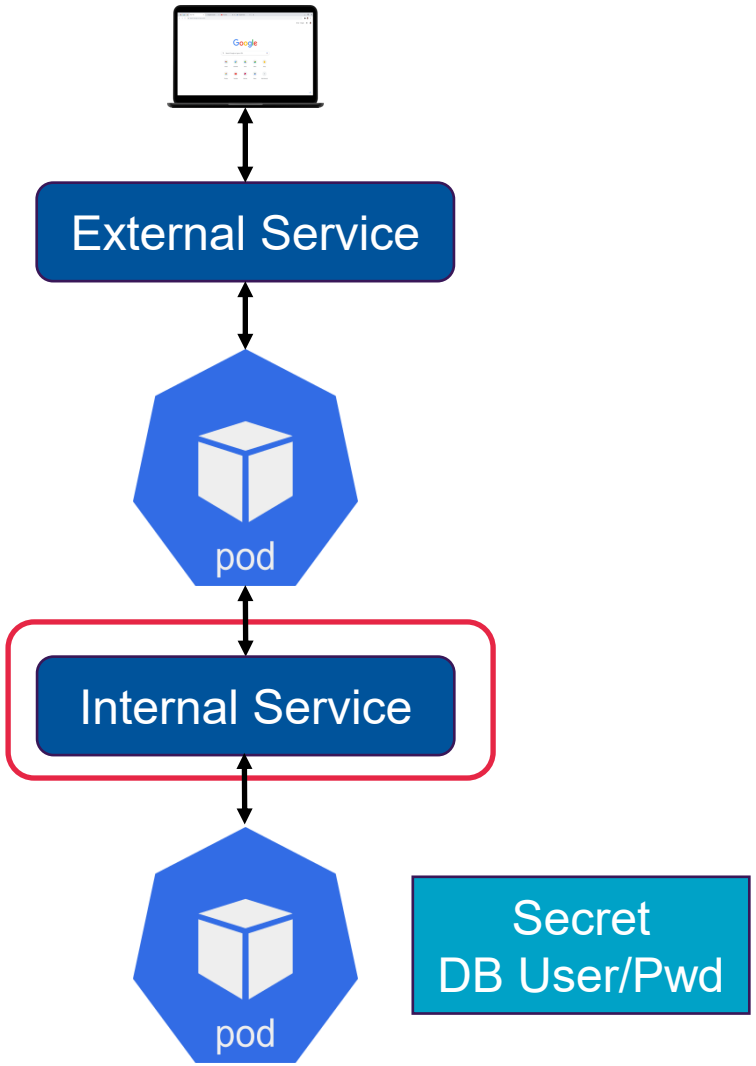
mysql_secret.yaml

Note: Secret must be created before the Deployment

Create mysql secret with kubectl apply

```
uqteaching@vm-global:~/lect6_demos/mysql$ kubectl apply -f mysql_secret.yaml
secret/mysql-secret created
uqteaching@vm-global:~/lect6_demos/mysql$ kubectl get secret
NAME           TYPE     DATA   AGE
mysql-secret   Opaque   1      7s
redis-secret   Opaque   2      5h7m
```

Internal Service



mysql_service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

mysql.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql-container
        image: mysql
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: mysql-root-pwd
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

MySQL Pod/Service/Deployment

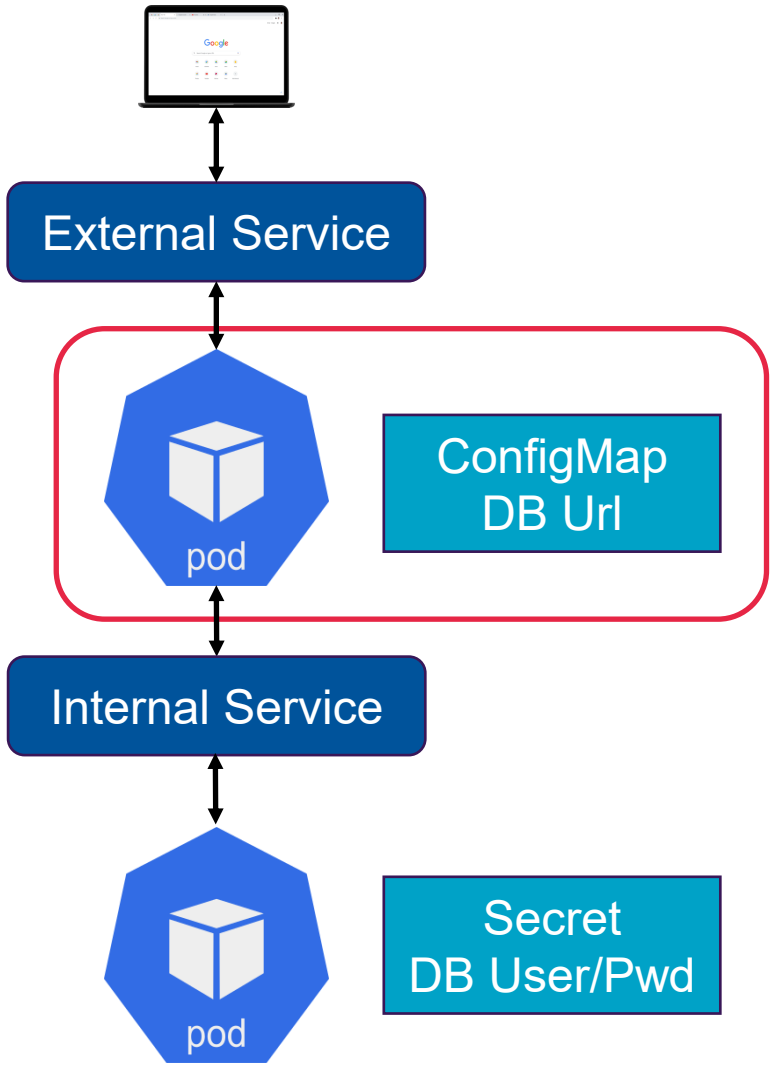
```
uqteaching@vm-global:~/lect6_demos/mysql$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/mysql-deployment-56c94b9c87-jv657   1/1     Running   0          2m56s

NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP    6h56m
service/mysql-service   ClusterIP   10.97.169.71   <none>        3306/TCP   10s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mysql-deployment    1/1     1            1           2m56s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/mysql-deployment-56c94b9c87    1         1         1       2m56s
uqteaching@vm-global:~/lect6_demos/mysql$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
mysql-deployment-56c94b9c87-jv657   1/1     Running   0          3m17s
uqteaching@vm-global:~/lect6_demos/mysql$
```

phpMyAdmin Pod



mysql_configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-configmap
data:
  db_url: mysql-service
```

phpmyadmin_deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phpmyadmin-deployment
  labels:
    app: phpmyadmin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phpmyadmin
  template:
    metadata:
      labels:
        app: phpmyadmin
    spec:
      containers:
      - name: phpmyadmin-container
        image: phpmyadmin
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
          protocol: TCP
        env:
        - name: PMA_HOST
          valueFrom:
            configMapKeyRef:
              name: mysql-configmap
              key: db_url
        - name: PMA_PORTS
          value: "3306"
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: mysql-root-pwd
```

phpMyAdmin Pod/Service/Deployment

```
uqteaching@vm-global:~/lect6_demos/mysql$ kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/mysql-deployment-56c94b9c87-jv657        1/1     Running   0          35m
pod/phpmyadmin-deployment-6f8d4f6854-cgfvz   1/1     Running   0          103s

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/kubernetes             ClusterIP      10.96.0.1        <none>        443/TCP          7h28m
service/mysql-service          ClusterIP      10.97.169.71     <none>        3306/TCP         32m
service/phpmyadmin-service     LoadBalancer   10.109.234.164   <pending>     8081:30001/TCP   4s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mysql-deployment        1/1     1            1           35m
deployment.apps/phpmyadmin-deployment   1/1     1            1           103s

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/mysql-deployment-56c94b9c87        1         1         1       35m
replicaset.apps/phpmyadmin-deployment-6f8d4f6854   1         1         1       103s
```

Proxy for Minikube Running on VM

```
uqteaching@vm-global:~/lect6_demos/mysql$ minikube service phpmyadmin-service
|-----------|---------------------|-------------|---------------------------|
| NAMESPACE |        NAME         | TARGET PORT |            URL            |
|-----------|---------------------|-------------|---------------------------|
| default   | phpmyadmin-service  |        8081 | http://192.168.49.2:30001 |
|-----------|---------------------|-------------|---------------------------|
🎉 Opening service default/phpmyadmin-service in default browser...
👉 http://192.168.49.2:30001
```

```
upstream app_server_30001 {
    server 192.168.49.2:30001;
}
server {
    listen 30001;
    location /proxy {
       proxy_pass http://app_server_30001/;
    }
}
```

```
uqteaching@vm-global:~/lect6_demos/mysql$ sudo vim /etc/nginx/conf.d/upstream.conf
uqteaching@vm-global:~/lect6_demos/mysql$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
uqteaching@vm-global:~/lect6_demos/mysql$ sudo systemctl reload nginx
uqteaching@vm-global:~/lect6_demos/mysql$
```

Kubernetes on the Platforms

Major cloud providers offer managed Kubernetes services to simplify cluster management tasks for users. Here's a comparison of Kubernetes services across three major cloud providers: AWS, GCP, and Azure:

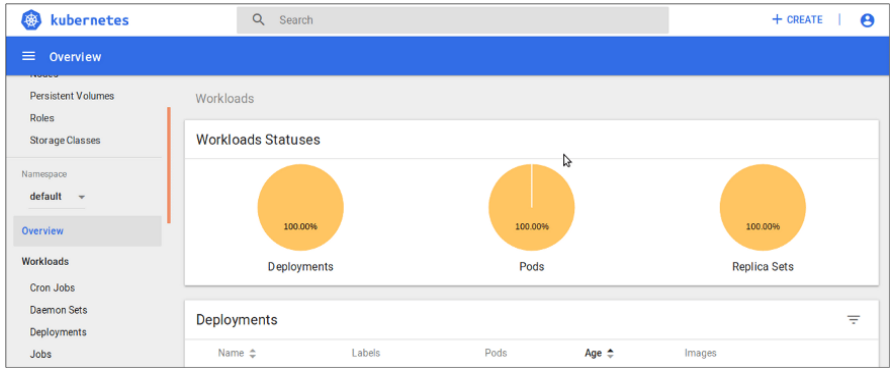
	Service Name	Integration with Native Services	Networking	Auto Scaling	Pricing
GCP	Google K8s Engine (**GKE**)	GKE has deep integrations with Google Cloud services like **BigQuery**, **Cloud Spanner**, etc.	Uses Google's global VPC for connectivity, providing strong network capabilities out of the box.	Yes	Charges for **worker node usage**; control plane (master node) usage is **FREE**.
AWS	Amazon Elastic K8s Services (**EKS**)	EKS integrates with many AWS services like Elastic Load Balancer (**ELB**), Identity and Access Management (**IAM**), and CloudWatch.	Supports VPC Peering, allowing private communication between different VPCs.	Yes	Charges for control plane and worker node usage*. Charges for resource usage*.
Azure	Azure K8s Service (**AKS**)	AKS integrates with Azure services such as **Azure Active Directory**, **Azure Monitor**, and **Azure Logic Apps**.	Offers Azure Virtual Network, ensuring private communication between different services.	Yes	Does not charge for the Kubernetes management infrastructure; users pay only for the virtual machines and associated resources**.

All three cloud providers offer extensions for developers, robust support options, and Service Level Agreements (SLAs) to ensure high availability and security.

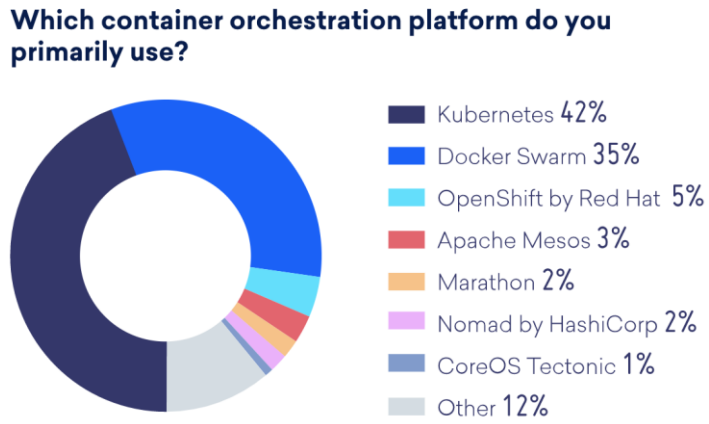
* Amazon EKS pricing model contains two major components: customers pay $0.10 per hour for each configured cluster, and pay for the AWS resources (compute and storage) that are created within each cluster to run Kubernetes worker nodes.
**https://azure.microsoft.com/en-au/pricing/details/kubernetes-service/#pricing

Kubernetes vs Docker Swarm

Features	Kubernetes	Docker Swarm
Installation & Cluster Configuration	Installation is complicated; but once setup, the cluster is very strong	Installation is very simple; but cluster is not very strong
GUI	GUI is the Kubernetes Dashboard	There is no GUI
Scalability	Highly scalable & scales fast	Highly scalable & scales 5x faster than Kubernetes
Auto-Scaling	Kubernetes can do auto-scaling	Docker Swarm cannot do auto-scaling
Load Balancing	Manual intervention needed for load balancing traffic between different containers in different Pods	Docker Swarm does auto load balancing of traffic between containers in the cluster
Rolling Updates & Rollbacks	Can deploy Rolling updates & does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
Data Volumes	Can share storage volumes only with other containers in same Pod	Can share storage volumes with any other container
Logging & Monitoring	In-built tools for logging & monitoring	3rd party tools like ELK should be used for logging & monitoring




```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```



Source: quarterly report on developer trends in the cloud by Digital Ocean

https://medium.com/edureka/kubernetes-vs-docker-45231abeeaf1

Review

- Introduction to K8s
- K8s Architecture and its Components
  - Control Plane Node and Work Node
- Declarative Model in K8s
- Reconciliation Loop
- Minikube & Kubectl
- K8s Objects:
  - Pod & ReplicaSet & Deployment
  - & Service & Ingress & StatefulSet
  - & ConfigMap & Secrets & Volumes

What's Next?

Getting more hands-on practice

- Play with Kubernetes classroom: https://training.play-with-kubernetes.com/kubernetes-workshop/
- Practice creating a K8s cluster on GCP and deploying a stateless application (practical session)

Getting prepared for certification

- https://cloud.google.com/certification/cloud-architect

Remember to stop GKE clusters when not in use…

```
uqyluo@cloudshell:~ (mythic-dynamo-300704)$ gcloud container clusters resize cluster-demo --zone=australia-southeast1-a --num-nodes=0
Pool [default-pool] for [cluster-demo] will be resized to 0.

Do you want to continue (Y/n)?  y
```

What's Next?

A few more things to explore…

- Kubernetes Dashboard (https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/)
- ConfigMaps (https://kubernetes.io/docs/concepts/configuration/configmap/)
- Threat Modeling (https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/)

https://kubernetes.io/docs/concepts/configuration/configmap/

Tutorial & Practical for Week 6

Tutorial 5 (Week 6)

1. Please describe what Kubernetes (K8s) is.
2. What are the motivations of using Kubernetes?
3. Please describe the components and their functions in a Kubernetes cluster?
4. Please generally summarise the workflow of Kubernetes

Practical 5 (Week 6)

1. Use Docker commands to deploy a PHP-based web development environment with four containers.
2. Creating K8s cluster
3. Deploy a stateful application on K8s
4. Consultation for Programming Task 1

Next (Week 6) Topic:

Databases in Cloud Computing I (Dr Zhi Chen)