universität innsbruck

Advanced Distributed Systems

Lecture 03-Kubernetes

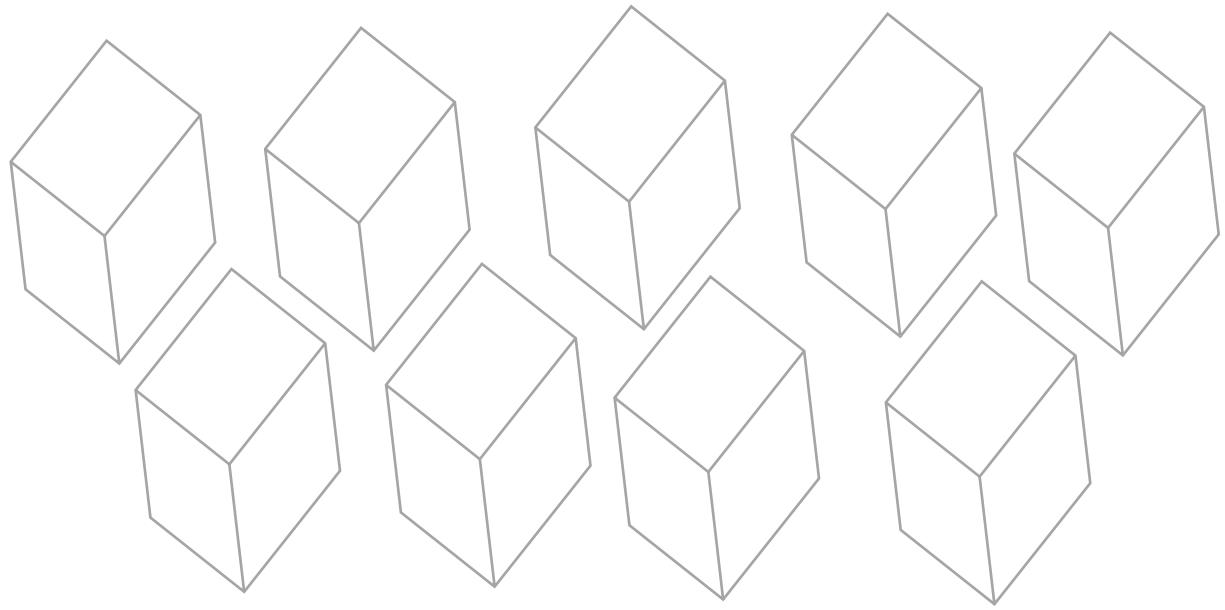
Dr. Zahra Najafabadi Samani

Agenda

- Introduction
- Worker node
- Mater node
- Networking
- Core objects
- Minikube

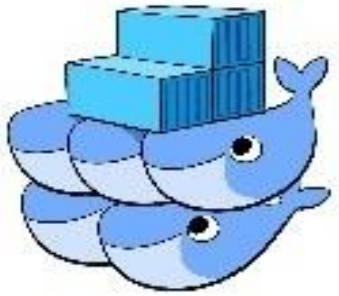
The need for a container orchestration tool

- Trend from monolith to microservices
- Increased usage of containers
- Difficulty of Managing the application containing hundreds of containers
- Demand for a proper way of managing those hundreds of containers



Container orchestration tools

Marathon (Mesosphere)

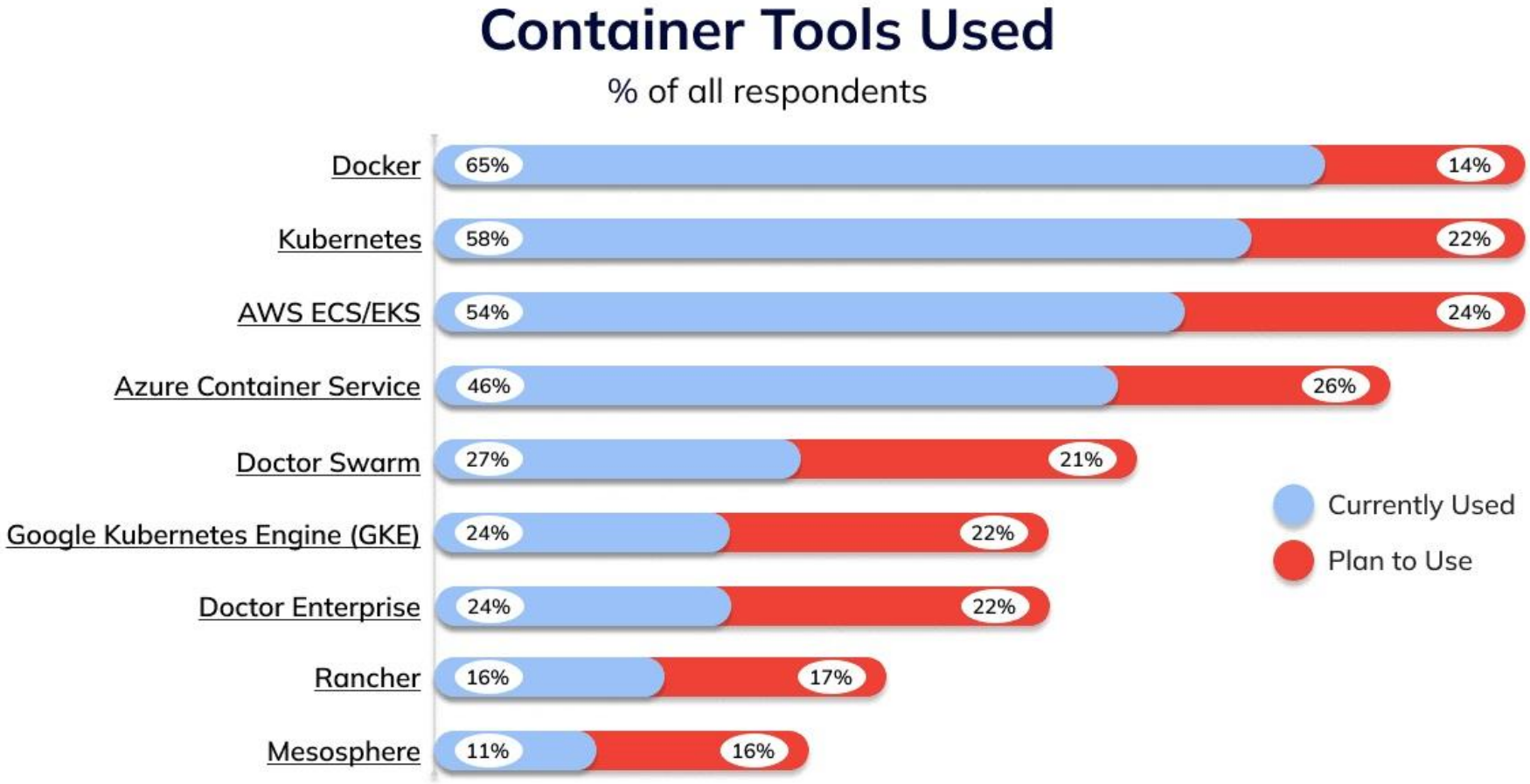


Docker Swarm

Nomad (HashiCorp)

Kubernetes

Container orchestration tools



Introduction

What is Kubernetes

- The name Kubernetes originates from Greek, meaning "helmsman" or "pilot", and is the root of "governor" and "cybernetic".
- K8s is an abbreviation derived by replacing the 8 letters "ubernete" with 8.
- With Kubernetes you can deploy a full cluster of multi-tiered containers (frontend, backend, etc.) with a single configuration fille and a single command.
- 100% Open source, written in Go
- Manage applications, not machines

What is Kubernetes

Official definition of Kubernetes:

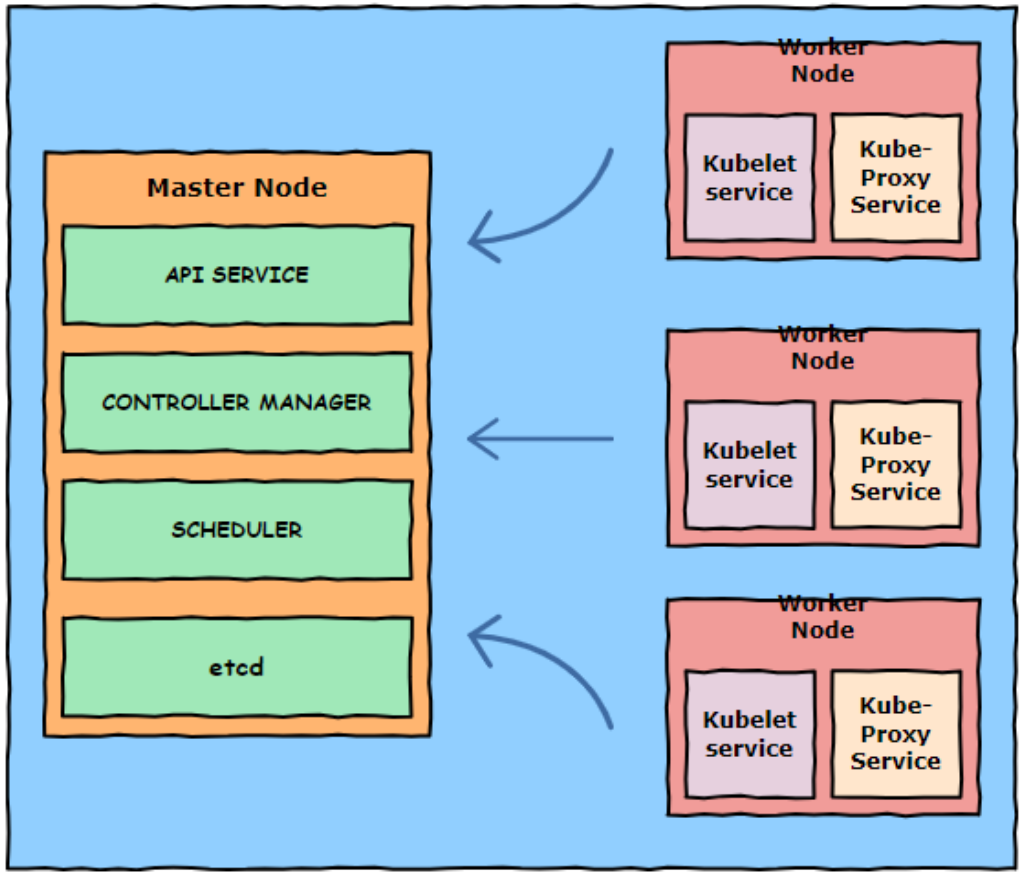
- Open source container orchestration tool
- Developed by google
- Helps to manage containerized applications which made of hundreds or thousands containers
- Helps to manage them in different environments such as:
  - Physical machines
  - Virtual machines
  - Cloud environments
  - Hybrids environments

Advantages of Kubernetes

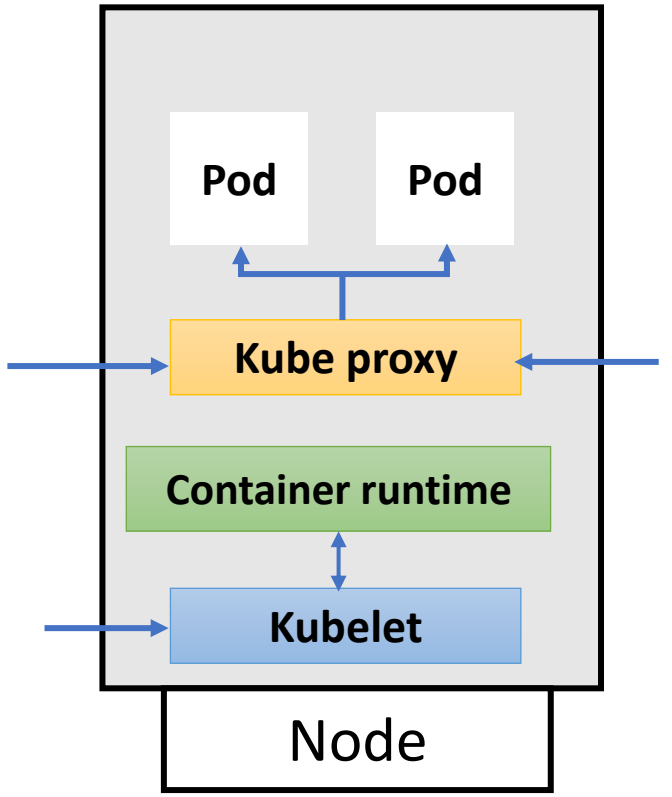
- High availability or no downtime
- High performance
- Disaster recovery- backup and restore
- Deploy your applications quickly
- Scale your application
- Seamlessly roll out new features
- Optimize use of your hardware by using only the resources you need
- Portable: public, private, hybrid, multi-cloud
- Self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

Kubernetes architecture

- A **Kubernetes cluster** is a cluster of nodes configured in a master/slave architecture.
- A Kubernetes cluster is made up of one **master** node and several **worker** nodes.
- The worker nodes are responsible for running the containers and doing any work assigned to them by the master node.
- The master node looks after:
  - scheduling and scaling pods
  - monitoring and maintaining the state of the cluster
  - implementing updates and joining new nodes

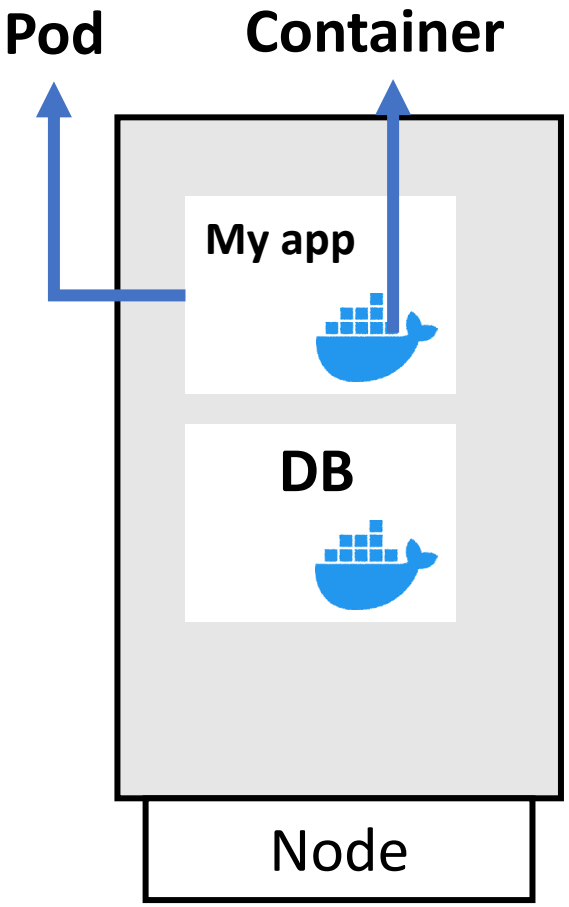


Worker node



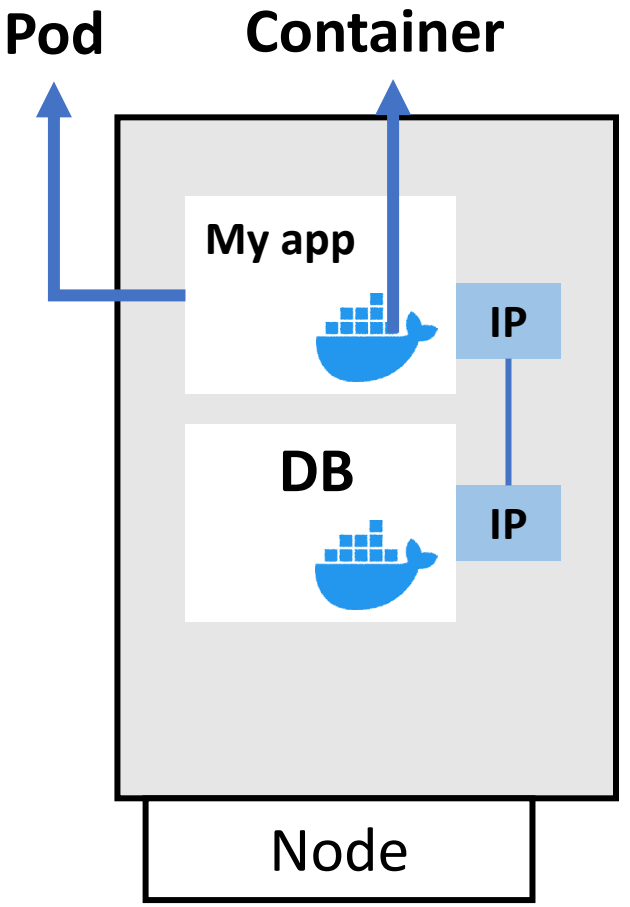
Pod

- Smallest and atomic unit of Kubernetes
- Abstraction that represents a group of one or more application containers, which are relatively tightly coupled.
- Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion.
- In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.



Pod

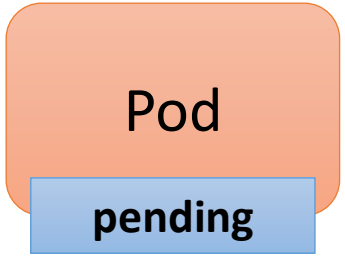
- Each Pod gets its own IP address
- IP addresses are not public
- Pods communicate with each other through IP addresses
- New IP address on re-creation:
  - If Pod dies due to crashing the application inside, a new one get created with new IP address



Lifecycle of a pod

The lifecycle of a pod goes through various phases.

Lifecycle of a pod



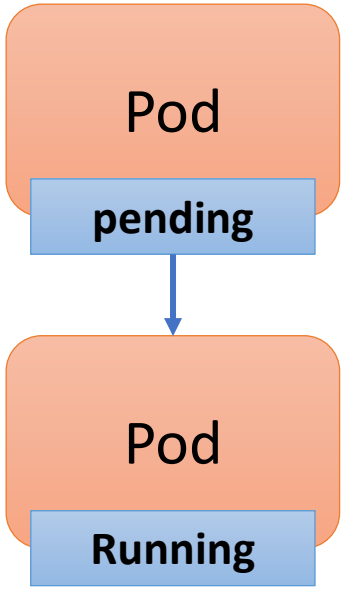
The lifecycle of a pod goes through various phases.

1. Pending:

- When a new pod is created, it initially enters the "Pending" phase.
- During this phase, the Kubernetes scheduler is responsible for assigning the pod to a node in the cluster. The scheduler takes into account resource requirements, node affinity, and other constraints.

Lifecycle of a pod

2. **Running**:
   - Once a pod is scheduled to a node and all its containers are successfully started, it enters the "Running" phase.
   - In this phase, the containers within the pod are executing, and they can serve their intended functions.
   - The pod will remain in the "Running" state as long as the containers continue to run without any issues.



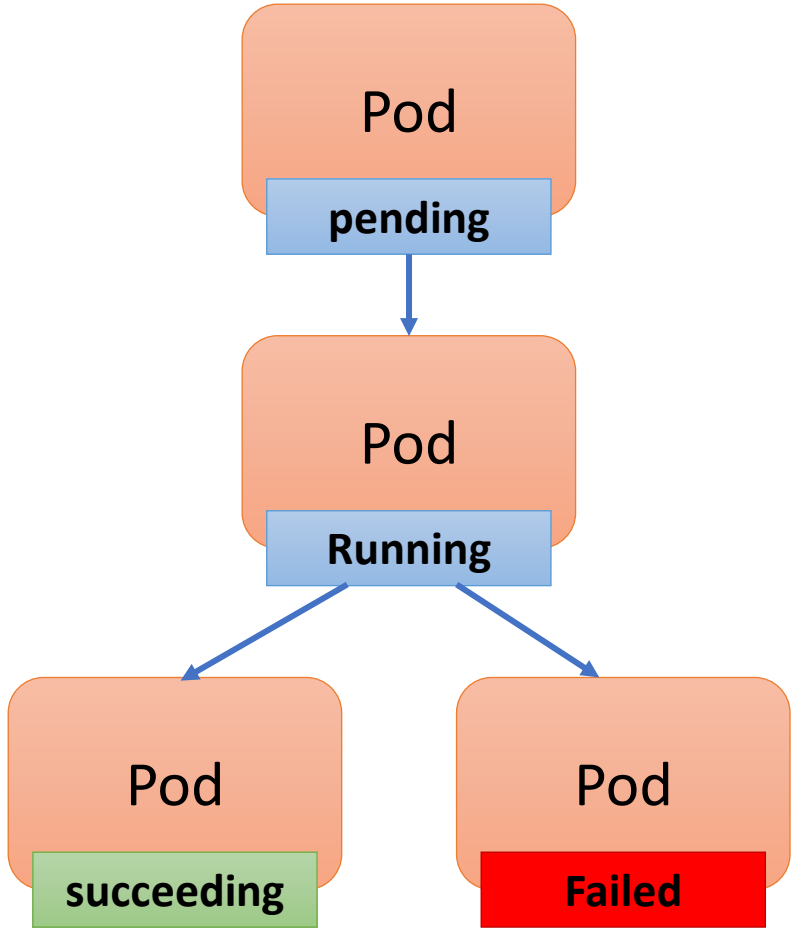
Lifecycle of a pod

1.Succeeded:

- This phase is typically seen in batch jobs or one-time execution tasks.
- All the containers within the pod have completed their tasks successfully and then terminated.
- The pod remains in this phase until it is terminated or deleted.

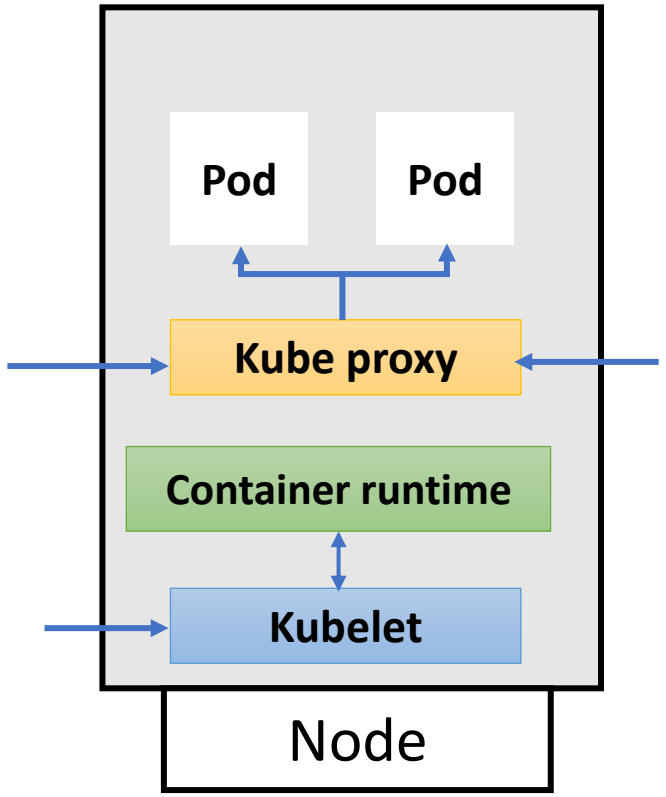
2.Failed:

- If any of its containers encounter errors or issues during execution that cause them to exit.
- If at least one container within it has failed. Other containers within the same pod may still be running.
- The pod remains in the "Failed" state until it is terminated or deleted.



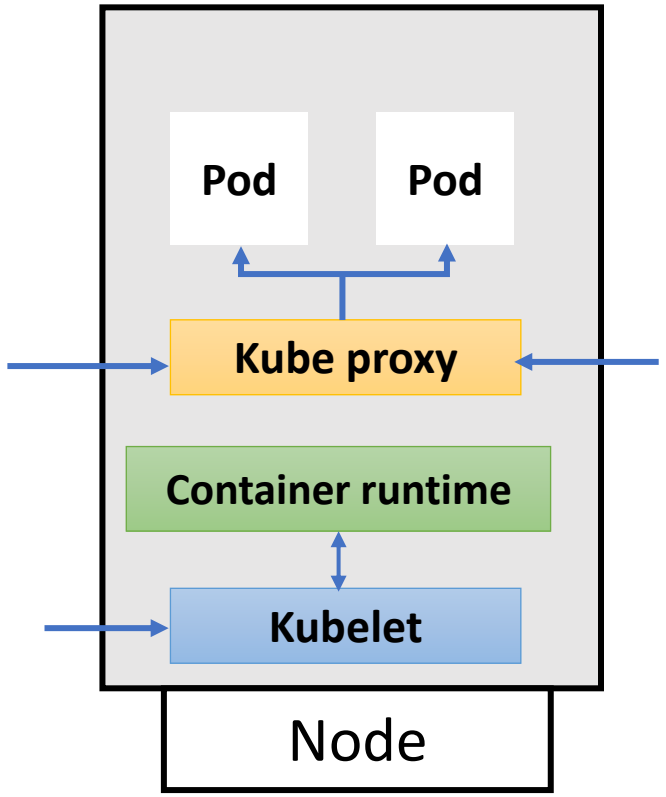
Kubelet

- An agent that runs on each node in the cluster.
- Makes sure that containers are running in a pod.
- Interacts with both the container and node.
- Is responsible for running and start a pod with the container inside.
- Assigns the resources to the container like CPU, RAM,…
- Takes a set of Pod Specs that are provided through various mechanisms and ensures that the containers described in those Pod Specs are running and healthy.



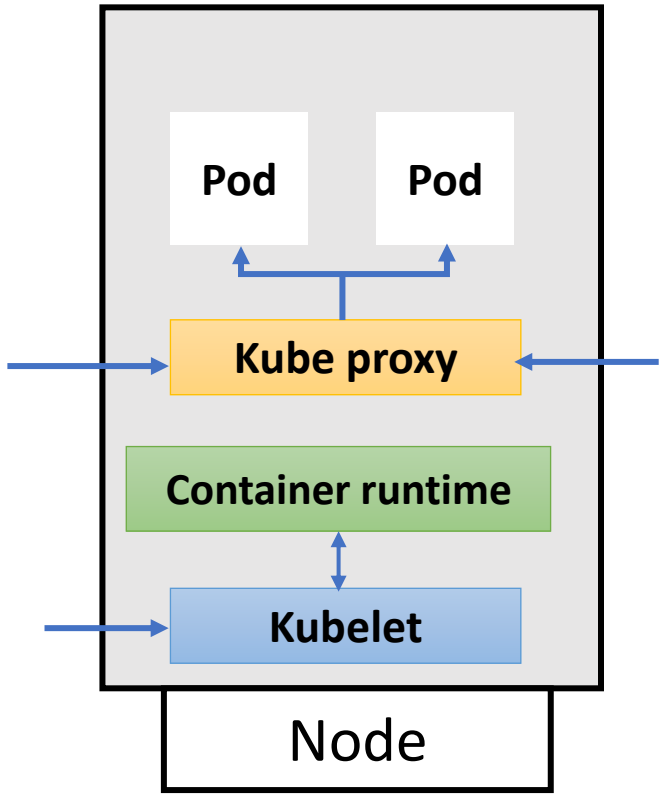
Container runtime engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - docker
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)



Kube proxy

- Manages the network rules on each node.
- Forwards the requests to the pod through a forwarding intelligent that makes sure that communication works in a correct way with low load.
- Performs connection forwarding or load balancing for kubernetes cluster services.



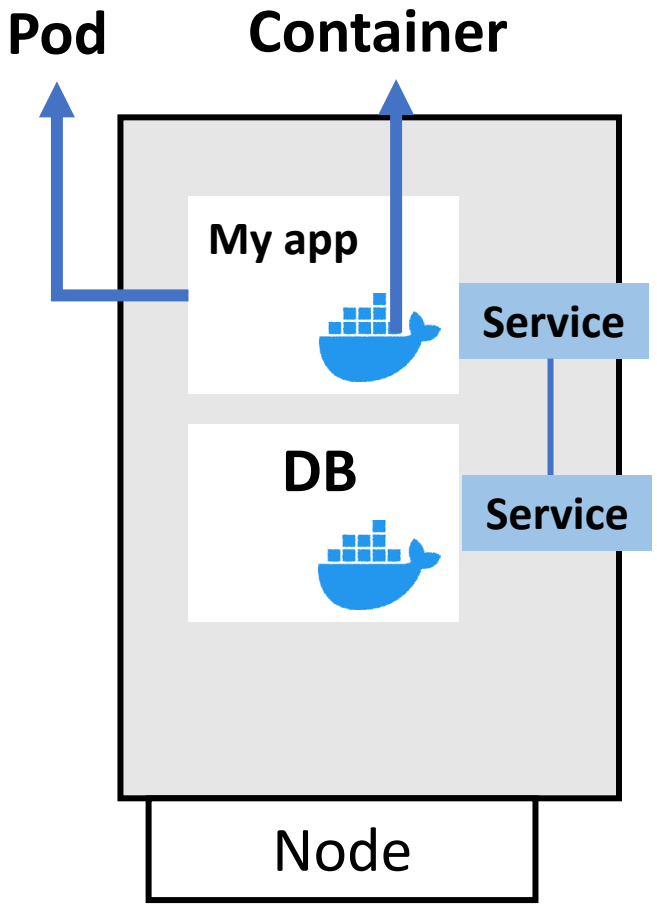
Service

Internal service:

- Services solve the problem temporary pod IP addresses
- Service is static or permanent IP address attached to each pod
- The lifecycle of pod and service are not connected
  - Even a pod dies the service and IP address will be stay

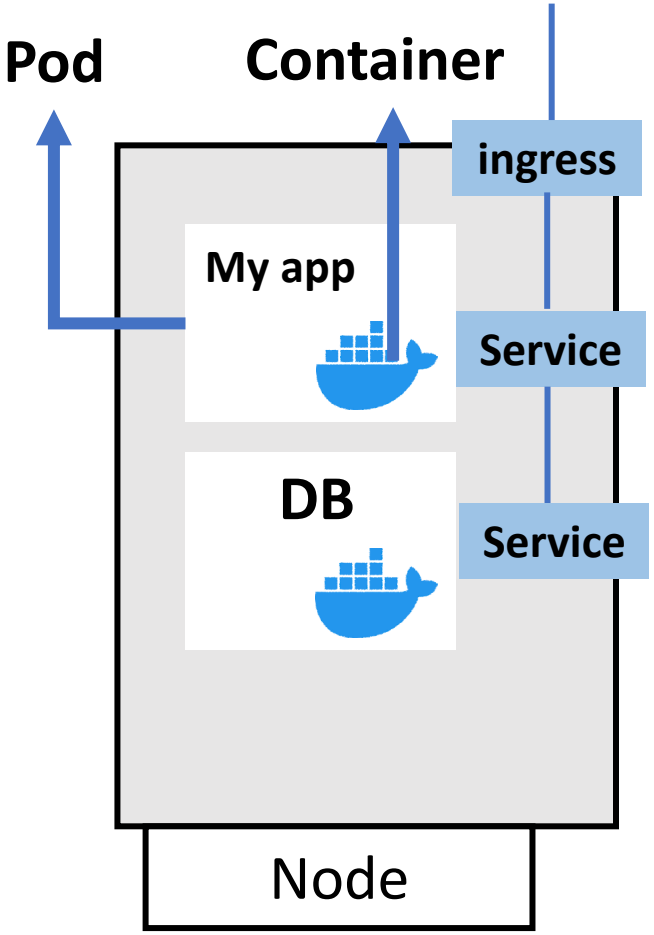
External service:

- Make the communications from external sources
- Make the pod (application) accessible through the browser
- URL: http:// Pod IP address. Port number of the service



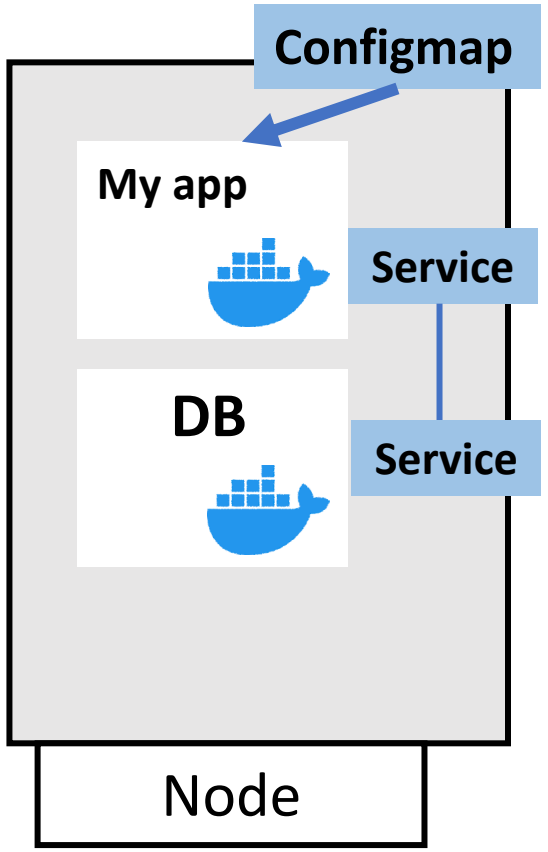
Ingress

- A component to make the pod (application) accessible through the browser
- An API object that manages external access to the services in a cluster
- The external requests go to the ingress
- Ingress forward the request to the service
- Provides load balancing, SSL termination and name/path based virtual hosting
- Support authentication and authorization mechanisms for securing access to services.
- Gives services externally reachable URLs



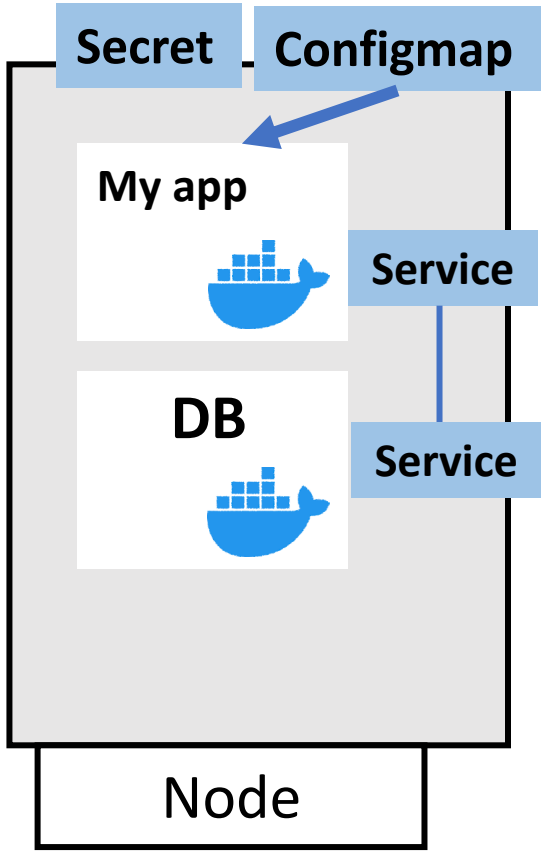
Configmap

- External configuration to your application
- It contains configuration data like URL of database or other services
- It is connected to the pod
- Pod get information from it



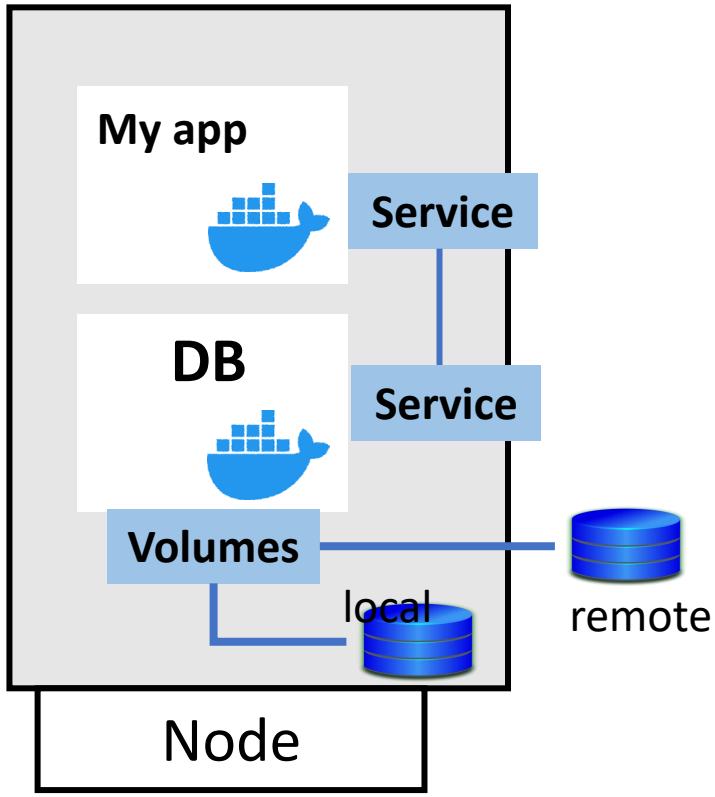
Secret

- Used to store secret data like credential in base64 encoded format
- It is connected to the pod
- Pod can read data from secret
- Use it as environment variables or as a properties file



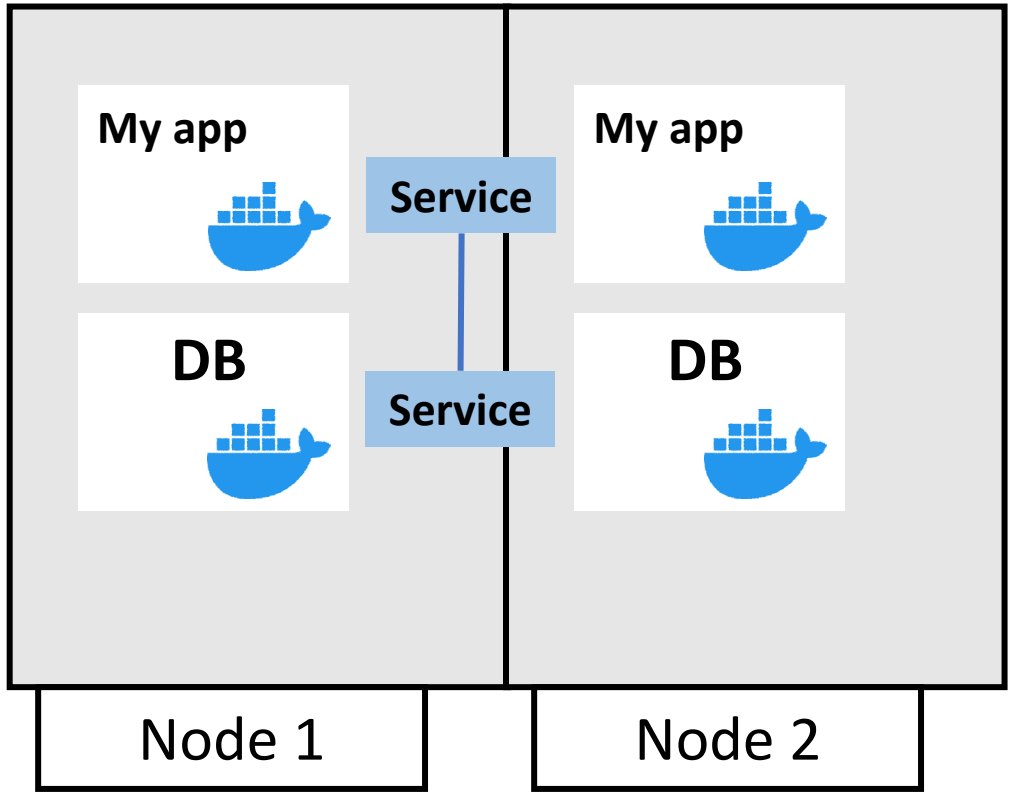
Volume

- Attach a physical storage (hard drive) to your pod
- Local storage in the local machine where the pod is running
- Remote storage, outside of the K8s cluster
  - Cloud storage
  - Private storage
- Create a persistent data storage that store the data when the database pod get restarted



Replication

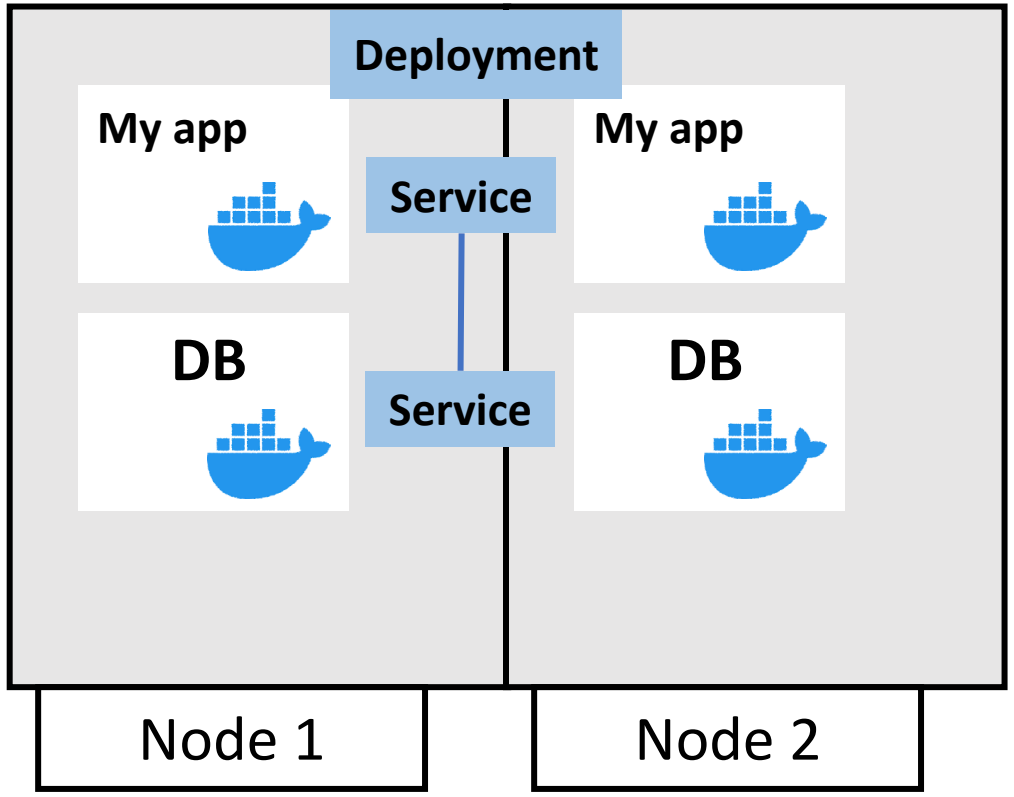
- It replicate every thing in the another node
- It avoid down time in case pod crashes or restart
- It is also connected to the service with two functionalities:
  - Permanent IP
  - Load balancer
- If one pods dies, the service forward the request to another replica



Deployment

- Blueprint for pods
- Abstraction of Pod and facilitate creating the pods
- We create deployment to create pods
- Deployment specify the number of replicas for each pod

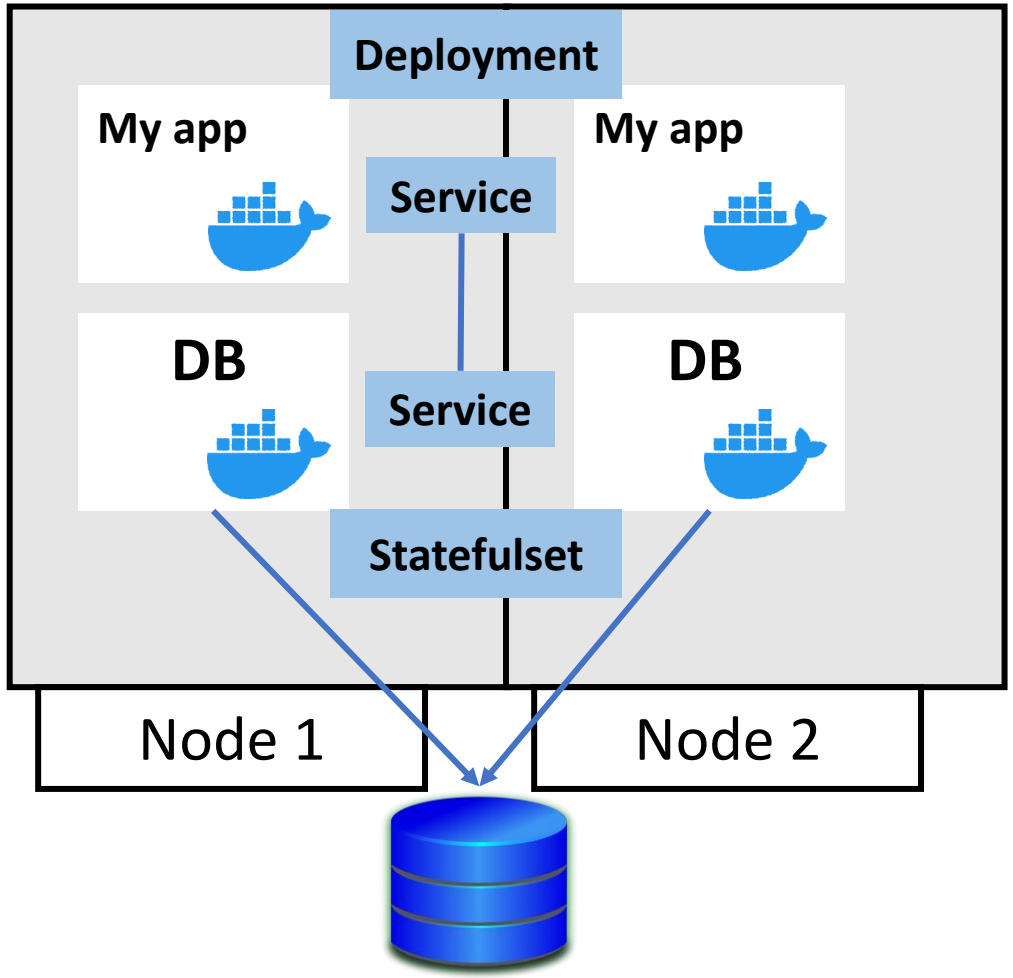
In practice we work with deployments not pods



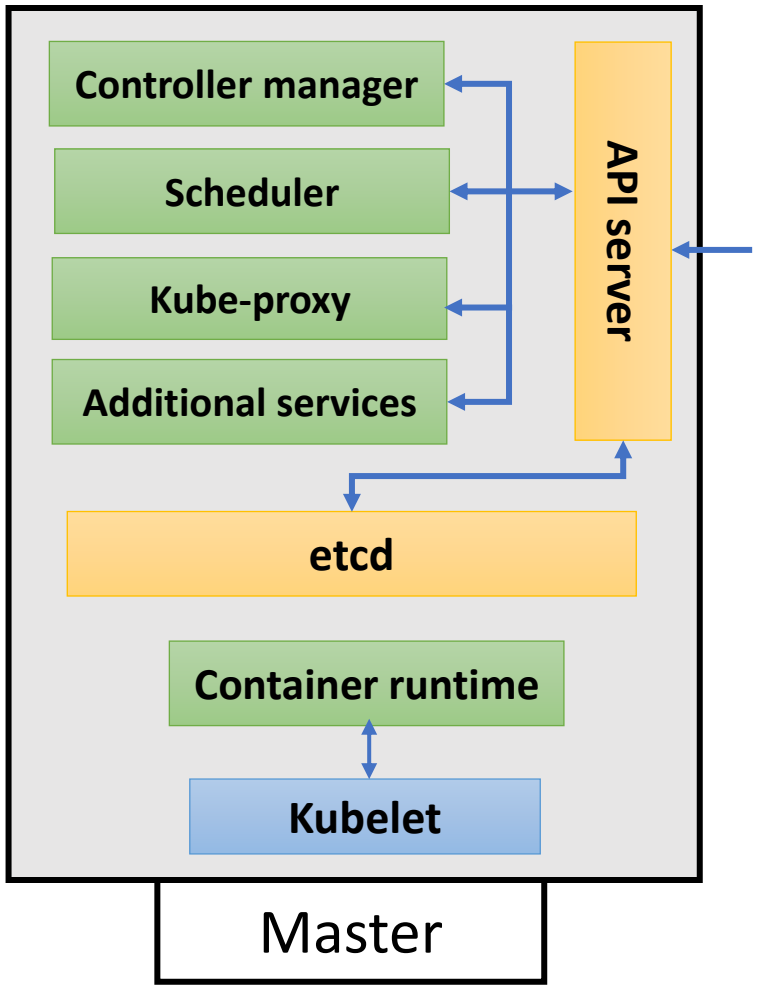
Statefulset

- Statefulset component is responsible for data consistency to specify with pod is writing/reading to/from the storage
- DB are often hosted outside of K8s cluster
- All databases have same data storage

Deployment is employed for stateless apps, while Statefulset is for stateful applications like date bases (mongoDB, elastic,..)

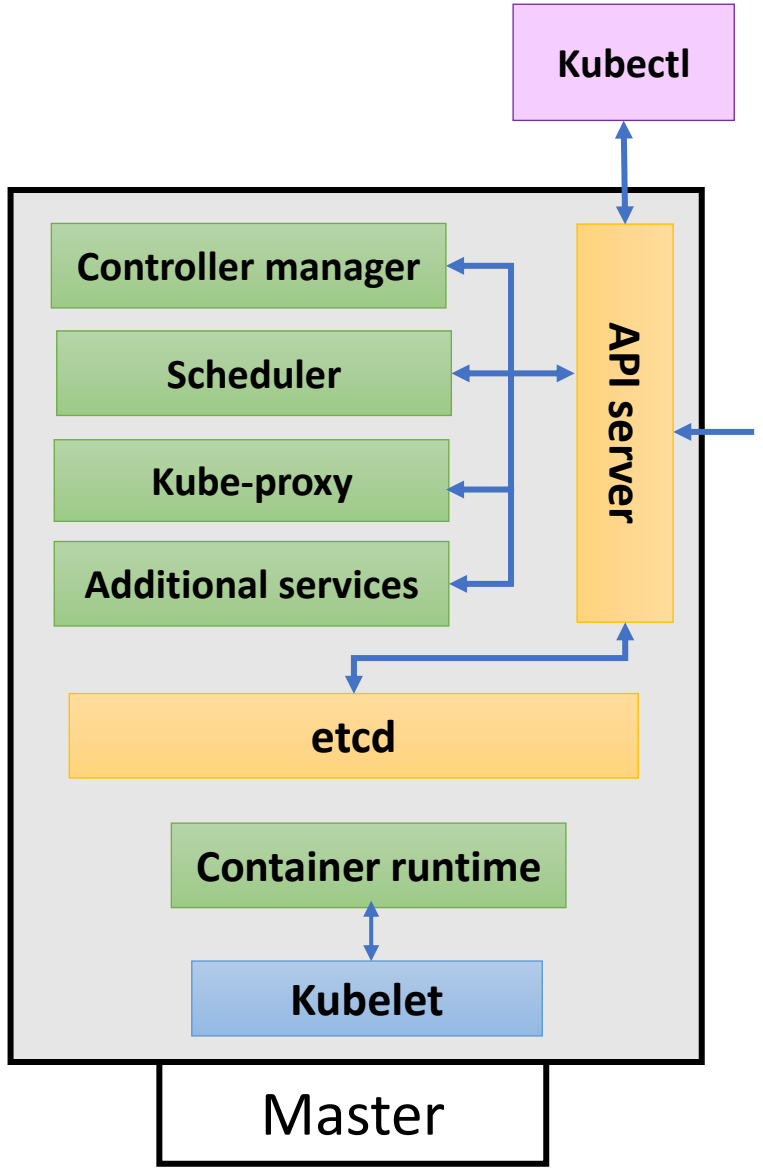


Master node


Controller manager
Scheduler
Kube-proxy
Additional services
API server
etcd
Container runtime
Kubelet
Master

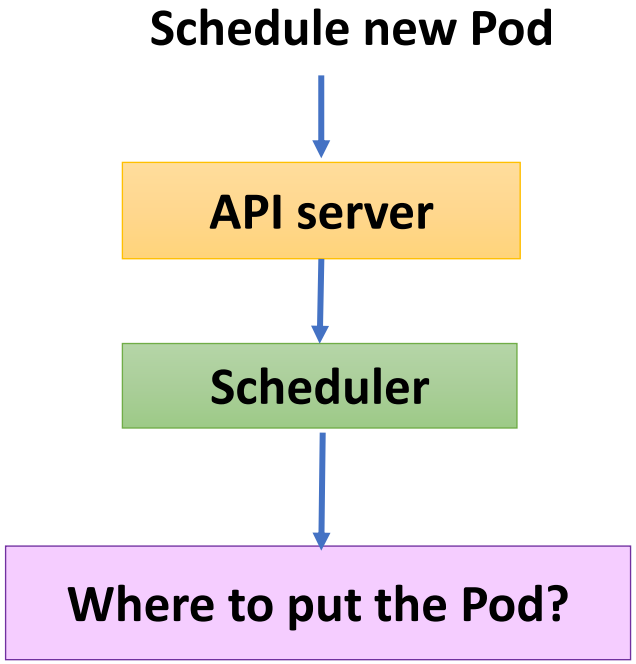
API server

- It Provides a forward REST interface into the Kubernetes control plane and datastore.
- It Is like a cluster gateway.
- All clients and applications interact with Kubernetes through the API Server using command line tool like kubectl.
- It acts as the gatekeeper to the cluster by handling
  - authentication and authorization
  - request validation
  - mutation
  - admission control



Scheduler

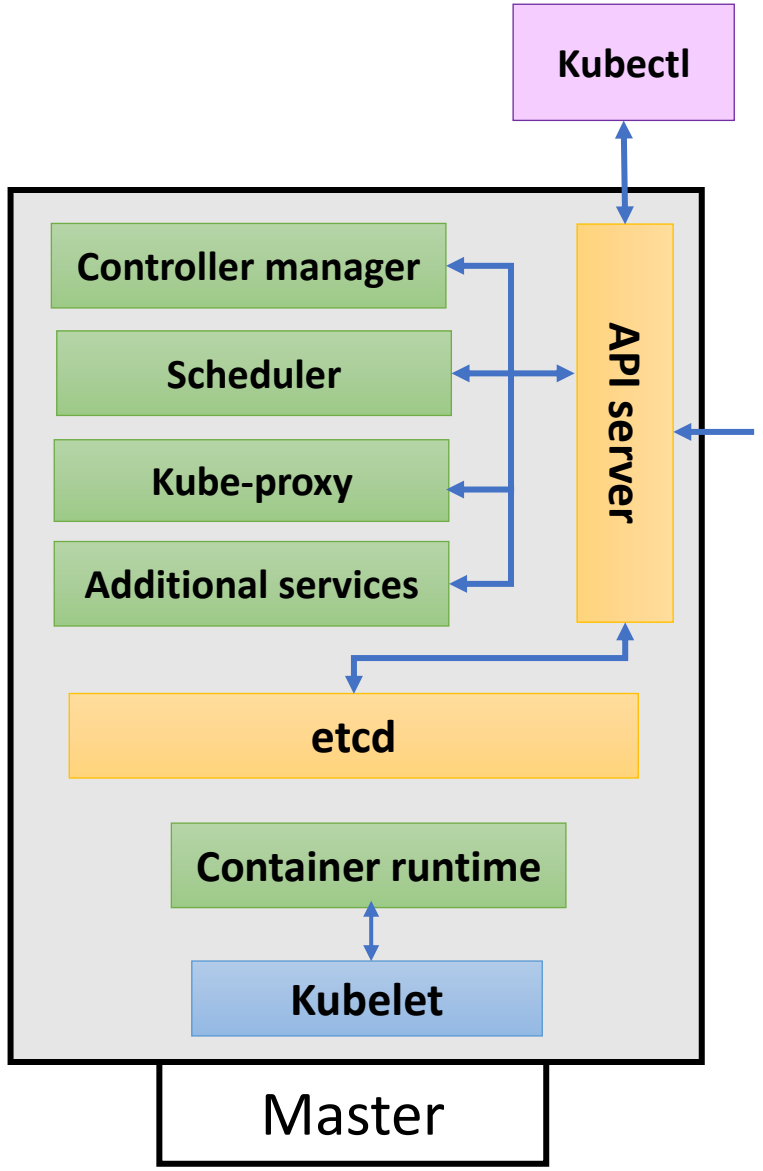
- Watches newly created pods that have no node assigned, and selects a node for them.
- Factors taken into account for scheduling decisions include:
  - individual and collective resource requirements,
  - hardware/software/policy constraints,
  - affinity and anti affinity specifications,
  - and deadlines



Controller manager

Monitors the cluster state via the API server and **steers the cluster towards the desired state**.

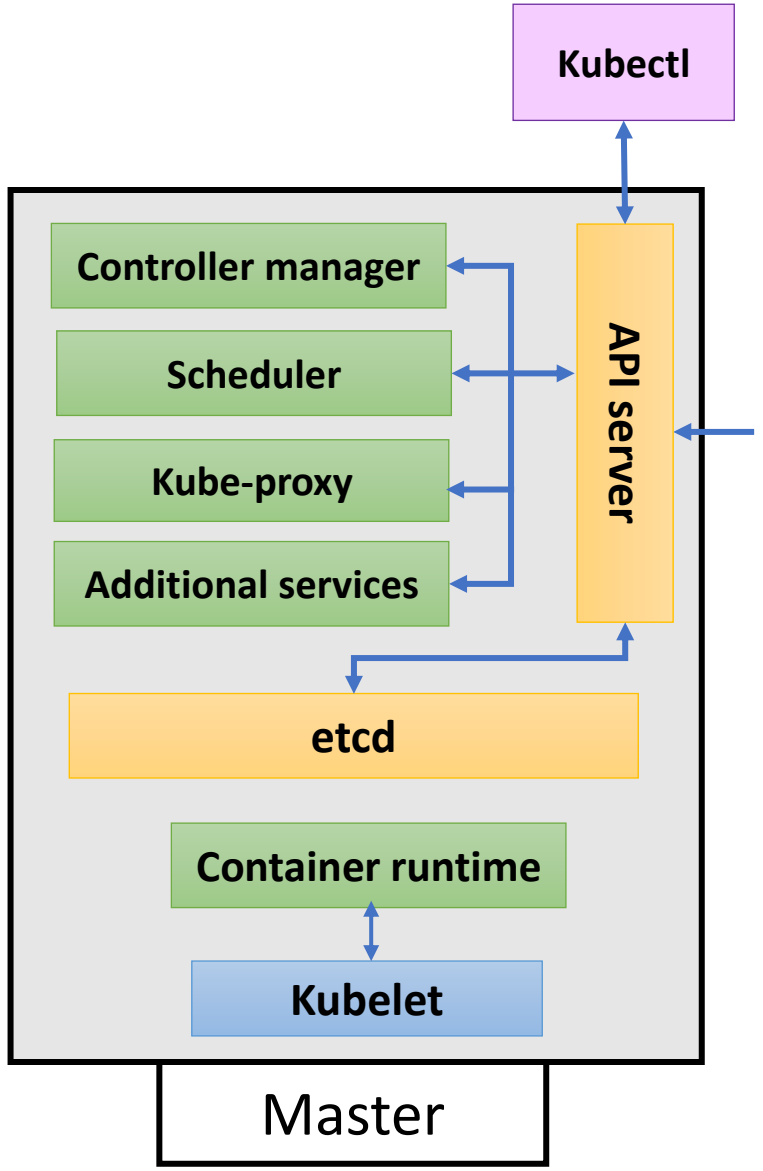
- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- Endpoints Controller: Populates the endpoints object (joins Services & Pods).
- Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.



etcd

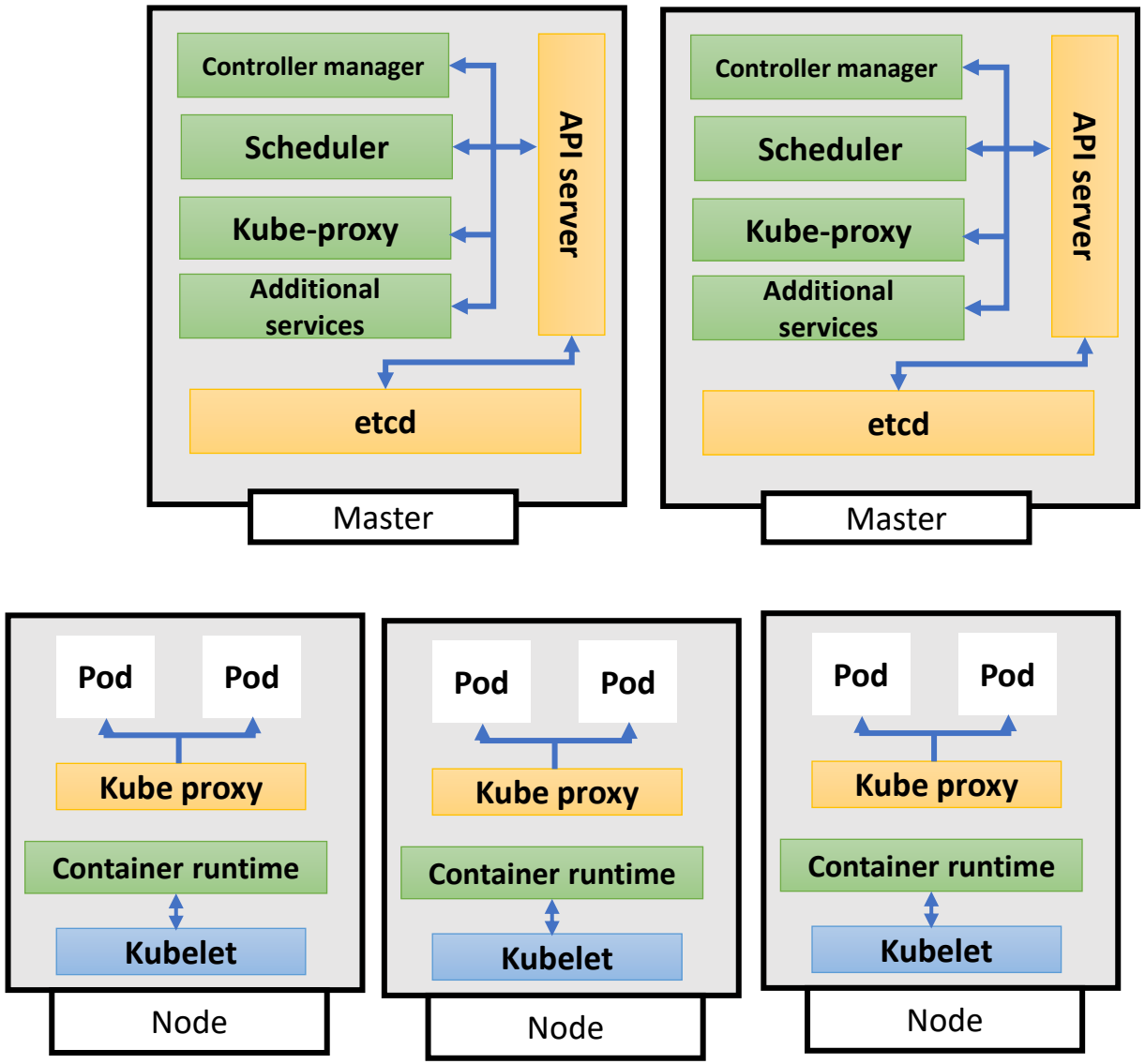
- Acts as the cluster datastore and any change in the cluster(e.g., when a pod get schedule, pod dies,…) get store in the key value store.
- Stores objects and config information.
- provides a strong, consistent and highly available key-value store for persisting cluster state.
- All the master component work based on the data in the etcd.
- Uses *“Raft Consensus”* among a quorum of systems to create a fault-tolerant consistent *“view”* of the cluster.

https://raft.github.io/



Cluster setup

- There is at least one master nodes in every cluster.
- Master nodes have less load and have less resources(RAM, CPU, Storage)
- Add new master/ worker node:
  - Get new bare server
  - Install all the master/worker node components
  - Add it to the cluster



Networking

Fundamental networking Rules

Container-to-Container

- Containers within a pod exist within the **same network namespace** and share an IP.
- Enables intra pod communication over *localhost*.

Pod-to-Pod

- Allocated **cluster unique IP** for the duration of its life cycle.
- Pods themselves are fundamentally ephemeral.

Fundamental networking Rules

Pod-to-Service

- Managed by **kube-proxy** and given a **persistent cluster unique IP**
- Exists beyond a Pod's lifecycle.

External-to-Service

- Handled by **kube-proxy.**
- Works in cooperation with a cloud provider or other external entity (load balancer).

Core objects

Name spaces

- Namespaces are a logical cluster or environment, and are the primary method of partitioning a cluster or scoping access.
- *Namespaces* provides a mechanism for isolating groups of resources within a single cluster.
- Multiple namespaces are intended for use in environments with many users spread across multiple teams, or projects.

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

```
$ kubectl get ns --show-labels
NAME          STATUS    AGE       LABELS
default       Active    11h       <none>
kube-public   Active    11h       <none>
kube-system   Active    11h       <none>
prod          Active    6s        app=MyBigWebApp
```

Pods example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

Key Pod container attributes

- name - the name of the container
- image - the container image
- ports - array of ports to expose. Can be granted a friendly name and protocol may be specified
- env - array of environment variables
- command - entrypoint array (equal to Docker ENTRYPOINT)
- args - arguments to pass to the command (equal to Docker CMD)

Container

```
name: nginx
image: nginx:stable-alpine
ports:
  - containerPort: 80
    name: http
    protocol: TCP
env:
  - name: MYVAR
    value: isAwesome
command: ["/bin/sh", "-c"]
args: ["echo ${MYVAR}"]
```

Pod template

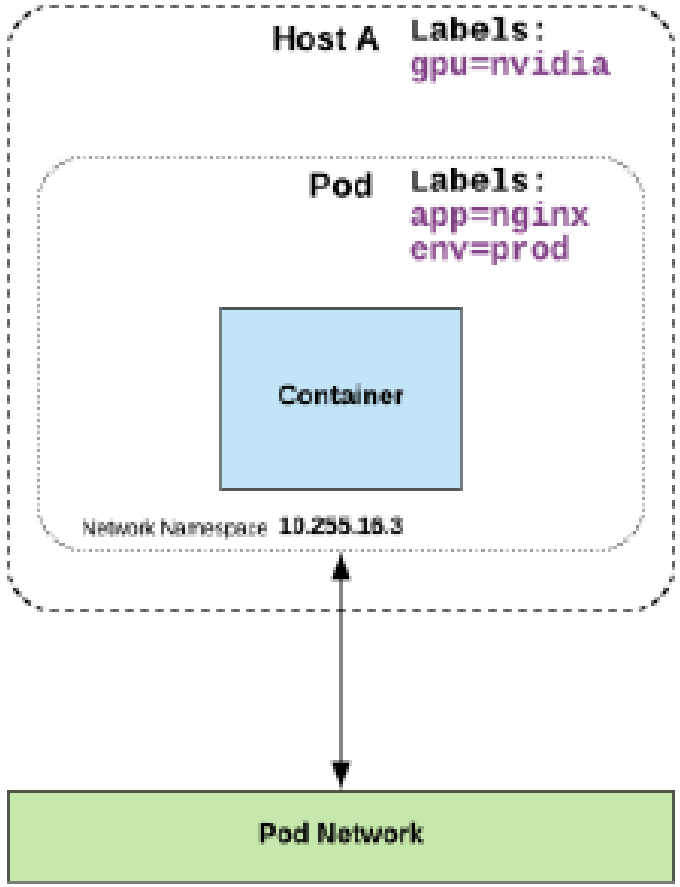
- Workload Controllers manage instances of Pods based on a provided template.
- The Pod Template defines the specification for the Pods that these controllers manage.
- Controllers use Pod Templates to make actual pods.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx
```

Labels

- *Labels* are key/value pairs that are attached to objects such as Pods.
- Labels can be used to organize and to select subsets of objects.
- Labels can be attached to objects at creation time and subsequently added and modified at any time.
- Each object can have a set of key/value labels defined. Each Key must be unique for a given object.



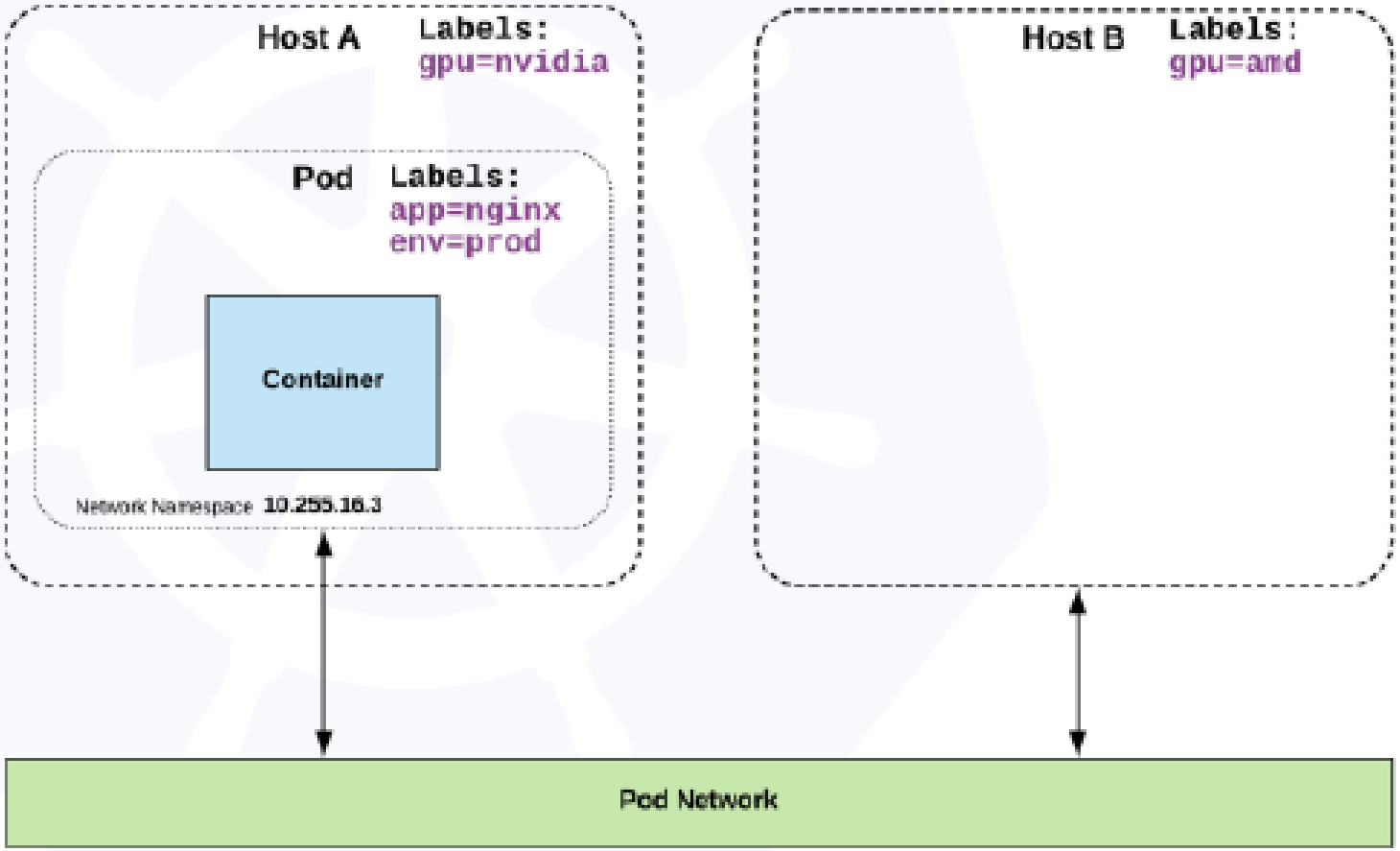
Selectors

Selectors use labels to filter or select objects, and are used throughout Kubernetes.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```

Selector example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```



Selector types

- **Equality based** selectors allow for simple filtering.
- **Set-based** selectors are supported on a limited subset or objects. However, they provide a method of filtering on a set of values, and supports multiple operators including in, notin, and exist.

```
selector:
  matchLabels:
    gpu: nvidia
```

```
selector:
  matchExpressions:
    - key: gpu
      operator: in
      values: ["nvidia"]
```

Set-based selectors are more flexible than equality-based selectors and are ideal when you need to filter resources based on labels that can have multiple values or when you need to check for the existence of a label.

Resource model

- Request: amount of a resource allowed to be used, with a strong guarantee of availability.
  - CPU(second), RAM(bytes)
  - Scheduler will not over-commit requests
- Limit: maximum amount of a resource that can be used, regardless of guarantees

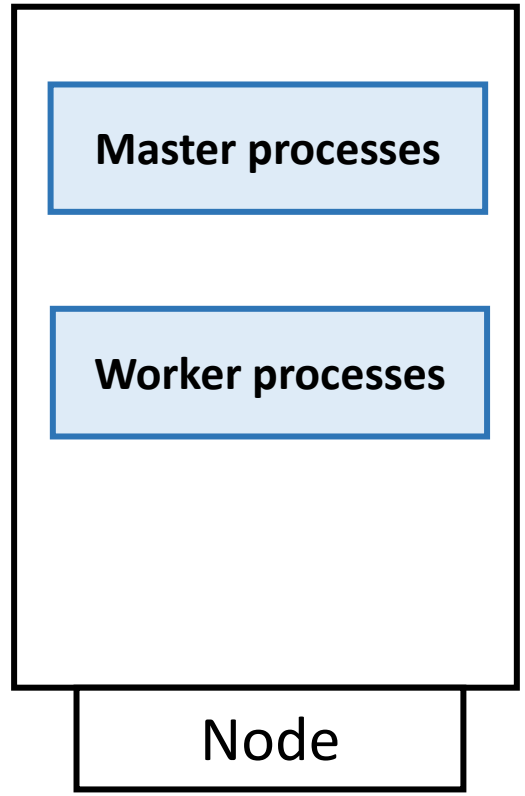
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- Mapping to Docker
  - —cpu-shares=requests.cpu
  - —cpu-quota=limits.cpu
  - —cpu-period=100ms
  - —memory=limits.memory

Minikube

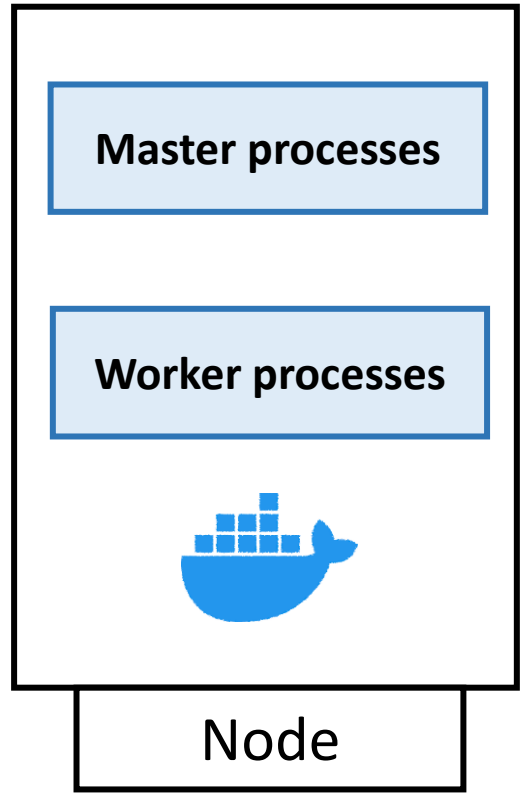
Minikube

- To get started with Kubernetes development, you can use Minikube.
- Minikube is a lightweight Kubernetes implementation
- Minikube creates a VM on your local machine and deploys a simple cluster containing only one node where master and worker processes run on one machine.
- Minikube is available for Linux, MacOS, and Windows systems.
- The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.



Minikube

- The node in the minikube has the Docker container preinstall.
- It needs to install hypervisor on your machine such as: Docker, QEMU, Hyperkit, HyperV, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation
- Node runs in that virtual box (hypervisor)
- Minikube is a 1 node Kubernetes cluster on virtual box for testing Kubernetes on the local machine



Kubectl

- Kubectl is the Kubernetes command-line tool.
- kubectl, allows you to run commands against Kubernetes clusters and create component in the node.
- You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

Install Minikube

- Step 1: Install hypervisor

operating system	Supported hypervisors
macOS	VirtualBox, VMware Fusion, HyperKit
Linux	VirtualBox, KVM
Windows	VirtualBox, Hyper-V

Install Virtualbox on Ubuntu

- Update the repository:

```
sudo apt-get update
```

- Download and install VirtualBox by running:

```
sudo apt-get install virtualbox
```

- Install the VirtualBox Extension Pack:

```
sudo apt-get install virtualbox—ext–pack
```

Install Minikube

- Step 2: Install kubectl on linux

1. Download the latest release with the command:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

2. Validate the binary (optional)

Download the kubectl checksum file:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
```

Install minikube

- Step 2: Install kubectl on linux

4. Install kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

5. Test to ensure the version you installed is up-to-date

```
kubectl version --client
```

Install Minikube

- To install the latest Minikube **stable** release on **x86-64 Linux** using **binary download**:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Minikube

- Check the kubectl commands:

```
kubectl
```

- Check the Minikube commands:

```
minikube
```

```
Basic Commands:
  start          Starts a local kubernetes cluster
  status         Gets the status of a local kubernetes cluster
  stop           Stops a running local kubernetes cluster
  delete         Deletes a local kubernetes cluster
  dashboard      Access the kubernetes dashboard running within the minikube
cluster
```

Minikube

- Create and start a Kubernetes cluster: We should specify that which hypervisor it should use to start the cluster

Minikube start --vm-driver=virtualbox

- Minikube has docker demean pre-installed

Minikube

- Get status of the nodes

kubectl get nodes

```
[~]$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   21h   v1.17.0
```

- Get status of Minikube

minikube status

```
[~]$ minikube status
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Basic kubectl commands

Create and debug pods in a minikube cluster

- CRUD commands:
  - Create deployment
  - Edit deployment
  - Delete deployment

```
kubectl create deployment [name]
kubectl edit deployment [name]
kubectl delete deployment [name]
```

- Get status of different K8s components

```
kubectl get nodes|pod|services|replicaset|deployment
```

- Debugging pods
  - Log to console
  - Get interactive terminal

```
kubectl logs[pod name]
kubectl exec -it [pod name] -- bin/bash
```

Get status of different components and create and edit a pod

Create and edit a pod

- Deployment is used to create a pod, since deployment is abstraction over pods

kubectl create deployment Name --image=image [--dry-run] [option]

- Example: Create a pod from nginx image

```
kubectl create deployment nginx-depl --image=nginx
deployment.apps/nginx-depl created
```

- Get status of the deployment

```
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx-depl1 0/1     1            0           17s
```

- Get status of the pod

```
kubectl get pod
NAME                          READY   STATUS              RESTARTS   AGE
nginx-depl-709447675c-j9j8k 0/1     ContainerCreating 0          31s
kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
nginx-depl-709447675c-j9j8k 1/1     Running   0          54s
```

Create and edit a pod

- Checking the number of replicas in replicaset

```
kubectl get replicaset
NAME                     DESIRED   CURRENT   READY   AGE
nginx-depl-709447675c    1         1         1       98s
```

- Edit the pod in deployment

```
Kubectl edit deployment name
```

- Get the statues of the pod

```
Kubectl logs pod-name
```

- Get the more information about the pod

```
Kubectl describe pod pod-name
```

Create and edit a pod

- Delete the pod (we should delete the deployment)

```
Kubectl delete deployment name
```

- Create configuration file to specify the pod components

```
Touch name.yaml
Vim name.yaml
```

Example:

```
[~]$ touch nginx-deployment.yaml
[~]$ vim nginx-deployment.yaml
```

Create and edit a pod

Kind: specify what we want to create

Name: the name of deployment

Spec: specification for the deployment

Replicas: the number replicas of the pods

Template: the blueprint of the pod

Spec: specification for the pods

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 80
```

create and edit a pod

- Apply configuration file to create and update the component

Kubectl apply –f name.yaml

Example:

```
[~]$ vim nginx-deployment.yaml
[~]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
[~]$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-594cc45b78-pq5dx   1/1     Running   0          7s
[~]$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1/1     1            1           52s
[~]$ vim nginx-deployment.yaml
[~]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment configured
[~]$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           115s
[~]$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-594cc45b78-ncs97   1/1     Running   0          39s
nginx-deployment-594cc45b78-pq5dx   1/1     Running   0          2m5s
```

Configuration file in Kubernetes

Every Kubernetes file has three part:

1. Metadata
2. Spec
3. Status

Metadata

The metadata section contains information about the component that helps identify and manage it. Common fields within the metadata section include:

- **name**: A unique name for the resource within its namespace.
- **namespace**: The Kubernetes namespace in which the component is created (optional, defaults to "default").
- **labels**: Key-value pairs that can be used for organizing and selecting components.
- **annotations**: Additional metadata that may contain useful information

Example Metadata Section:

```yaml
metadata:
  name: my-pod
  namespace: my-namespace
  labels:
    app: my-app
  annotations:
    description: This is my pod.
```

Specification(Spec)

- The specification section defines the desired state of the component, including configuration settings and parameters.
- The exact contents of the spec section depend on the component type.
- Example
  - In a Pod, the spec defines the container(s),
  - In a Service, the spec defines the network rules for load balancing.

Example Spec Section (Pod):

```
spec:
  containers:
  - name: my-container
    image: nginx:latest
    ports:
    - containerPort: 80
```

Status

- The status section is maintained by the Kubernetes control plane and provides information about the current observed state of the resource within the cluster.
- It is read-only and should not be modified directly in the component manifest.
- Kubernetes controllers and the control plane components update this section as the component's actual state changes.

Example Status Section (Pod):

```
status:
  phase: Running
  conditions:
  - type: Ready
    status: True
```

Connecting components

- The components are connected to each other through:
  - Labels
  - Selectors
  - Ports
- The metadata contains labels and specifications contains selectors
- In metadata we give component like deployment and key-value pair for component

Labels and selectors

- The metadata contains labels

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:…
```

Labels and selectors

- The metadata contains labels
- Specifications contains selectors

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:…
```

Labels and selectors

- The metadata contains labels
- Specifications contains selectors
- In metadata we give component like deployment and key-value pair for component

```
labels:
  app: nginx
```

Deployment

```
1   apiVersion: apps/v1
2   kind: Deployment
3   metadata:
4     name: nginx-deployment
5     labels:
6       app: nginx
7   spec:
8     replicas: 2
9     selector:
10      matchLabels:
11        app: nginx
12    template:
13      metadata:
14        labels:
15          app: nginx
```

Service

```
1   apiVersion: v1
2   kind: Service
3   metadata:
4     name: nginx-service
5   spec:
6     selector:
7       app: nginx
8   >   ports: …
12
```

Connecting Services to Deployment

- Selector in the service make the connection between deployment or its pods
- This way specify which pod belong to that service

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:…
```

Ports

Deployment

```
8   replicas: 2
9   selector:
10    matchLabels:
11      app: nginx
12  template:
13    metadata:
14      labels:
15        app: nginx
16    spec:
17      containers:
18      - name: nginx
19        image: nginx:1.16
20        ports:
21        - containerPort: 8080
```

Service

```
1   apiVersion: v1
2   kind: Service
3   metadata:
4     name: nginx-service
5   spec:
6     selector:
7       app: nginx
8     ports:
9       - protocol: TCP
10        port: 80
11        targetPort: 8080
12
```

Pod and service

```
kubectl get pod
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-7d644fb574-fklxj 1/1     Running   0          10s
nginx-deployment-7d644fb574-v7mwj 1/1     Running   0          10s

kubectl get service
NAME            TYPE          CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes      ClusterIP     10.96.0.1      <none>        443/TCP   2d
nginx-service   ClusterIP     10.96.25.229   <none>        80/TCP    19s

kubectl describe service nginx-service
Name:              nginx-service
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"
Selector:          app=nginx
Type:              ClusterIP
IP:                10.96.25.229
Port:              <unset> 80/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.6:8080,172.17.0.7:8080
Session Affinity:  None
```

Pod and service

- Get more information about the pod

Kubectl get pod -o wide

```
kubectl get pod -o wide
NAME                                READY STATUS   RESTARTS AGE   IP          NODE
nginx-deployment-7d644fb574-fklxj   1/1   Running  0        2m17s 172.17.0.7 minikube
nginx-deployment-7d644fb574-v7mwj   1/1   Running  0        2m17s 172.17.0.6 minikube
```

Status

- Get the status which automatically generated by Kubernetes

Kubectl get deployment nginx-deployment –o yaml

```
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2020-01-24T10:54:59Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: "2020-01-24T10:54:56Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: ReplicaSet "nginx-deployment-7d64f4b574" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
```

Status

- Get the status which automatically generated by Kubernetes

Kubectl get deployment nginx-deployment –o yaml

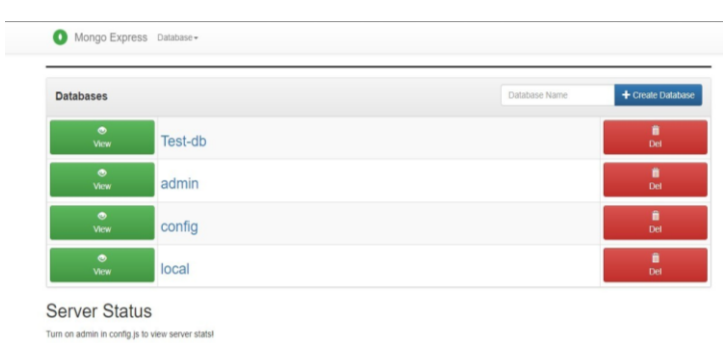
- Get the status in file

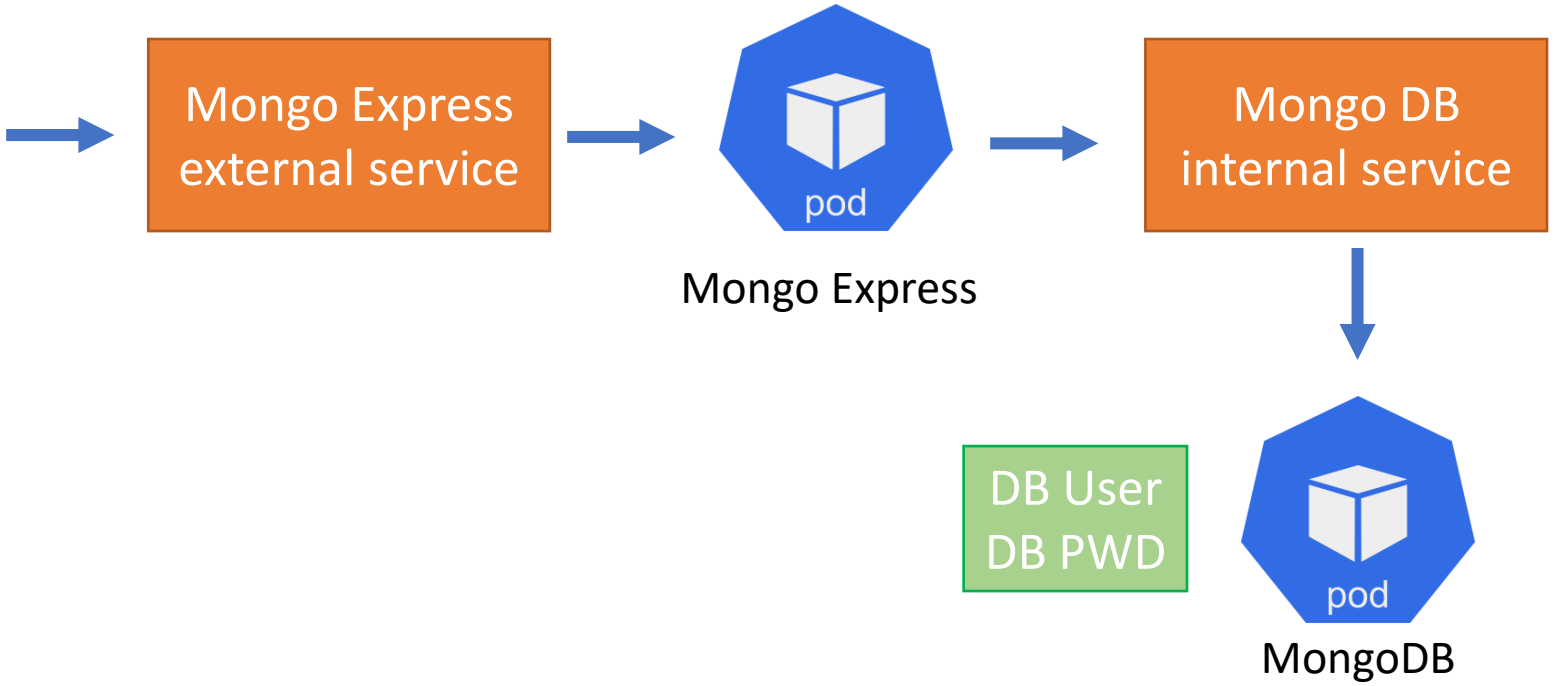
Kubectl get deployment nginx-deployment –o yaml > nginx-deployment-result.yaml

```
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2020-01-24T10:54:59Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: "2020-01-24T10:54:56Z"
    lastUpdateTime: "2020-01-24T10:54:59Z"
    message: ReplicaSet "nginx-deployment-7d64f4b574" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
```

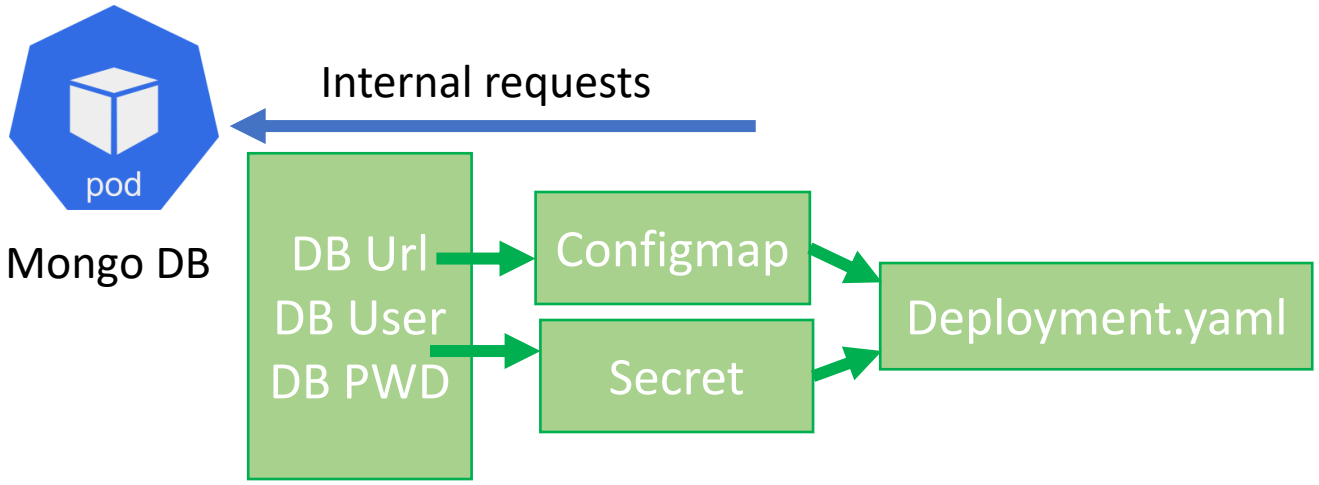
Application setup in Kubernetes

- This shows a simple setup of a web application and its database

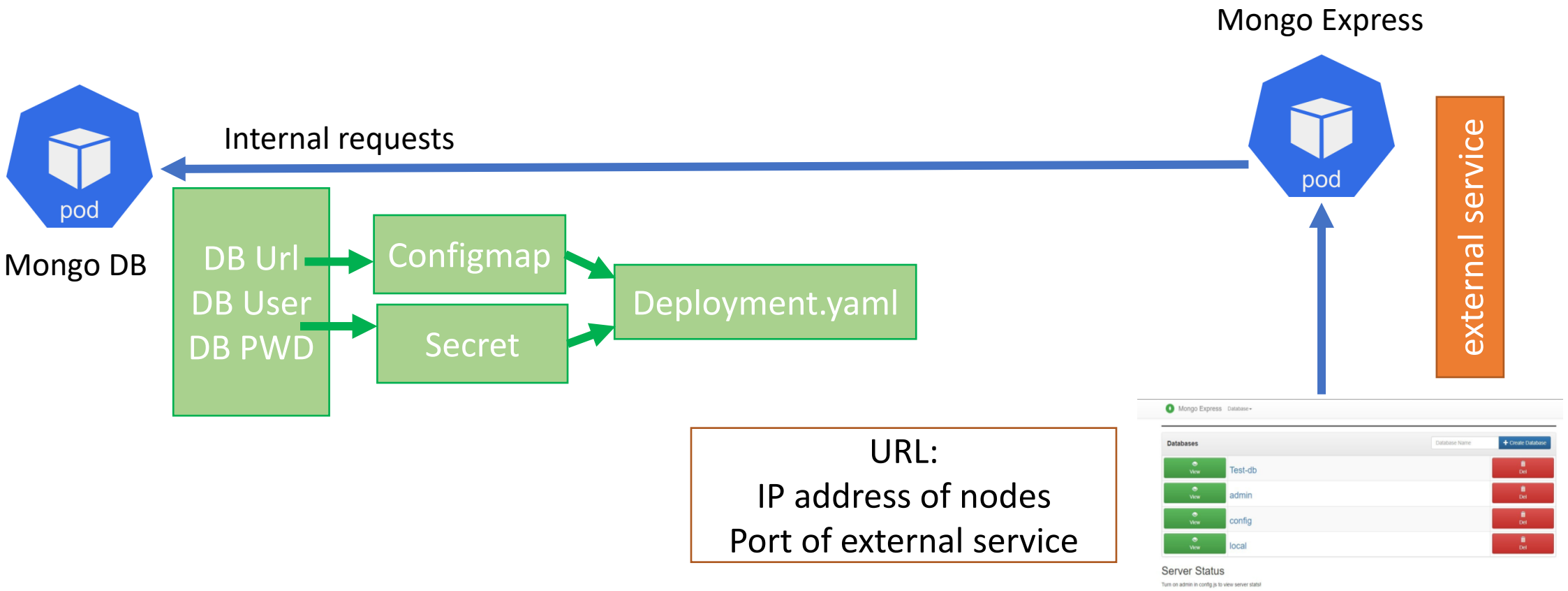




Application setup in Kubernetes



Application setup in Kubernetes



Application setup in Kubernetes

- Create a MongoDB secret file (yaml configuration file)
- Create a MongoDB deployment file (yaml configuration file)
- Create internal service
- Create a configmap file
- Create a Mongo express deployment file
- Create external service to access the Mongo express from the browser

Secret configuration-MongoDB

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username:
  mongo-root-password:
```

"Opaque"-default for arbitrary key-value pairs

The values for username and password are not plain text, they are base 64 encode

```
Echo –n 'username' | base64
```

Deployment configuration- MongoDB

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb-container
        image: mongo:latest
        ports:
        - containerPort: 27017
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-username
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-password
```

Create Secret and deployment configuration

```
cd k8s-configuration/
ls
mongo-secret.yaml    mongo.yaml
kubectl apply -f mongo-secret.yaml
secret/mongodb-secret created
```

```
[k8s-configuration]$ kubectl apply -f mongo.yaml
deployment.apps/mongodb-deployment created
[k8s-configuration]$ kubectl get all
NAME                                      READY   STATUS              RESTARTS
  AGE
pod/mongodb-deployment-78444d94d6-zsrcl   0/1     ContainerCreating   0
  7s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   25m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb-deployment   0/1     1            0           7s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-deployment-78444d94d6   1         1         0       7s
```

Internal service configuration-MongoDB

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb  # Replace 'app' label with your MongoDB Pod's label
  ports:
    - protocol: TCP
      port: 27017  # The port your MongoDB instance is listening on
      targetPort: 27017  # The port your MongoDB Pod is listening on
```

Internal service configuration-MongoDB

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb  # Replace 'app' label with your MongoDB Pod's label
  ports:
    - protocol: TCP
      port: 27017  # The port your MongoDB instance is listening on
      targetPort: 27017  # The port your MongoDB Pod is listening on
```

To see all the component of the application

Kubectl get all | grep mongodb

Internal service configuration

```
[k8s-configuration]$ kubectl get service
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP     36m
mongodb-service   ClusterIP   10.96.86.105    <none>        27017/TCP   51s
[k8s-configuration]$ kubectl describe service mongodb-service
Name:              mongodb-service
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotatio
ns":{},"name":"mongodb-service","namespace":"default"},"spec":{"ports":[{"port"
:...
Selector:          app=mongodb
Type:              ClusterIP
IP:                10.96.86.105
Port:              <unset>  27017/TCP
TargetPort:        27017/TCP
Endpoints:         172.17.0.6:27017
Session Affinity:  None
```

Deployment configuration- Mongo express

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express-deployment
spec:
  replicas: 1  # You can adjust the number of replicas as needed
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
        - name: mongo-express
          image: mongo-express
          ports:
            - containerPort: 8081  # Port for the mongo-express application
          env:
          - name: ME_CONFIG_MONGODB_ADMINUSERNAME
            valueFrom:
              secretKeyRef:
                name: mongodb-secret
                key: mongo-root-username
          - name: ME_CONFIG_MONGODB_ADMINPASSWORD
            valueFrom:
              secretKeyRef:
                name: mongodb-secret
                key: mongo-root-password
          - name: ME_CONFIG_MONGODB_SERVER
            valueFrom:
              configMapKeyRef:
                name: mongodb-configmap
                key: database_url
```

Deployment configuration- Mongo express

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express
  labels:
    app: mongo-express
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
   spec:
     containers:
     - name: mongo-express
     image: mongo-express
     ports:
       - containerPort: 8081
     env:
     - name: ME_CONFIG_MONGODB_ADMINUSERNAME
       valueFrom:
         secretkeyRef:
           name: mongodb-secret
           key: mongo-root-username
     - name: ME_CONFIG_MONGODB_ADMIN
       valueFrom:
         secretkeyRef:
           name: mongodb-secret
           key: mongo-root-password
     - name: ME_CONFIG_MONGODB_SERVER
       valueFrom:
         configmapkeyRef:
           name: mongodb-configmap
           key: database_url
```

The connection requirements:

- The database it should connect to:
  - MongoDB address/ internal service

Deployment configuration- Mongo express

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express-deployment
spec:
  replicas: 1  # You can adjust the number of replicas as needed
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
        - name: mongo-express
          image: mongo-express
          ports:
            - containerPort: 8081  # Port for the mongo-express application
          env:
          - name: ME_CONFIG_MONGODB_ADMINUSERNAME
            valueFrom:
              secretKeyRef:
                name: mongodb-secret
                key: mongo-root-username
          - name: ME_CONFIG_MONGODB_ADMINPASSWORD
            valueFrom:
              secretKeyRef:
                name: mongodb-secret
                key: mongo-root-password
          - name: ME_CONFIG_MONGODB_SERVER
            valueFrom:
              configMapKeyRef:
                name: mongodb-configmap
                key: database_url
```

Same username and password in the mongodb secret

The connection requirements:

- The database it should connect to:
  - MongoDB address/ internal service
- The credentials to authenticate
  - Adminusername
  - Adminpassword

Configmap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  database_url: mongodb-service
```

Create configmap and deployment- Mongo express

```
[k8s-configuration]$ kubectl apply -f mongo-configmap.yaml
configmap/mongodb-configmap created
[k8s-configuration]$ kubectl apply -f mongo-express.yaml
deployment.apps/mongo-express created
[k8s-configuration]$ kubectl get pod
NAME                                 READY   STATUS              RESTARTS   AG
E
mongo-express-797845bd97-p9grr       0/1     ContainerCreating   0          5s
mongodb-deployment-78444d94d6-zsrcl  1/1     Running             0          23
```

```
[k8s-configuration]$ kubectl logs mongo-express-797845bd97-p9grr
Waiting for mongodb-service:27017...
Welcome to mongo-express
------------------------


Mongo Express server listening at http://0.0.0.0:8081
Server is open to allow connections from anyone (0.0.0.0)
basicAuth credentials are "admin:pass", it is recommended you change this in yo
ur config.js!
Database connected
Admin Database connected
```

external service configuration

```
apiVersion: v1
kind: Service
metadata:
   name: mongo-experss-service
spec:
    selector:
      app: mongo-express
    type: LoadBalancer
    ports:
      - protocol: TCP
        port: 8081
        targetPort: 8081
        nodePort: 30000
```

Assigns services an external IP address to accept external requests

The port for external IP address, the port into the browser
It must be between: 30000-32767

Create external service

```
[k8s-configuration]$ kubectl apply -f mongo-express.yaml
deployment.apps/mongo-express unchanged
service/mongo-express-service created
[k8s-configuration]$ kubectl get service
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
kubernetes              ClusterIP      10.96.0.1       <none>        443/TCP
      62m
mongo-express-service   LoadBalancer   10.96.178.16    <pending>     8081:30000/
TCP   6s
mongodb-service         ClusterIP      10.96.86.105    <none>        27017/TCP
      26m
```

```
[k8s-configuration]$ minikube service mongo-express-service
|-----------|----------------------|-------------|---------------------------|
| NAMESPACE |         NAME         | TARGET PORT |            URL            |
|-----------|----------------------|-------------|---------------------------|
| default   | mongo-express-service |             | http://192.168.64.5:30000 |
|-----------|----------------------|-------------|---------------------------|
  Opening service default/mongo-express-service in default browser...
```

References

- https://kubernetes.io/docs/

universität innsbruck

Thank you for your attention☺

Dr. Zahra Najafabadi Samani

Email: Zahra.Najafabadi-Samani@uibk.ac.at