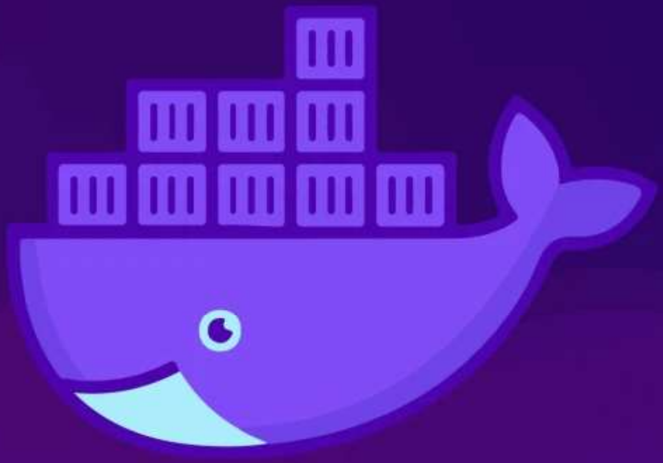


docker

Day 4 and Day 5

Docker Swarm

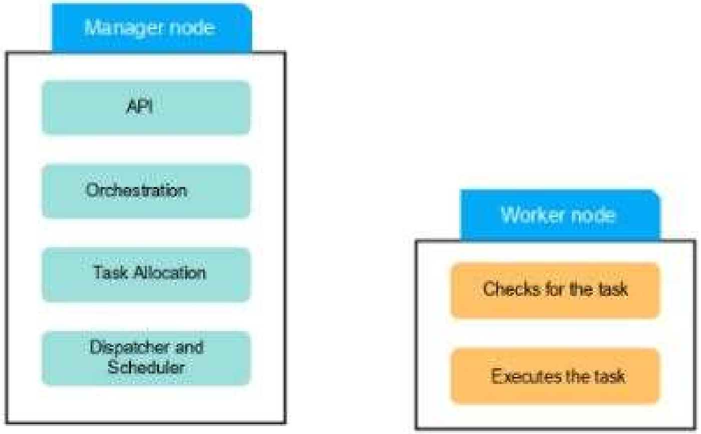
- Docker Swarm is an orchestration management tool that runs on Docker applications. It helps end-users in creating and deploying a cluster of Docker nodes.
- Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API. Each container within the Swarm can be deployed and accessed by nodes of the same cluster.

Key Concepts

- Service and Tasks
  - Docker containers are launched using services.
  - Services can be deployed in two different ways - global and replicated.
  - Global services are responsible for monitoring containers that want to run on a Swarm node. In contrast, replicated services specify the number of identical tasks that a developer requires on the host machine.
  - Services enable developers to scale their applications.
- Node
  - A Swarm node is an instance of the Docker engine.
  - It is possible to run multiple nodes on a single server. But in production deployments, nodes are distributed across various devices.

Key Concepts

- There are three types of nodes in Docker Swarm:
  - Manager node: Maintains cluster management tasks
  - Worker node: Receives and executes tasks from the manager node
  - Leader node: makes all of the swarm management and task orchestration decisions for the swarm

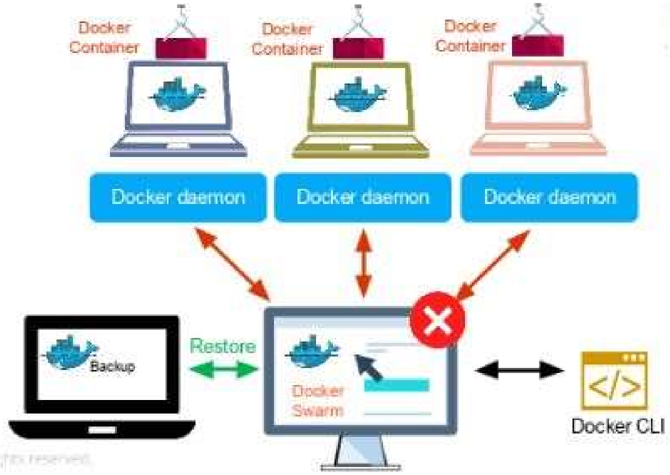


Working with Swarm

- A service is created based on the command-line interface.
- The API that we connect in our Swarm environment allows us to do orchestration by creating tasks for each service.
- The task allocation will enable us to allocate work to tasks via their IP address.
- The dispatcher and scheduler assign and instruct worker nodes to run a task.
- The Worker node connects to the manager node and checks for new tasks.
- The final stage is to execute the tasks that have been assigned from the manager node to the worker node.

Failure Correction Mechanism

- Consider an environment with docker containers
- If one of the containers fails, we can use the Swarm to correct that failure. Docker Swarm can reschedule containers on node failures. Swarm node has a backup folder which we can use to restore the data onto a new Swarm.

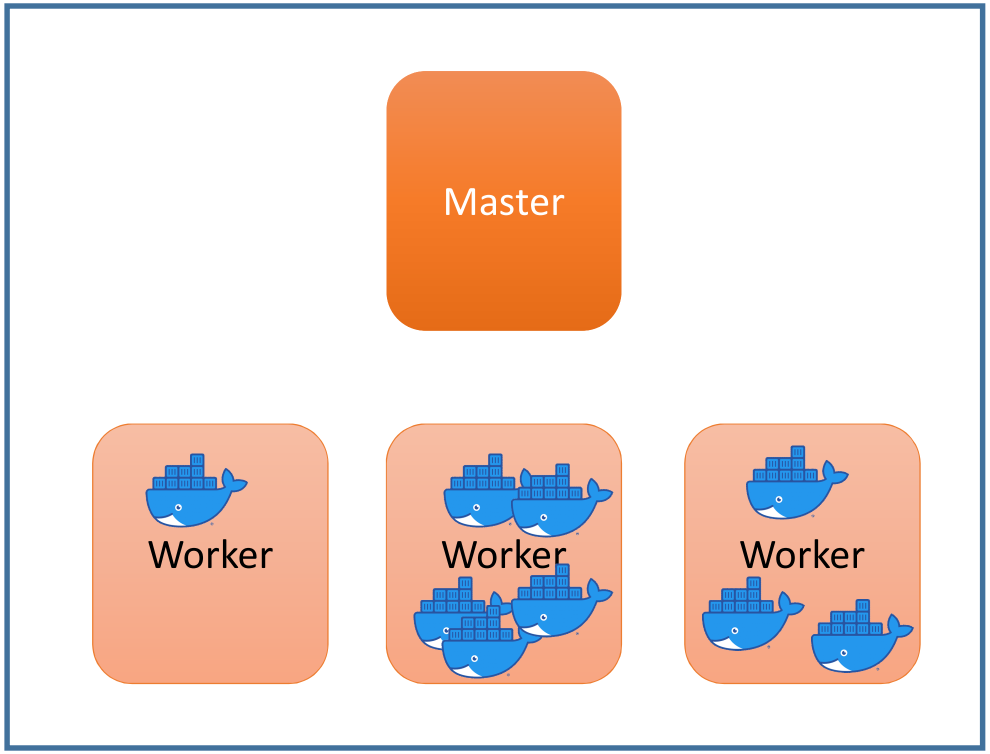


Kubernetes

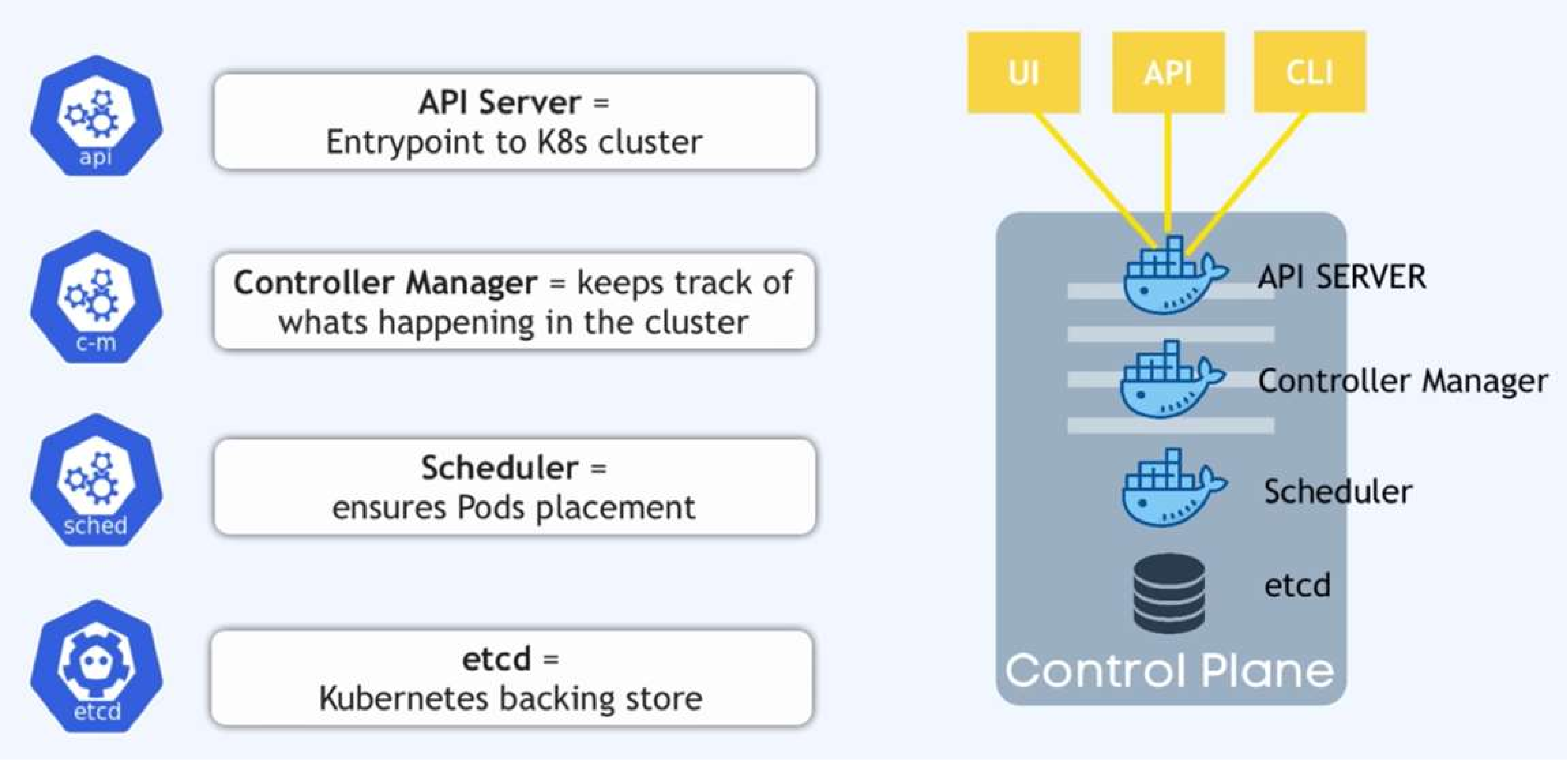
- Kubernetes is a container orchestration tool
- Helps manage containerized applications in different deployment environments such a physical machines, virtual machines, cloud and hybrid environments
- Need for orchestration tool:
  - Raise from monolith to microservices
  - Increases use of containers and need for a way to run them
  - Need for the management of containers

Kubernetes

- Features of orchestration tools:
    - High availability and zero down time
    - Scalability or high performance
    - Disaster recovery – backup and restore
- Kubernetes Architecture:
    - Master Node
    - Worker Nodes



Master Node



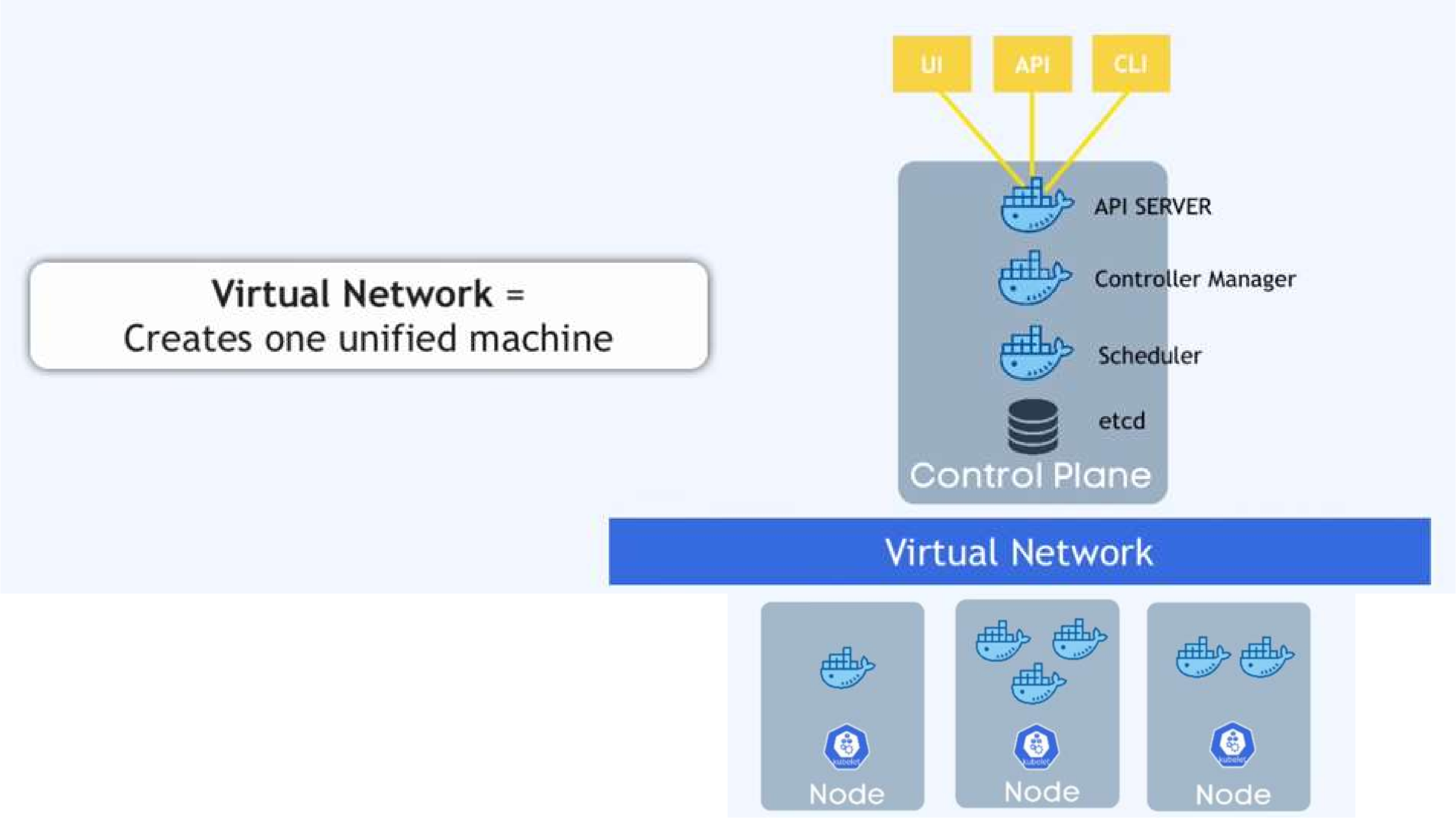
API Server =
Entrypoint to K8s cluster

Controller Manager = keeps track of whats happening in the cluster

Scheduler =
ensures Pods placement

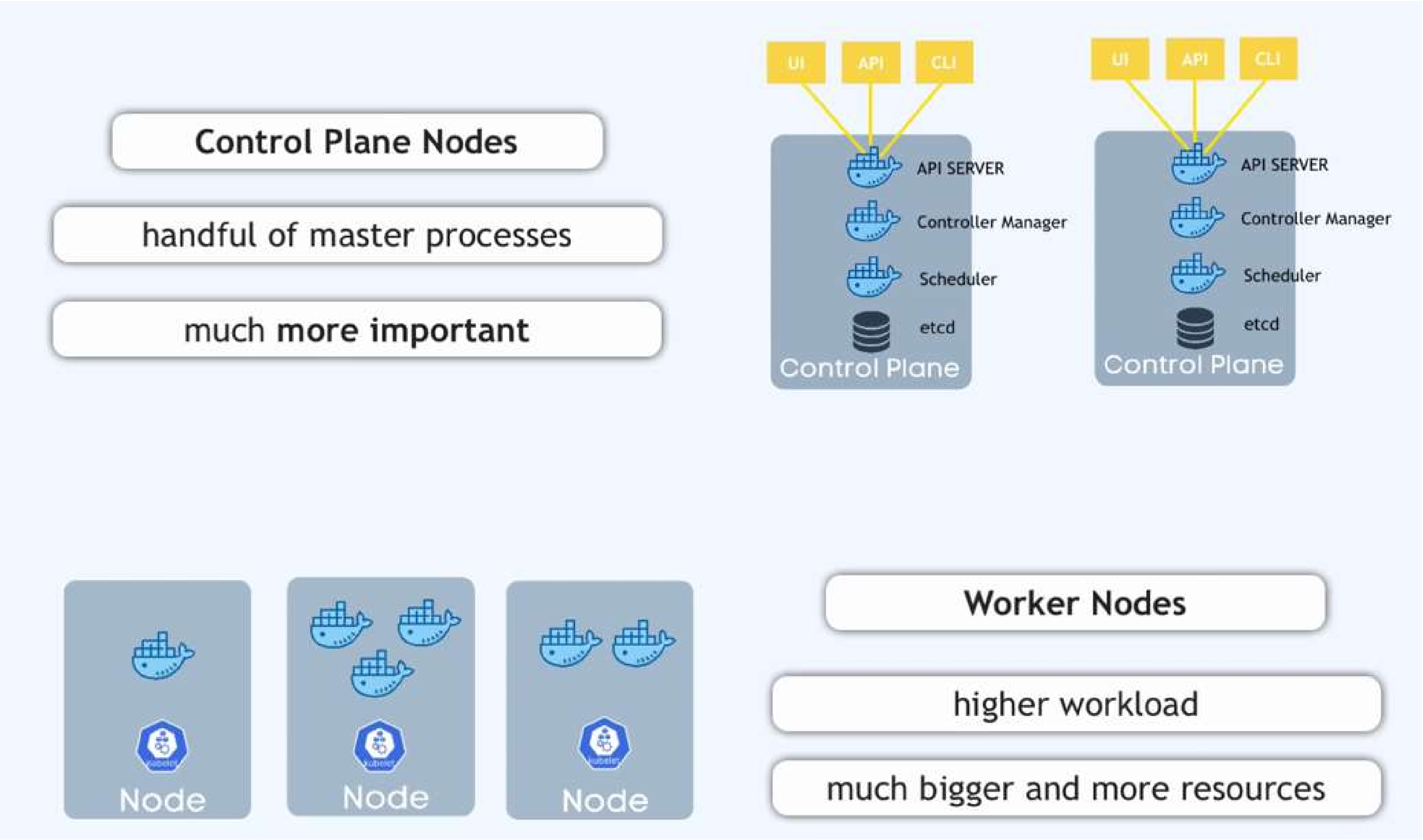
etcd =
Kubernetes backing store

Virtual Network



Virtual Network =
Creates one unified machine

Control Plane Backups



Control Plane Nodes

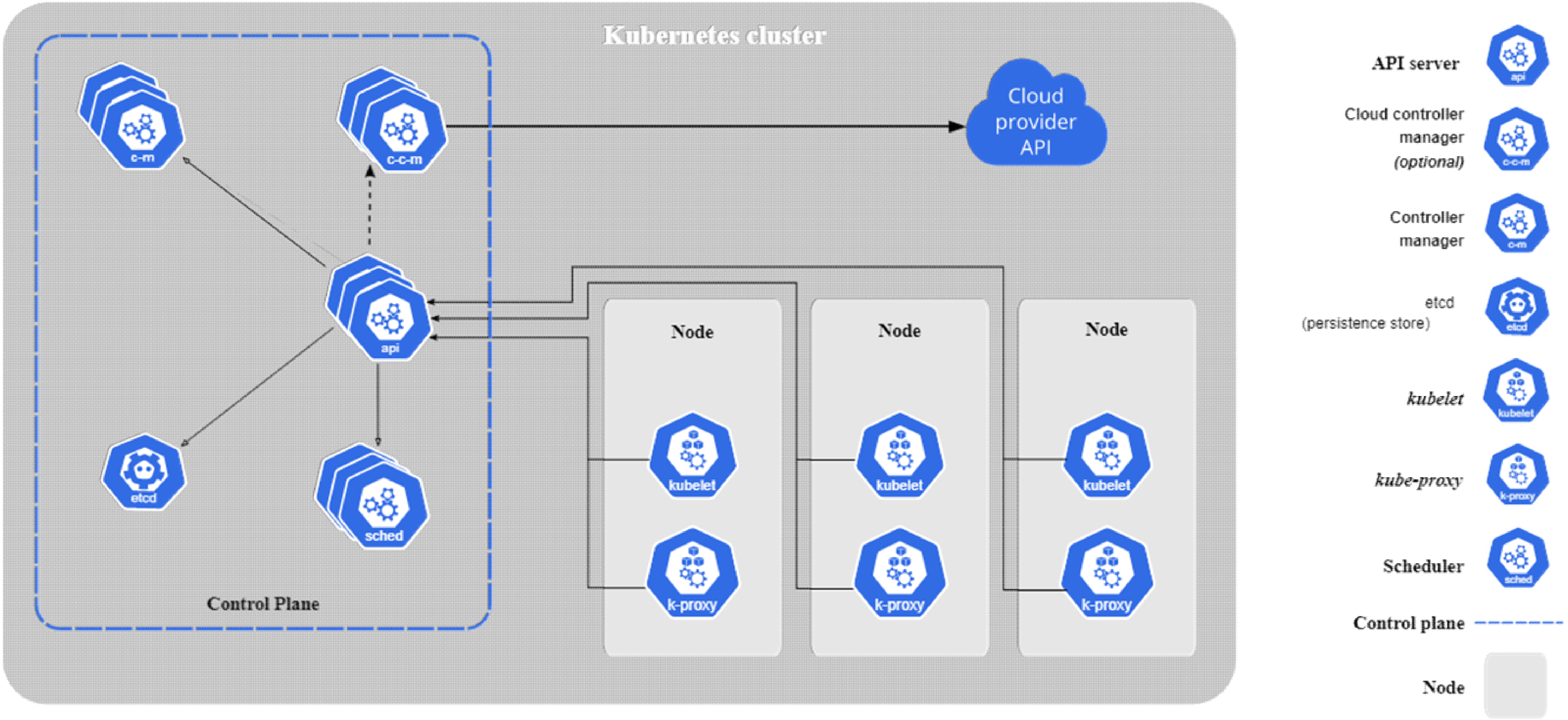
handful of master processes

much **more important**

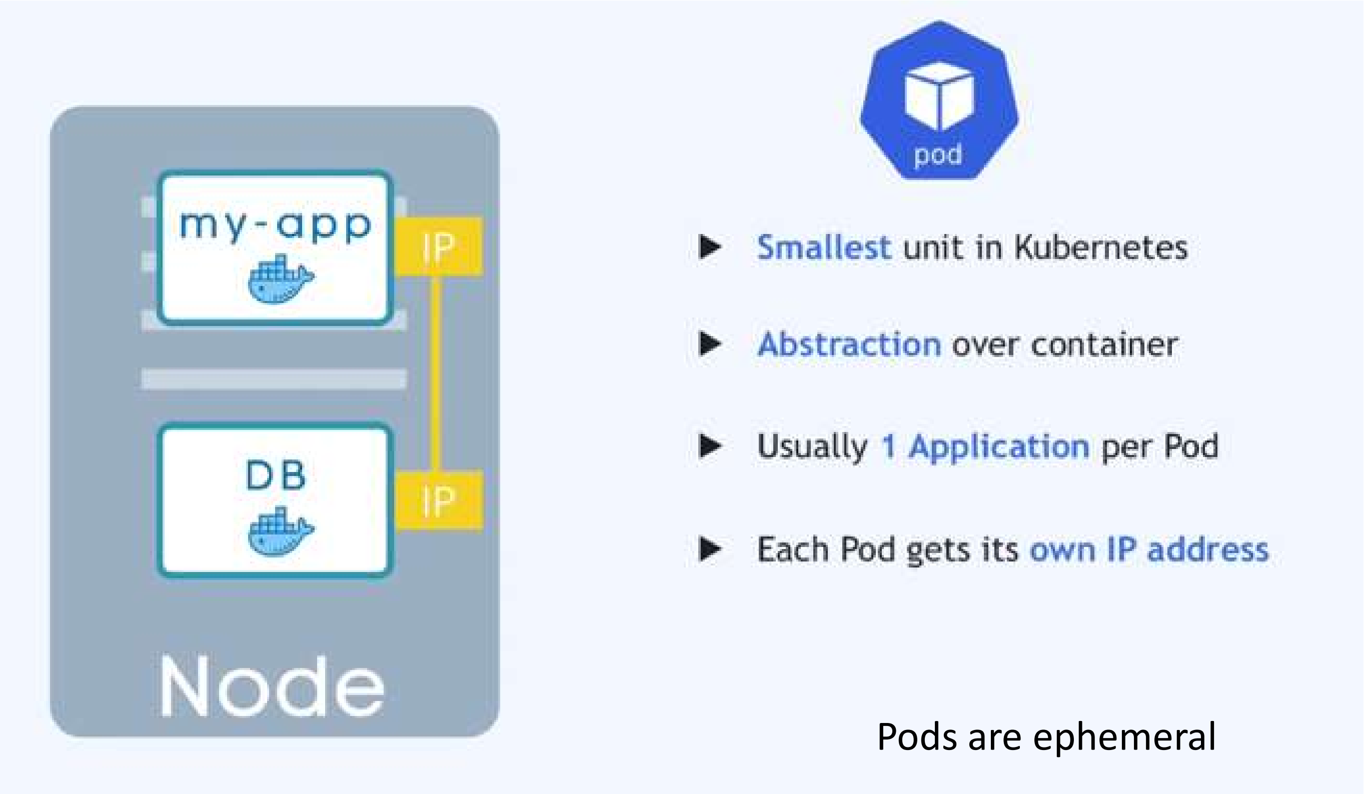
Worker Nodes

higher workload

much bigger and more resources


Kubernetes cluster
c-m
c-c-m
Cloud provider API
api
etcd
sched
Control Plane
Node
kubelet
k-proxy
Node
kubelet
k-proxy
Node
kubelet
k-proxy
API server
api
Cloud controller manager (optional)
c-c-m
Controller manager
c-m
etcd (persistence store)
etcd
kubelet
kubelet
kube-proxy
k-proxy
Scheduler
sched
Control plane
Node

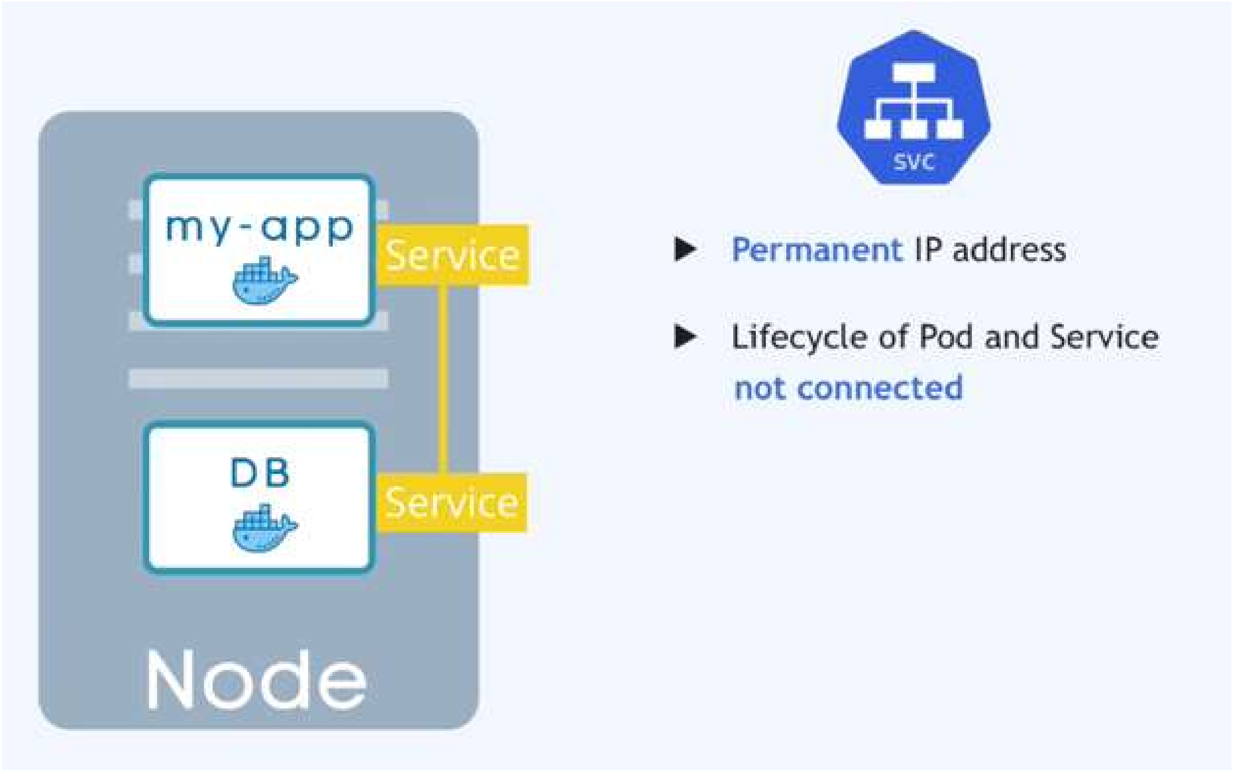
Nodes and Pods



- Smallest unit in Kubernetes
- Abstraction over container
- Usually 1 Application per Pod
- Each Pod gets its own IP address

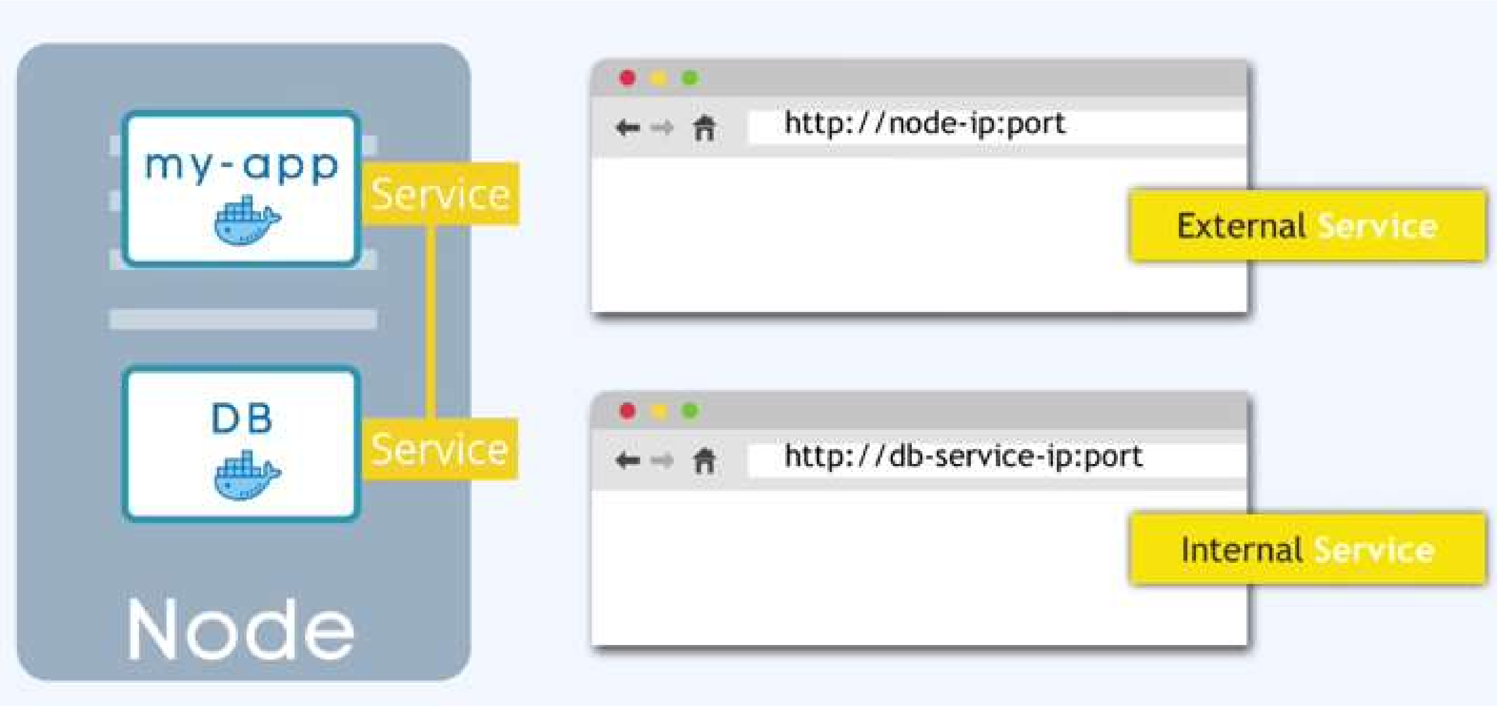
Pods are ephemeral

Service

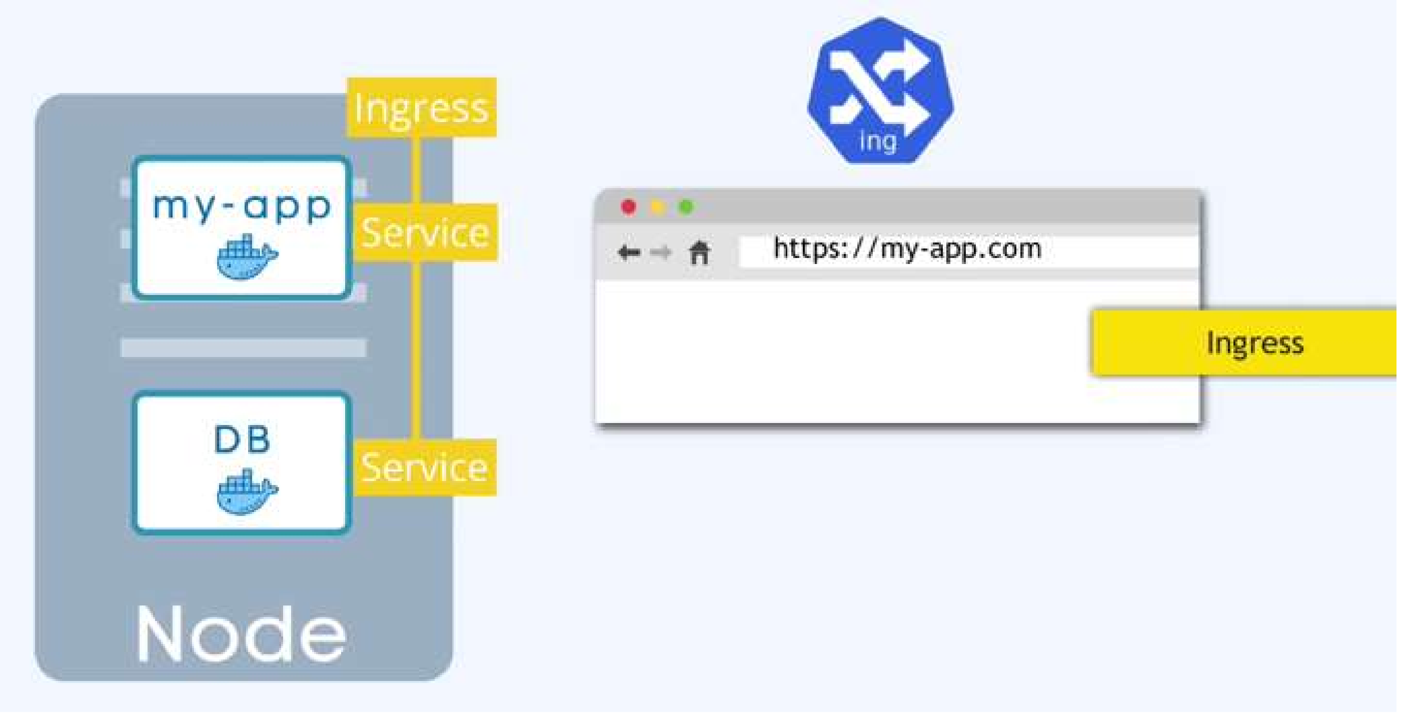


- Permanent IP address
- Lifecycle of Pod and Service not connected

Service

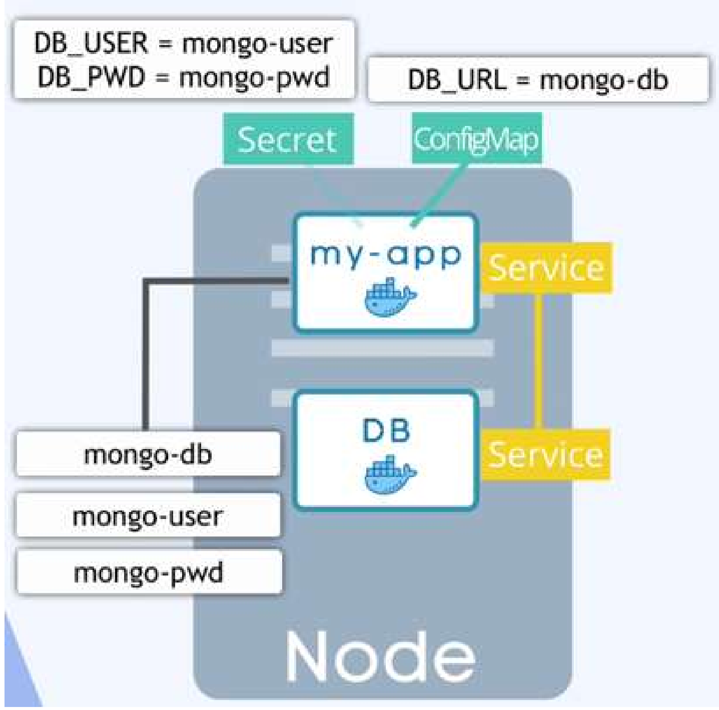

my-app
Service
DB
Service
Node
http://node-ip:port
External Service
http://db-service-ip:port
Internal Service

Ingress


Ingress
my-app
Service
DB
Service
Node
ing
https://my-app.com
Ingress

Configmap and Secret

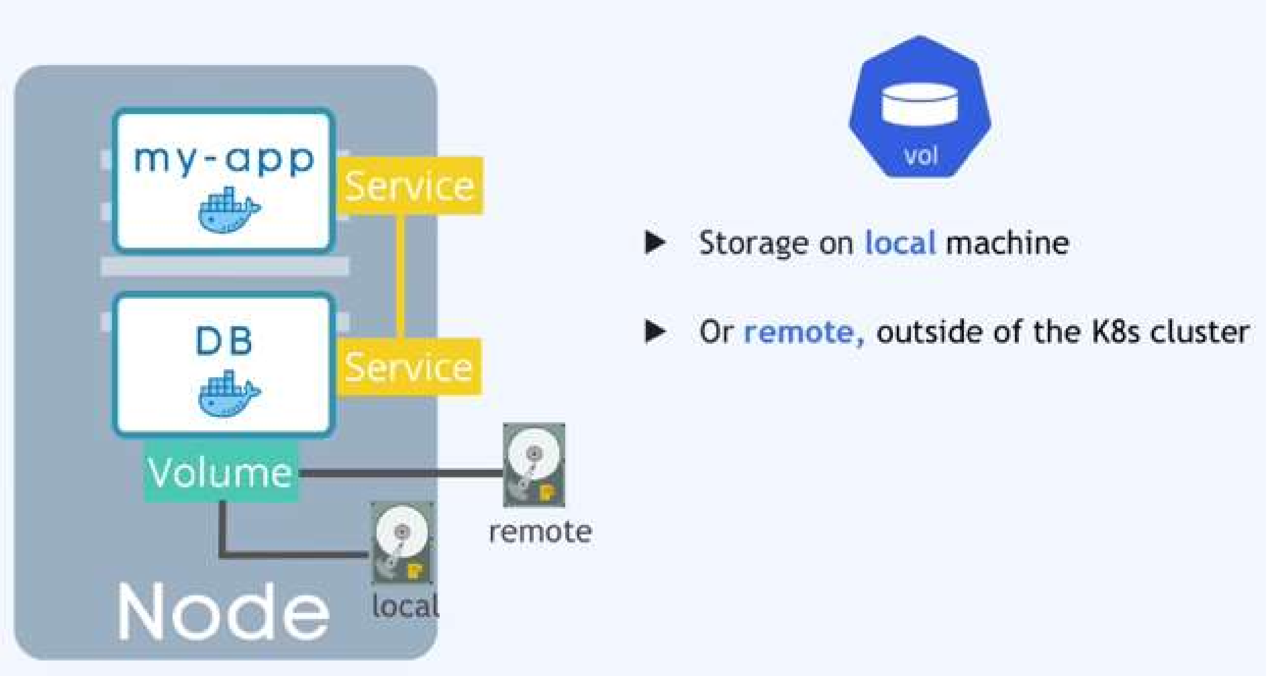
- External Configuration of your application



- Used to store secret data

Reference them in the pod

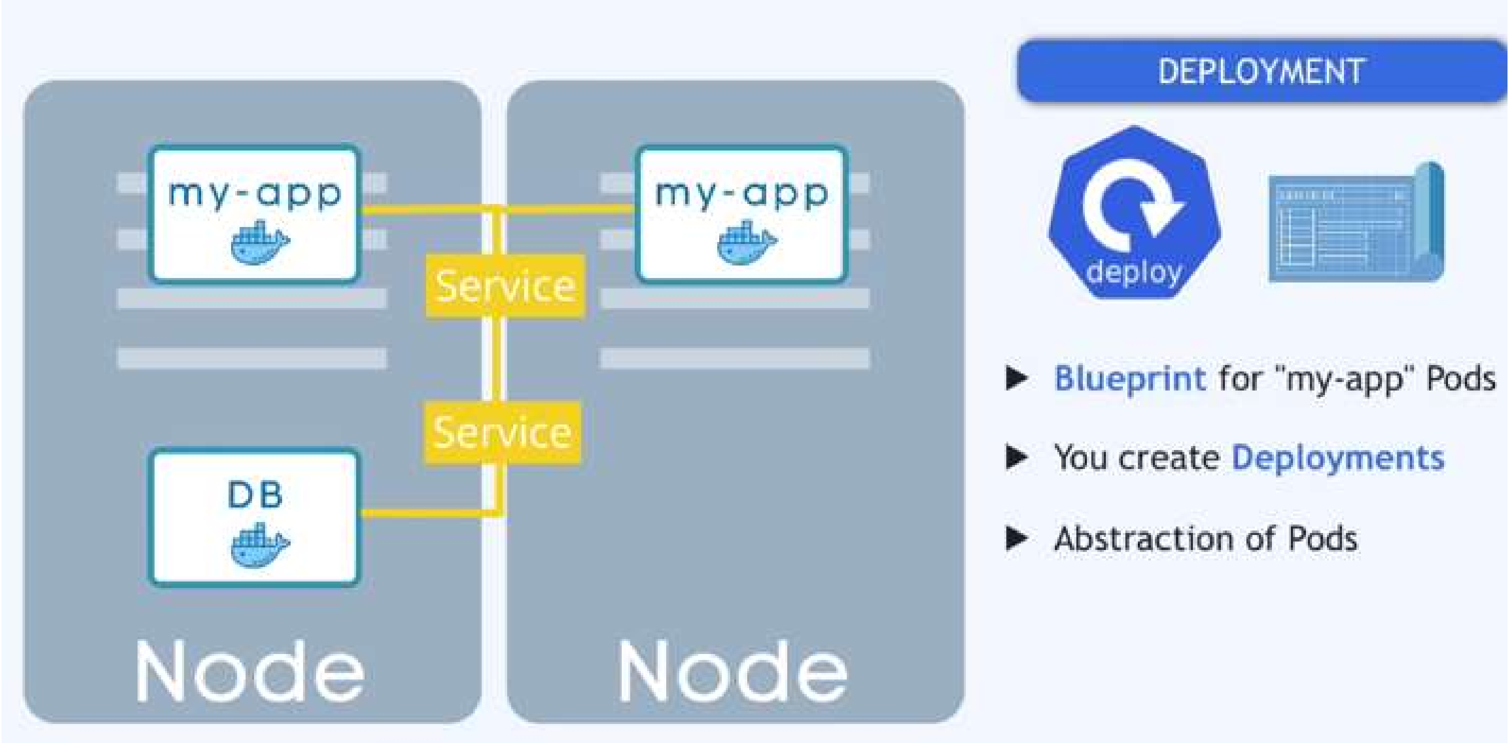
Volumes



- Storage on local machine
- Or remote, outside of the K8s cluster

Kubernetes does not manage any persistence. You as Kubernetes user are responsible for managing the data.

Deployment

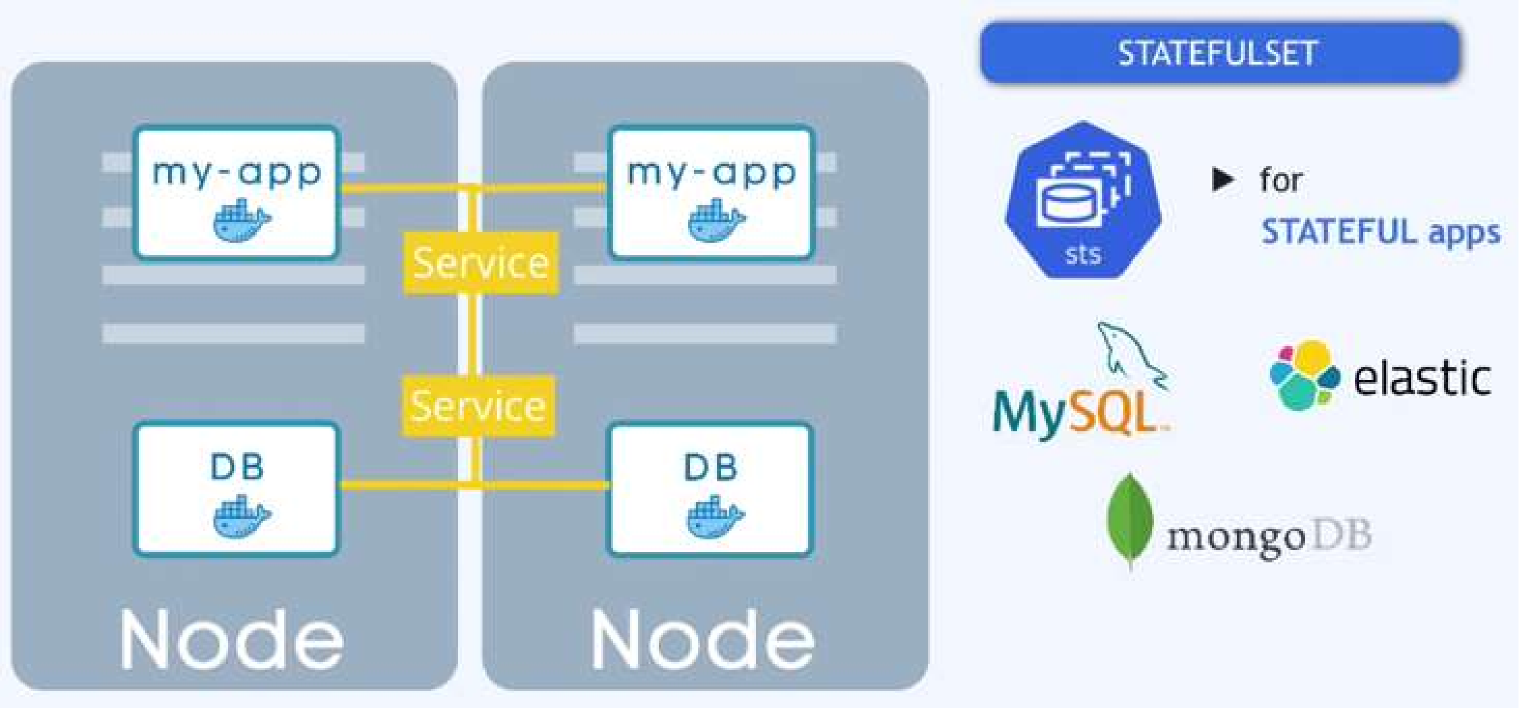


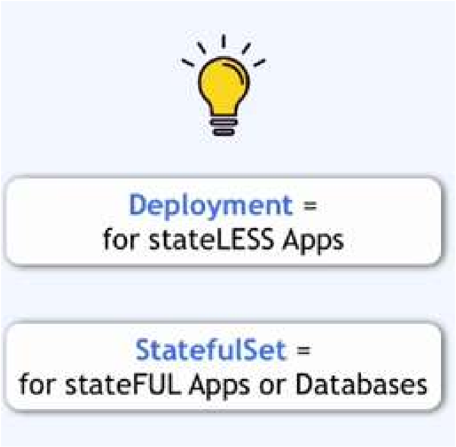
- Blueprint for "my-app" Pods
- You create Deployments
- Abstraction of Pods

It’s a common practice to keep replicas in separate nodes

Databases cannot be replicated because it has state

Stateful Set

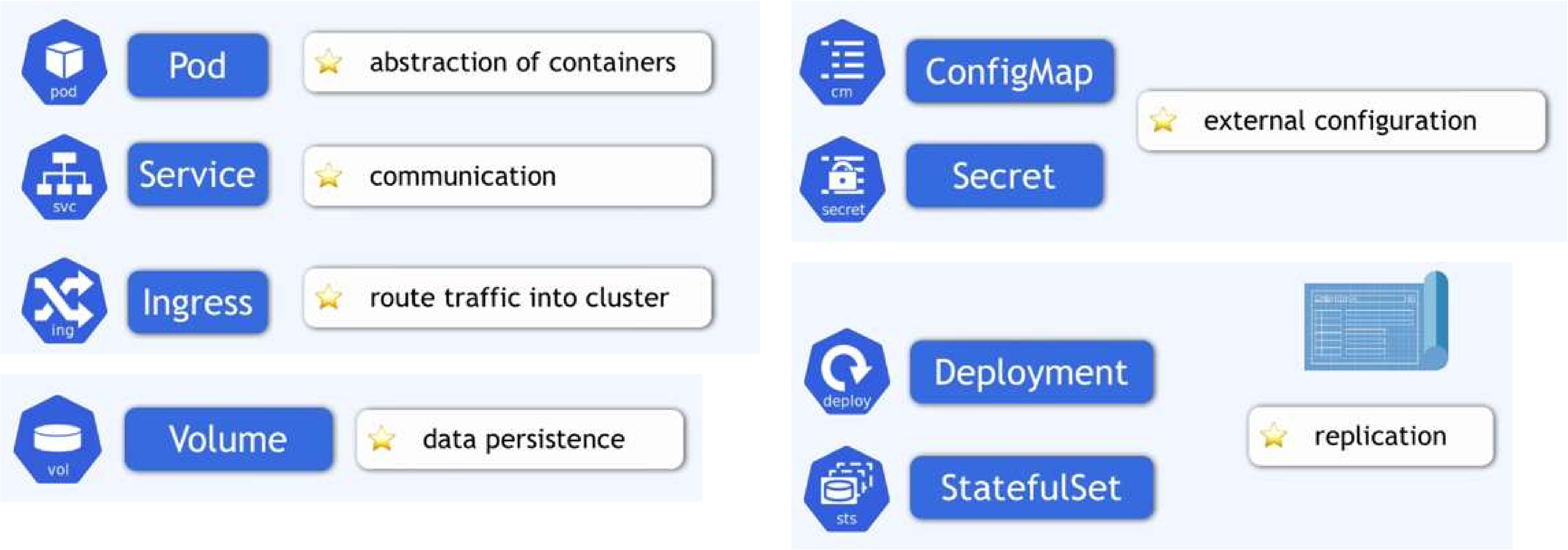



Deployment =
for stateLESS Apps

StatefulSet =
for stateFUL Apps or Databases

Databases are generally hosted outside the Kubernetes cluster as creating stateful set is not easy

Summary



Pod – ⭐ abstraction of containers

Service – ⭐ communication

Ingress – ⭐ route traffic into cluster

Volume – ⭐ data persistence

ConfigMap, Secret – ⭐ external configuration

Deployment, StatefulSet – ⭐ replication

Kubernetes Configuration

Deployment = template for creating pods

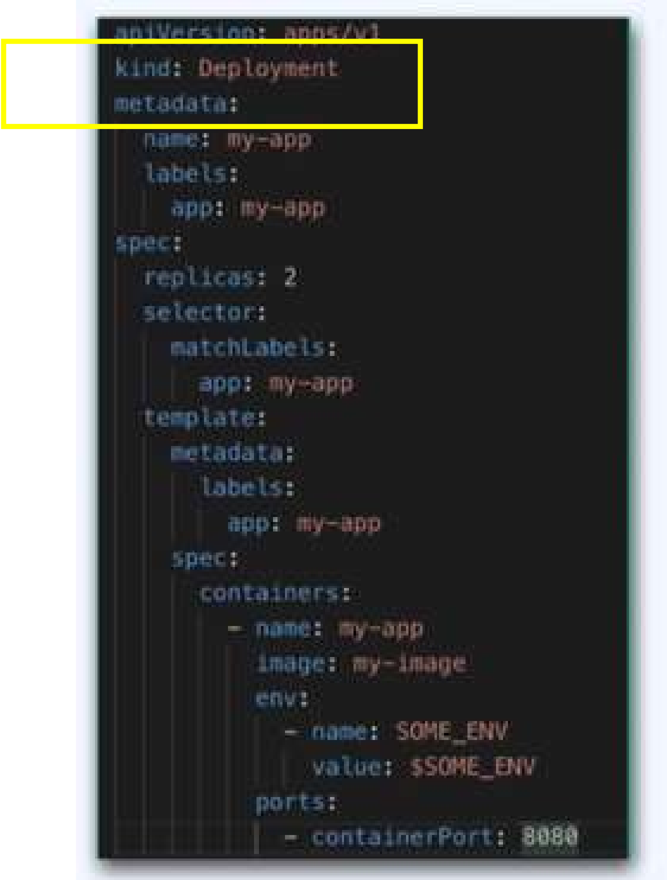
Declarative in nature

Metadata

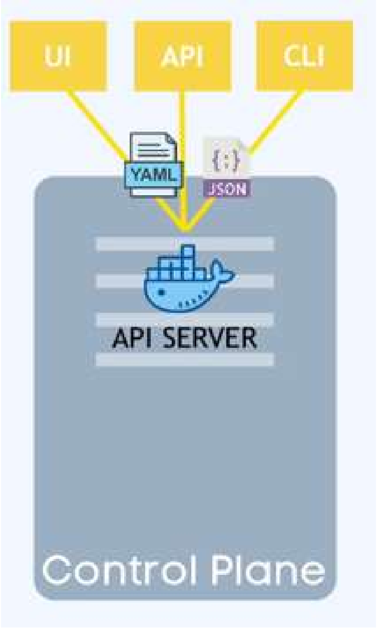
Specification

Stutus -> Automatic

Desired state and actual state, Kubernetes checks and if there is a mis-match it self-heals

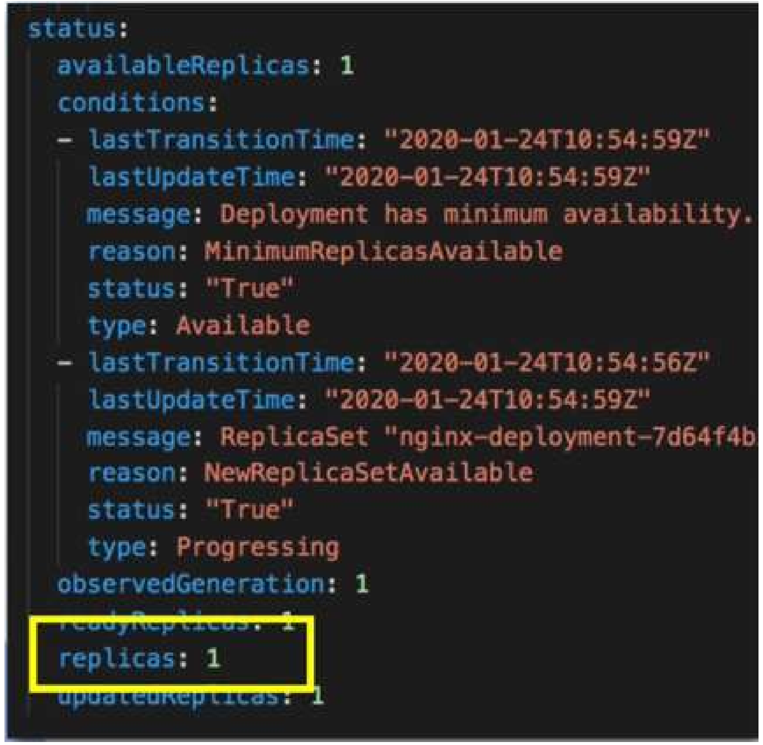


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: my-image
          env:
            - name: SOME_ENV
              value: $SOME_ENV
          ports:
            - containerPort: 8080
```

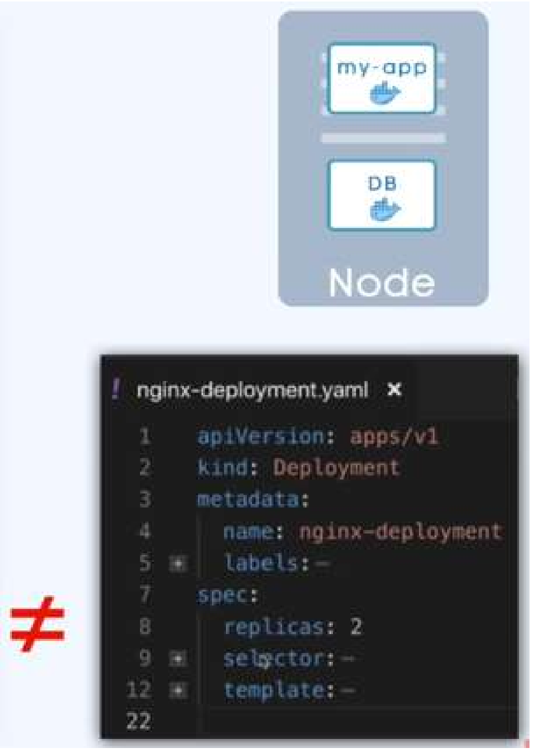


Kubernetes COnfiguration

Kubernetes gets the state of the cluster from etcd



```
status:
  availableReplicas: 1
  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-7d64f4b
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 1
  replicas: 1
  updatedReplicas: 1
```



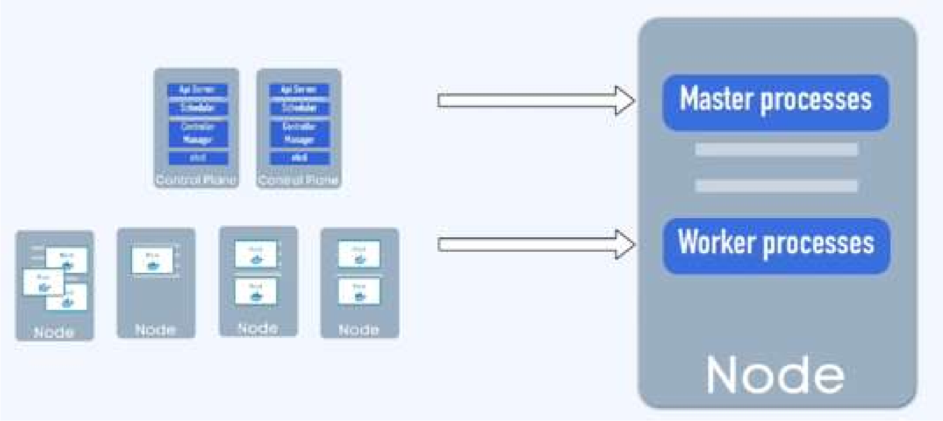
nginx-deployment.yaml ×

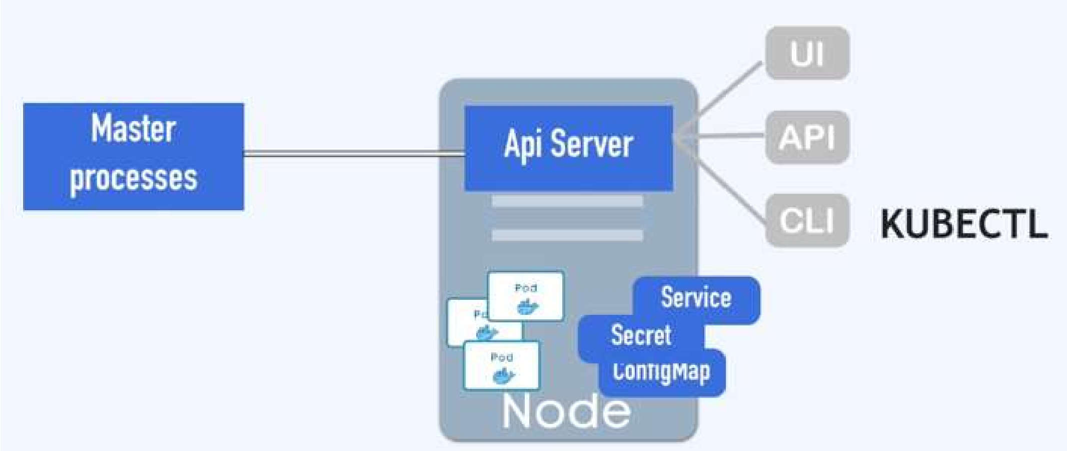
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:…
spec:
  replicas: 2
  selector:…
  template:…
```

One more node will be created as the replicas are not matching in the specification

Minikube and Kubectl

Minikube = Local/Test setup for running Kubernetes cluster, master and node processes run in a single machine





Connecting to the node
Can be used not only on the local cluster, but can be used to talk to any type of Kubernetes cluster such as cloud Kubernetes cluster