Introduction to docker & k8s

Saman Hoseini
Cloud Engineer, *Snapp!*

Me :D

Saman Hoseini

Computer Engineering B.Sc @ Amirkabir University of Technology

Back-End Software Engineer @ ***Snapp!***

saman2000h@aut.ac.ir, saman.hoseini@snapp.cab

aut.ac.ir, snapp.ir

Agenda

docker

- why?
- what?

k8s

- why?
- what?

k8s structure

- master nodes
- worker nodes

k8s resources

- pods
- deployment
- replica set
- config map
- secret
- service
- endpoints
- stateful set
- namespace

Docker

What is it and why do we need it?

docker **k8s** **k8s structure** **k8s resources**

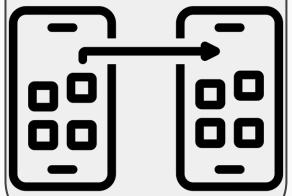
Docker

What is it and why do we need it?

Why containerization?

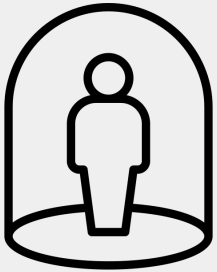
Consistency

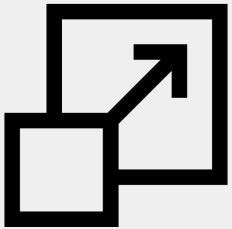
Portability



Resource Efficiency

Isolation





Scalability

Development Speed

Containerization

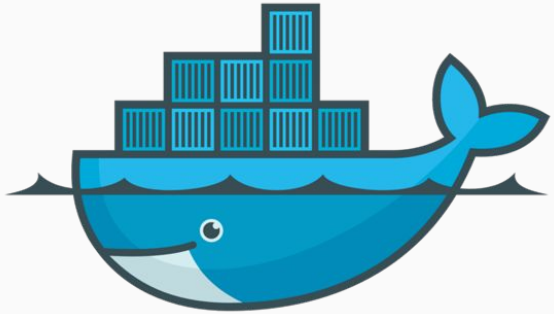
- package applications, libraries, and dependencies
- isolated environment
- sharing the host OS kernel
- single, lightweight, and portable unit
- can run consistently across different environments
- fundamental building block for cloud computing

Why Docker?

User-friendly

Standardization

Large Ecosystem



docker

Community Support

Compatibility

Versatility

Installation and Setup

1. In case of emergency break the sanctions 🚨 (Shecan)
2. Install docker desktop using official documentation
3. Go through post installation steps (Linux only)
4.

```
~
docker --version
Docker version 24.0.6, build ed223bc
```

Lets get hands on

commit

tag

push

pull

ps

inspect

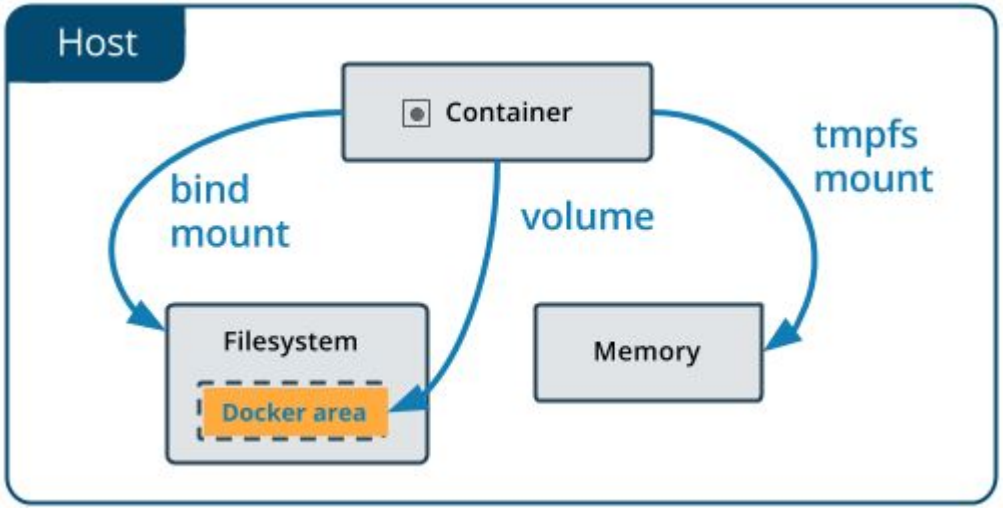
exec

run

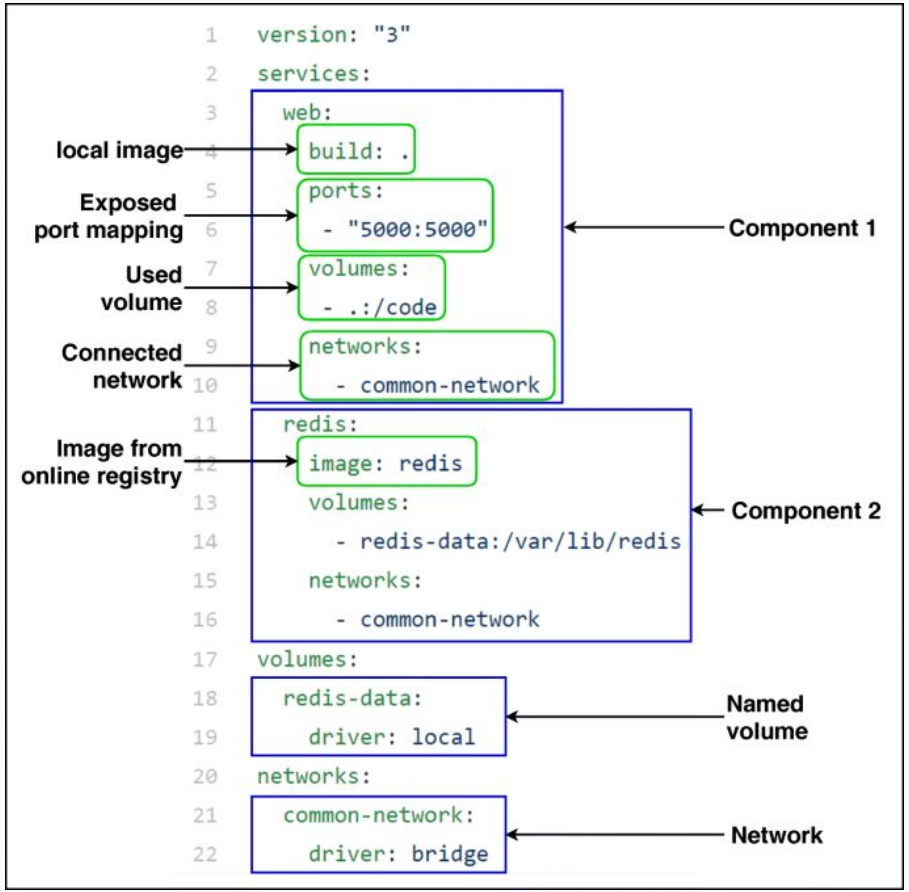
rm

docker mount types

- Volumes
  - docker manages the mount points and data
- Bind mounts
  - save data locally on the host itself
- tmpfs mounts
  - best for sensitive data or information

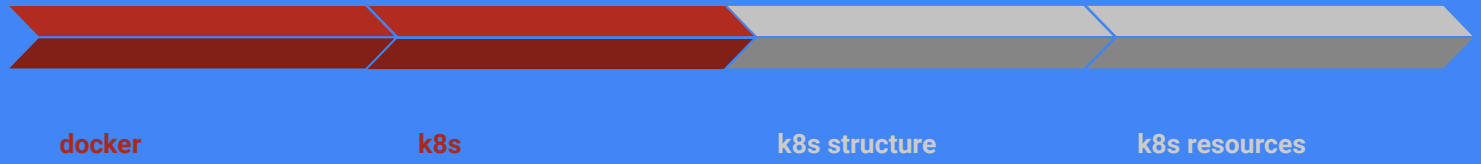


docker compose



Kubernetes

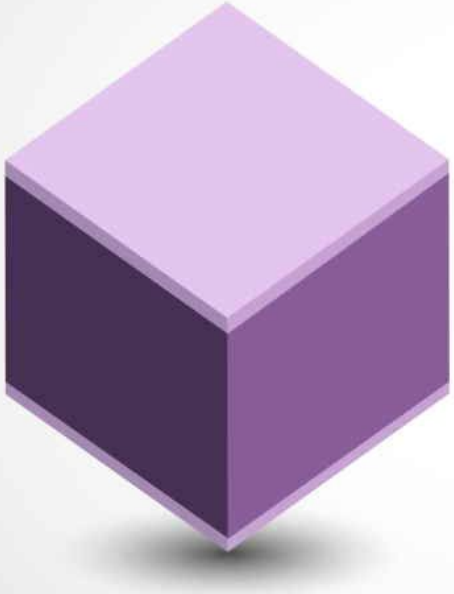
What is it and why do we need it?



Monolithic applications

- Collection of tightly coupled components
- Have to be developed, deployed and managed as one entity
- You must redeploy the entire application on each update
- Scaling up is easy but scaling out requires major code changes
- If a single part of an application is unscalable then the whole application becomes unscalable

Monolithic



Microservices

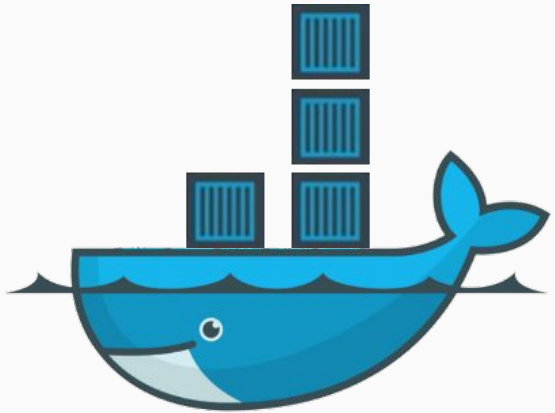
Microservice Architecture

These and other problems have forced us to start splitting complex monolithic applications into smaller independently deployable components called microservices.
Microservices communicate through:

1. Synchronous protocols such as REST API or gRPC

2. Asynchronous protocols such as AMQP & MQTT

- docker run my-image



Microservice drawbacks

Service Discovery

- Microservices perform their work together as a team, so they need to find and talk to each other.
- With increasing numbers of microservices, this becomes tedious and error-prone.

Microservice drawbacks (cont.)

Component Management

Increase in number of components:

- deployment-related decisions become increasingly difficult
- the number of deployment combinations increase
- the number of inter-dependencies between the components increases by an even greater factor

Microservice drawbacks (cont.)

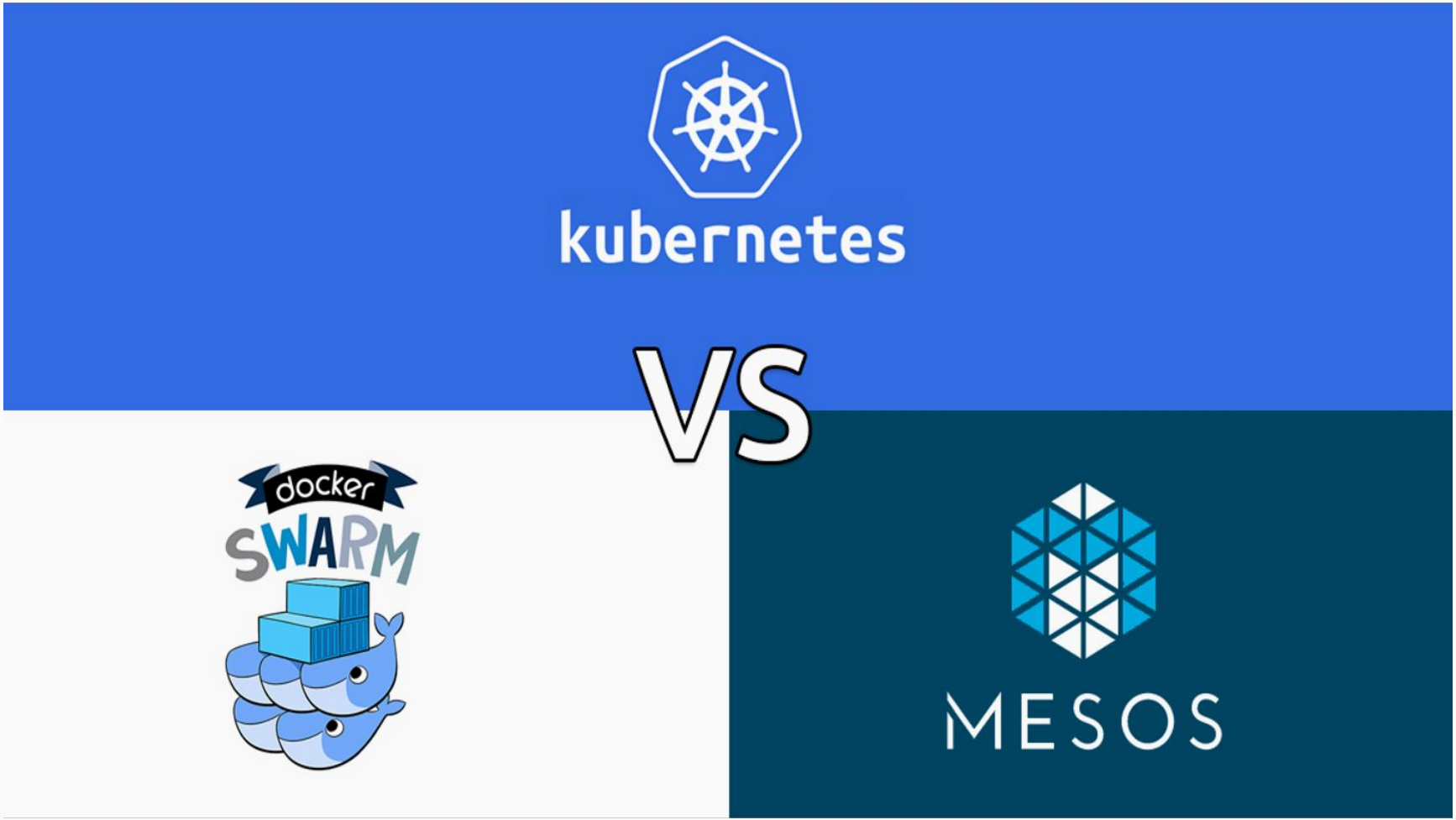
Library Conflicts

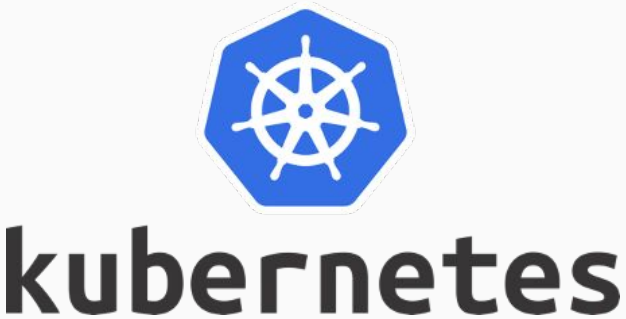
When each component has its own development team, nothing impedes each team from using different libraries and replacing them whenever the need arises.

Solution:
Container Orchestration

Container orchestration advantages

1. Scaling according to load of the system
2. Advanced network between containers in different hosts
3. Sharing storage between the hosts
4. Configuration managements
5. Clusters security


kubernetes
VS
docker
SWARM
MESOS

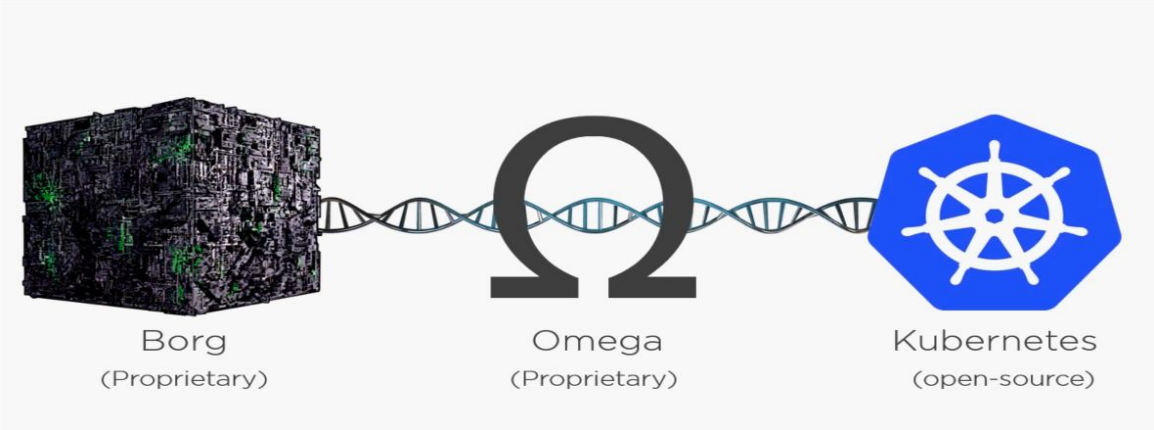

kubernetes

Overview

- Born in Google
- Donated to CNCF in 2014 (open source)
- Written in Go/Golang
- https://github.com/kubernetes/kubernetes
- Often shortened to k8s

DNA

- Borg: a cluster manager used by Google
- Omega: an offspring of Borg

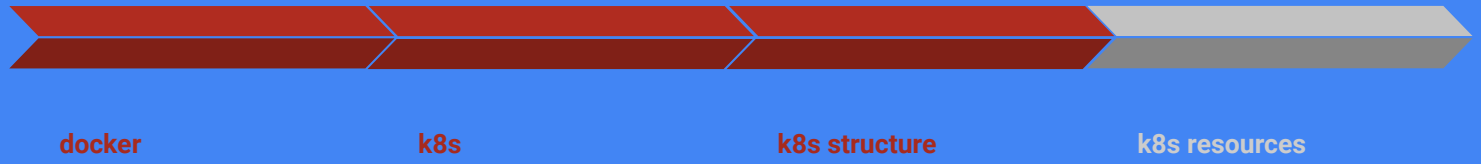


Goal

- Scale containers
- Storage orchestration
- Load balancer
- Update containers without bringing down the application
- Eliminate single points of failure – Self-healing

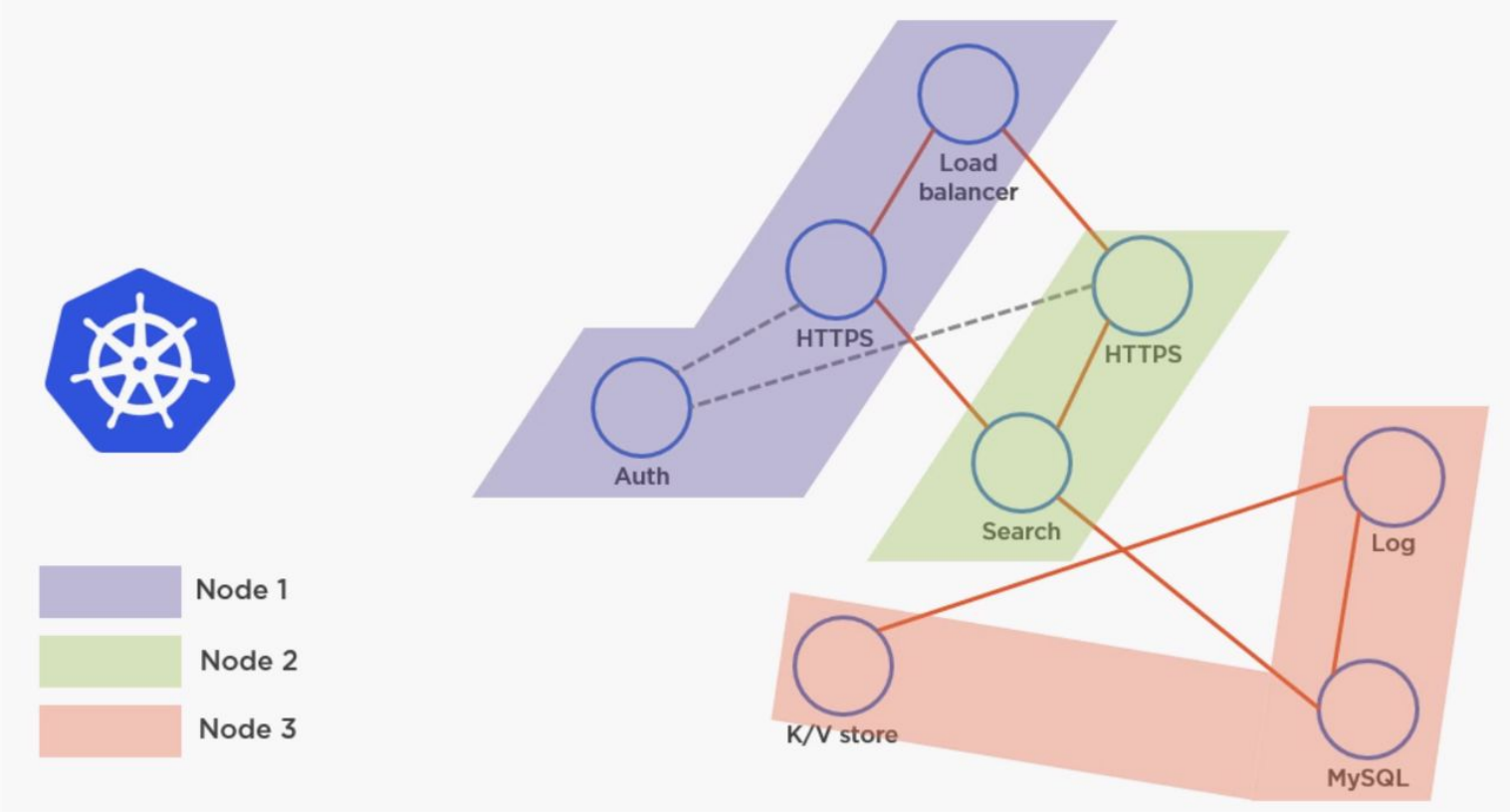
Kubernetes Structure

How does kubernetes components work together?

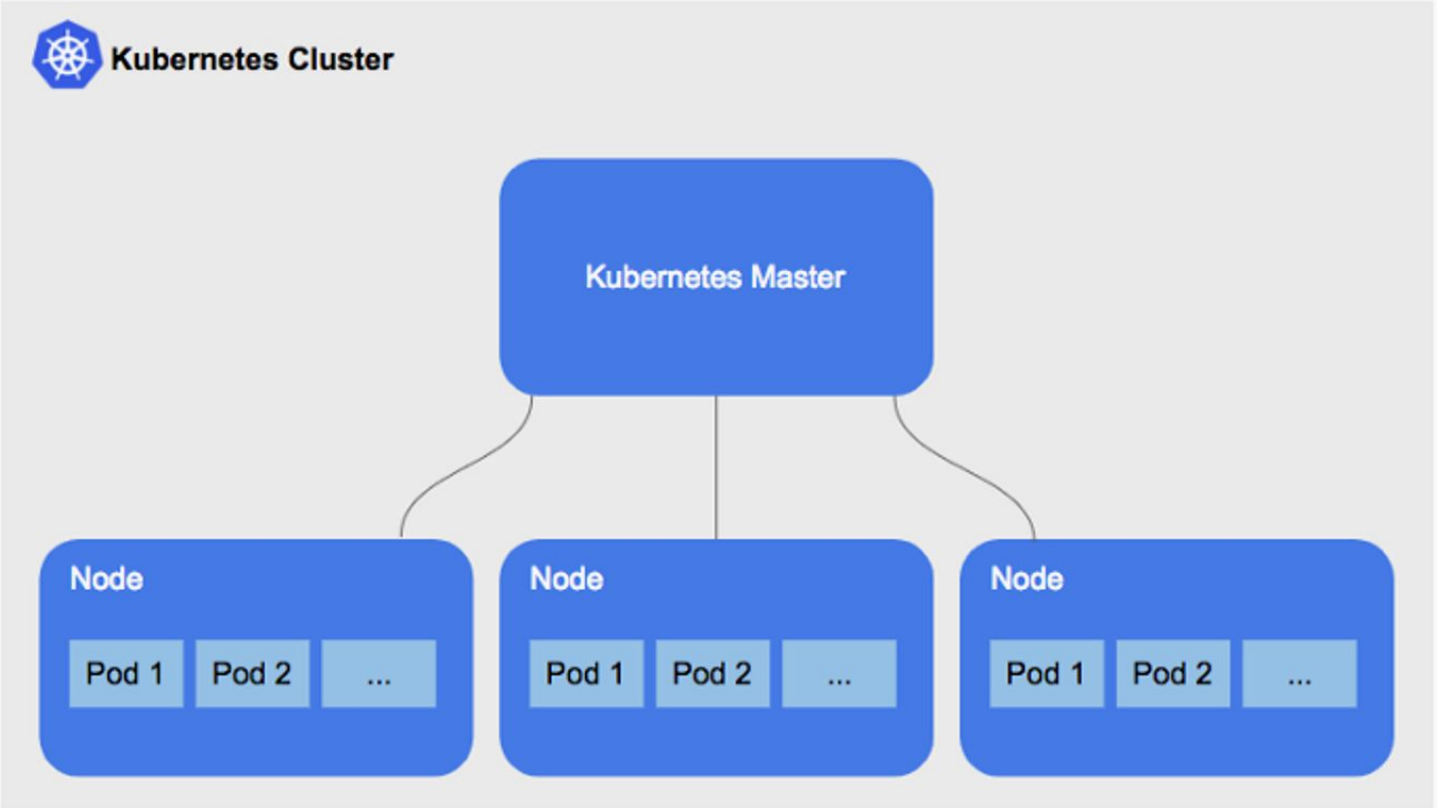


Big picture view

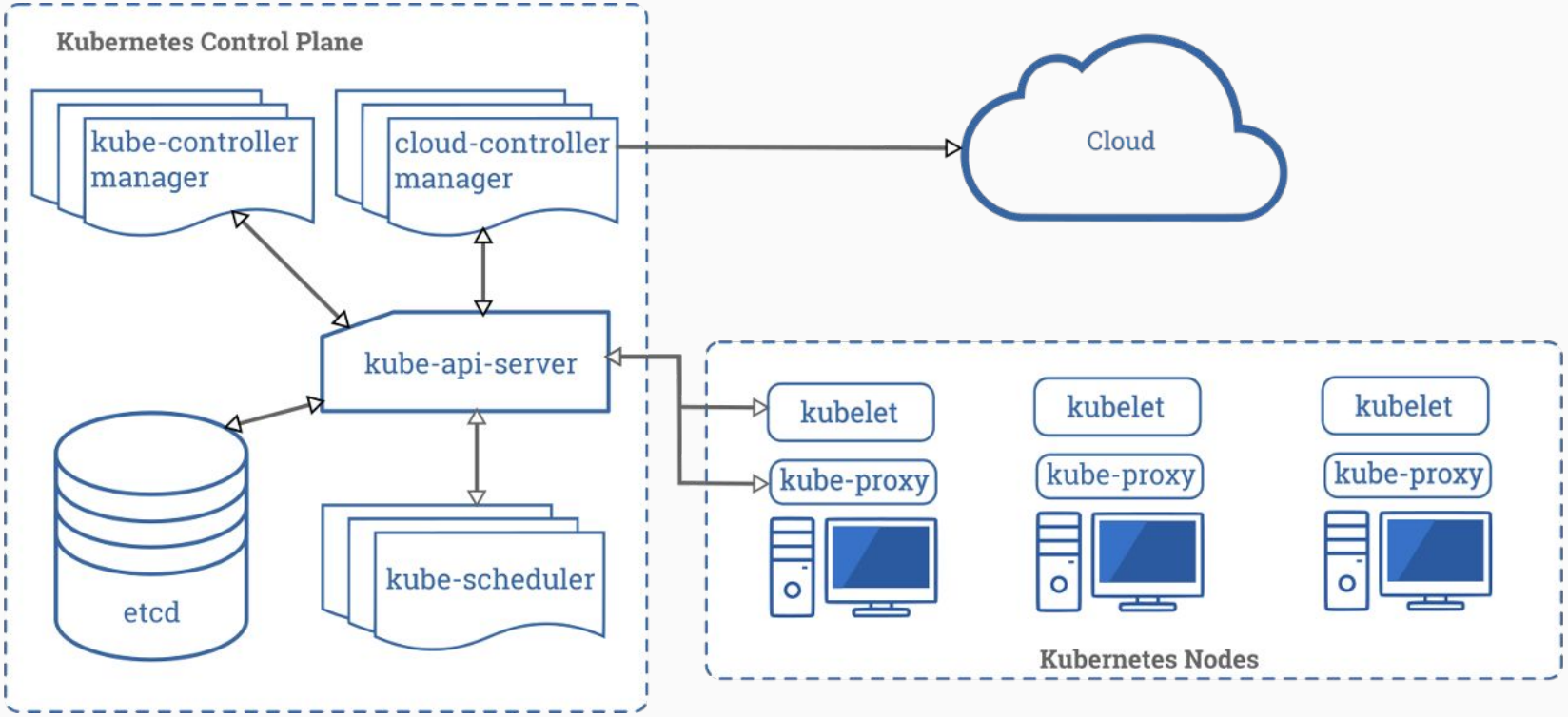
An orchestrator for microservice apps



An orchestrator for microservice apps

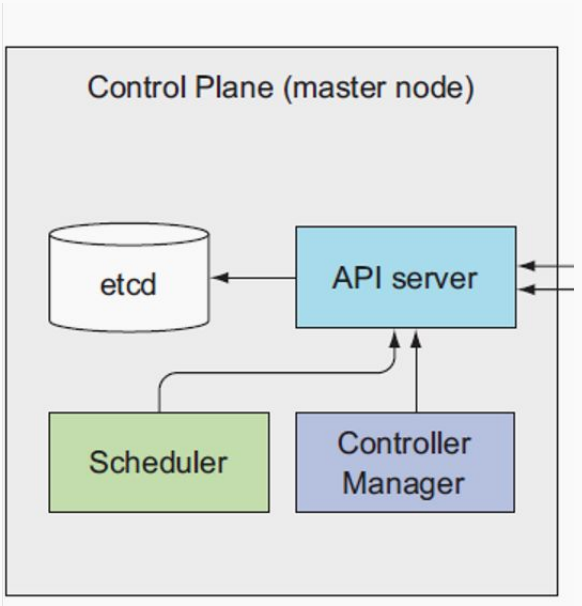


Kubernetes control plane (master node)



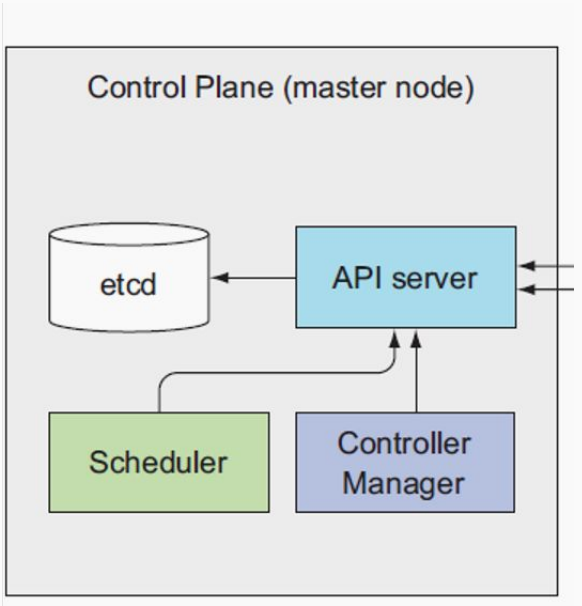
API server

- Front-end to the control plane
- Exposes the API (REST)
- Consumes JSON



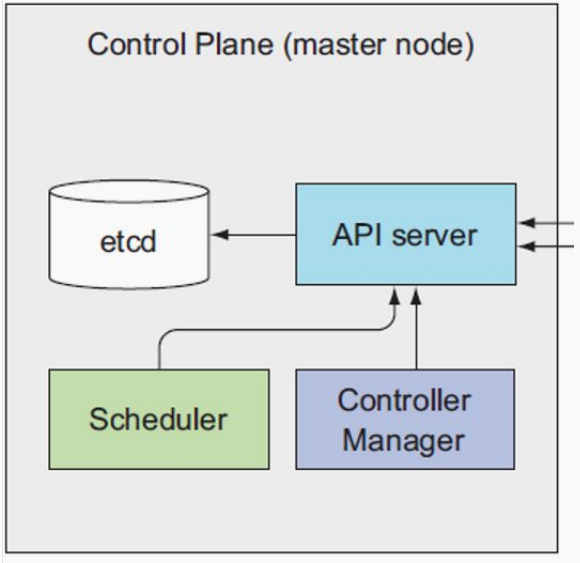
Cluster Store

- Persistent storage
- Cluster state and config
- It uses etcd
  - etcd is a distributed, reliable key-value store for the most critical data of a distributed system



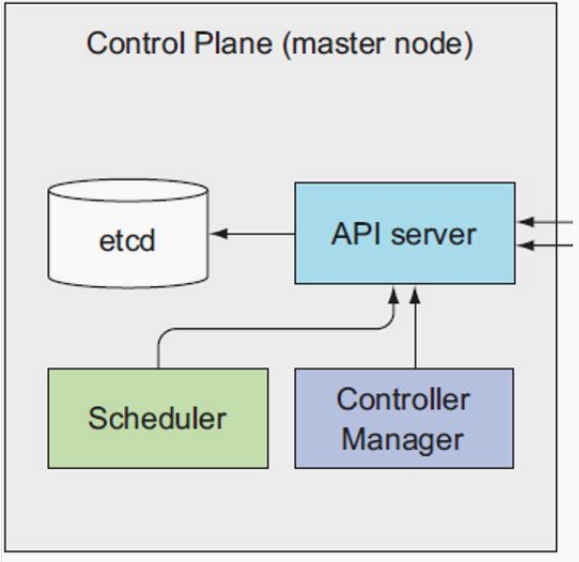
Controller Manager

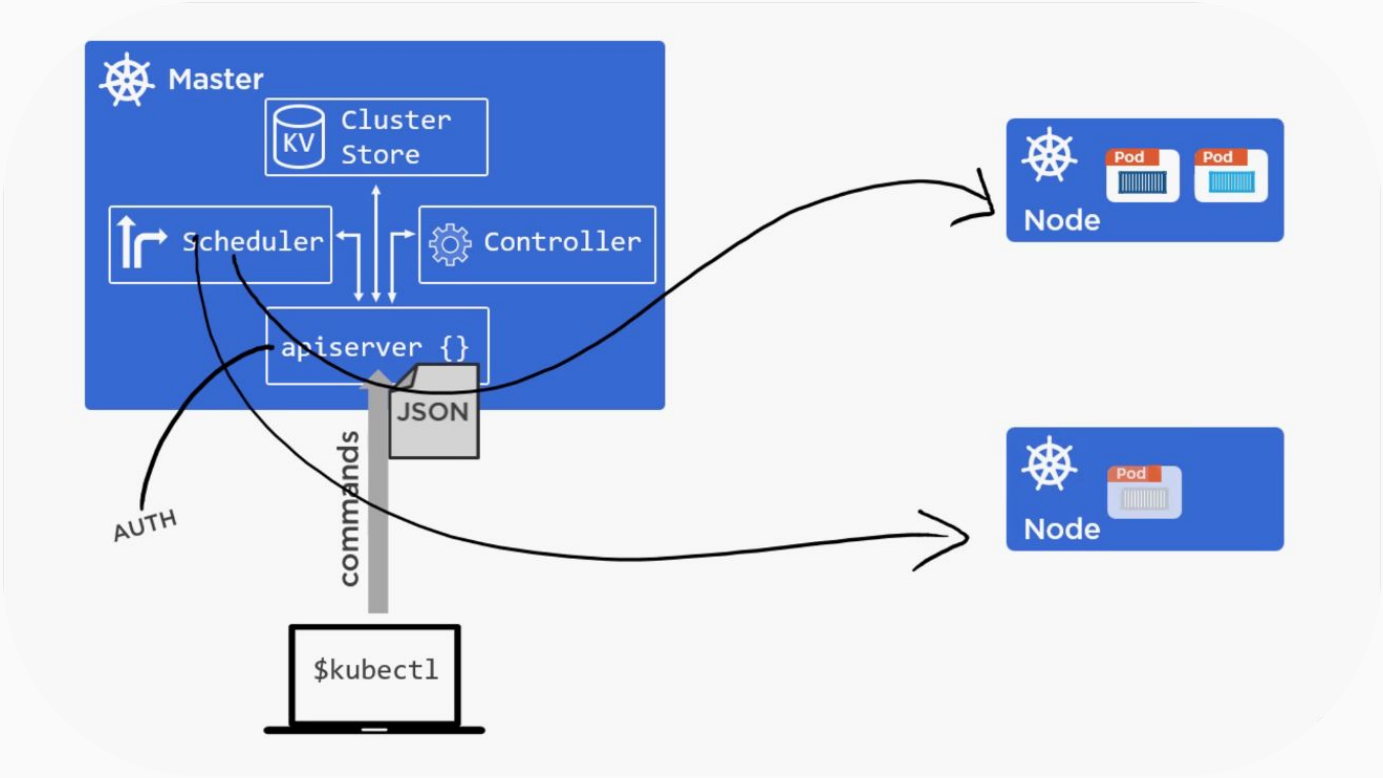
- Controller of controllers:
  - Node controller
  - Endpoints controller
  - Namespace controller
  - ...
- Watches for changes
- Helps maintain desired state



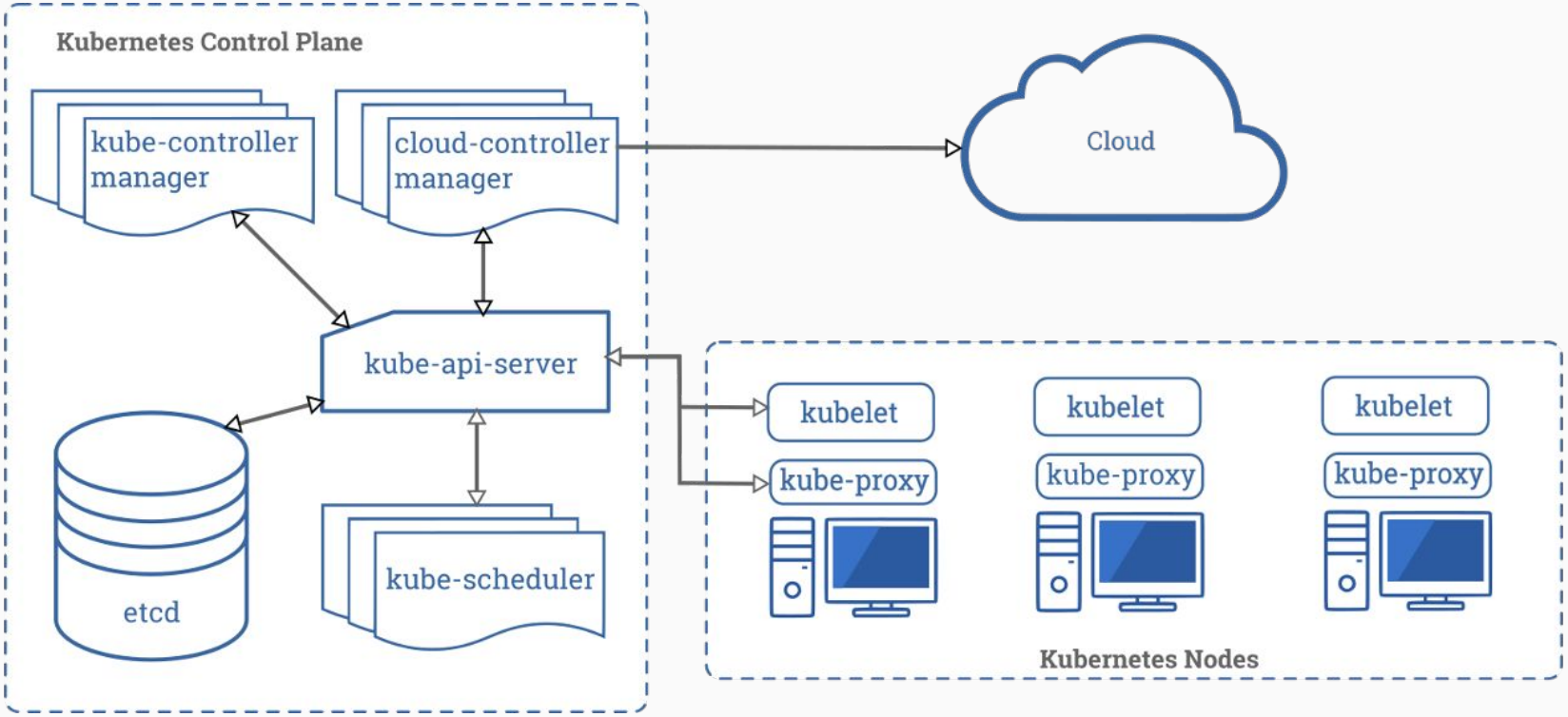
Scheduler

- Watches api-server for new pods
- Assign work to nodes



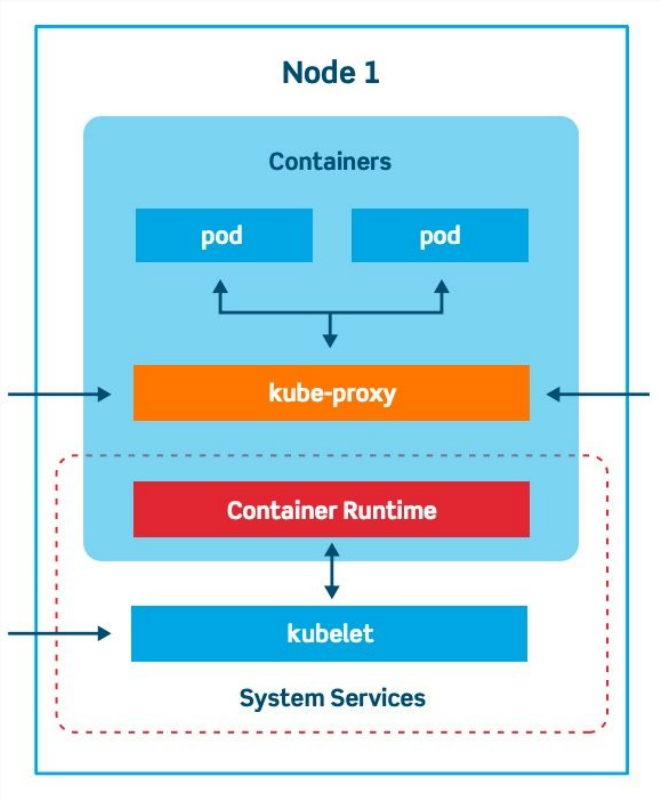

Master
KV
Cluster Store
Scheduler
Controller
apiserver {}
JSON
commands
AUTH
$kubectl
Pod
Pod
Node
Pod
Node

Worker node


Kubernetes Control Plane
kube-controller manager
cloud-controller manager
Cloud
kube-api-server
etcd
kube-scheduler
kubelet
kube-proxy
kubelet
kube-proxy
kubelet
kube-proxy
Kubernetes Nodes

Kubelet

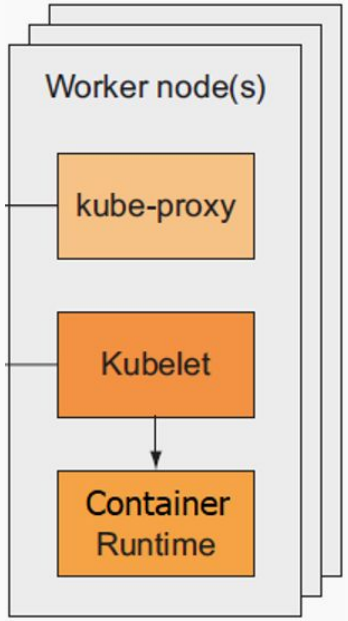
- The main Kubernetes agent
- Registers node with cluster
- Watches api-server
- Instantiates pods
- Reports back to master



Container Runtime

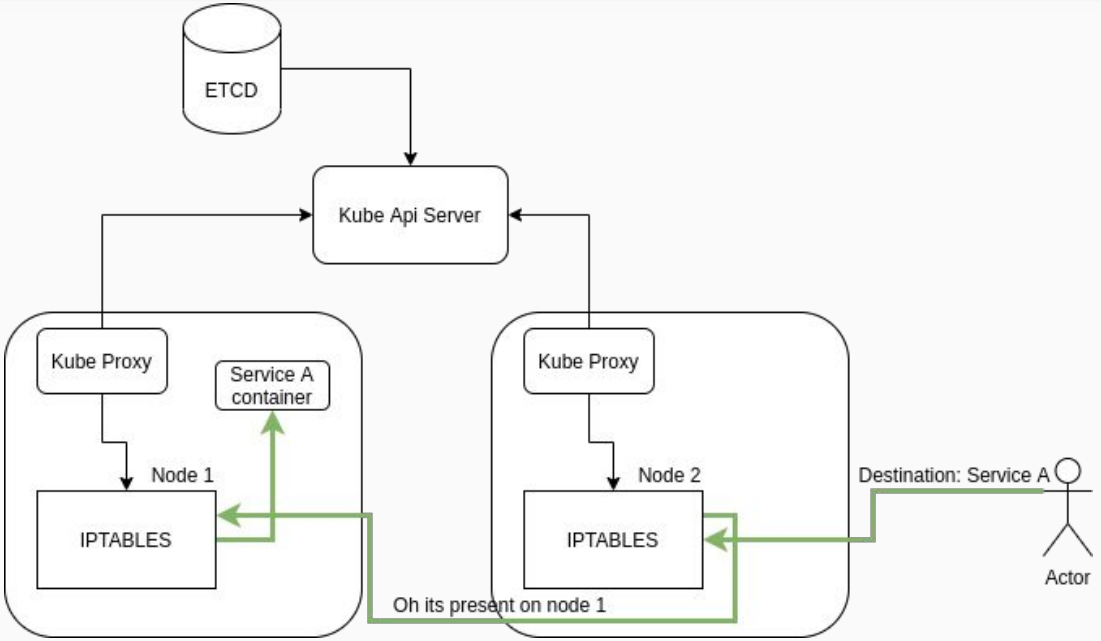
- Does container management:
  - Pulling images
  - starting/stopping containers
- Pluggable:
  - Usually Docker 

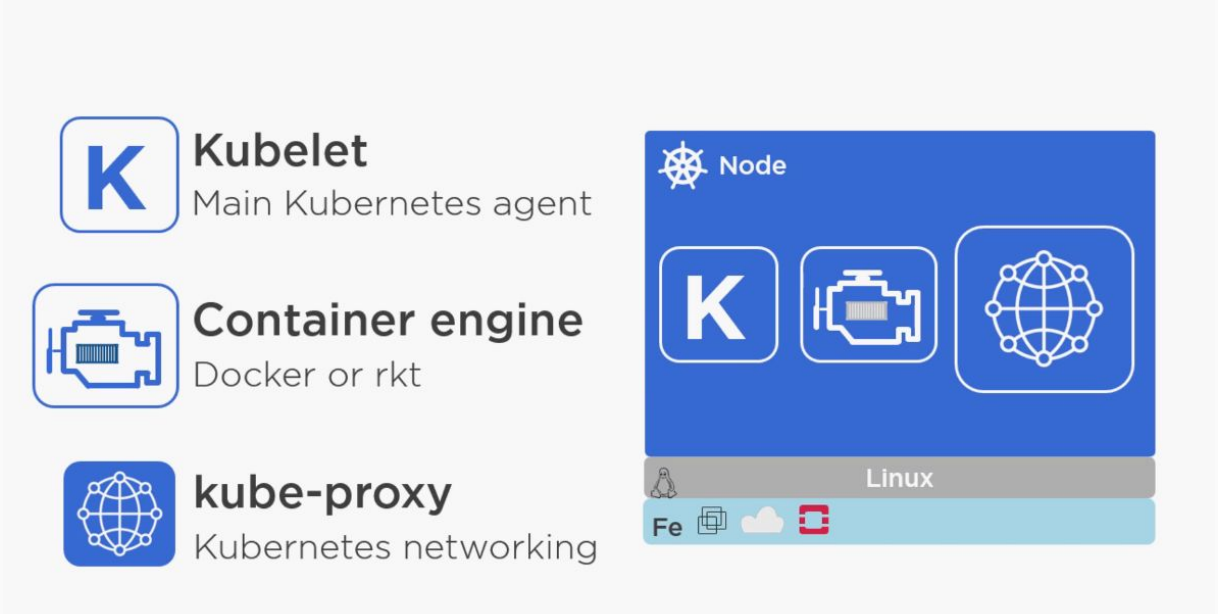
  - Can be rkt 



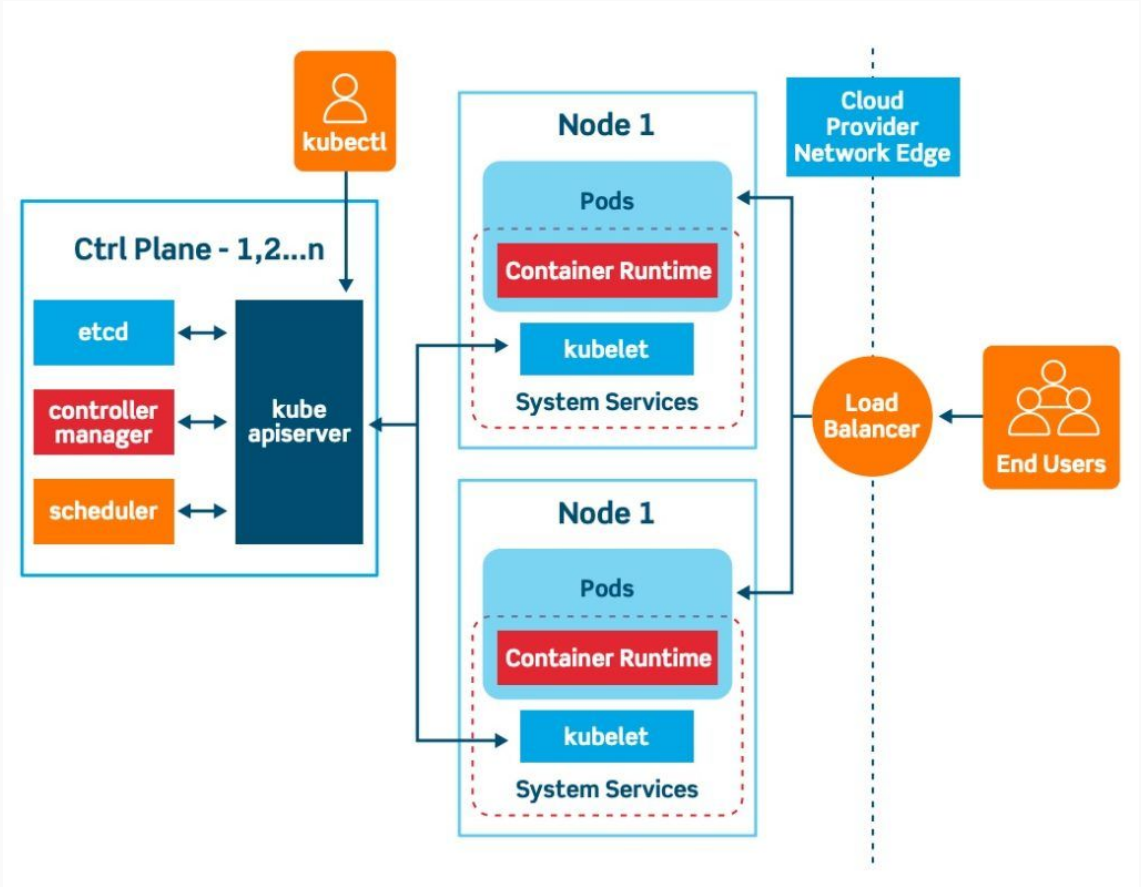
kube-proxy

- Kubernetes networking:
  - Pod IP addresses
  - Load balancer for pods

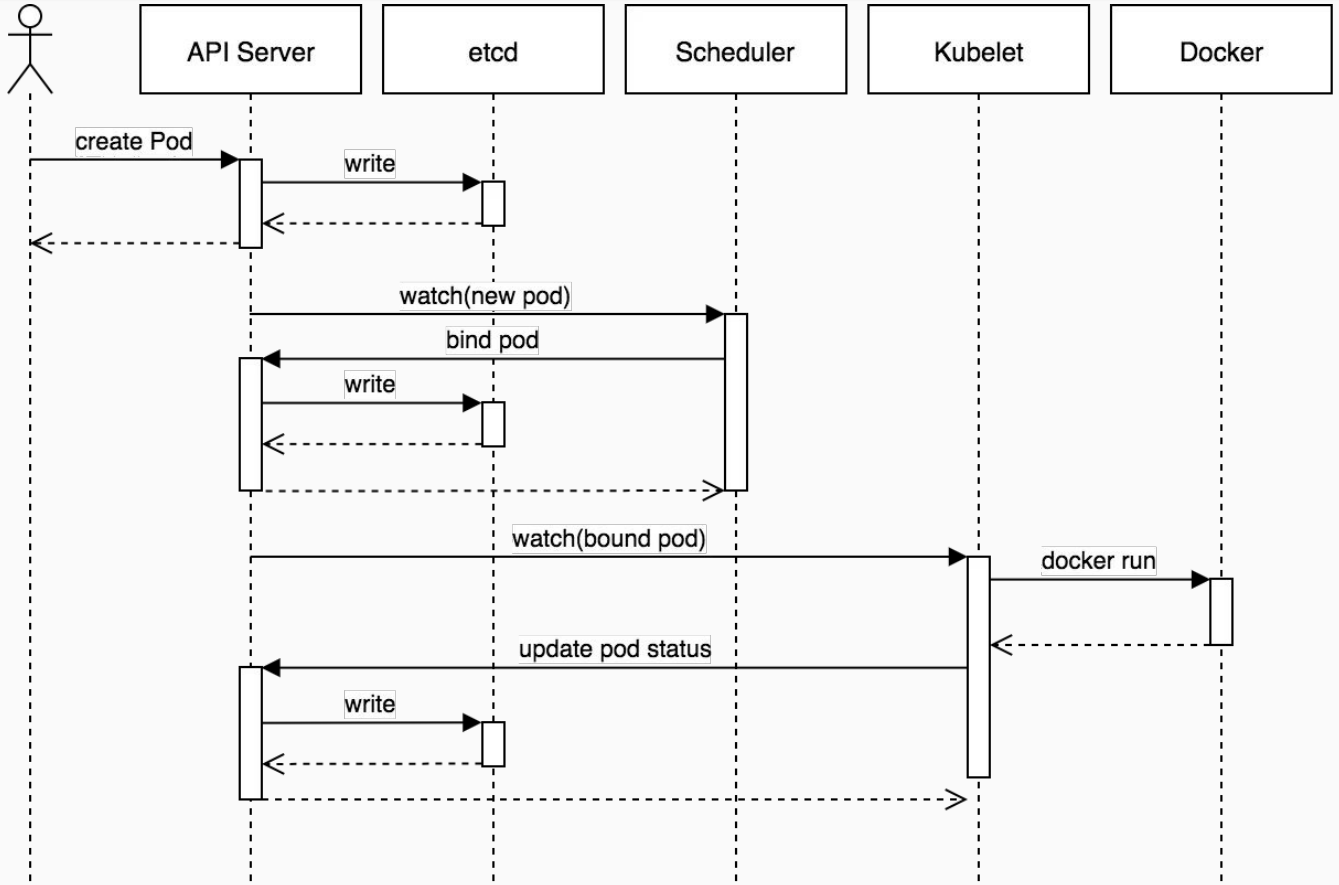



Kubelet
Main Kubernetes agent
Container engine
Docker or rkt
kube-proxy
Kubernetes networking
Node
Linux
Fe

Conclusion

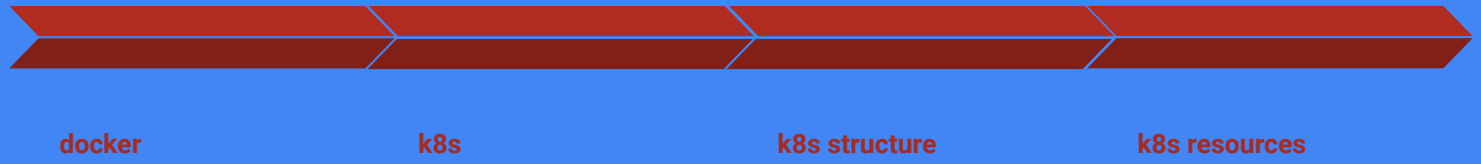

kubectl
Ctrl Plane - 1,2...n
etcd
controller manager
scheduler
kube apiserver
Node 1
Pods
Container Runtime
kubelet
System Services
Node 1
Pods
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users

Conclusion


API Server
etcd
Scheduler
Kubelet
Docker
create Pod
write
watch(new pod)
bind pod
write
watch(bound pod)
docker run
update pod status
write

Kubernetes Objects

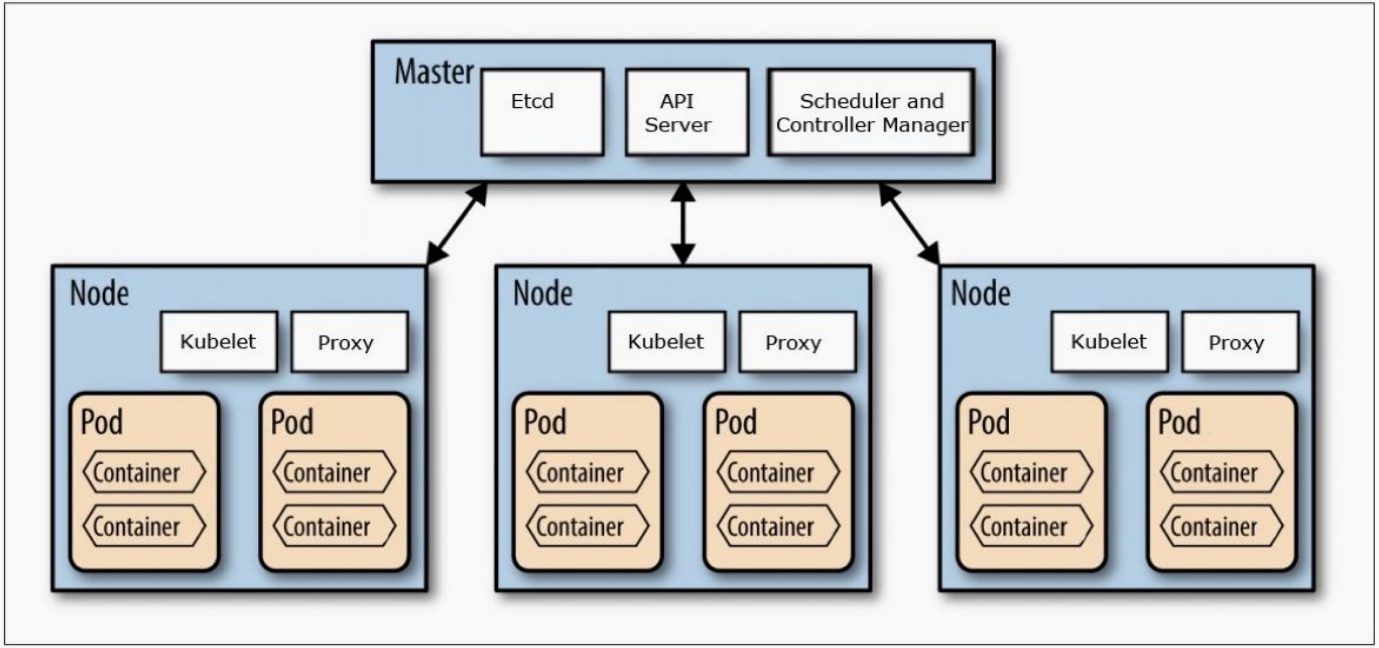
Kubernetes main objects



Pods

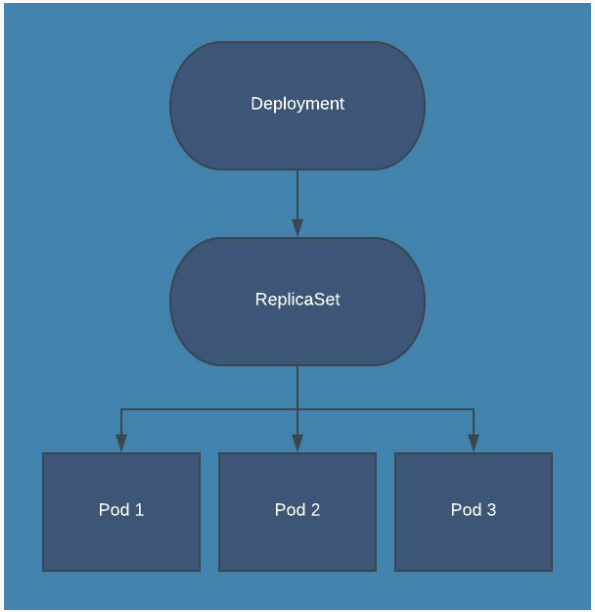
- Ring-fenced environment
  - Network stack
  - Kernel namespaces
  - …
- n containers
- All containers in pod share the environment

Pods (cont.)


Master
Etcd
API Server
Scheduler and Controller Manager
Node
Kubelet
Proxy
Pod
Container
Container
Pod
Container
Container
Node
Kubelet
Proxy
Pod
Container
Container
Pod
Container
Container
Node
Kubelet
Proxy
Pod
Container
Container
Pod
Container
Container

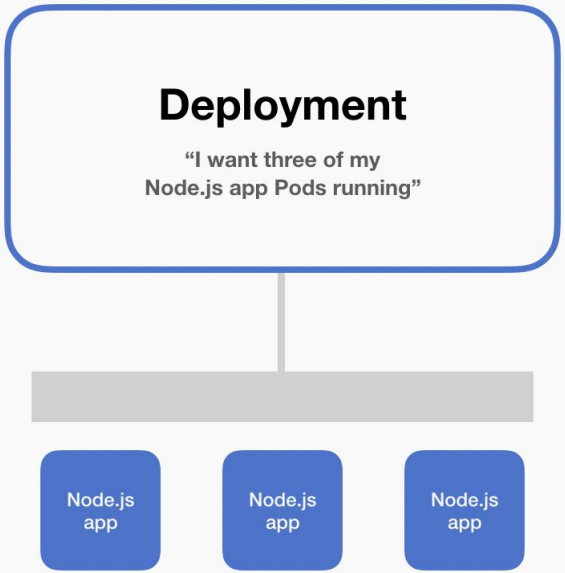
ReplicaSet

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.



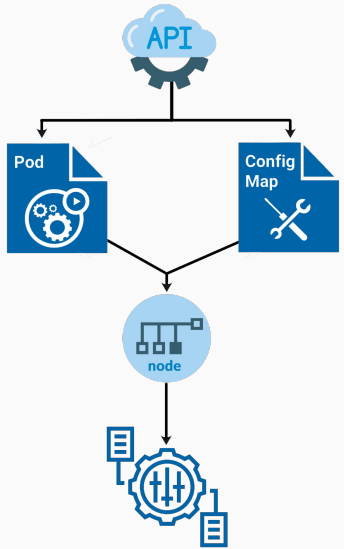
Deployment

A ReplicaSet ensures that a specified number of pod replicas are running at any given time. However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.



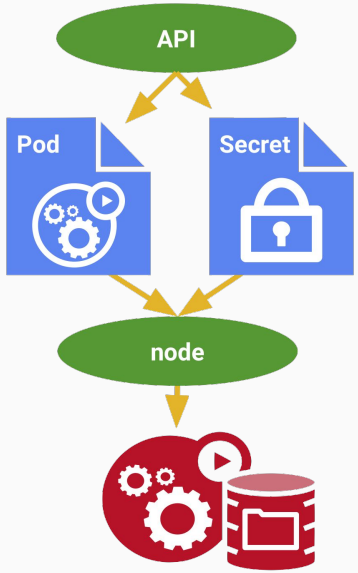
ConfigMaps

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.



Secret

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image.

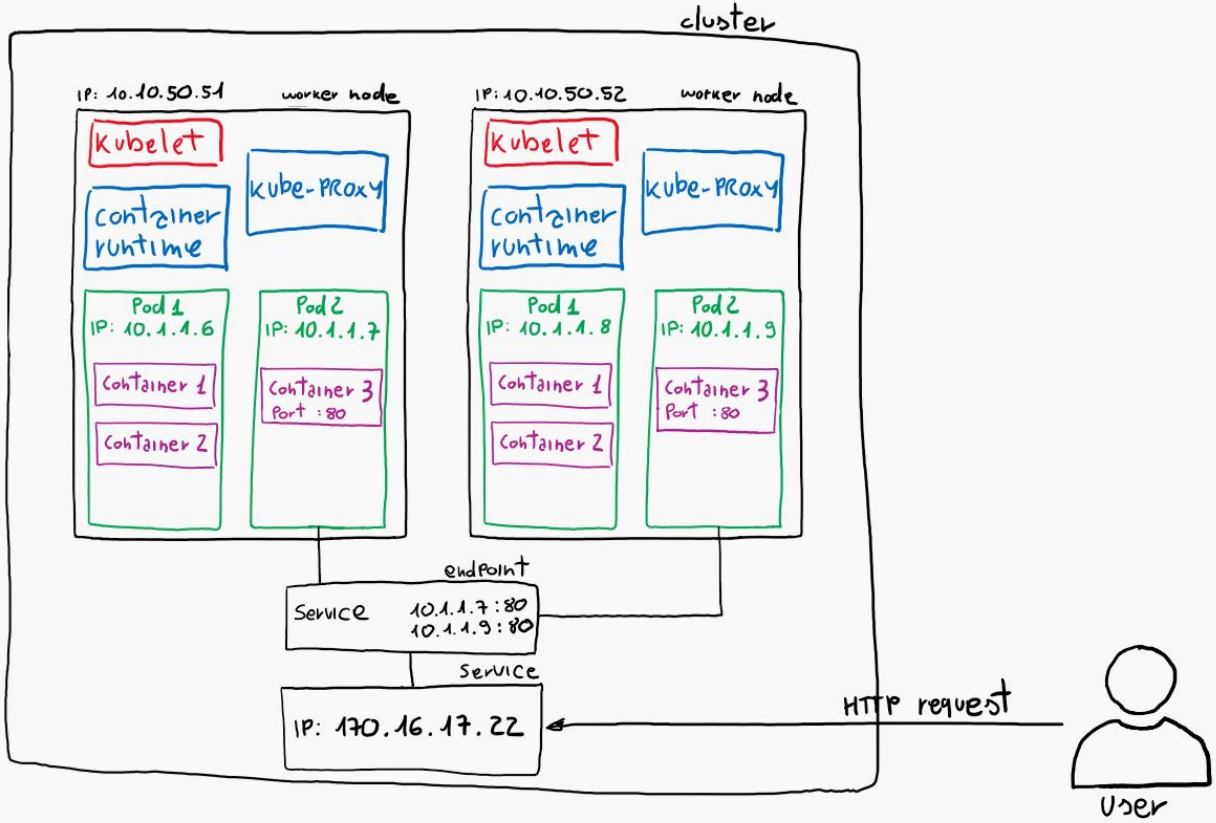


Service

An abstract way to expose an application running on a set of Pods as a network service:

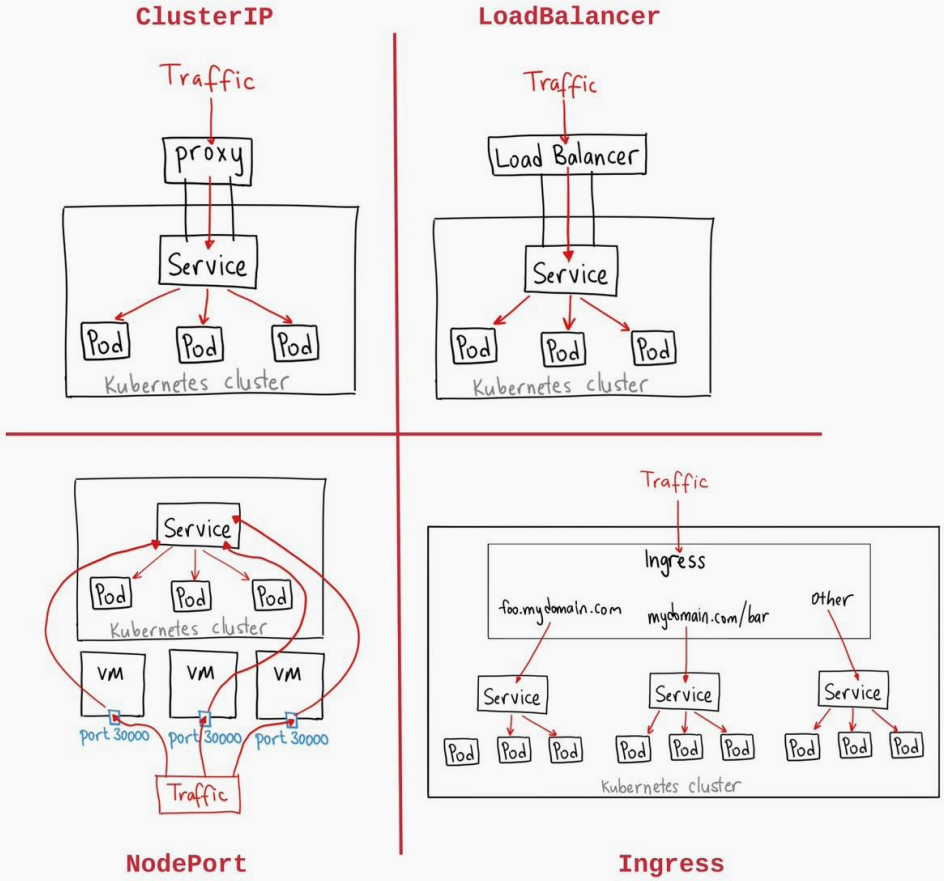
- Pods are non permanent resources
- Each Pod gets its own IP address
- Single DNS name for a set of Pods
- load-balance across them

Service (cont.)



Service (cont.)

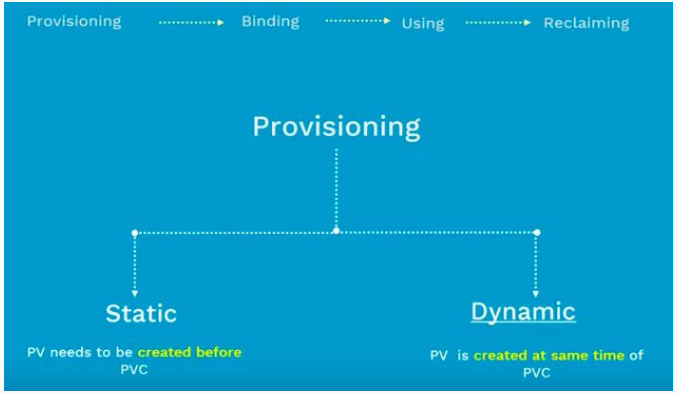
Service (cont.)



Persistent Volumes

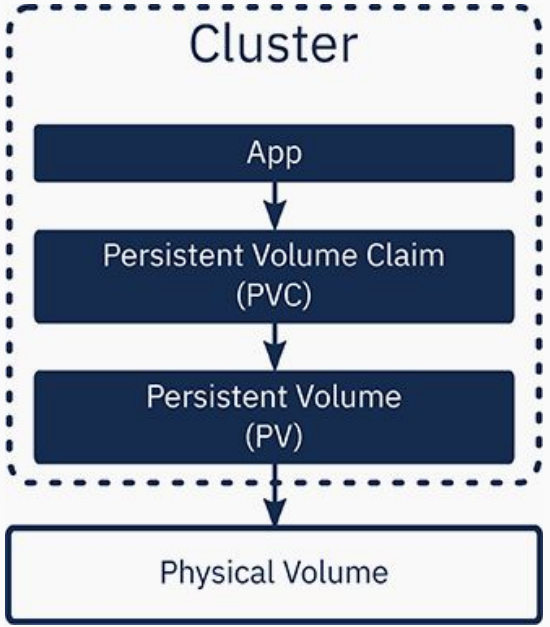
On-disk files in a container are ephemeral:

- the loss of files when a container crashes
  - kubelet restarts the container but with a clean state
- sharing files between containers running together in a Pod



Persistent Volume Claims

- A persistent volume claim (PVC) is a request for storage by a user from a PV
- Claims can request specific size and access modes
  - once read/write
  - many times read-only



PVC ReadWriteOnce access mode

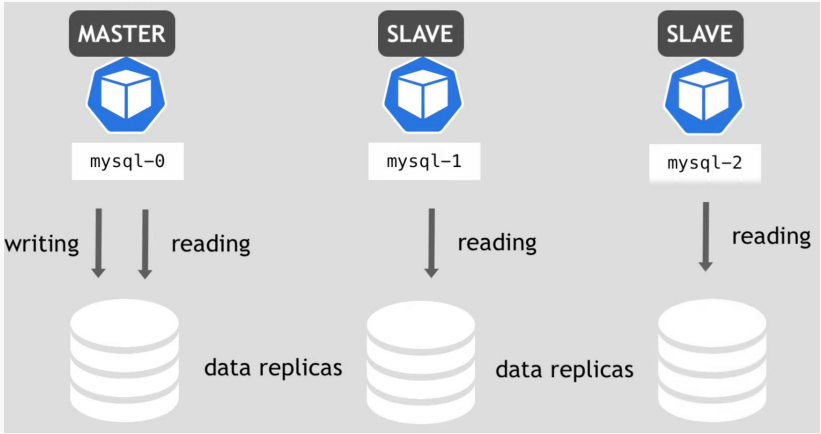
PVC with once read/write access mode can be mounted as read-write by a single node.

So you won’t face any errors on minikube with single node, but it may cause concurrency problems (race condition).

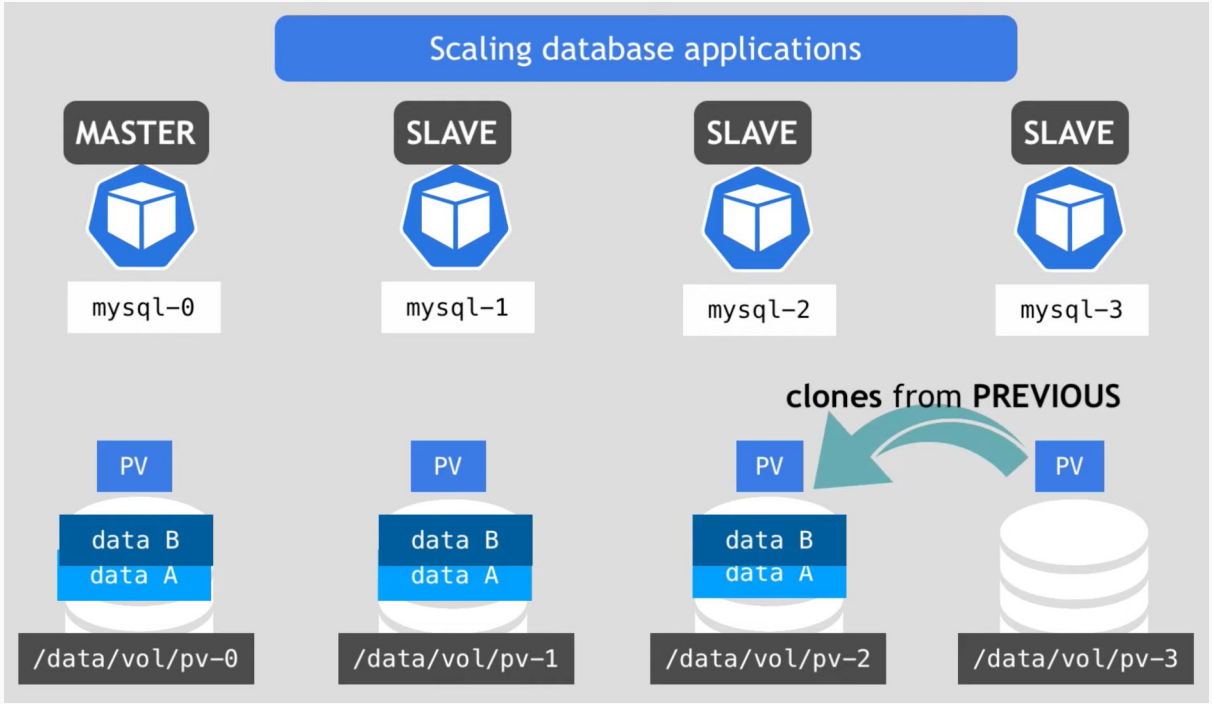
StatefulSet

StatefulSets are valuable for applications that require one or more of the following:

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, automated rolling updates.

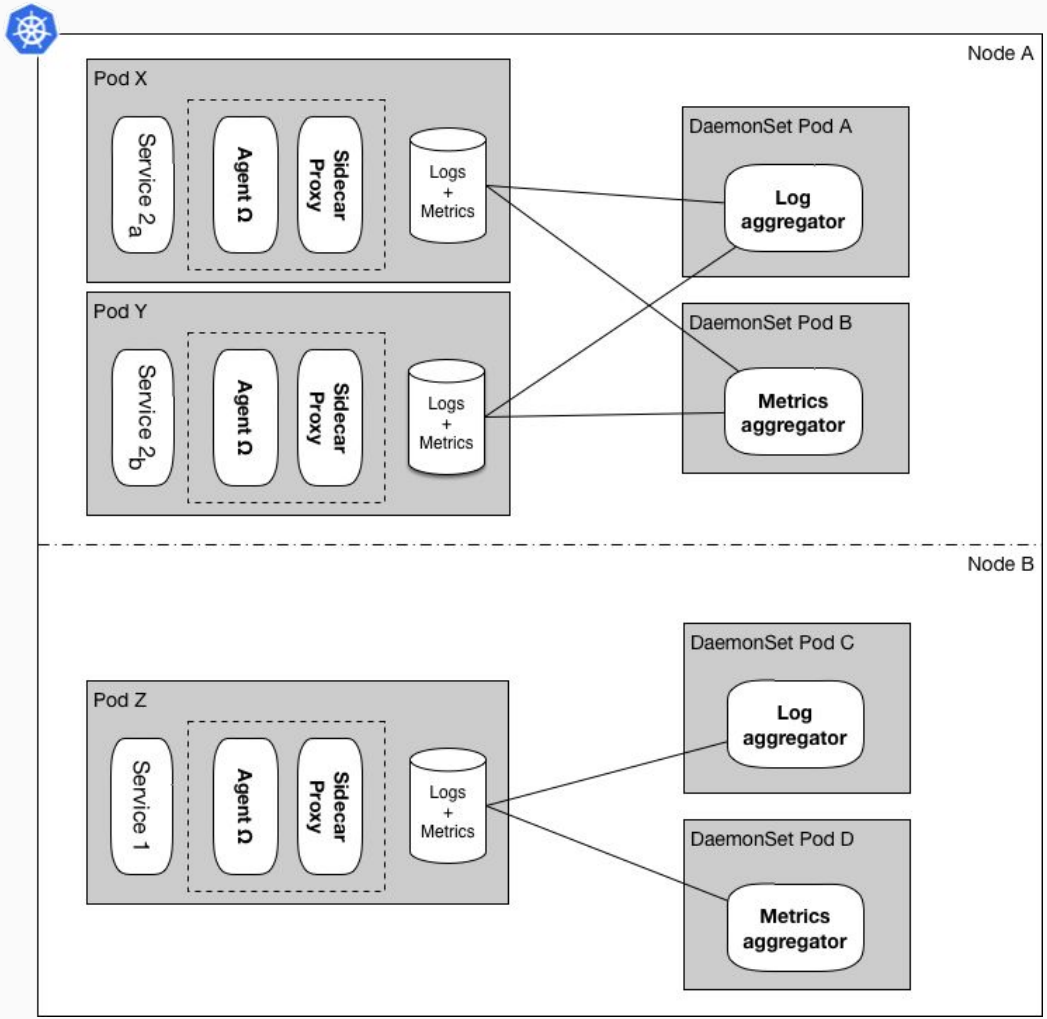


StatefulSet (cont.)



DaemonSet

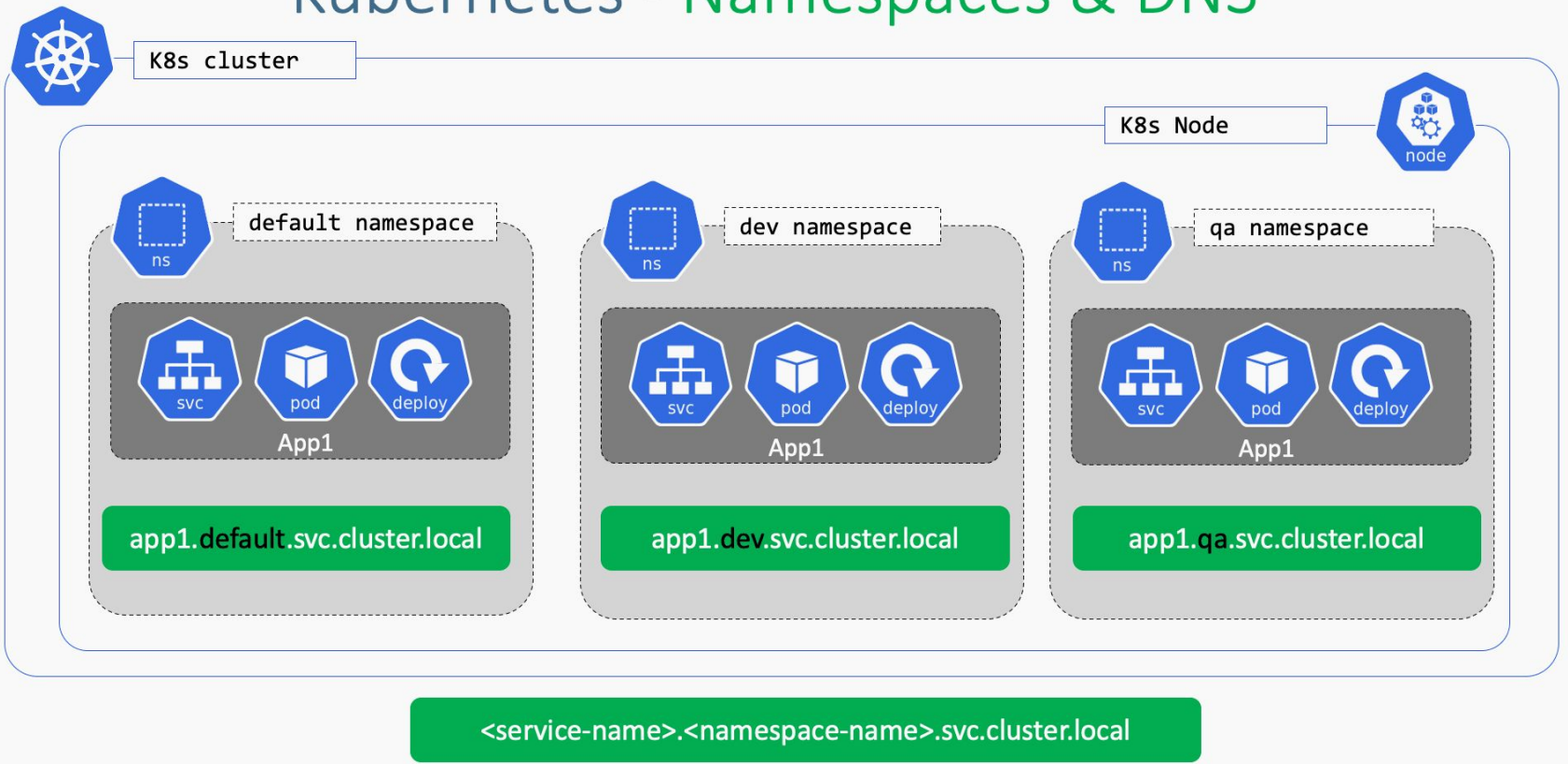
Ensures that all (or some) Nodes run a copy of a Pod.



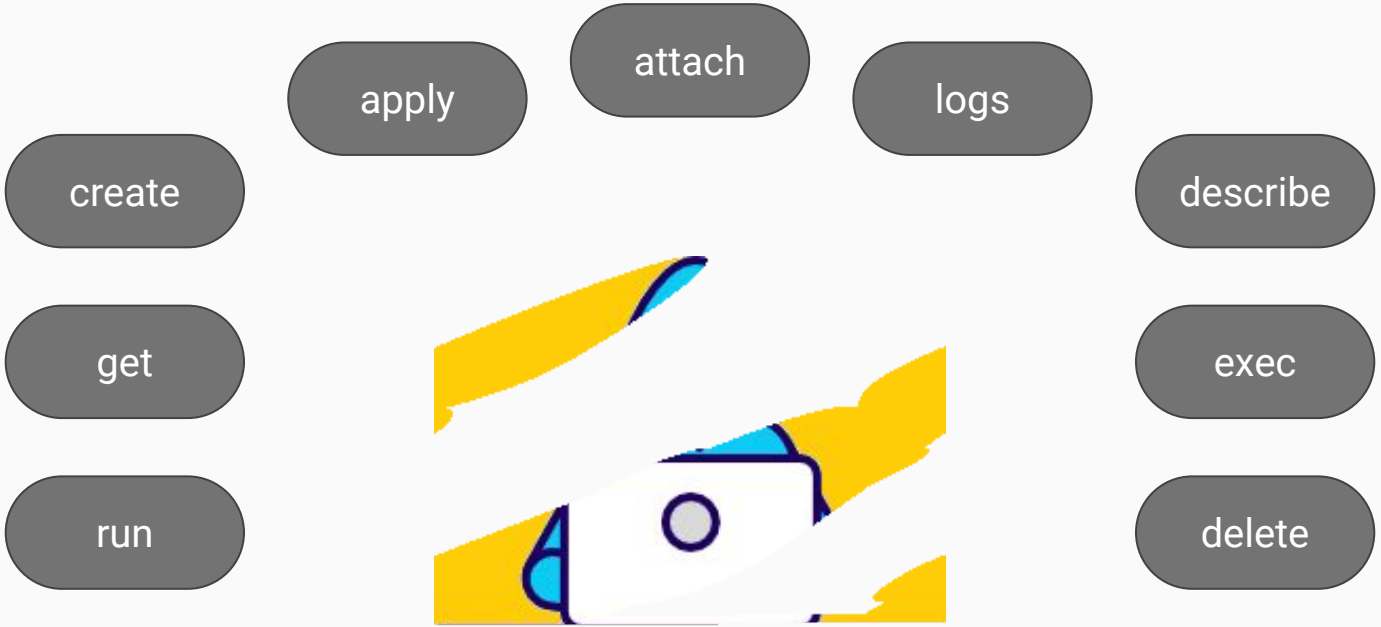
Namespace

- Organize resources
- Virtual cluster inside k8s cluster
- Group k8s resources
- Access and resource limit

Kubernetes - Namespaces & DNS



Lets get hands on



Pods and Basics

- kubectl run net-utils --image=saman2000h/net-utils:1.2
- kubectl get pods
- kubectl logs net-utils
- kubectl describe pod net-utils
- kubectl run -i -t net-utils --image=saman2000h/net-utils:1.2
- kubectl attach net-utils -c net-utils -i -t
- kubectl exec net-utils -it -- bash
- kubectl run net-utils --image=saman2000h/net-utils:1.2 sleep infinite
- kubectl logs net-utils

Deployments and Basics

- kubectl create deployment net-utils --image=saman2000h/net-utils:1.2
- kubectl create deployment net-utils --image=saman2000h/net-utils:1.2 --dry-run=client -o yaml > deployment.yaml
- kubectl scale deployment net-utils --replicas=2
- kubectl explain pods
- kubectl explain deployment.spec
- selector: matchLabels: app: net-utils ⇒ template: metadata: labels:
- kubectl api-resources

Deploy visitors

- kubectl apply -f secret.yaml
- kubectl apply -f redis-deployment.yaml
- kubectl apply -f redis-service.yaml
- kubectl apply -f config-map.yaml
- kubectl apply -f visitors-deployment.yaml
- kubectl apply -f visitors-service.yaml
- kubectl replace -f manifest.yaml

helm charts

```
deepakgupta@197nodnb3072391:~/opstree/helm$ sudo helm create gowebapp
Creating gowebapp
deepakgupta@197nodnb3072391:~/opstree/helm$ tree gowebapp/
gowebapp/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 9 files
```

Useful links

- https://docs.docker.com/engine/install/
- https://minikube.sigs.k8s.io/docs/start/
- https://helm.sh/
- https://kubernetes.io/docs/home/
- https://github.com/ahmetb/kubectx
- https://github.com/saman2000hoseini/k8s-training

References

- https://kubernetes.io/docs/home/
- Kubernetes Tutorial for Beginners [FULL COURSE in 4 Hours]
- https://stacksimplify.com/azure-aks/azure-kubernetes-service-namespaces-imperative/
- https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59