Cloud Native certified

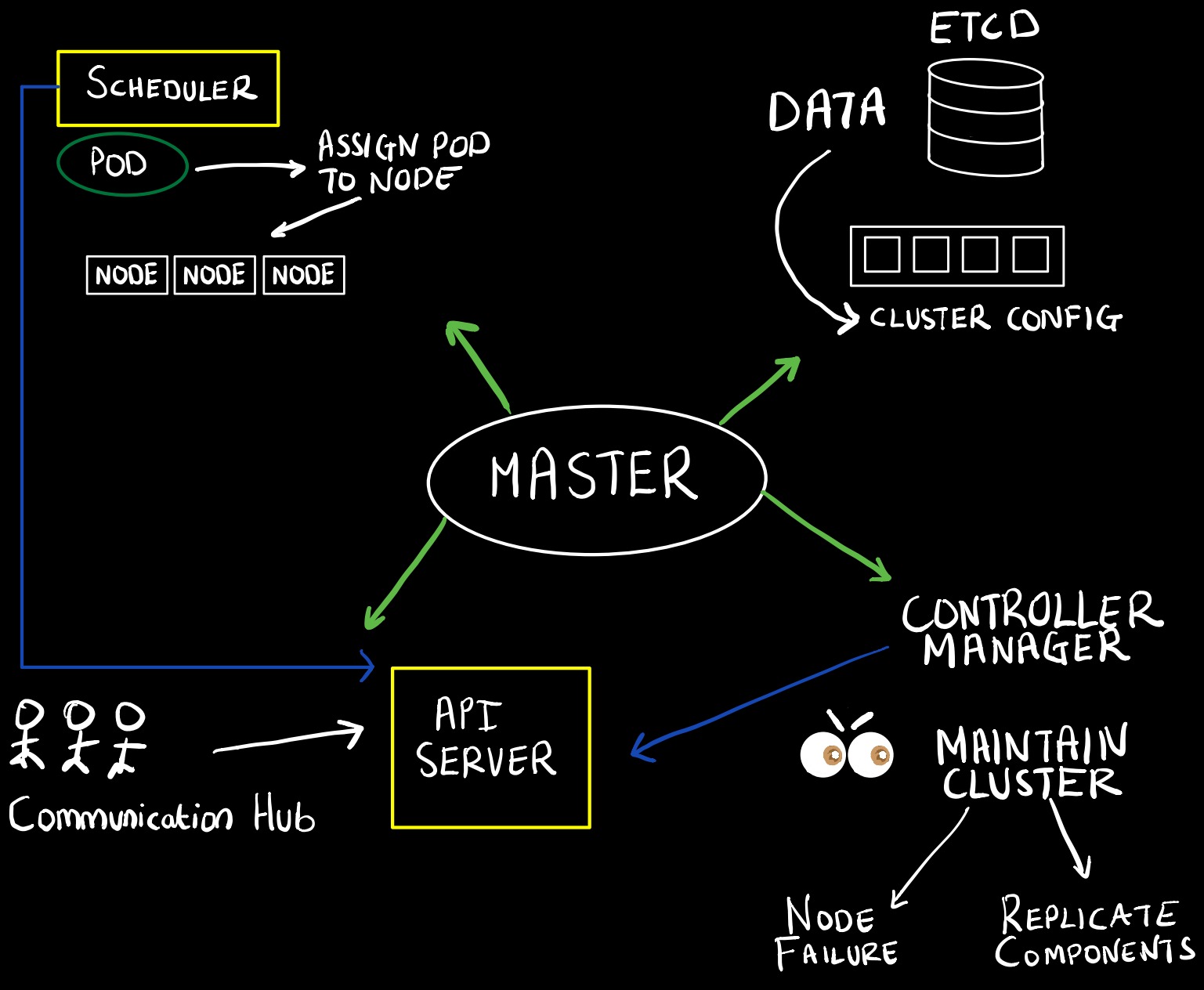
Kubernetes Administrator

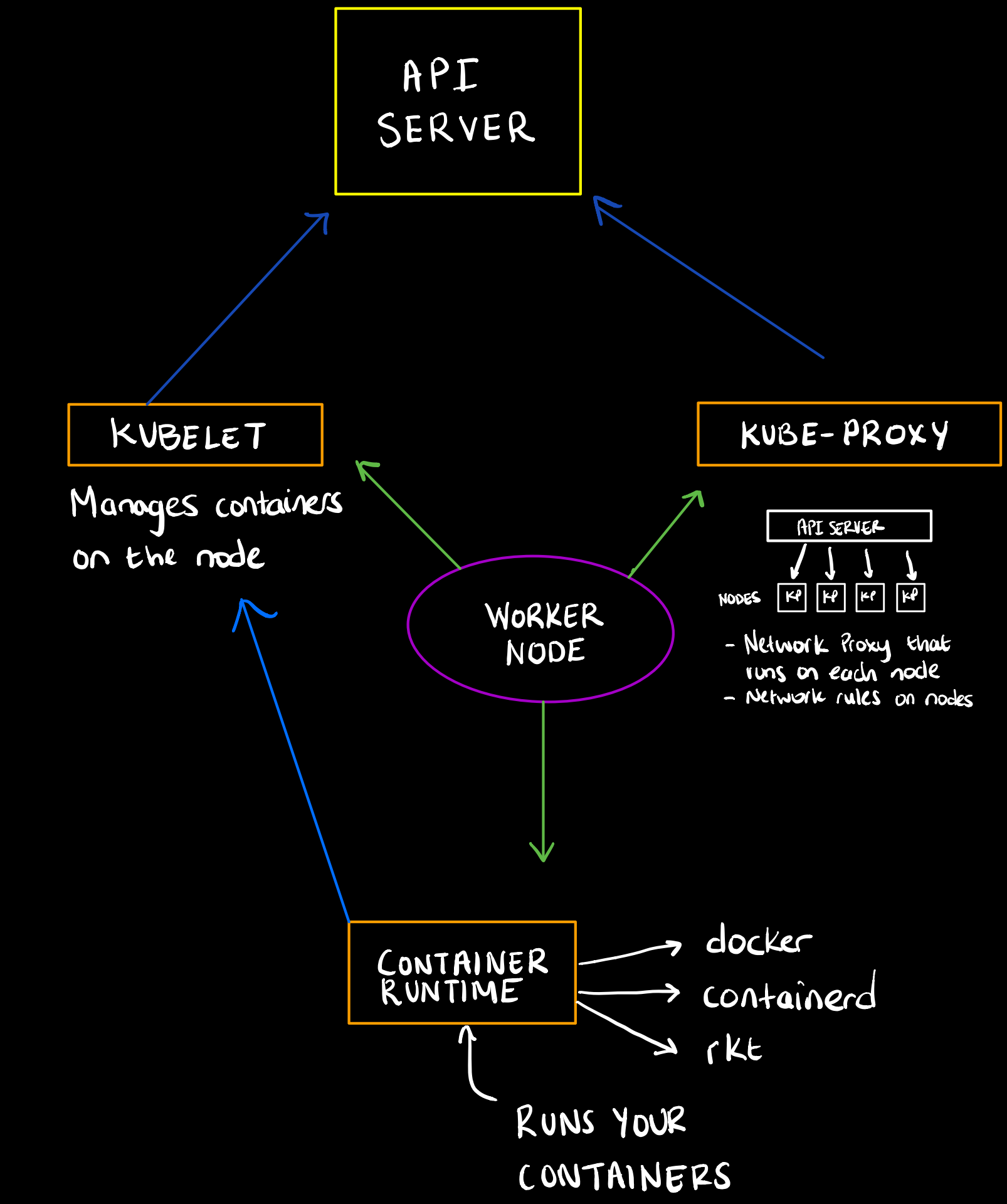
Adnan Rashid

instagram.com/
adnans_techie_studies

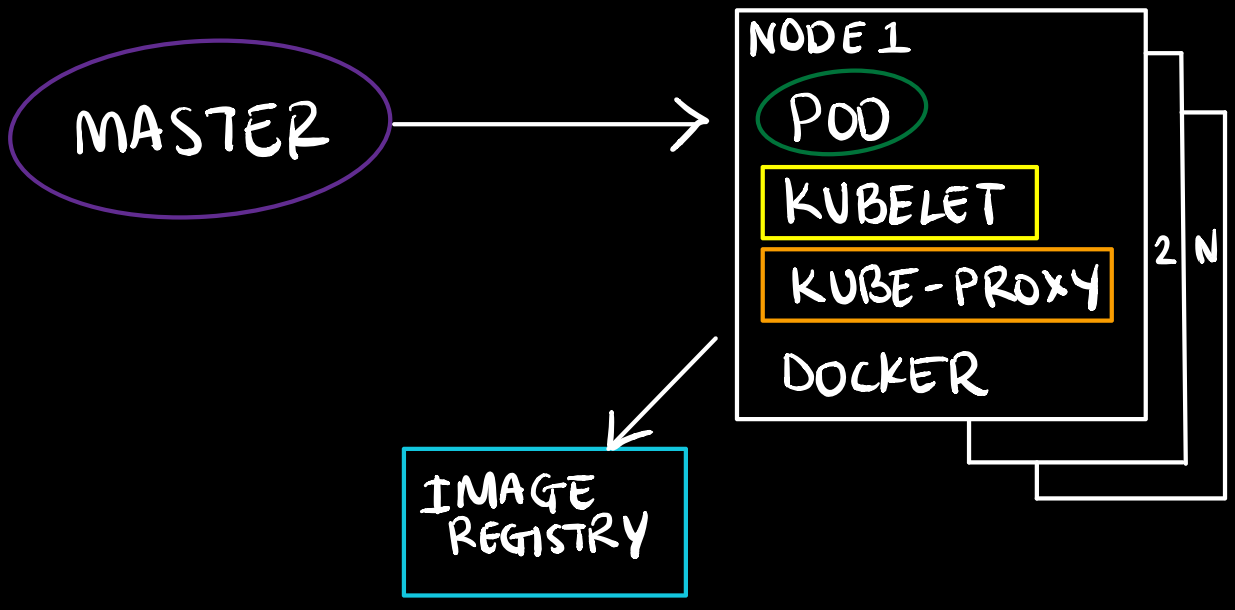
UNDERSTANDING THE KUBERNETES ARCHITECTURE

Cluster Architecture



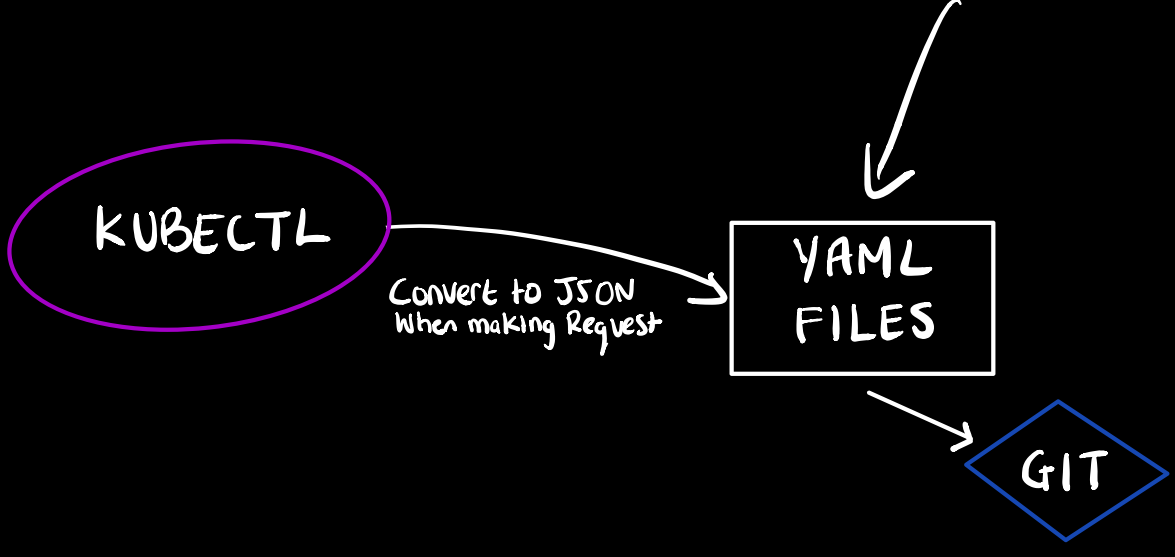

API SERVER
KUBELET
Manages containers on the node
KUBE-PROXY
API SERVER
NODES
KP
KP
KP
KP
- Network Proxy that runs on each node
- Network rules on nodes
WORKER NODE
CONTAINER RUNTIME
docker
containerd
rkt
RUNS YOUR CONTAINERS

Application Running on K8S



API Primitives

API server is the only one that communicates with etcd
Every single component speak with API server only not to each other
Objects like pods and services are declarative intents



YAML File Composition

API VERSION → clear consistent view of resources

KIND → The kind of object you want to create
- → Pod
- → Deployment
- → Job

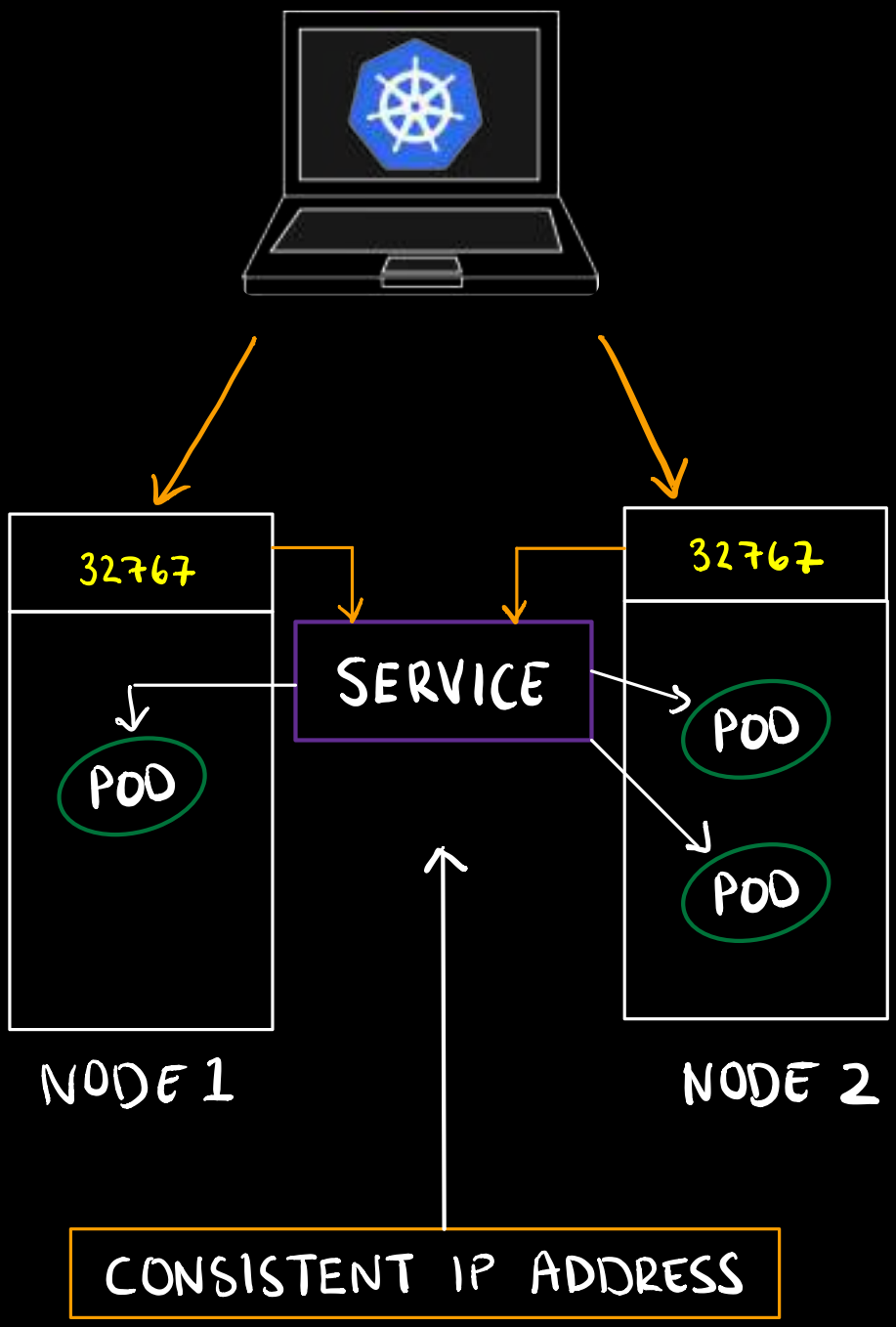
METADATA → uniquely identify the object
- Name String
- UID
- namespace

SPEC → container image volume exposed ports

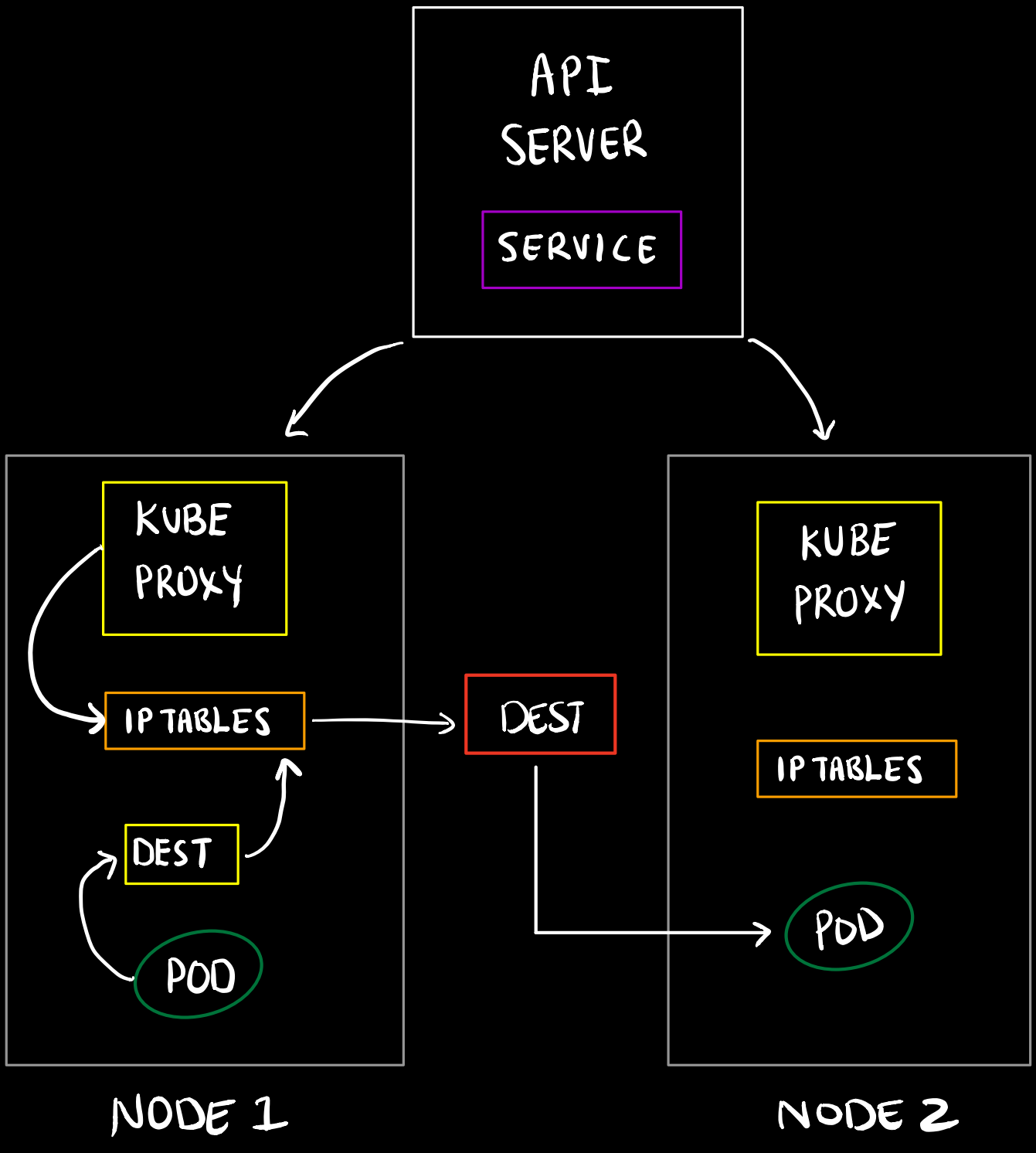
STATUS → State of the object
- ↓ match desired states

Services and Network Primitives

Services allow you to dynamically access a group of replica pods



KUBE-PROXY



KUBE-PROXY HANDLES THE TRAFFIC ASSOCIATED WITH A SERVICE BY CREATING IP TABLE RULES

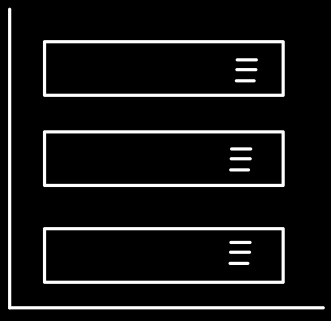
BUILDING THE KUBERNETES CLUSTER

Release Binaries, Provisioning & Types of Clusters

Picking the right Solution

CLOUD

VS



on-prem

ALSO

CUSTOM

- → Install Manually
- → Configure your own network Fabric
- → Locate the release binaries
- → Build your own Images
- → Secure cluster comms

VS

Pre-Built

- → Minikube
- → Minishift
- → MicroK8s
- → Ubuntu on LXD
- → AWS, Azure, GCP

✓ using minikube locally on MacOS

Installing Kube Master And Nodes

MASTER + WORKERS

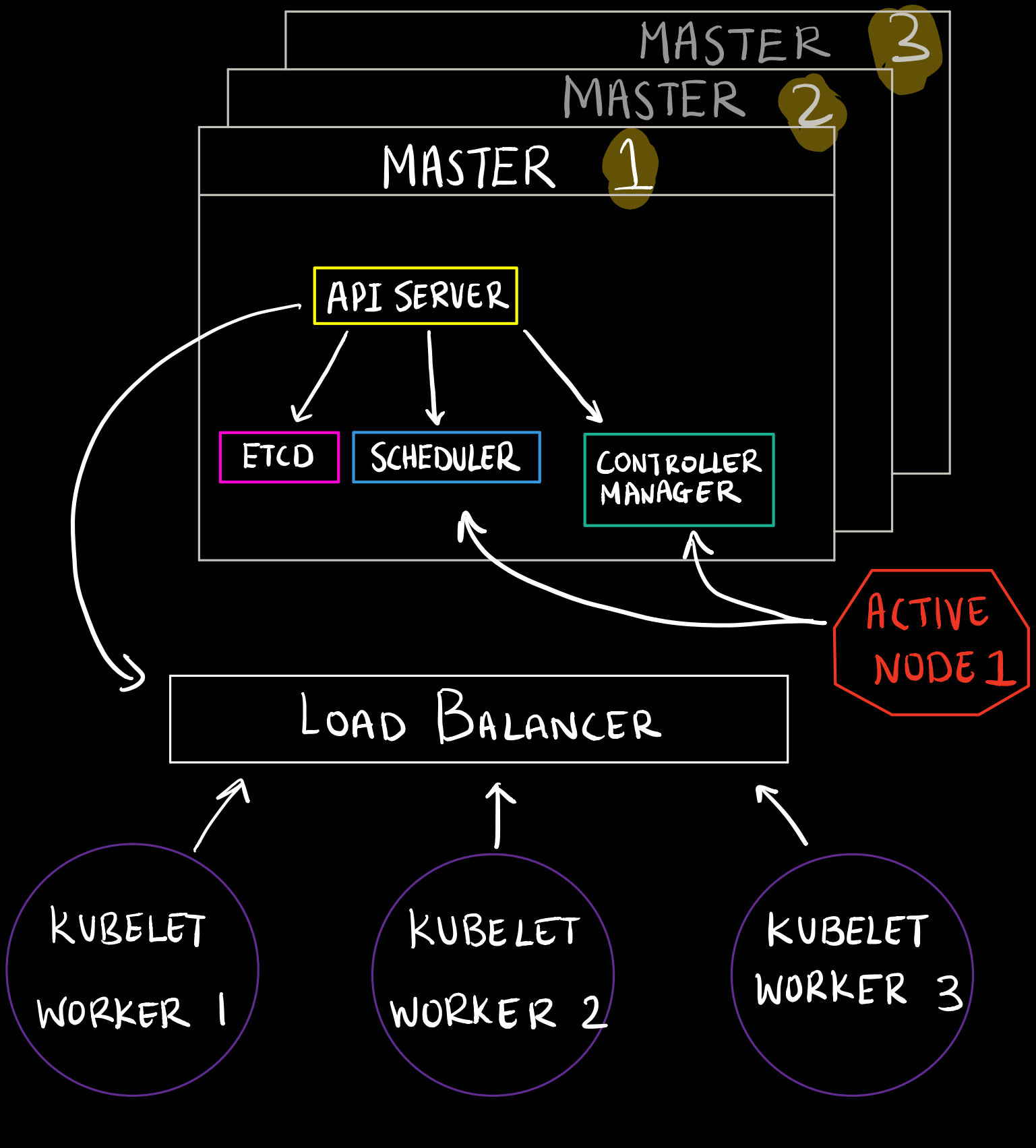
1. Docker + Kube → GPG KEY
   → ADD REPOS
2. UPDATE PACKAGES
3. Install Docker, kubelet, kubeadm, kubectl
4. Modify bridge adapter settings

MASTER ONLY

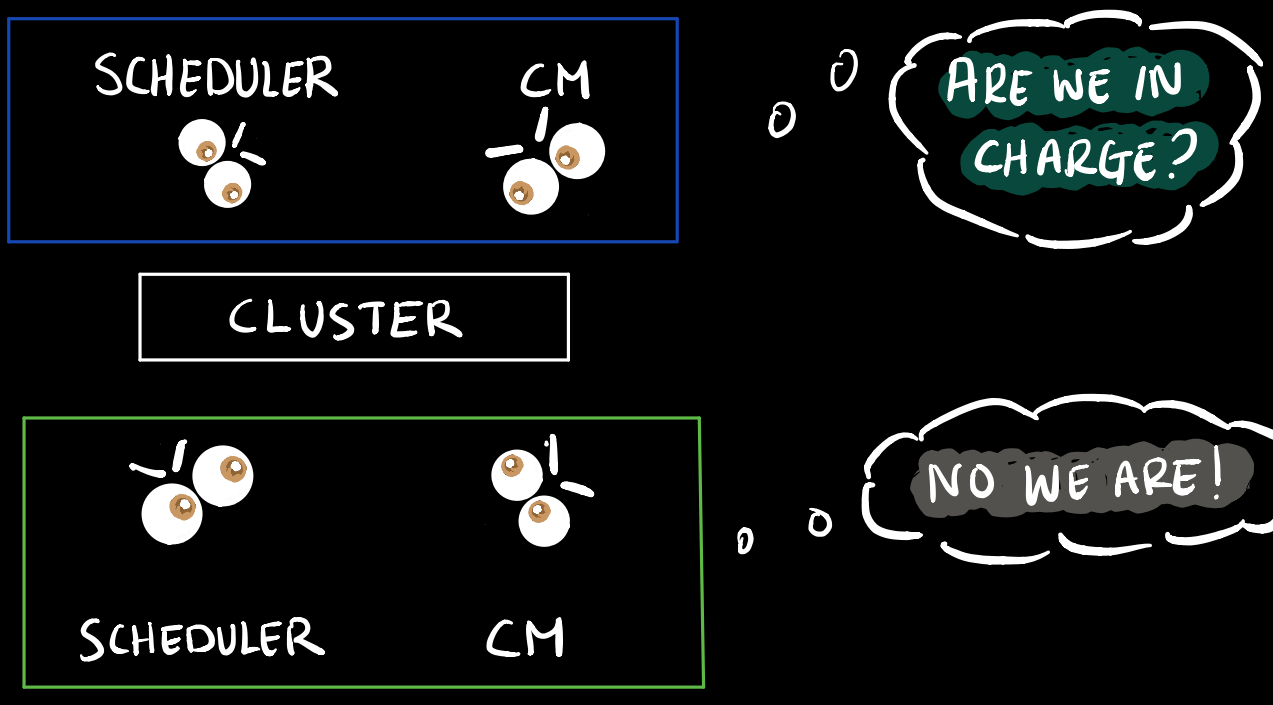
1. Initialise Cluster
2. Make directory for kube
3. Copy kube config
4. Change ownership of config
5. Apply Flannel CNI

Building Highly Available Cluster

All components can be replicated, but only certain can operate simultaneously



The controller manager and the scheduler actively watch the cluster state and take action when it changes



→ Leading to duplicate resources or corruption. → HOW DO WE DECIDE?

LEADER ELECT OPTION → creates endpoint resource
↑ see in scheduler YAML
↑ holderIdentity

Replicating etcd

TOPOLOGY

Stacked

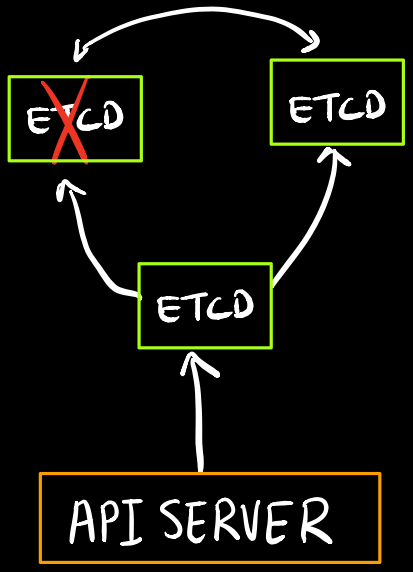
External to Kube Cluster

Each Control Plane node creates local

etcd member

only communicates

API SERVER



RAFT Consensus Algorithm

Requires majority

There must be more than half taking place in the state change

∴ there must be odd number of nodes

Configuring Secure Cluster Communications

→ All communication via Https

→ Kube CTL → Translator → Kubernetes API

Kubernetes API ← Provides CRUD

CRUD: create, read, update, delete

CRUD → Cluster State

Return Response

→ Kube CTL → CREATE A POD → HTTP POST → API SERVER → State Stored in ETCD

API SERVER → Multiple Plugins: [Authentication | Authorisation | Admission | Validation]

Admission → READ → Skip

Admission → create, modify, delete

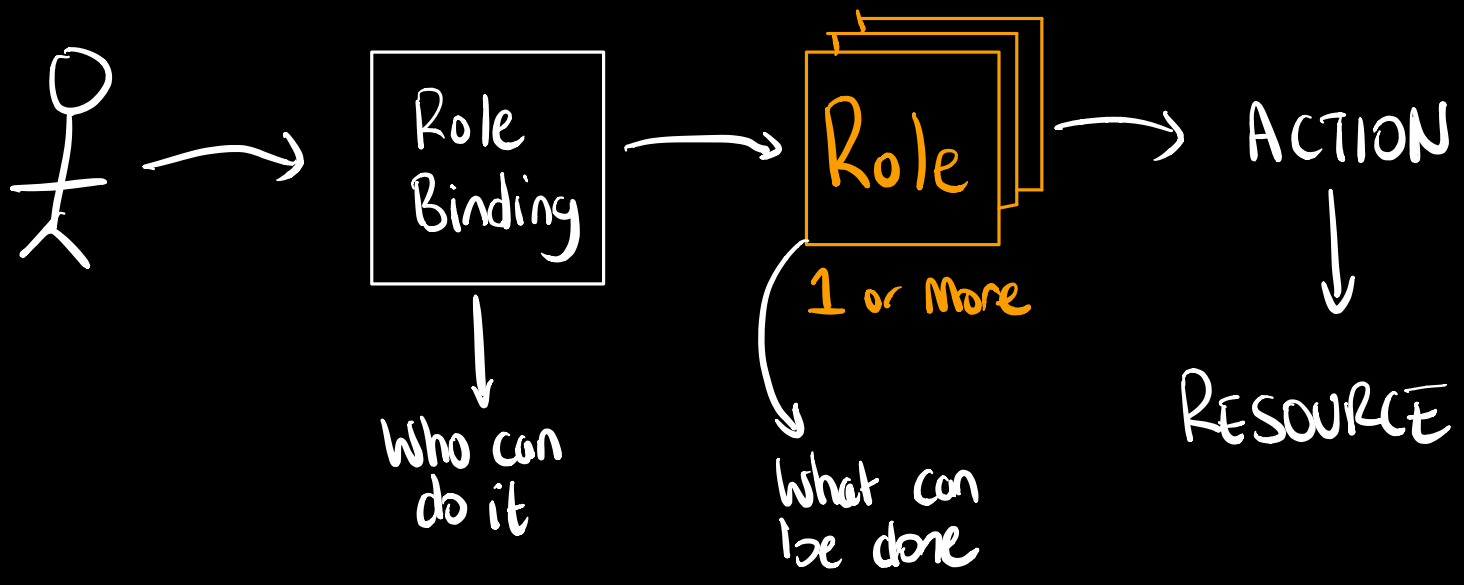
Authorisation → Can this User perform this action?

Calls to determine request

- HTTP HEADER
- CERTIFICATE

Roles And Access

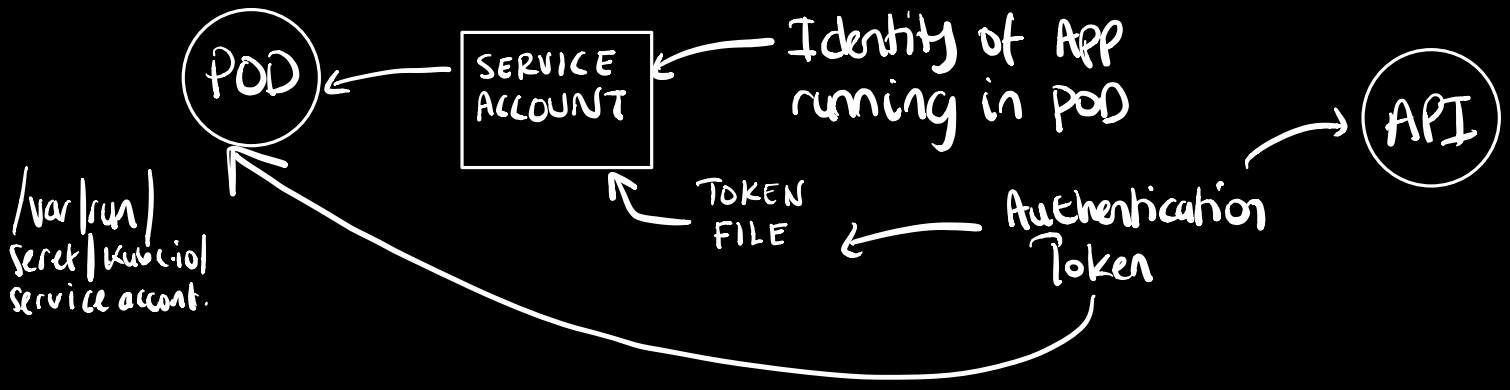
RBAC is used to prevent unauthorised users from modifying the cluster state

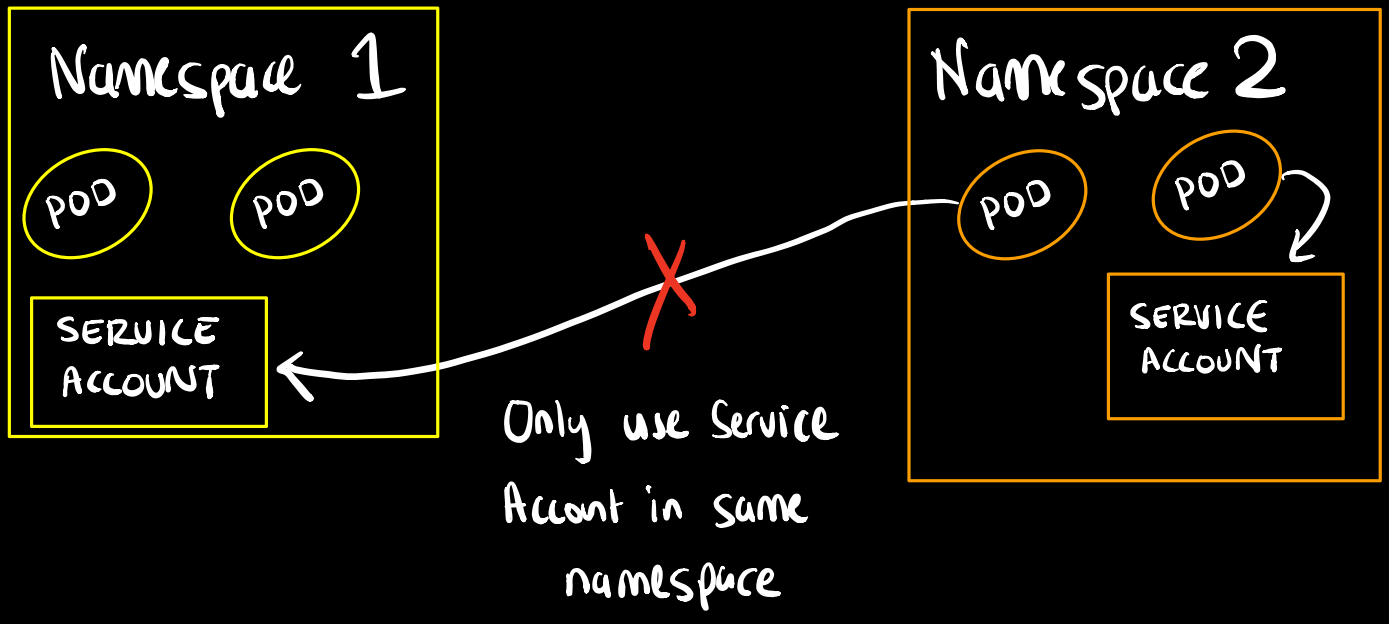


Role	Role Binding	→ Namespace Resource

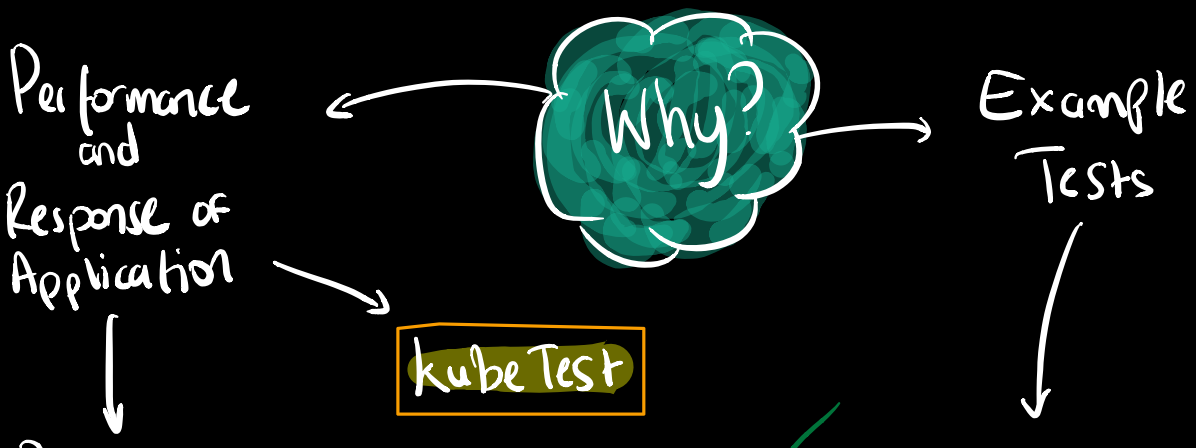
ClusterRole	ClusterRoleBinding	→ Cluster Level Resources

Service Account





Running End to End Tests On Cluster



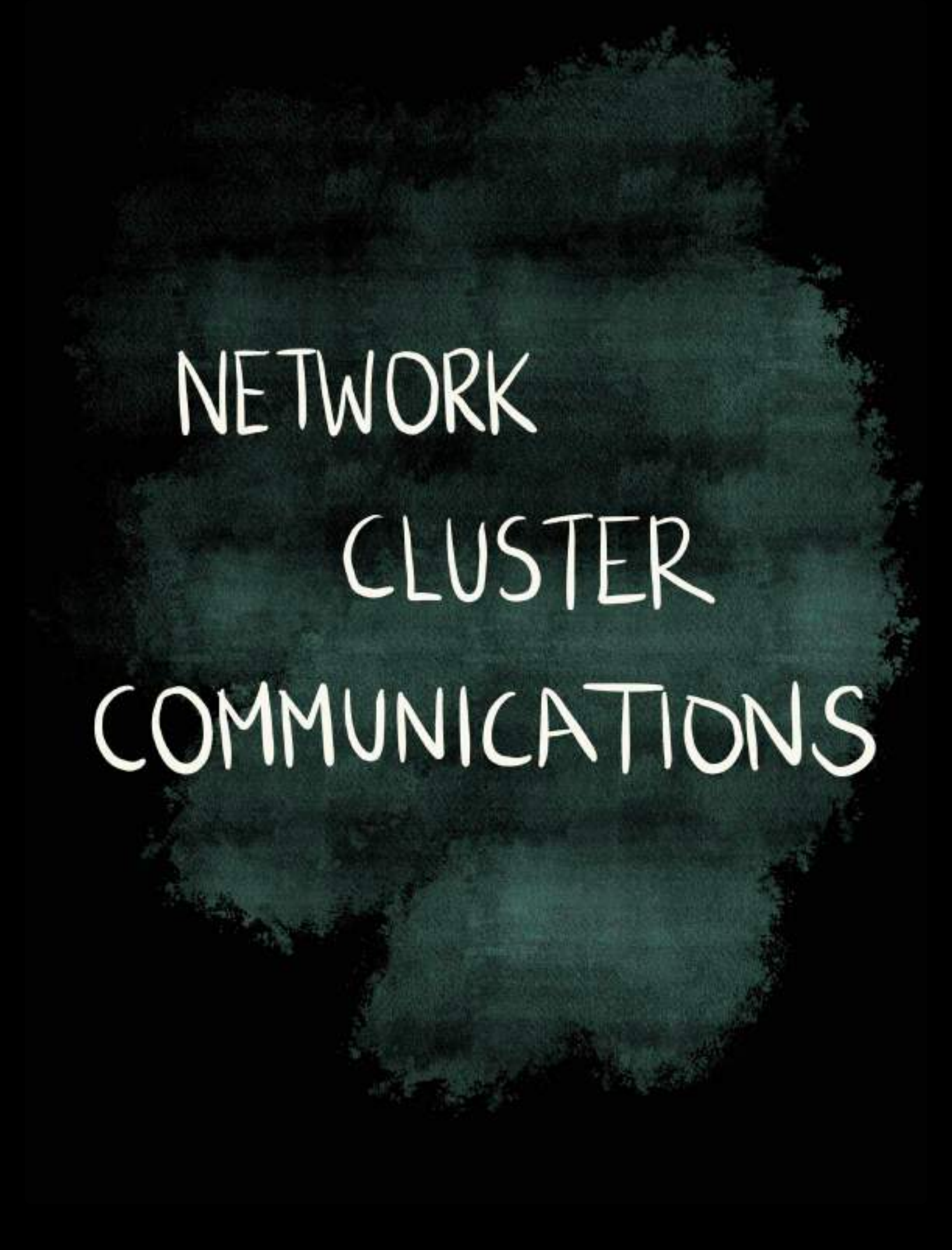
- ✓ Deployments can run
- ✓ Pods can run
- ✓ Pods can be directly accessed
- ✓ Logs can be collected
- ✓ Commands run from pod
- ✓ Services can provide access
- ✓ Nodes are healthy
- ✓ Pods are healtl

Managing Cluster

Upgrading Cluster
- Kubeadm
- Kubectl

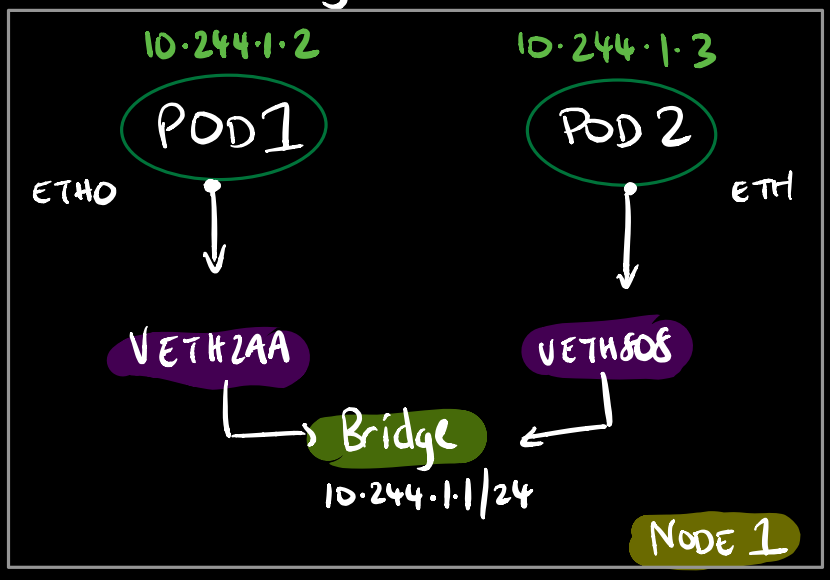
Operating System Upgrades
- Use Deployments or replica sets
- Get pods
  Drain Node
  Uncordon
  [Put back to Service

Back-up and Restore Cluster
- cluster state → etcd
  - Save externally

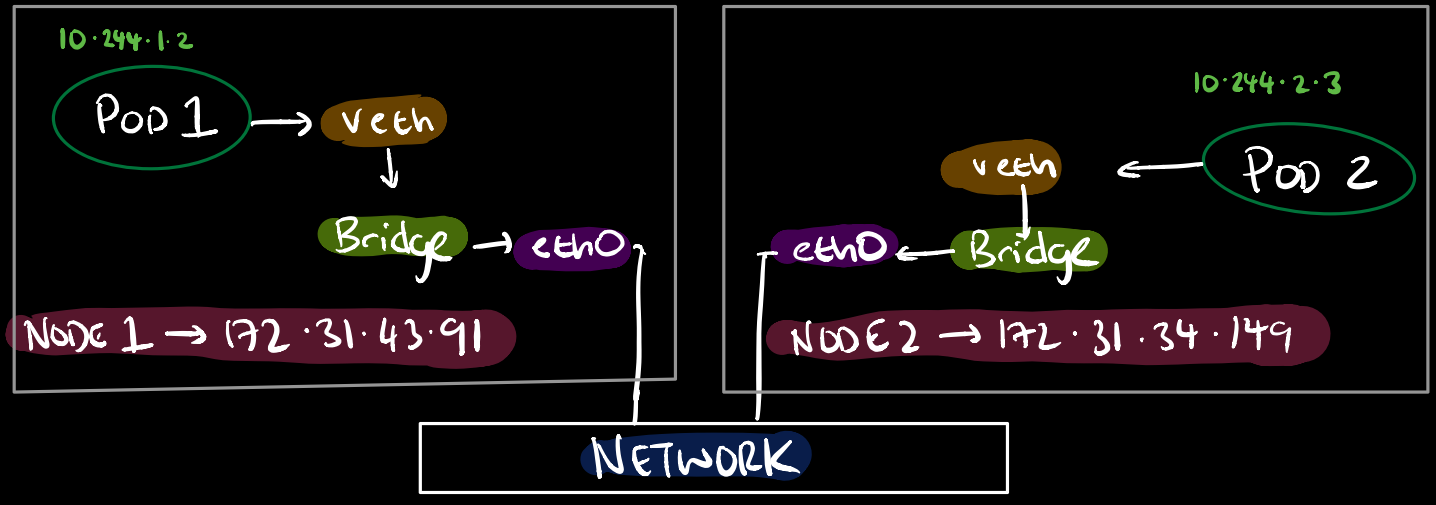

NETWORK
CLUSTER
COMMUNICATIONS

Pod and Node Networking

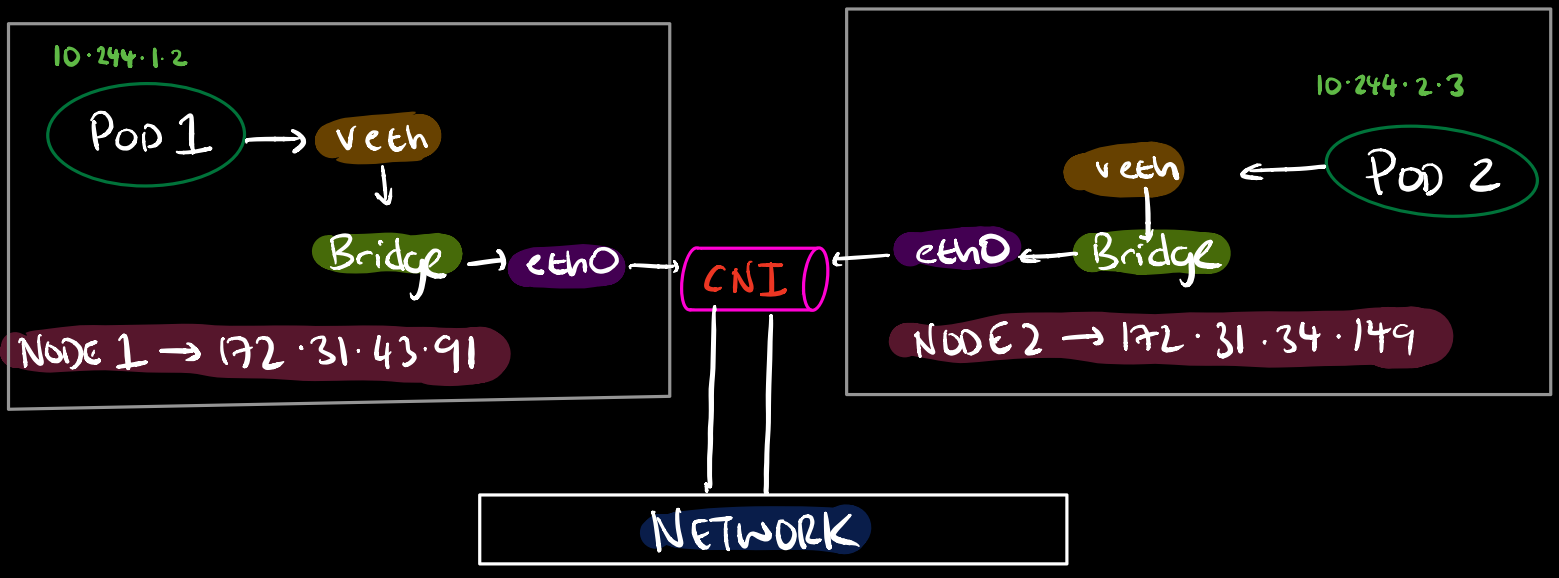
Networking within a node



Networking Outside of the Node



Container Network Interface



CNI is a network Overlay

↳ Allows building tunnel between nodes

↳ Sits on top of existing networks

↳ Encapsulates Packet

↳ changes SRC & DST

How CNI do this? → There is a mapping associated in user space

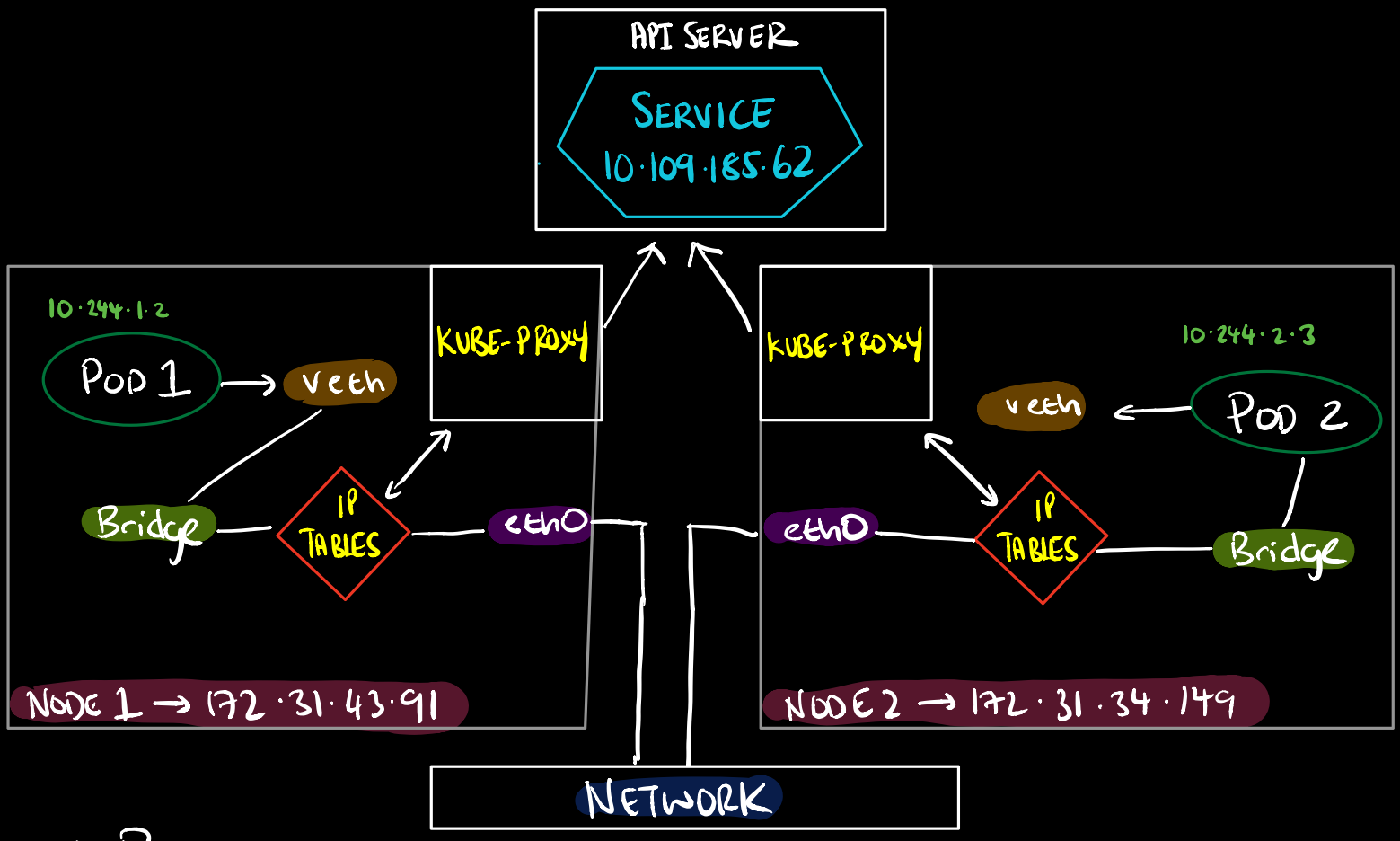
↳ Program all pod ip addresses to node IP, when reach other node, de-encapsulate packet and give to bridge

Example CNI

- Calico
- Flannel

Appears Local to Node!

Service Networking



→ Pods come and go.

↳ How does cluster keep track?

↳ Services!

↳ Provides virtual interface → Auto assigned to pods behind interface.

Example — Cluster IP Services → Auto created on cluster creation

↳ Takes care of internal routing → No matter where moves other pods know how to communicate to it

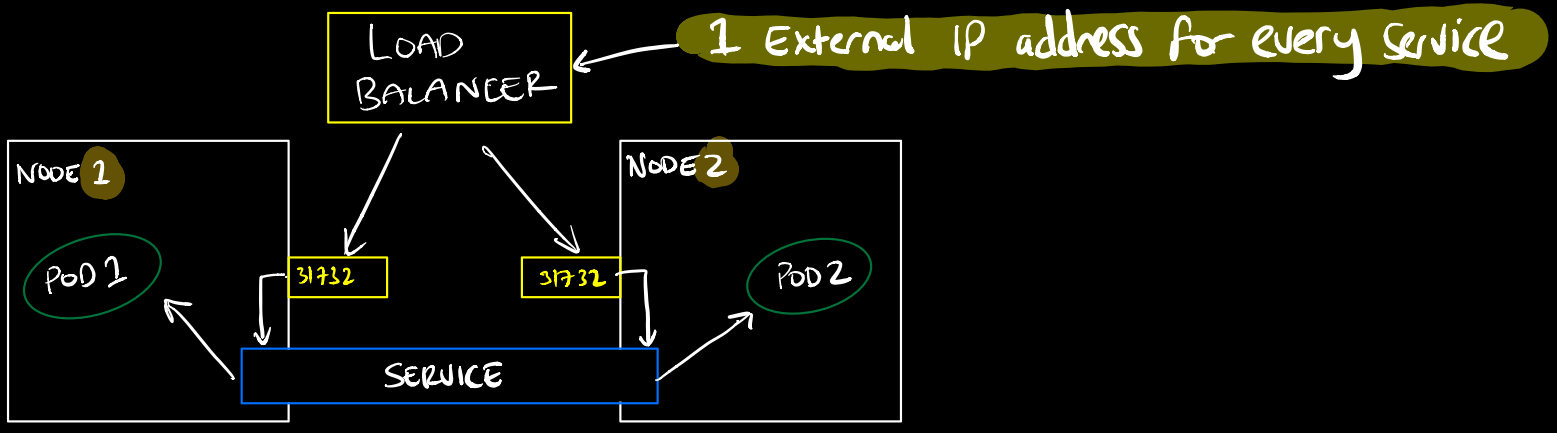
Ingress Rules and Load Balancers

Load Balancer

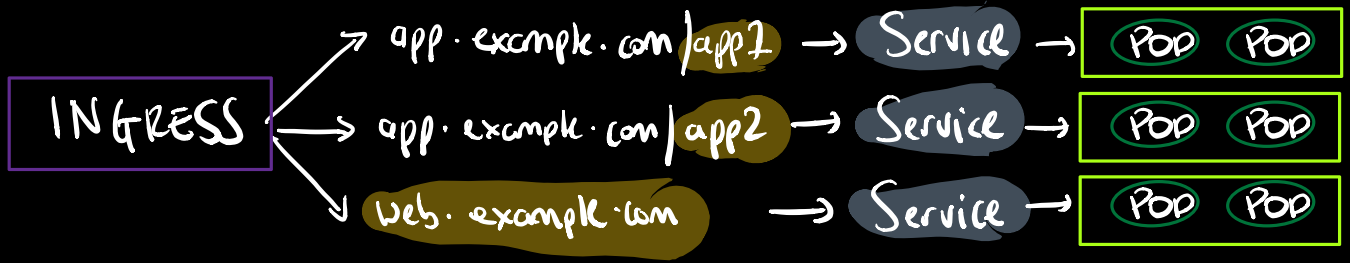
- Redirect traffic to all nodes + ports
- Clients talk to LB to access application

Service → Talks to 1 node at a time
- ↳ cannot split traffic like a load balancer

Only accessible internally → does not have external IP



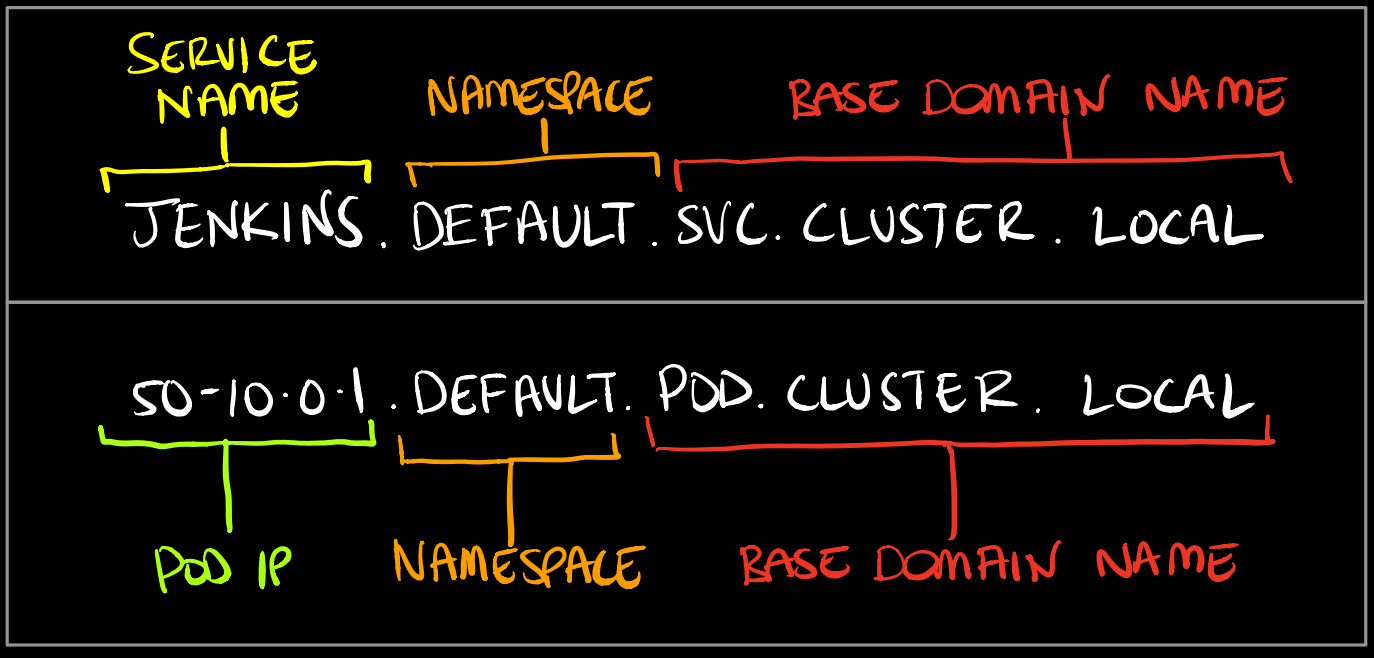
Ingress



Access multiple Services with Single IP Address

Cluster Dns

Every service defined in the cluster is assigned a DNS name



A pods DNS search will include the pods own namespace and the clusters default domain

POD SCHEDULING WITHIN THE KUBERNETES CLUSTER

Configuring the Kubernetes Scheduler

Scheduler responsible for assigning pod to node based on resource requirements of the pod

RULES ARE PLACED BY DEFAULT

HOWEVER, CAN CREATE OWN

WHY??

SAME NODE TO SAVE MONEY

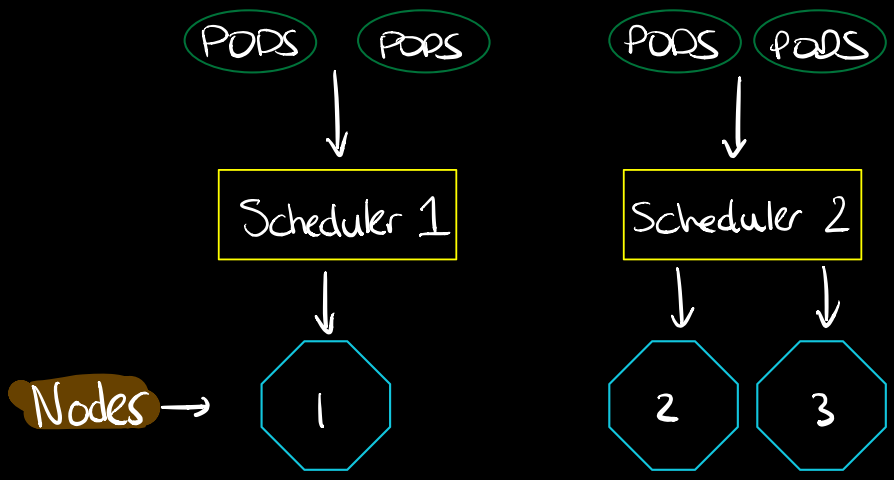
WORKER NODES HAVE DIFFERENT DISKS

SCHEDULER →

1. DOES THE NODE HAVE ADEQUATE HARDWARE RESOURCES?
2. IS THE NODE RUNNING OUT OF RESOURCES
3. DOES THE POD REQUEST A SPECIFIC NODE?
4. DOES THE NODE HAVE A MATCHING LABEL?
5. IF POD REQUESTS A PORT, IS IT AVAILABLE?
6. IF POD REQUESTS A VOLUME, CAN IT BE MOUNTED?
7. DOES THE POD TOLERATE THE TAINTS OF THE NODE?
8. DOES THE POD SPECIFY NODE OR POD AFFINITY?

Running Multiple Schedulers for Multiple Pods

It is possible to have 2 schedulers working alongside each other.



Sheduling Pods with Limits and Label Selectors

Taints → Repel Work → Example Master Node No Schedual

Tolerations → Allow you to tolerate a taint → Example kube-proxy ← Daemon set pod must run on all nodes

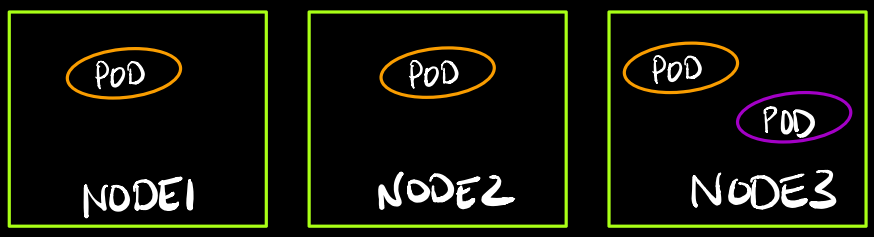
CPU/memory
- Sheduler looks at the sum of resources requested by existing pods
- Pod may not be using all requested resource at a given time.

Scheduler
Post 8 steps
- Most requested priority
- Least requested priority

Why? ← Cloud environments you are paying for all resources

DaemonSets

DaemonSets ensure that a single replica of a pod is running on each node at all times

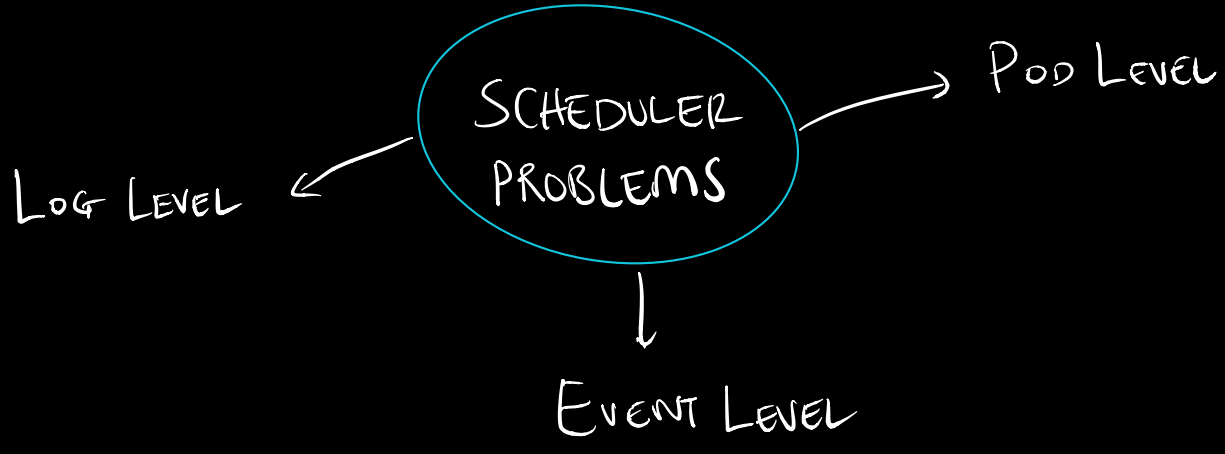


POD DAEMONSET POD

POD REPLICASET POD

If you try delete a daemonset pod, it will simply recreate it.

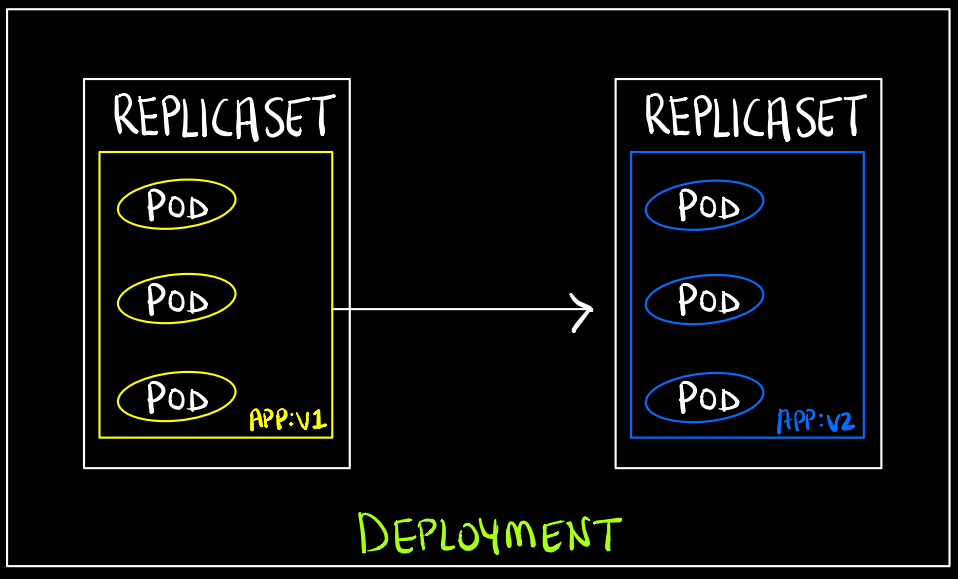
Display Scheduler Events



DEPLOYING APPLICATIONS IN THE KUBERNETES CLUSTER

Deploying an Application, Rolling Updates, and Rollbacks

Deployments → High level resource for deploying and updating apps



kubectl apply → mooify objects to existing yaml & if deployment not created → also create

kubectl replace → replaces old with new and object must exist.

Rolling update → preferred way → service not interrupted → fastest way

kubectl rollout → roll back previous version

Configuring an App for HA and Scale

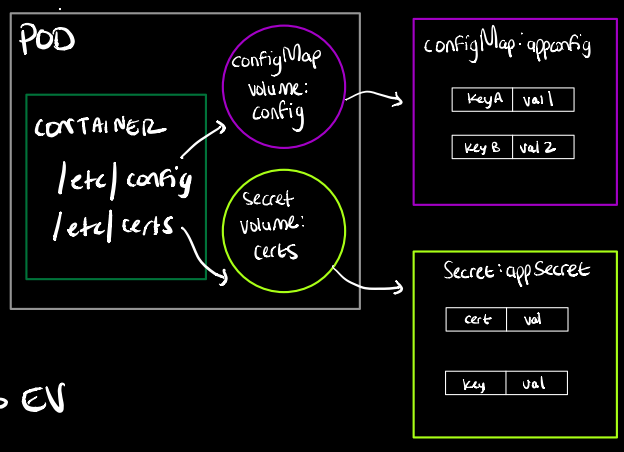
Avoiding bad versions → Block bad version release

- Min ready seconds → How long a newly created pod should be ready before considered available
- \+
- Readiness probe → Determines if a specific pod should recieve client request or not.

Deployment v1 → Min ready seconds → readiness probe → Pod Errors at 5 seconds → Pod NOT RELEASED

- Min ready seconds: 10
- readiness probe: must return success
  - check every 1 second
  - port 80
- Pod NOT RELEASED → Roll back version

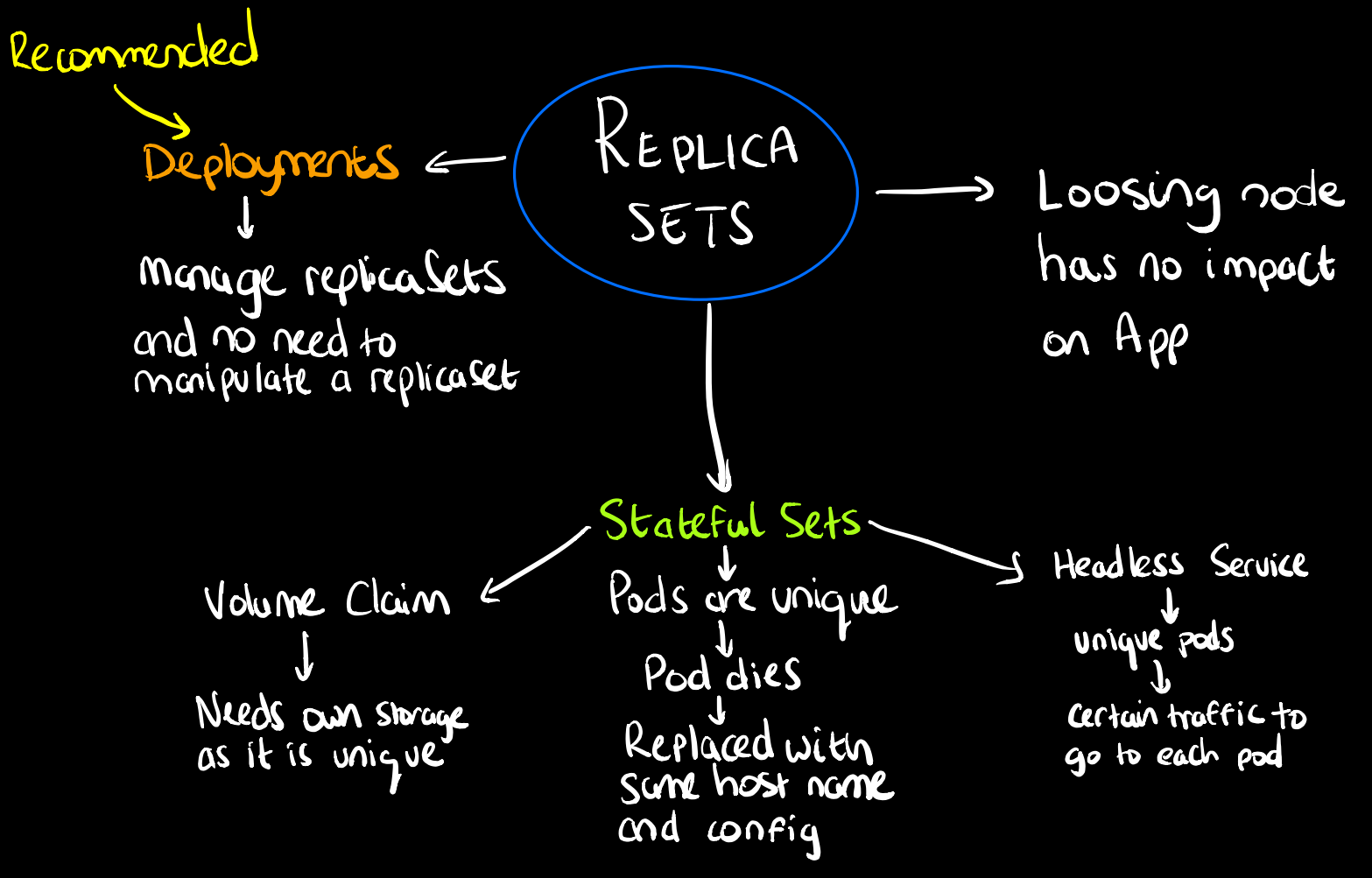
Passing configuration options to app



- Environment variables
  - Store in config map
  - Create secret & pass to ev
    - Just update → no need to rebuild image
    - Multiple containers can use same

Creating a Self-Healing App

ReplicaSets ensure that a identically configured pods are running at the desired replica count

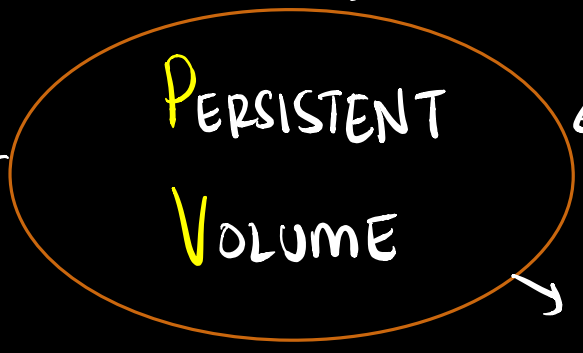


MANAGING DATA IN KUBERNETES CLUSTER

Persistent Volumes

→ Pods are ephemeral → Pod terminated ↓ Storage terminated

→ Storage must be independent → If pod moves ↓ Storage follows



← Storage Classes

↘ Resource in Cluster

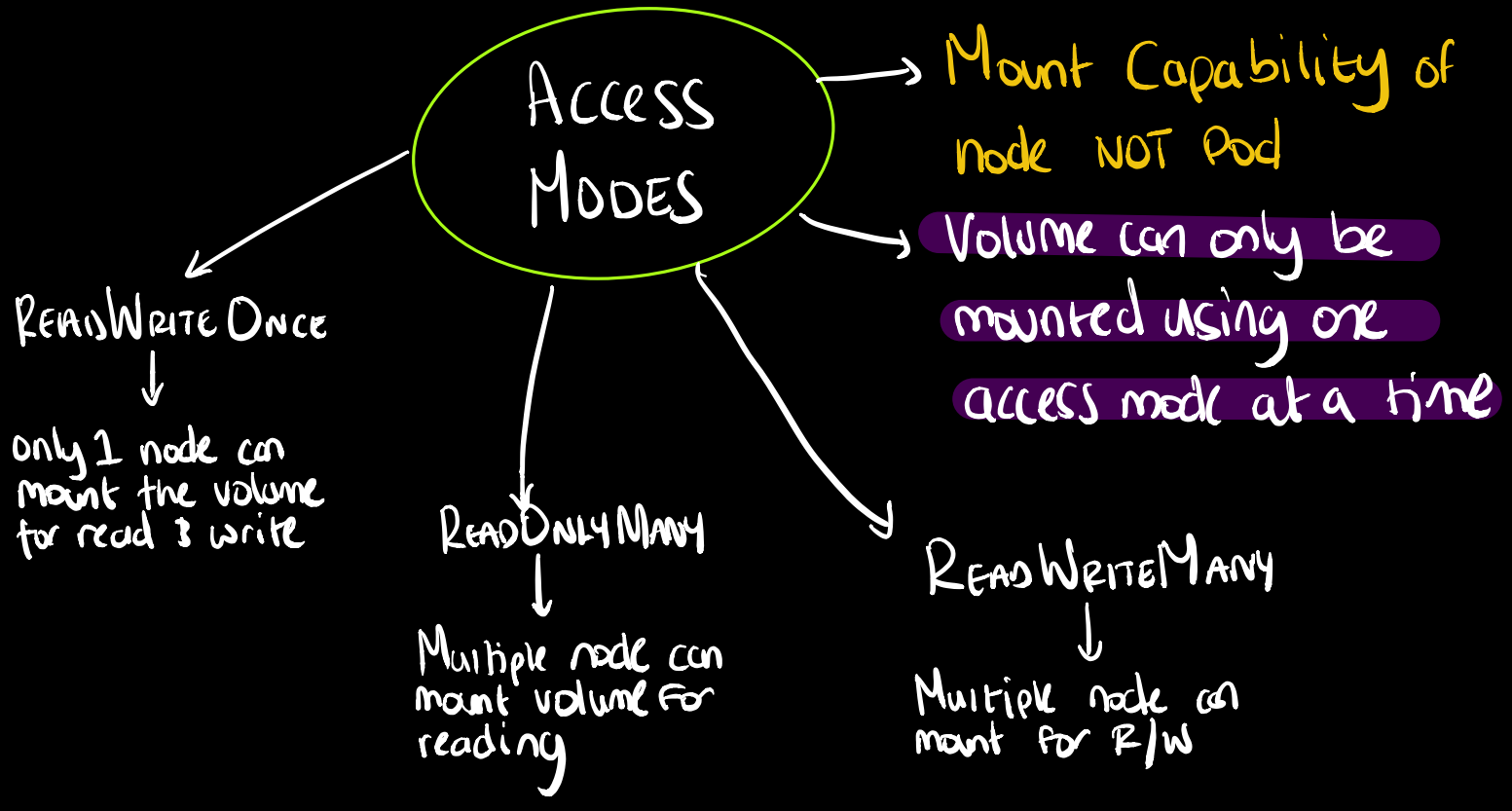
Provisioning

↓ Static ↓ Cluster admin creates and avail for consumption

↘ Dynamic ↳ PVC must request a storage class

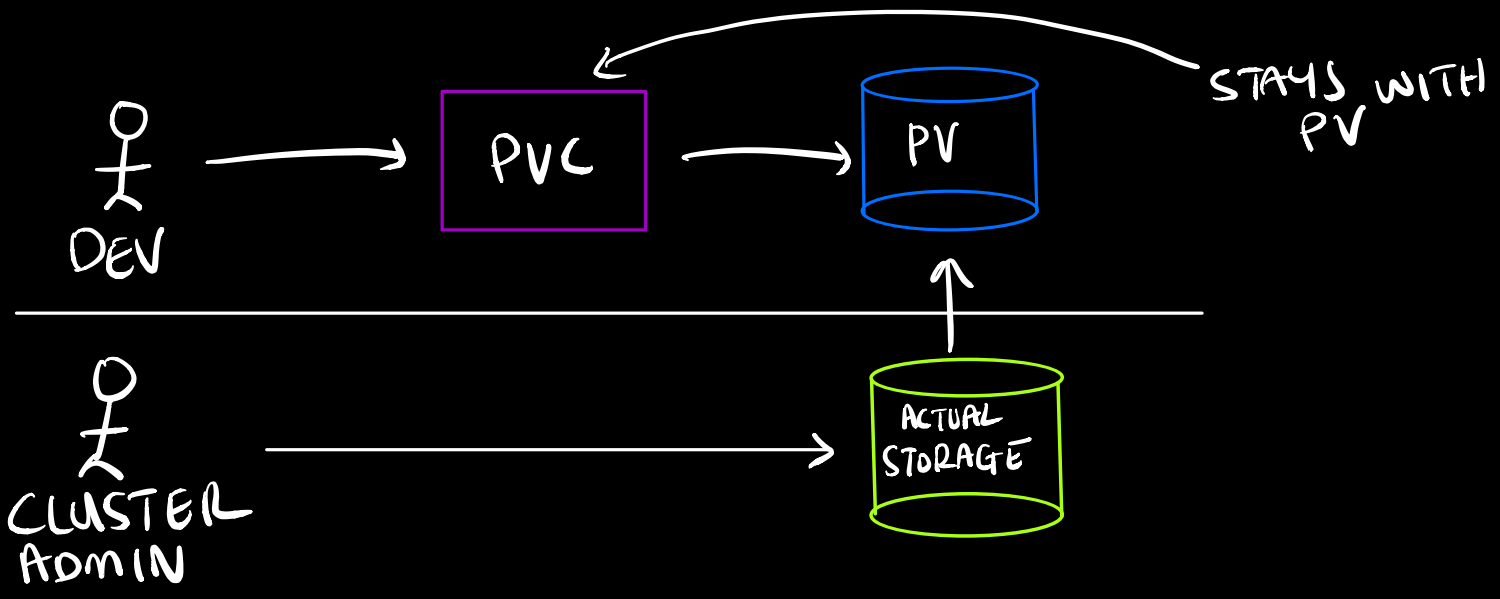
Volume Access Modes

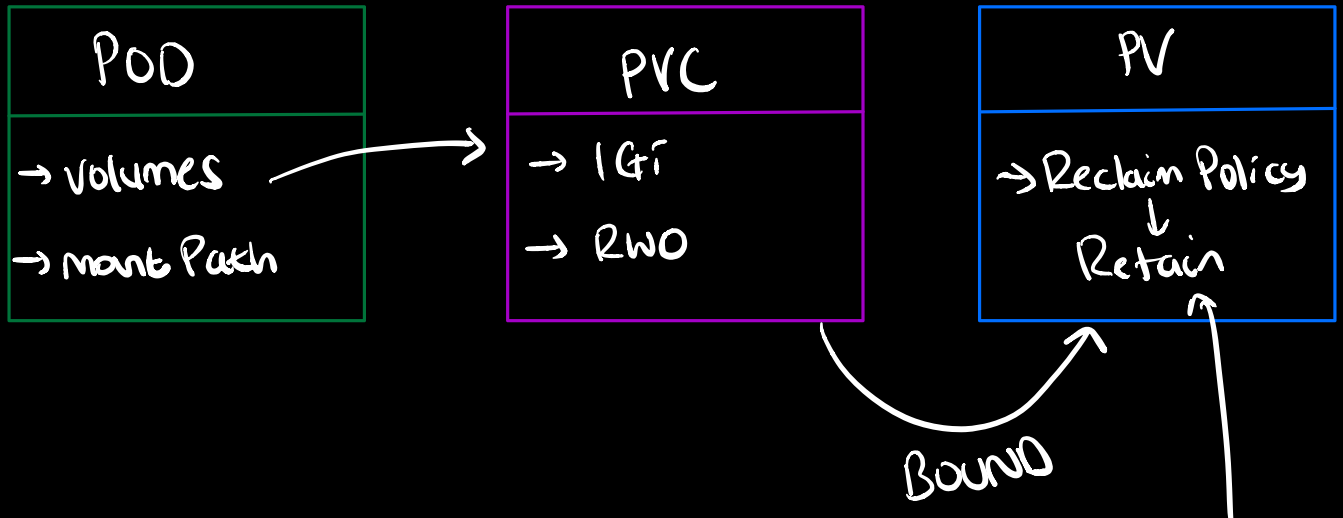
By specifying an access mode with your PV, you allow the volume to be mounted to one or many nodes, as well as read by one or many



Pesistent Volume Claims (PVC)

PVC allows the application developer to request storage for the application, without having to know underlying infra.





RETAIN DATA IN VOLUME

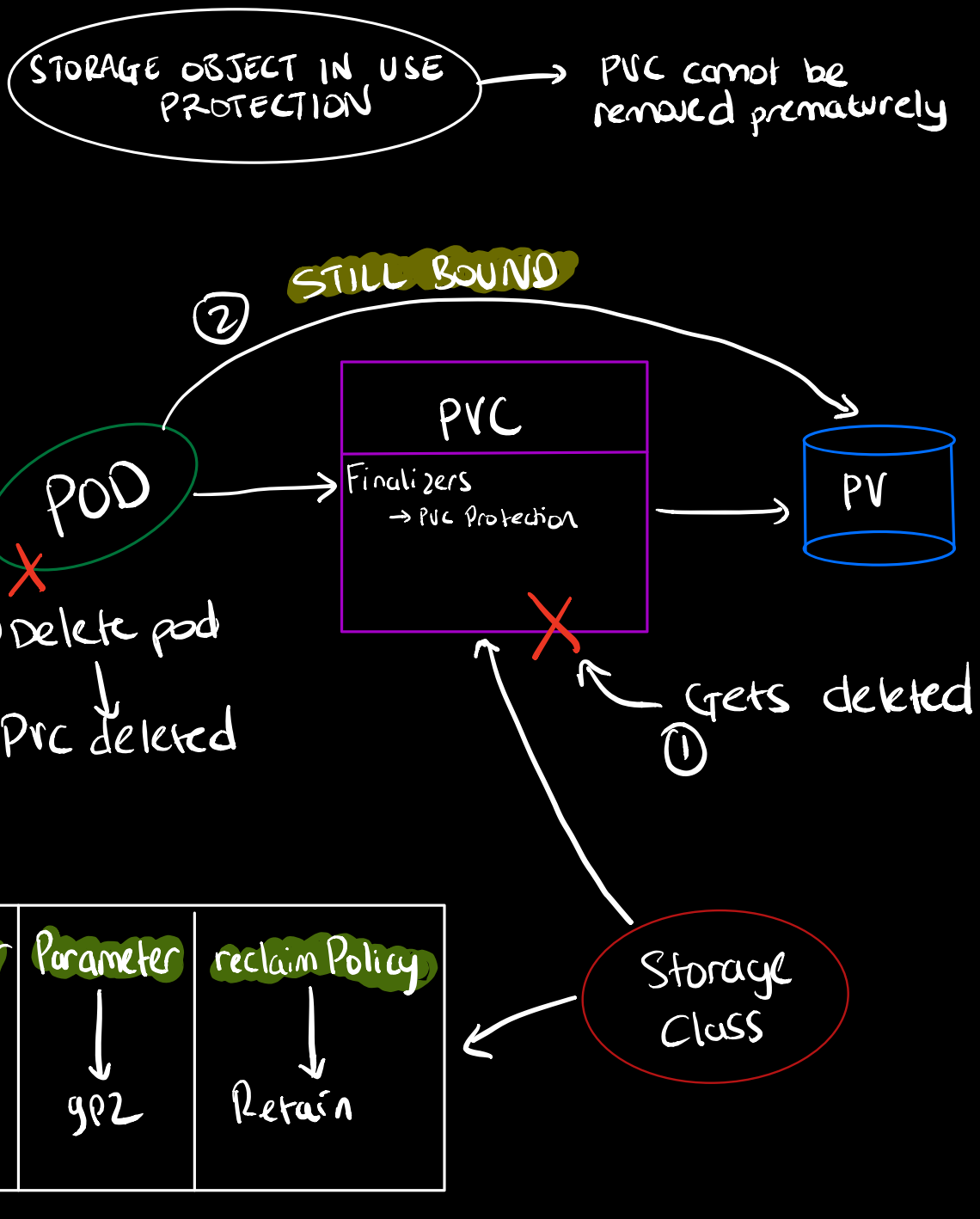
Could also be recycle or delete

deletes content of volume

delete underlying storage

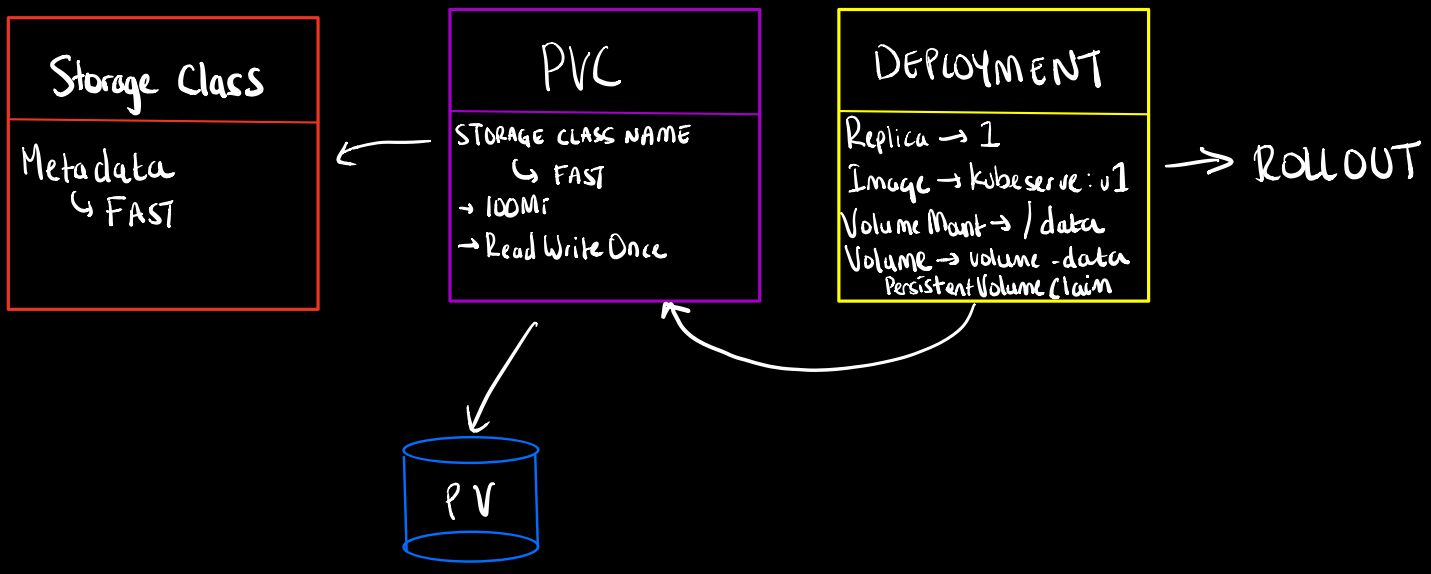
Storage Objects

Volumes that are already in use by a pod are protected against data loss. This means even if you delete a PVC, you can still access volume from POD.



Applications with Persistent Storage

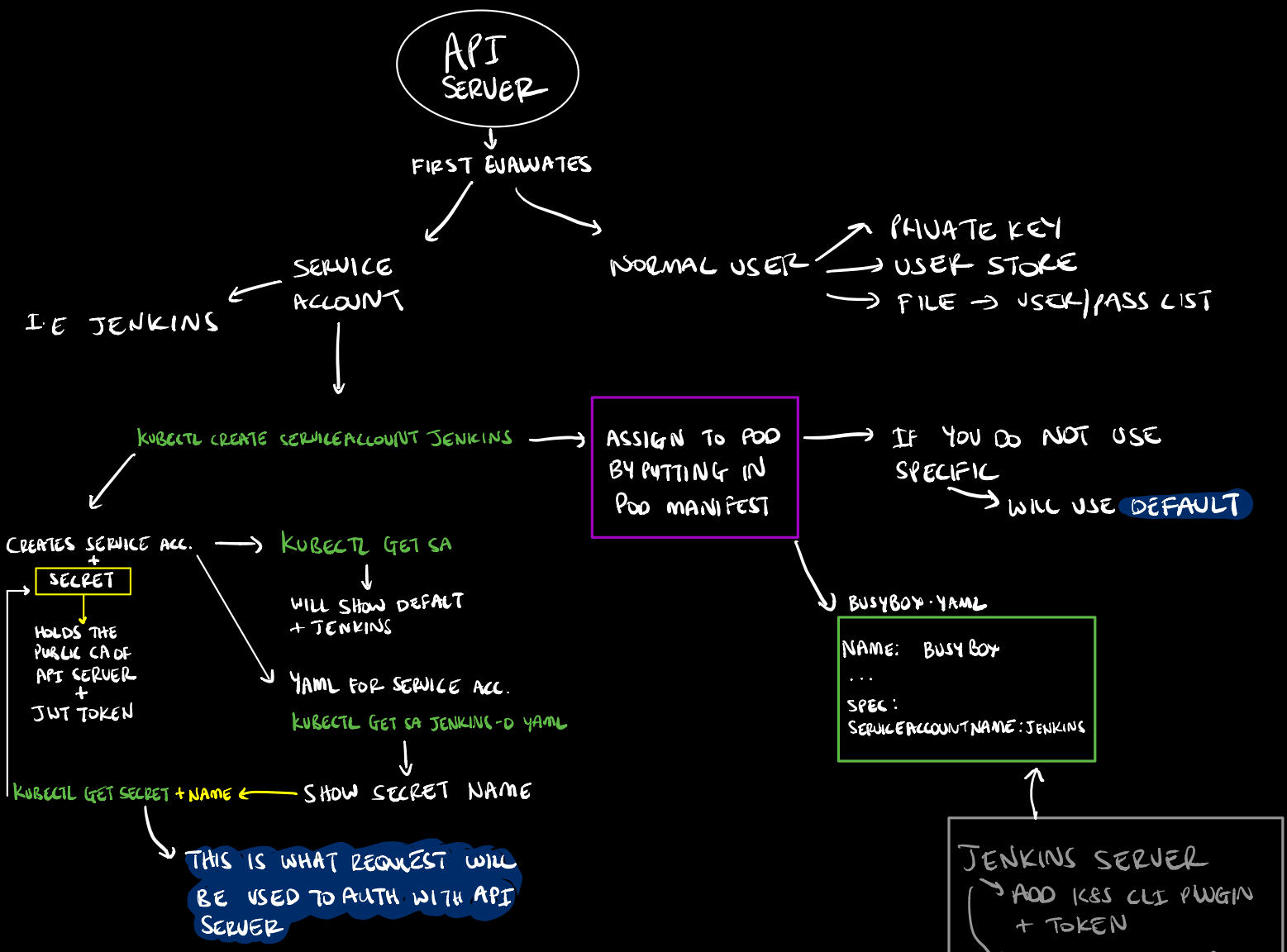
Example →



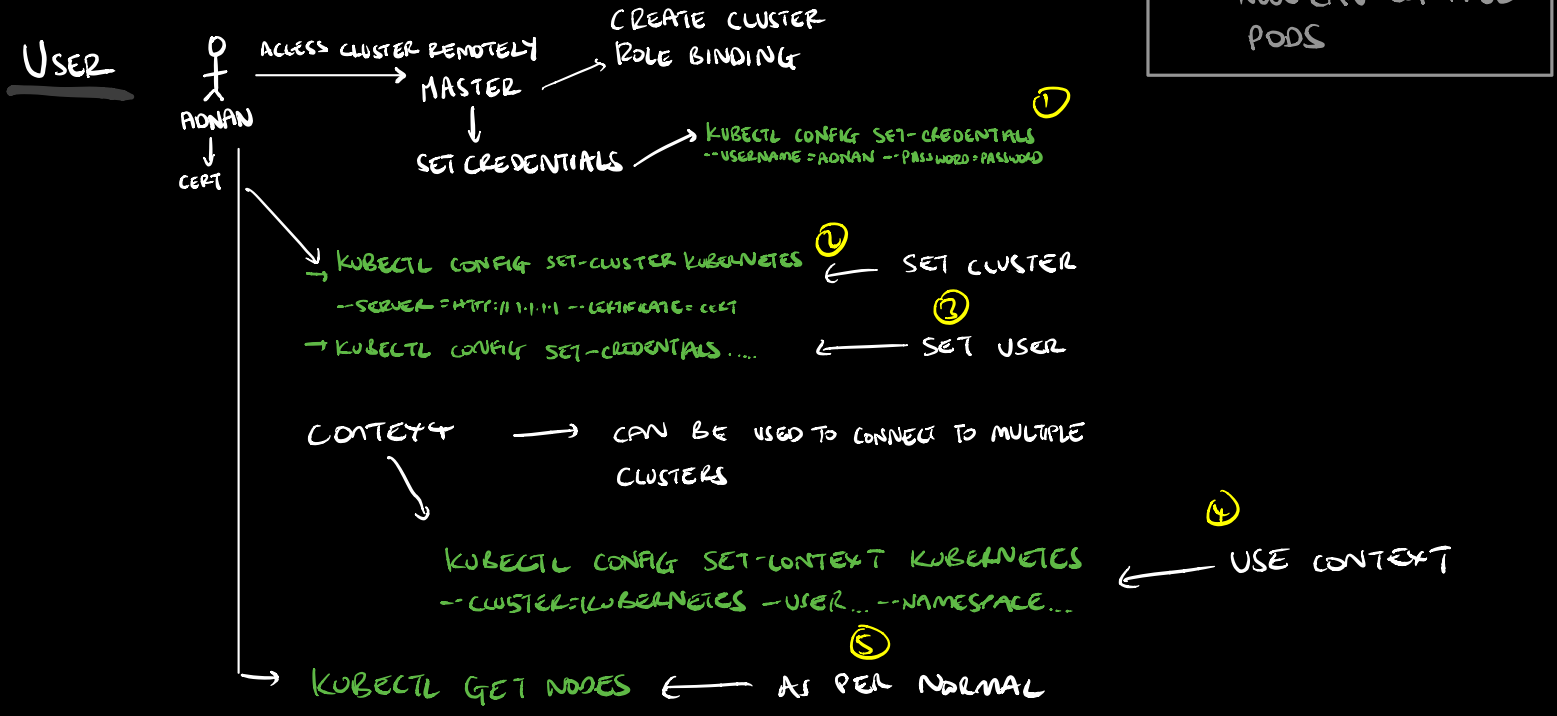
1. Create storage class object
2. Create PVC object
3. Create deployment
4. Rollout deployment
5. Check pods
6. Create file on mark
7. List contents.


SECURING
THE
KUBERNETES
CLUSTER

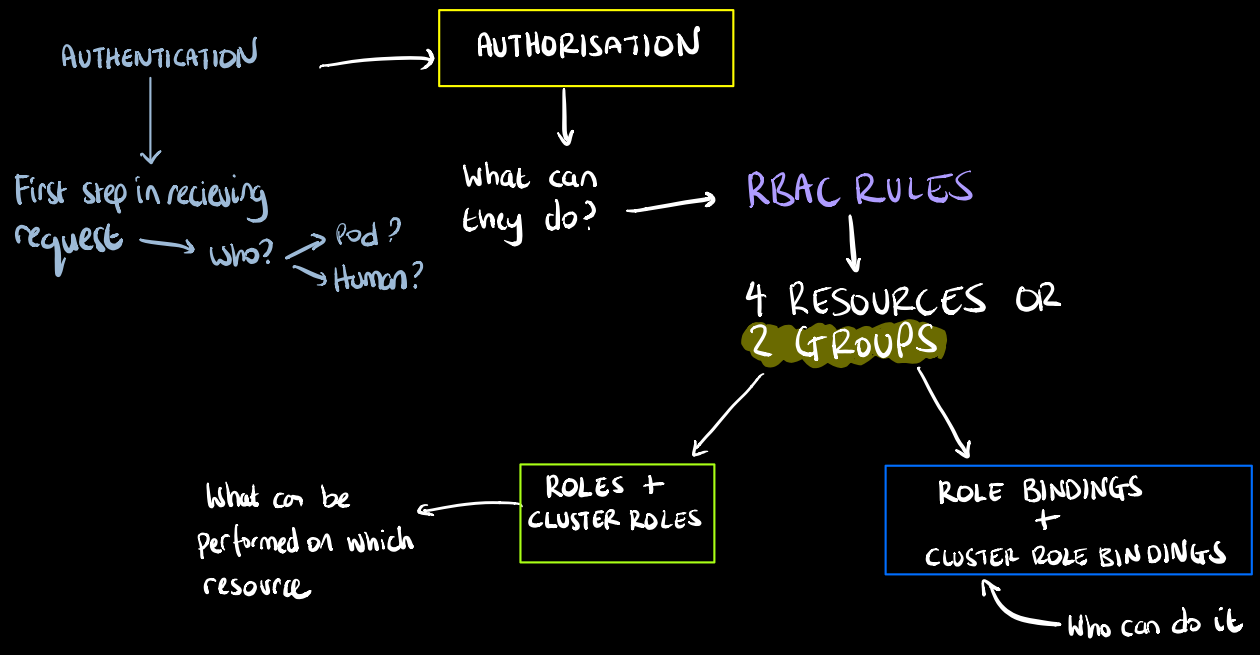
Service Accounts and Users

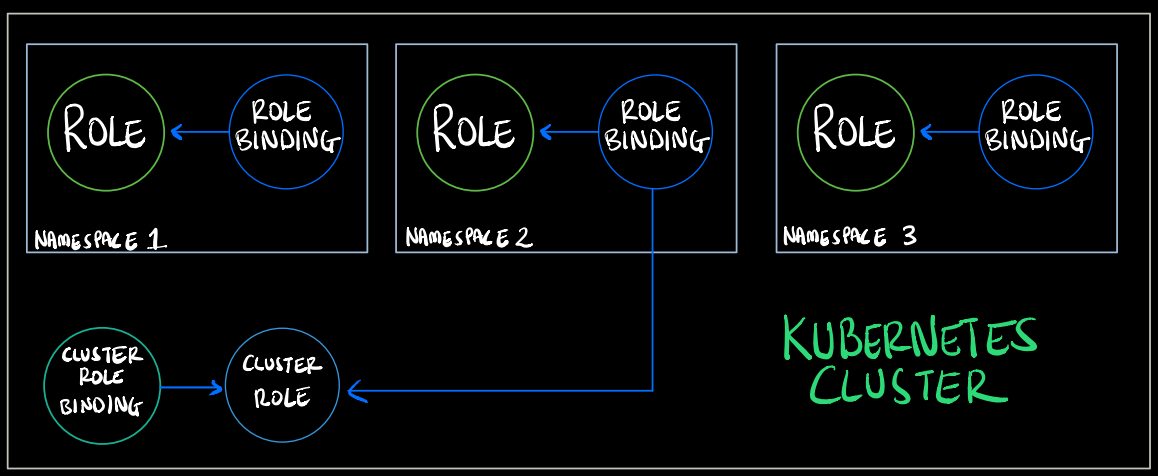


User

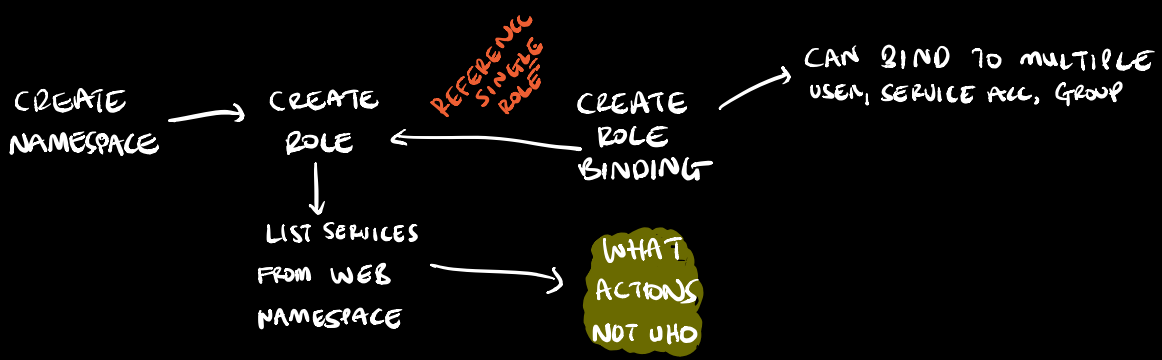


Cluster Authentication and Authorisation

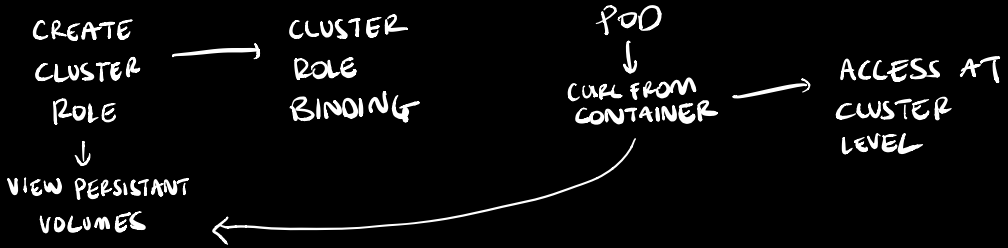




ROLE

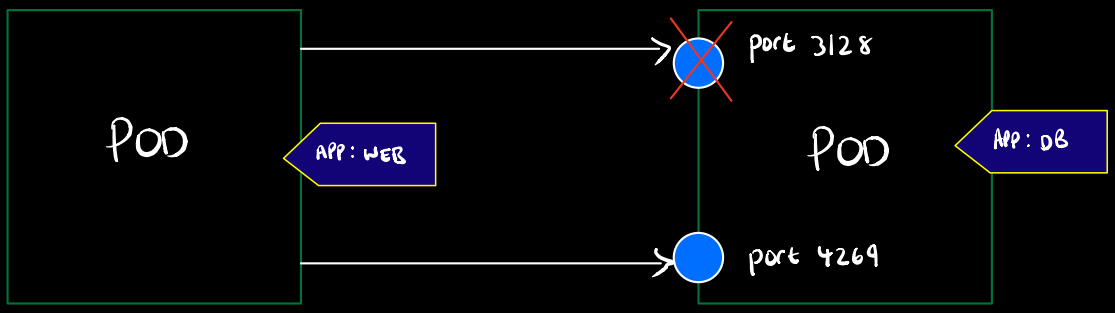


CLUSTER ROLE



Configuring Network Policies

Network policies use selectors to apply rules to pods for communication throughout the cluster



How?

Network Plugin → Canal Plugin → Create Network Policy → Create Deployment

Create Network Policy → Deny All Net

Blank ∴ all inherit →

```
Kind: Network Policy
Name: Deny All
podSelector: {}
Policy Type: Ingress
```

Create Deployment → Expose Deployment

Try Access → Timeout ✓

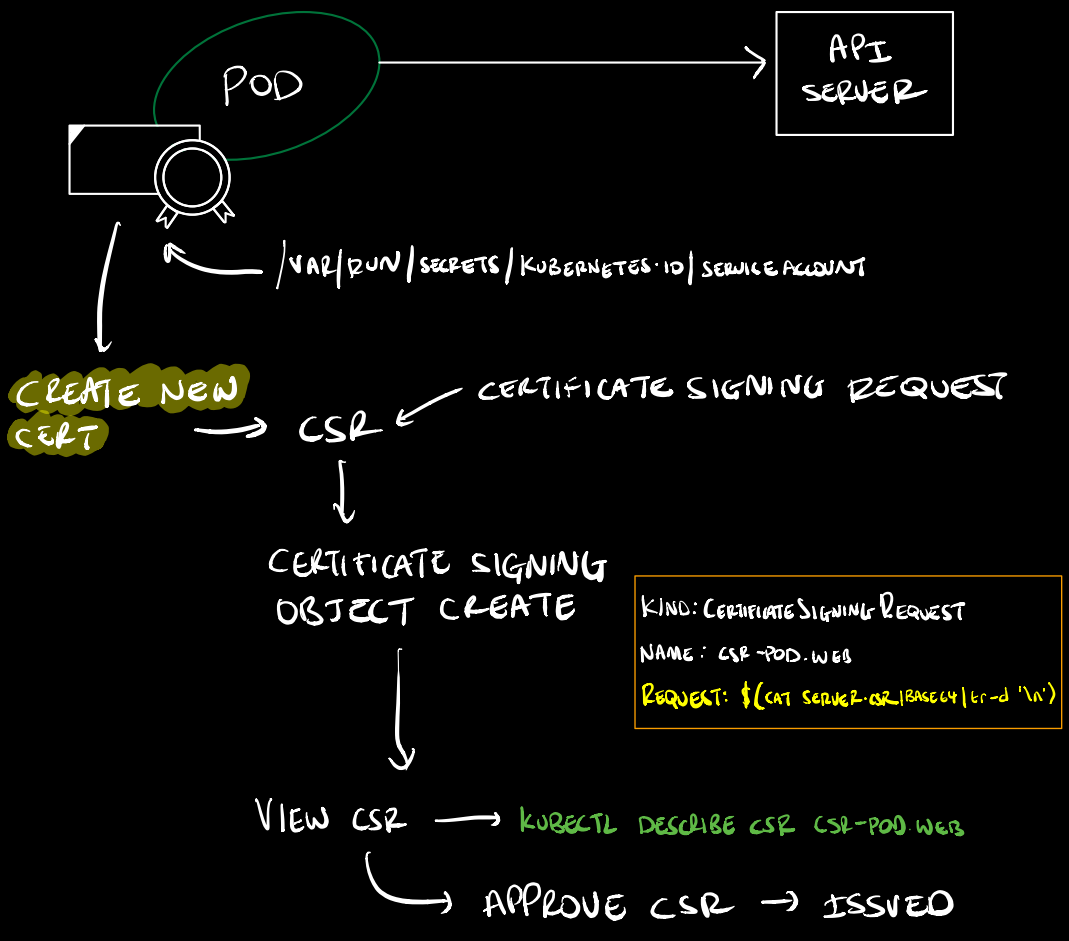
Create Network Policy → Label Pods

```
podSelector:
  matchLabels:
    app: db
ingress:
- ...
  matchLabels
    app: web
  port:
  - port: 4269
```

Label Pods → Allow comms between pods and specified port

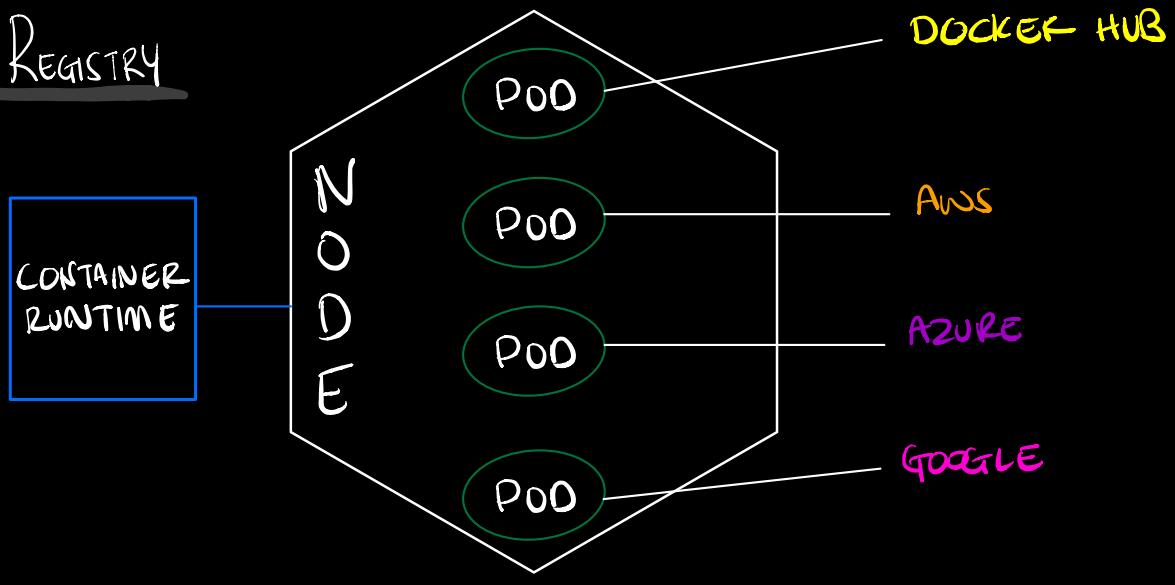
Creating TLS Certificates

The CA is used to generate a TLS certificate and authenticate with the API Server.



Secure Images

Private Registry



CONTROLLING IMAGES THAT GO INTO PRODUCTION

- → VULNERABILITIES → CLAIR (SCANNING)
- → SOMETHING ON IT CAUSE NODE CRASH
- → LOGIN TO PRIVATE REGISTRY
  - → TAG DOCKER IMAGE
    - → PUSH TO PRIVATE REGISTRY

How?

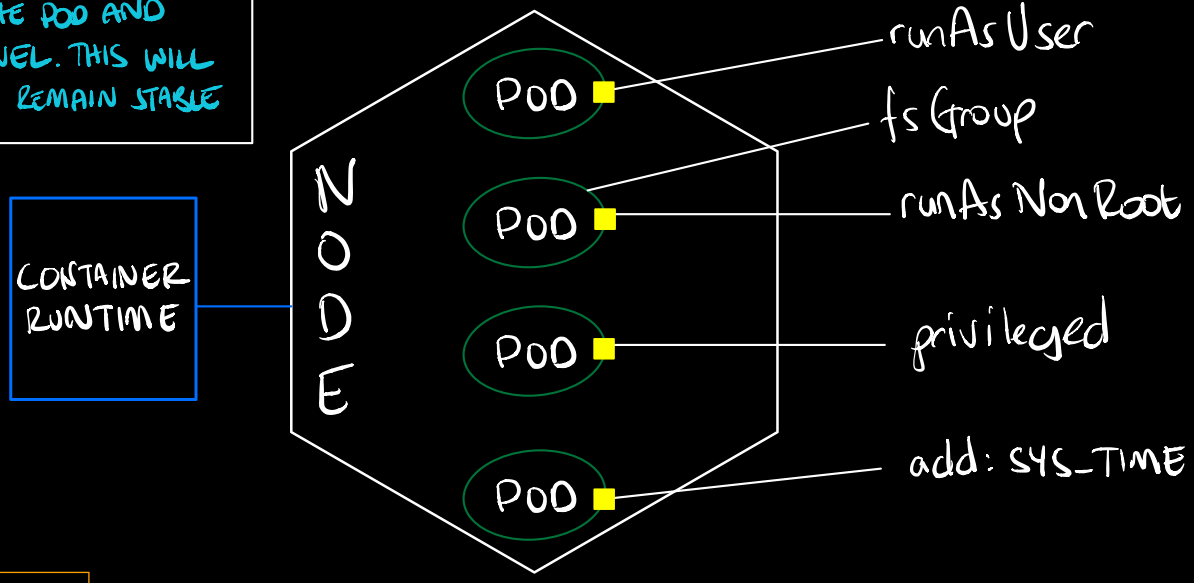
KUBERNETES CREATE SECRET → DOCKER REGISTRY TYPE
- → docker-server
- → docker-username
- → docker-password

DOCKER REGISTRY TYPE → MODIFY SERVICE ACCOUNT (USE FOR IMAGE PULLS)
- → kubectl patch

MODIFY SERVICE ACCOUNT → POD
- → PRIVATE REGISTRY IMAGE

Defining Security Context

LIMIT ACCESS TO CERTAIN OBJECTS AT THE POD AND CONTAINER LEVEL. THIS WILL ALLOW IMAGES TO REMAIN STABLE




```
kind: Pod
image: alpine
SecurityContext:
    runAsUser: 405
```

→ Run Pod as 405

Can Also put 'runAsNonRoot'

Ability to run as privileged → 'privileged: true'

CONTAINER LEVEL

ABILITY TO LOCK DOWN KERNEL LEVEL FEATURES ON CONTAINER → SETTING CAPABILITIES ON POD LEVEL

ADD →

```
SecurityContext:
  add:
    - SYS_TIME
    - NET_ADMIN
```

REMOVE →

```
SecurityContext:
  drop:
    CHOWN
```

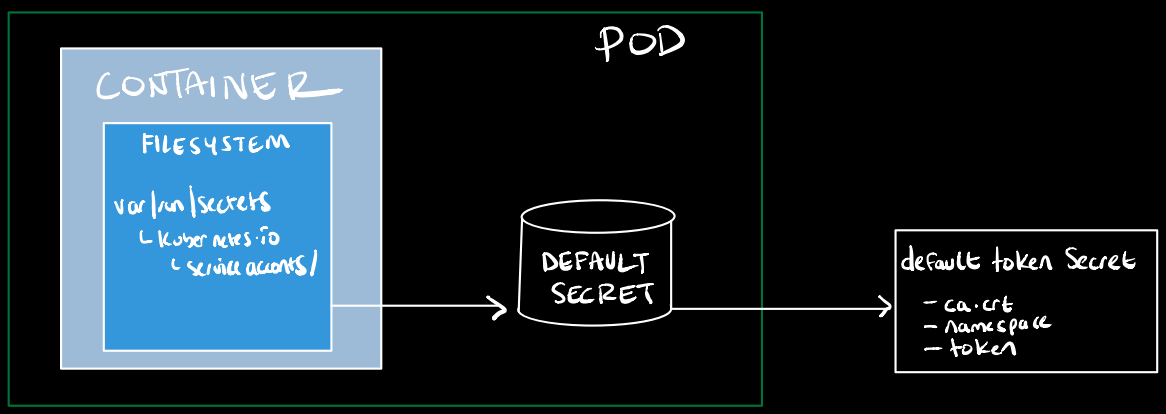
Securing Persistent Key/Value Store

> Secrets allow you to expose entries as files in a volume. Keeping this data secure is critical to cluster security

Data must live beyond life of pod → **Secrets** → key/value pair

key/value pair → pass as env var → not best practice → may be output to log files

key/value pair → or expose as files in volume



Https to website

Generate certificate → key, cert, file → Secret → Mount in pod → Pod

Secret → combined files → Mount in pod

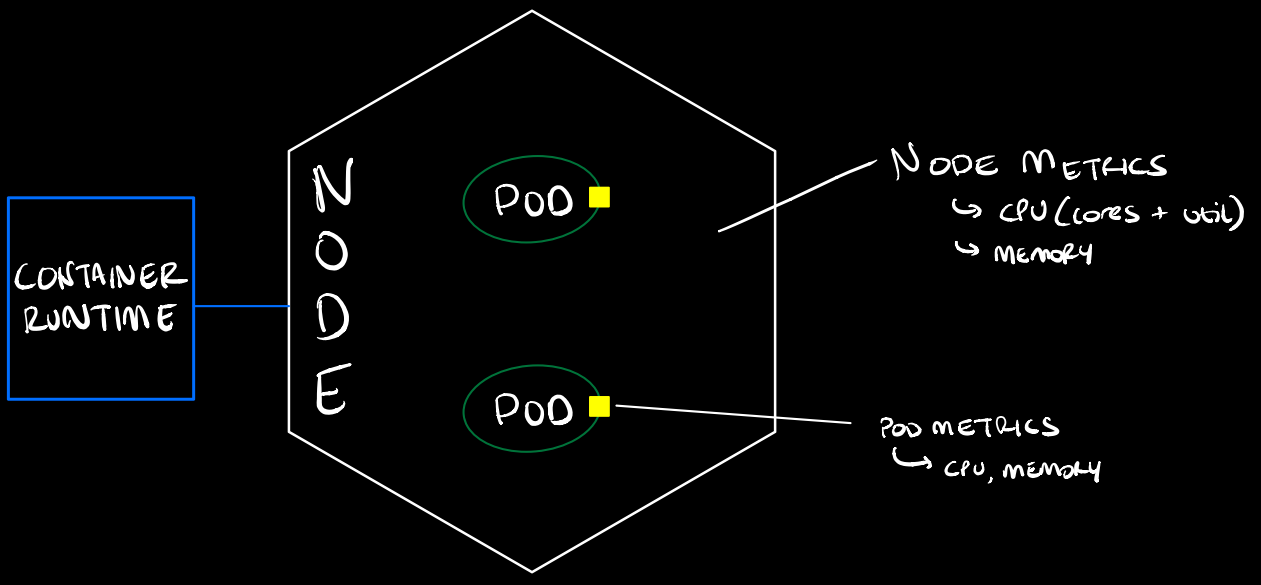
Mount in pod → use secret

Pod → in memory file system → secret not written to disk

MONITORING CLUSTER COMPONENTS

Monitoring the Cluster Components

The metric server allows you to collect CPU and memory data from the nodes and pods in your cluster

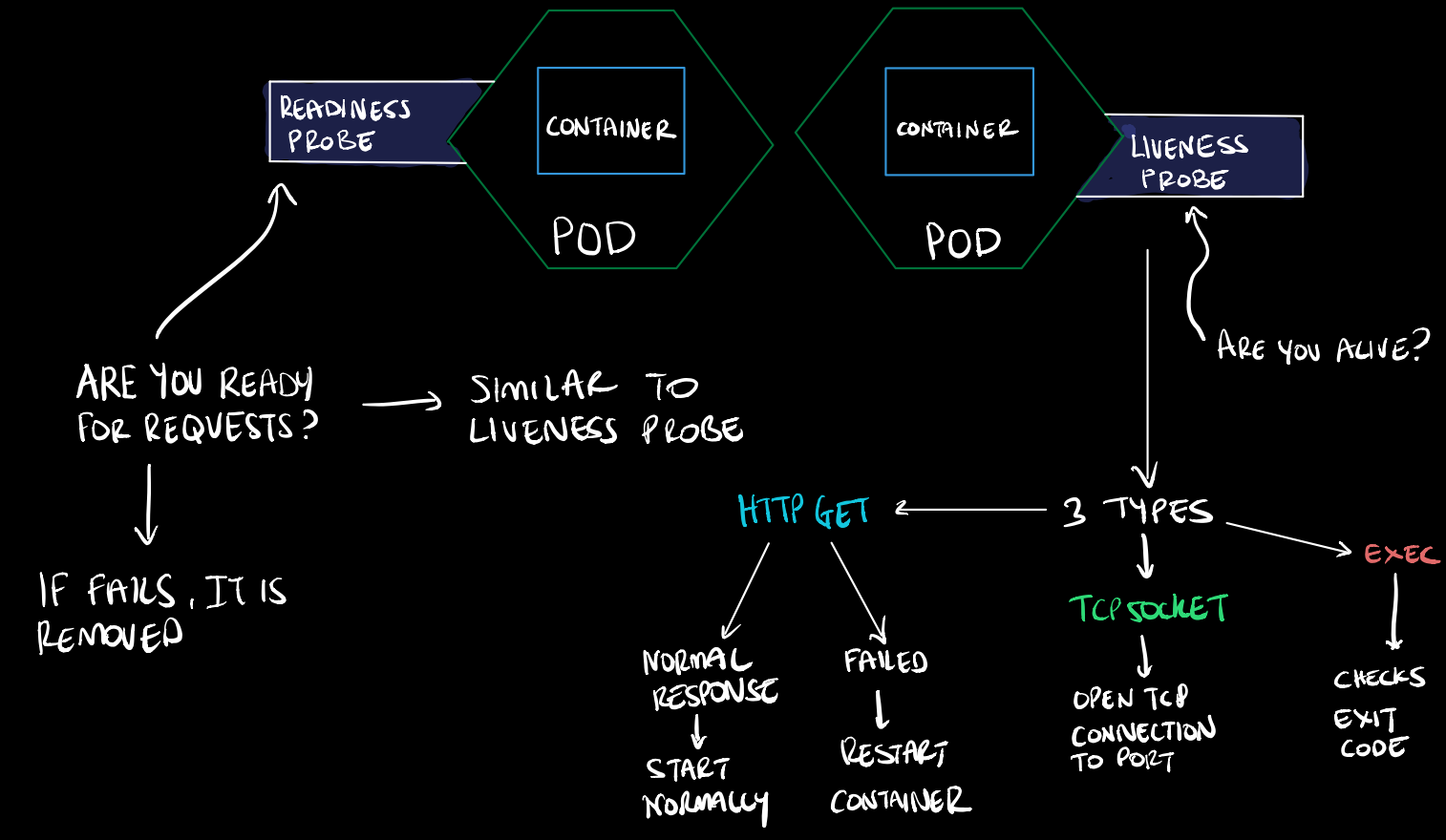


Install metric server →

- `kubectl top node` → CPU/memory for all the nodes
- `kubectl top pod` → CPU/memory for all the pods
- `kubectl top pod --all-namespaces` → all namespace
- `kubectl top pod -n kube-system` → kube-system namespace
- `kubectl top group-context --containers` → pods containers

Monitoring the Apps Running Within a Cluster

Liveness and Readiness probe



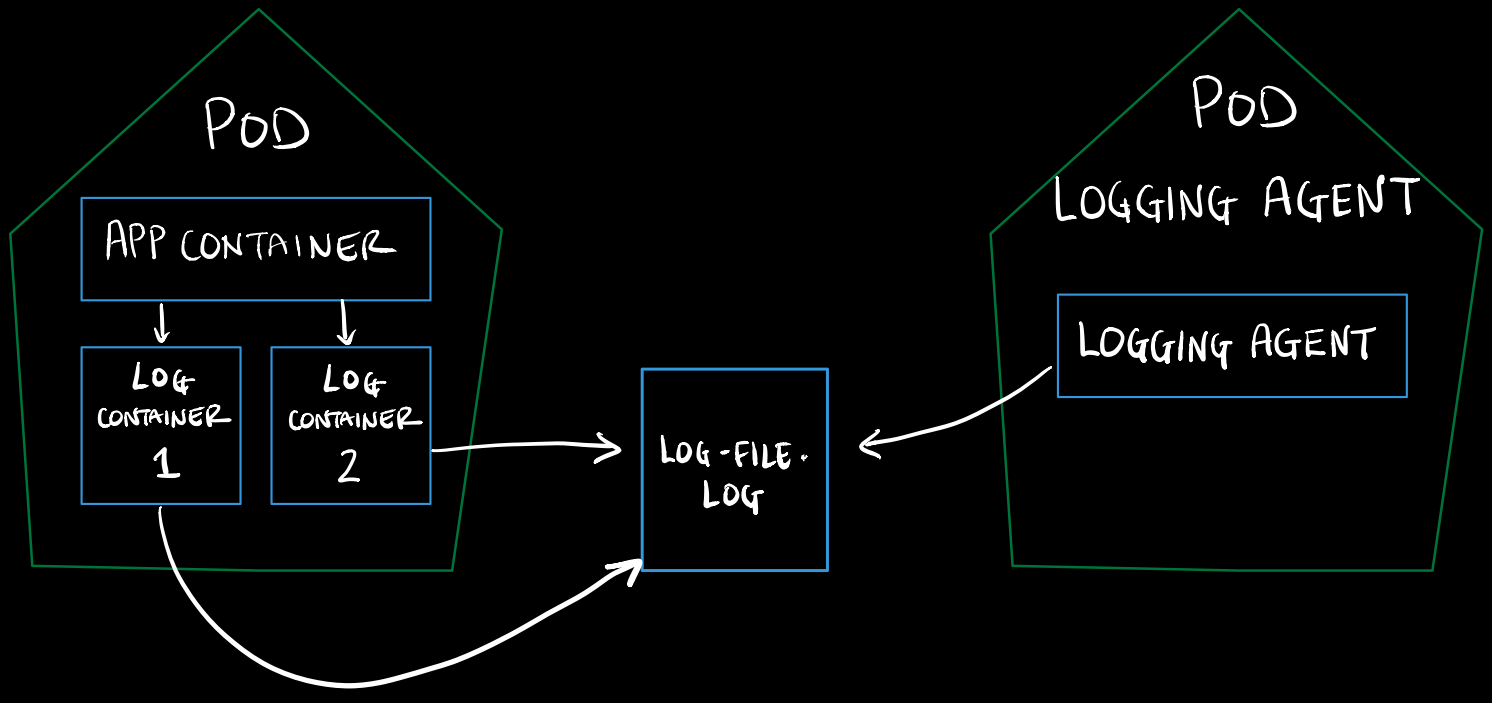
Managing Cluster Component Logs

App and System Logs

- Logs to container streamed to stdout → use `kubectl logs`
- → understand what is happening inside cluster
- Debugging
- Logs accumulate over time
- If many microservice → which logs are which?

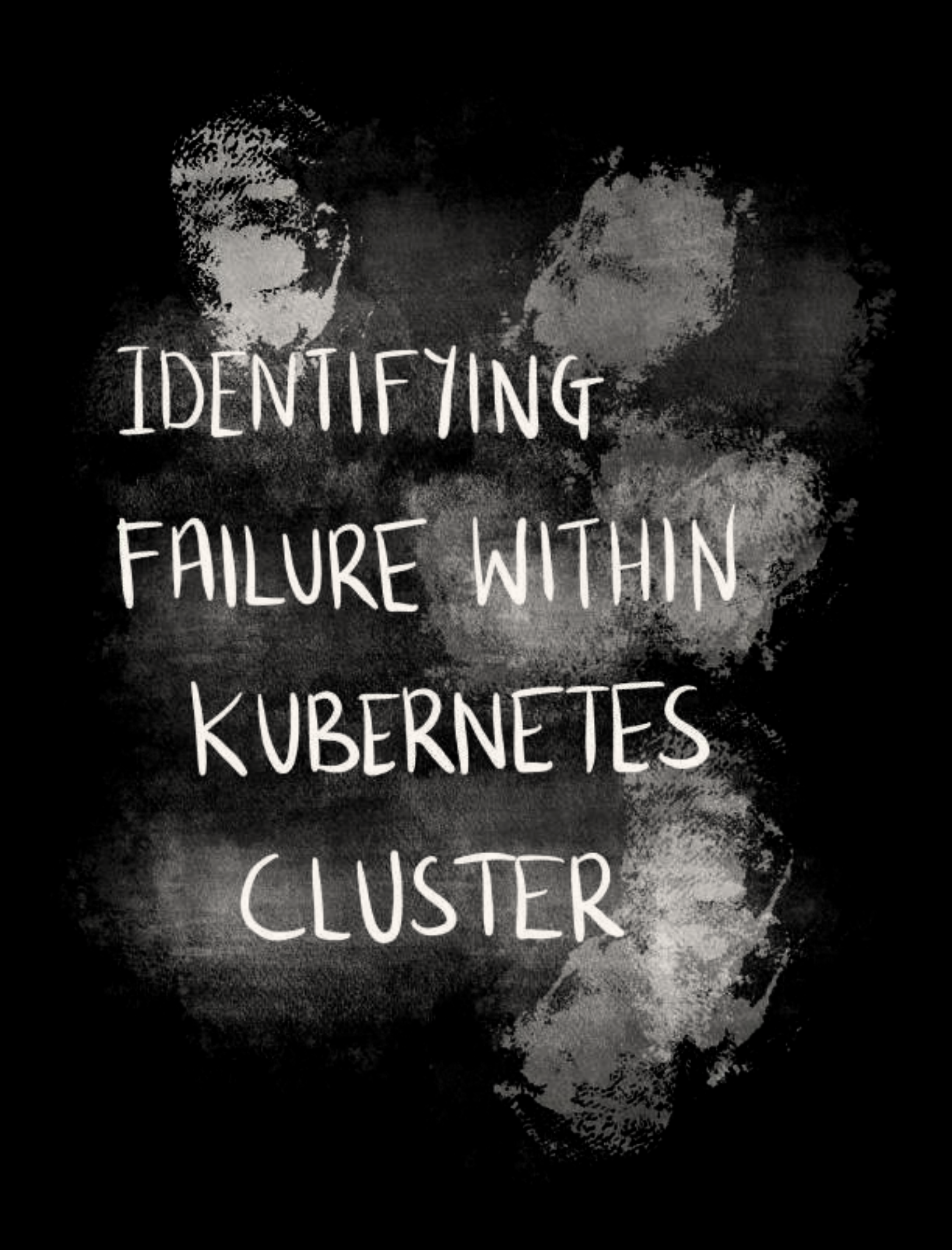
Log directory → /var/log/containers

↑ accumulates logs



→ Have sidecar container to do logging so you can access specific logs

→ Able to rotate logs using other tooling → no native


IDENTIFYING
FAILURE WITHIN
KUBERNETES
CLUSTER

Troubleshooting Application Failure

Ability to write termination message to specific file on container →

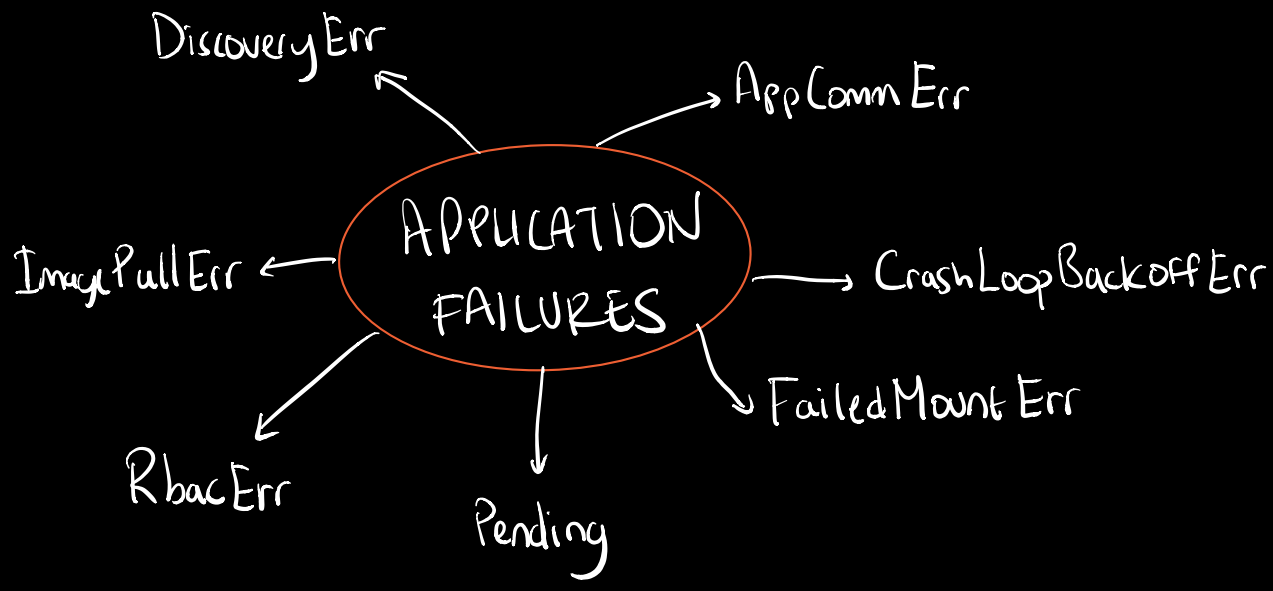
pod1.yaml

```
kind : pod
name : pod1
image: busybox
command :
....
terminationMessagePath
```

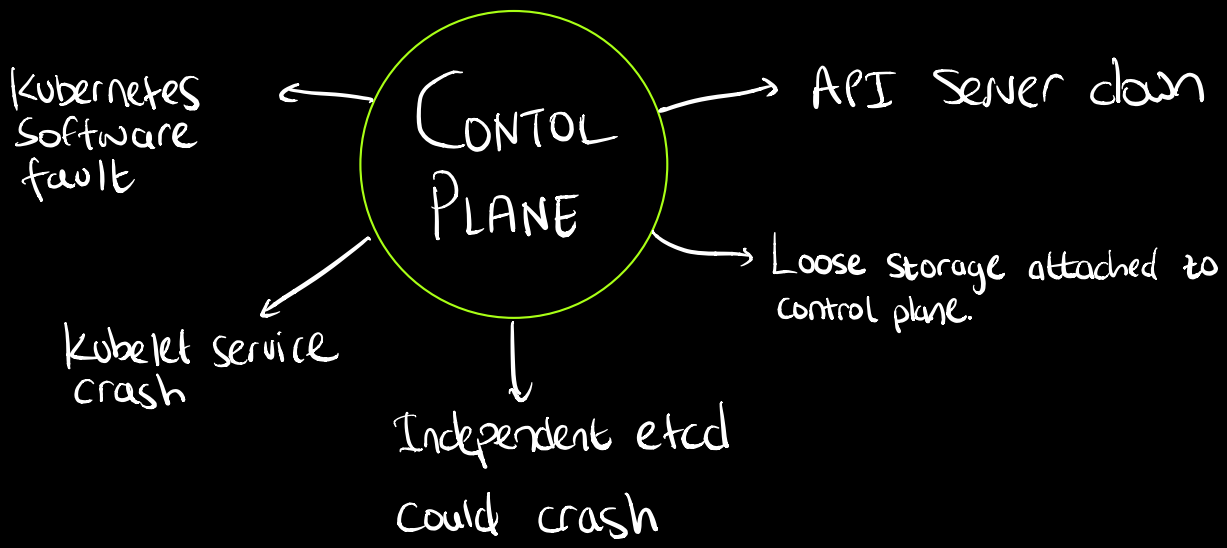
→ kubectl describe ↓ will show error message

→ Only particular fields can be changed i.e Image

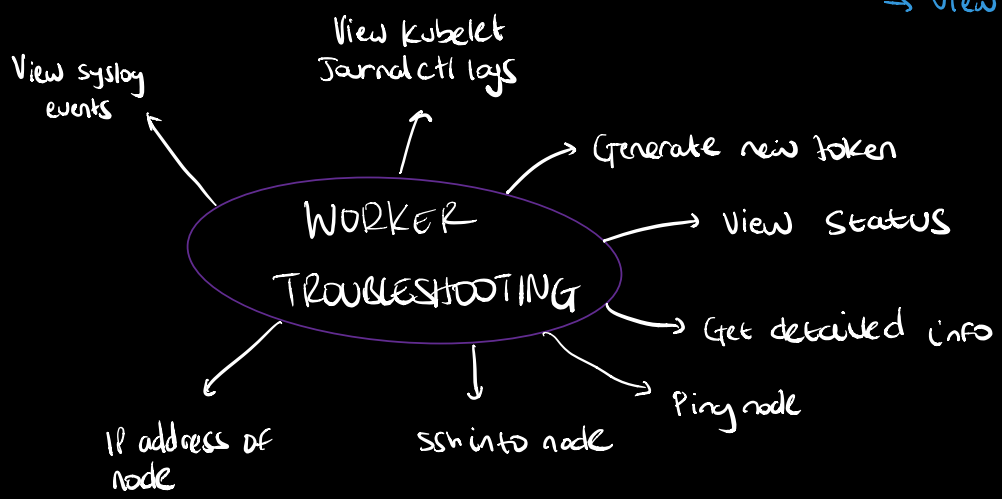
→ To change other fields of failured pod ↓ export configuration → kubectl get po mem2 -o yaml --export ↓ modify yaml ie change memory request

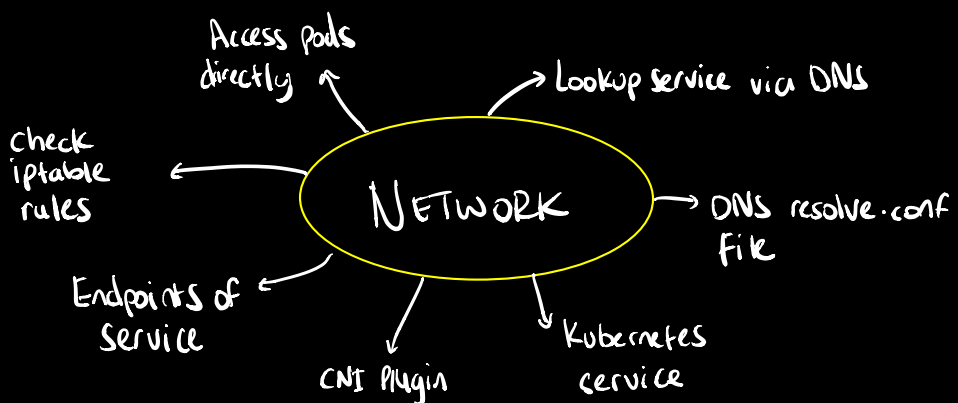


Troubleshoot Failures



- → View the events from control plane components
- → View logs for control plane pods
- → Check status of docker service
- → Check status of kubelet service
- → Disable swap
- → check firewalld service
- → View kube config





Thank You For Reading

Please Like and Share For More

Latest Notes Available On

Instagram → adnans_techie_studies