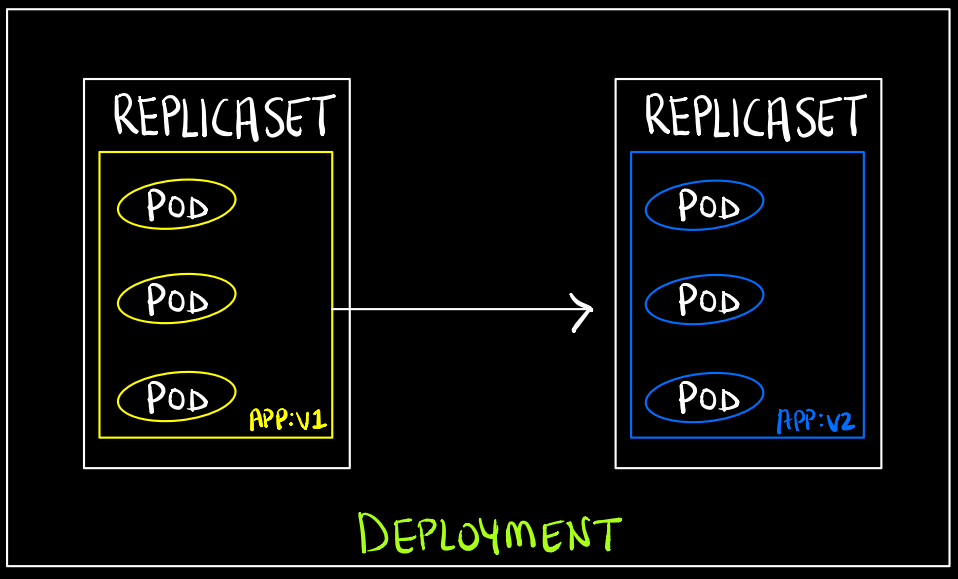
DEPLOYING APPLICATIONS IN THE KUBERNETES CLUSTER

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

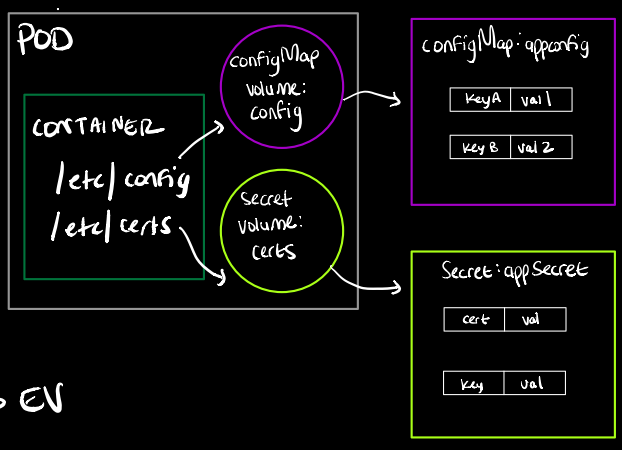
- **Min ready seconds** → How long a newly created pod should be ready before considered available
- **+**
- **Readiness probe** → Determines if a specific pod should recieve client request or not.

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

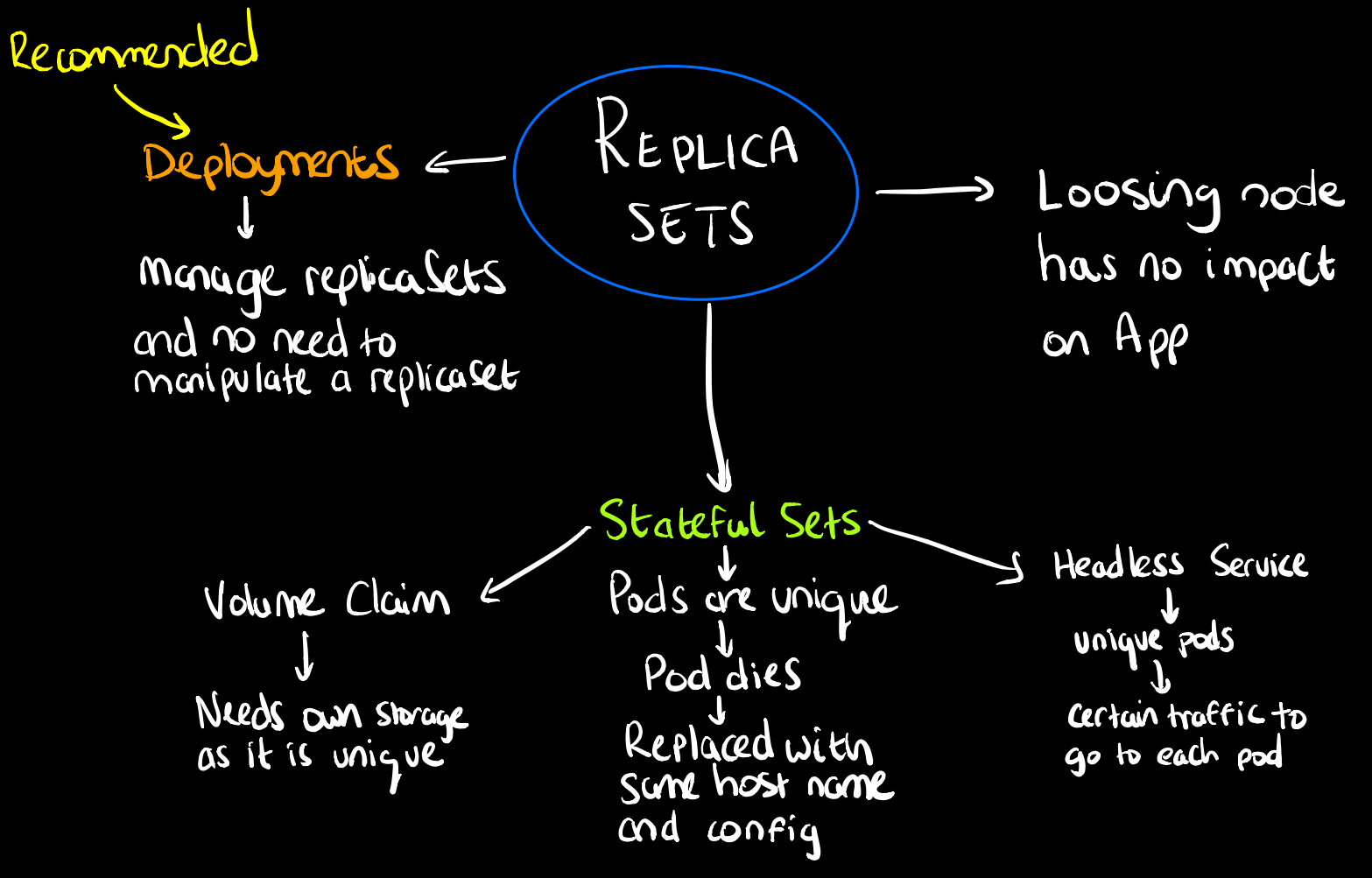
Passing configuration options to app

- 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

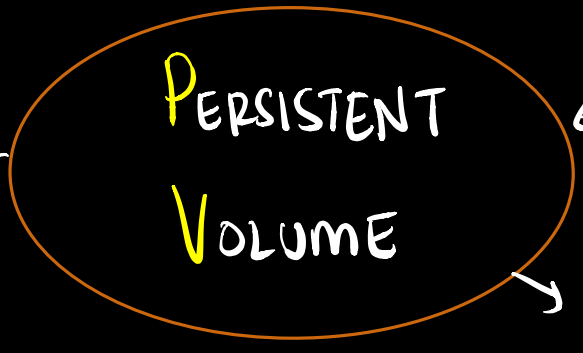


MANAGING DATA IN KUBERNETES CLUSTER

Persistent Volumes

→ Pods are ephemeral → Pod terminated → Storage terminated

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



Storage Classes

Resource in Cluster

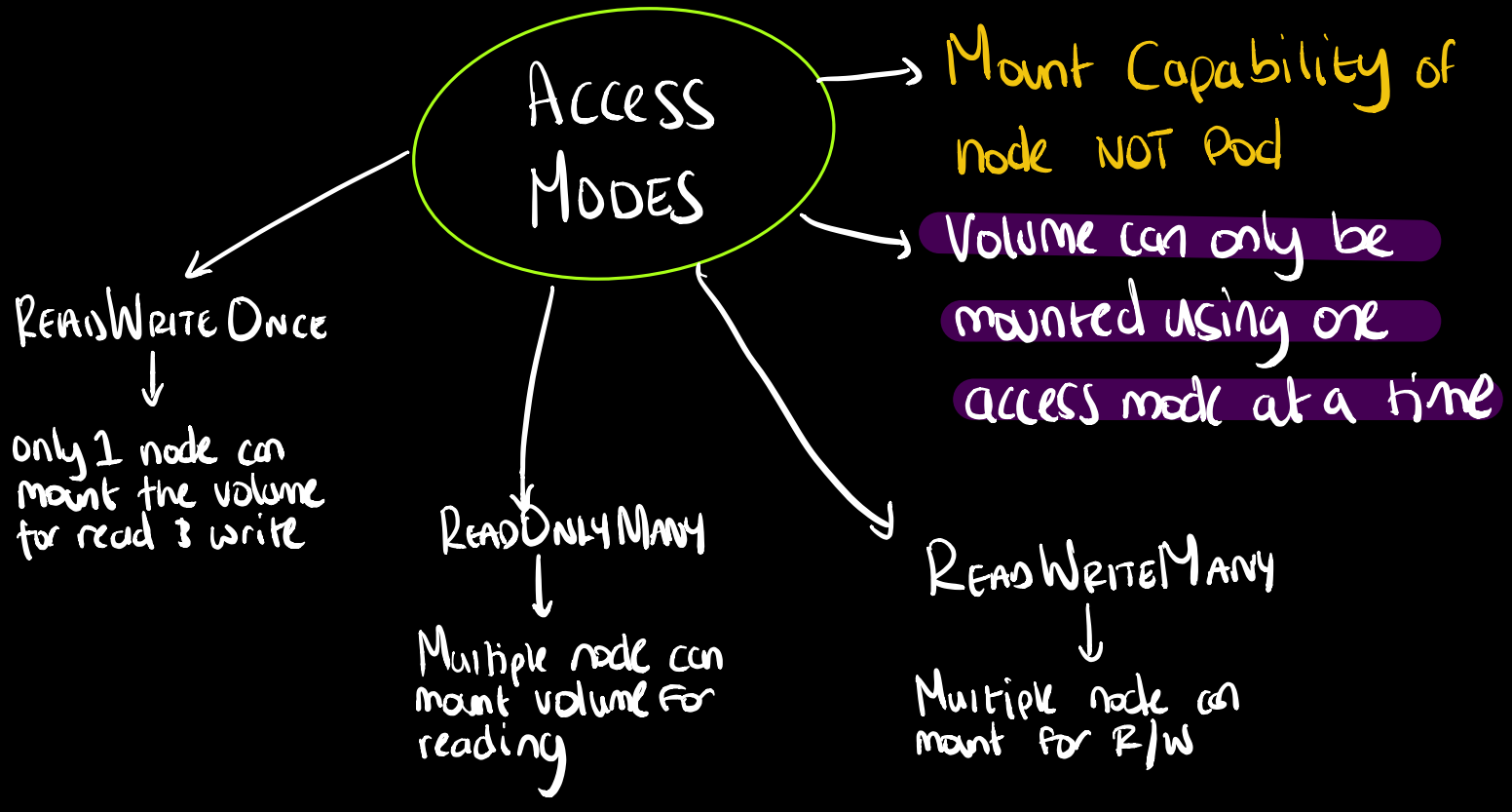
Provisioning

Static → Cluster admin creates and avail for consumption

Dynamic → PVC must request a storage class

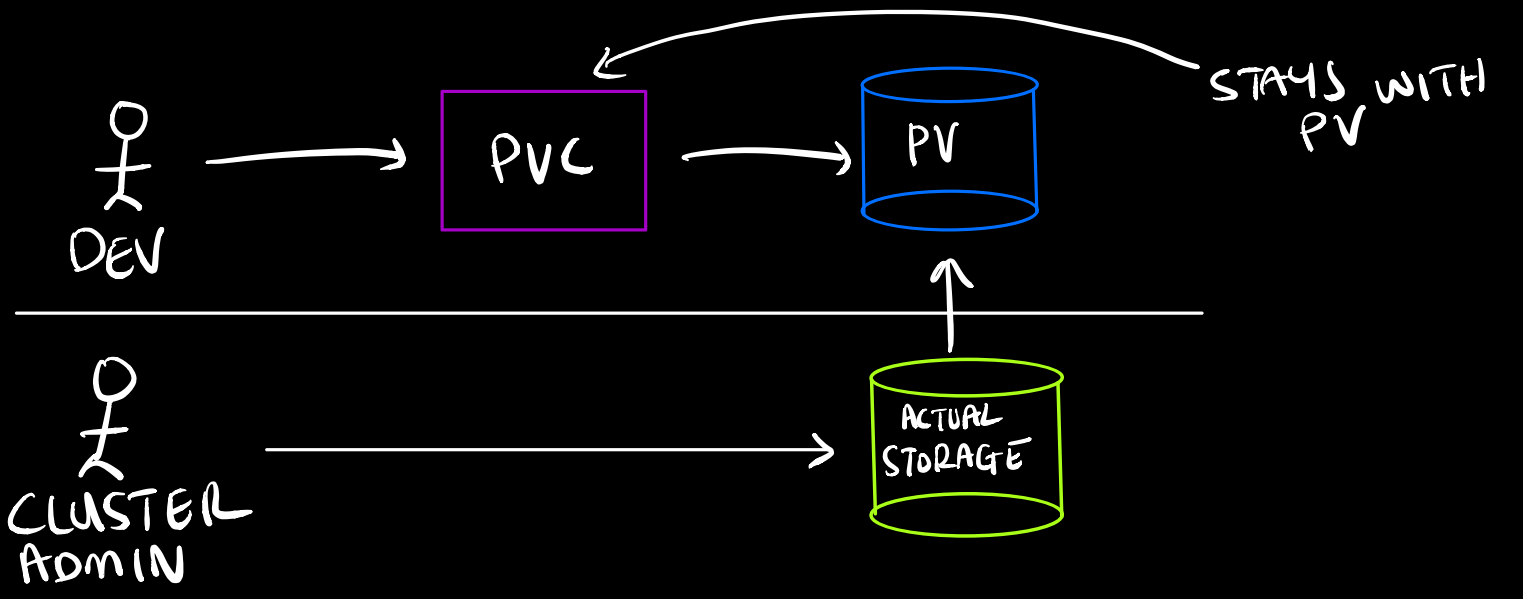
Volume Access Modes

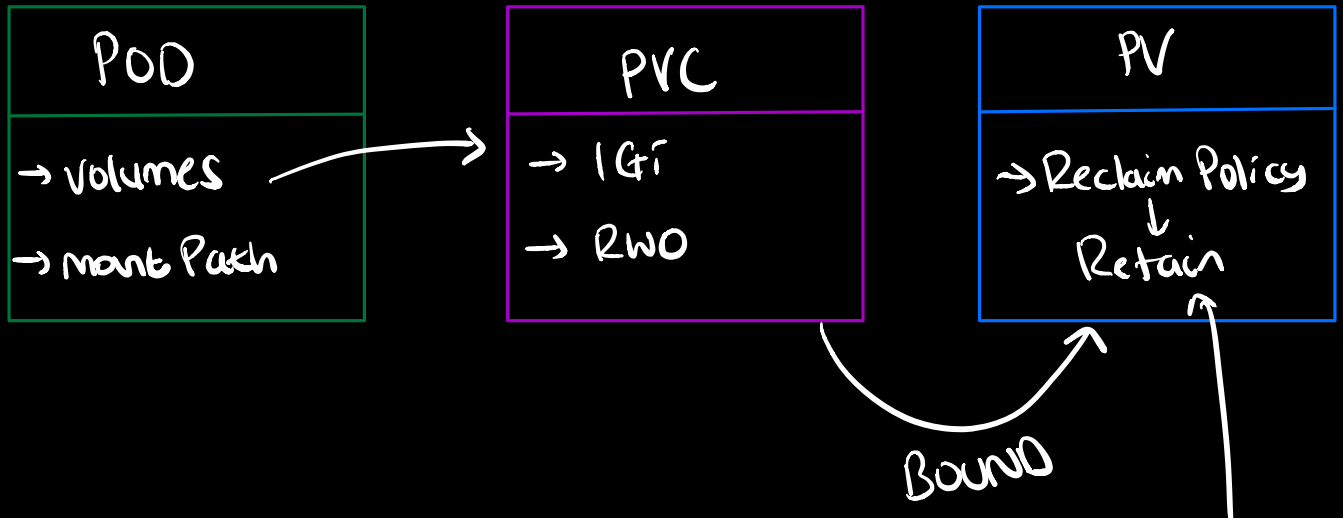
By specifying an access mode with your PV, you allow the volume to be mounted to one or many nodes, as well as read by one or many



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

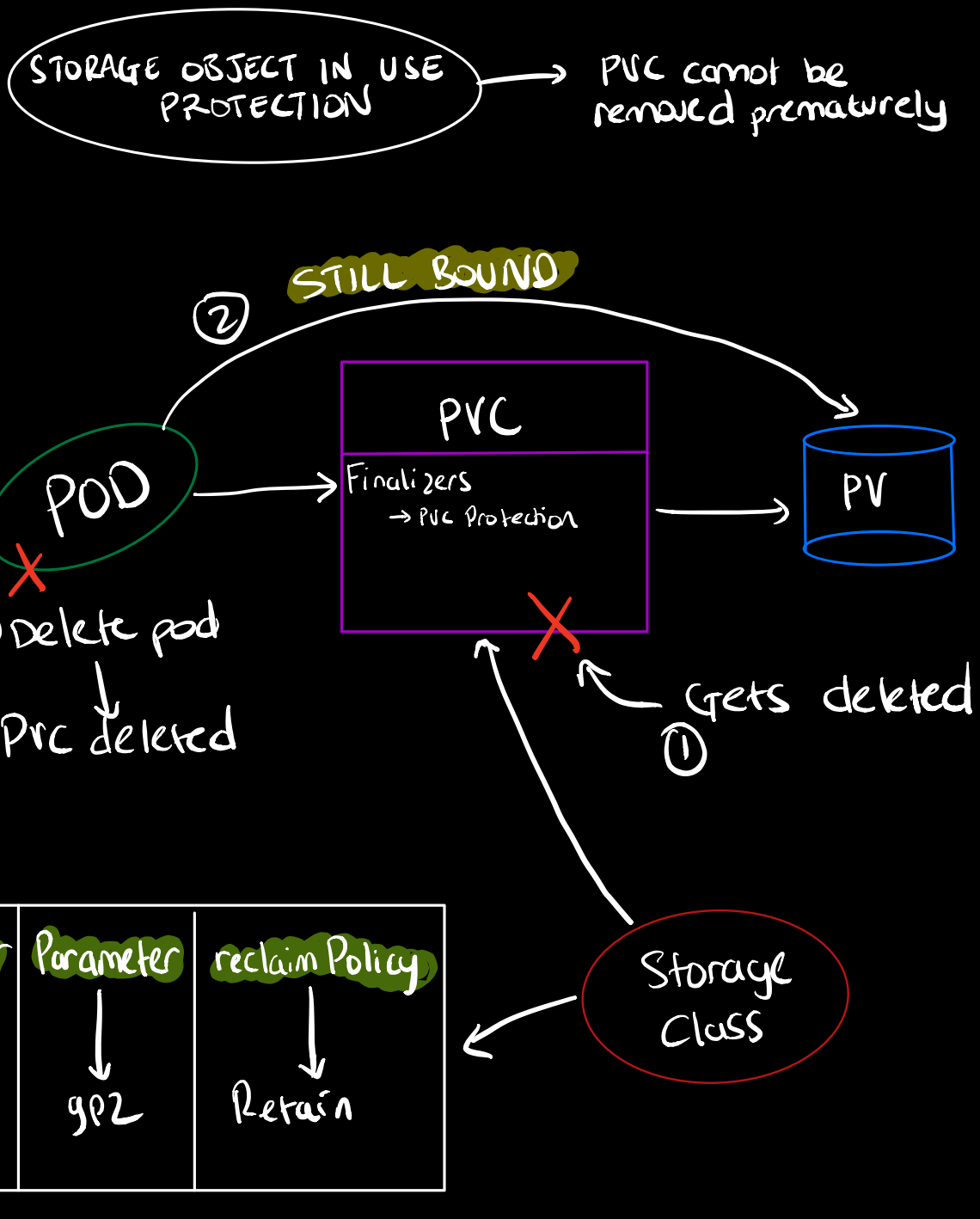
Could also be recycle or delete

deletes content of volume

delete underlying storage

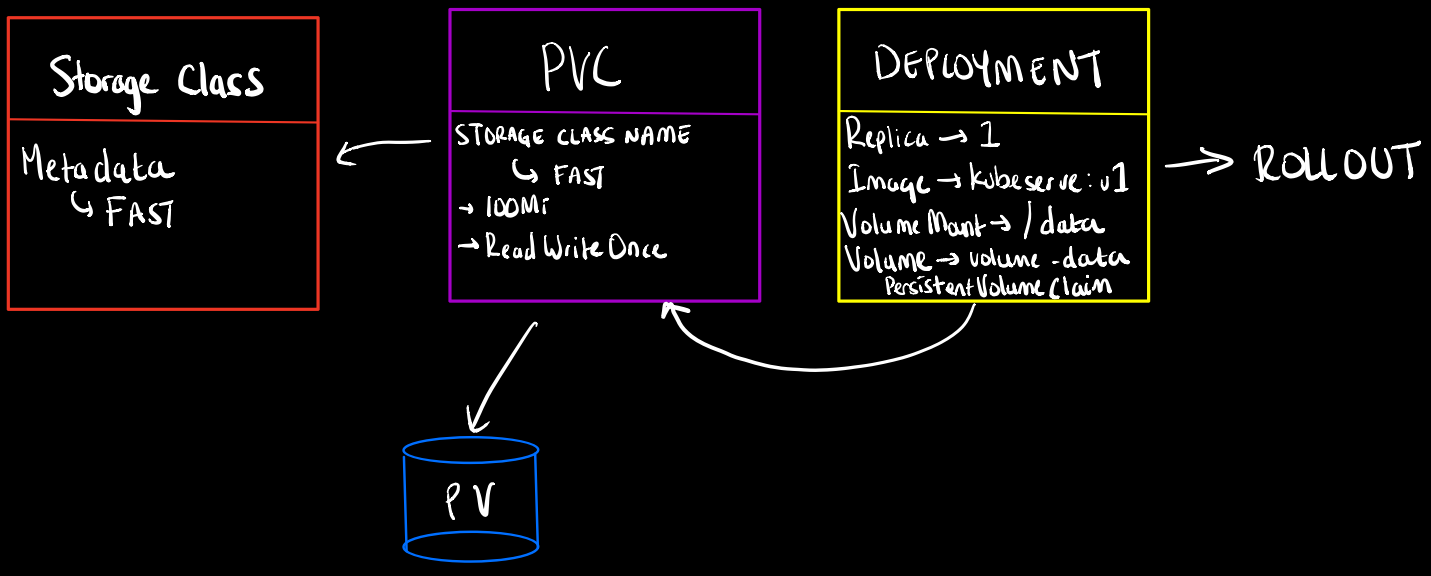
Storage Objects

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

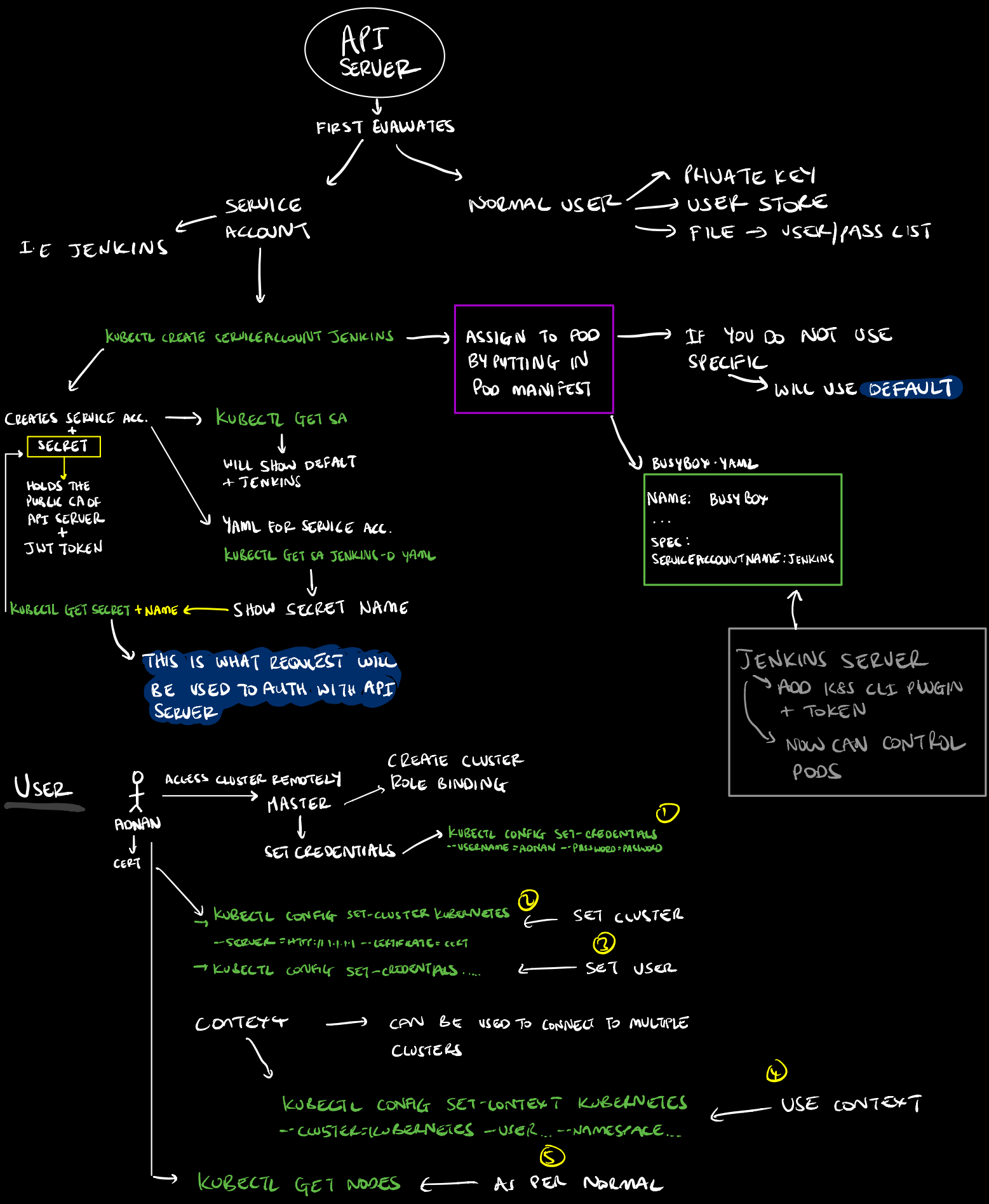
Example →



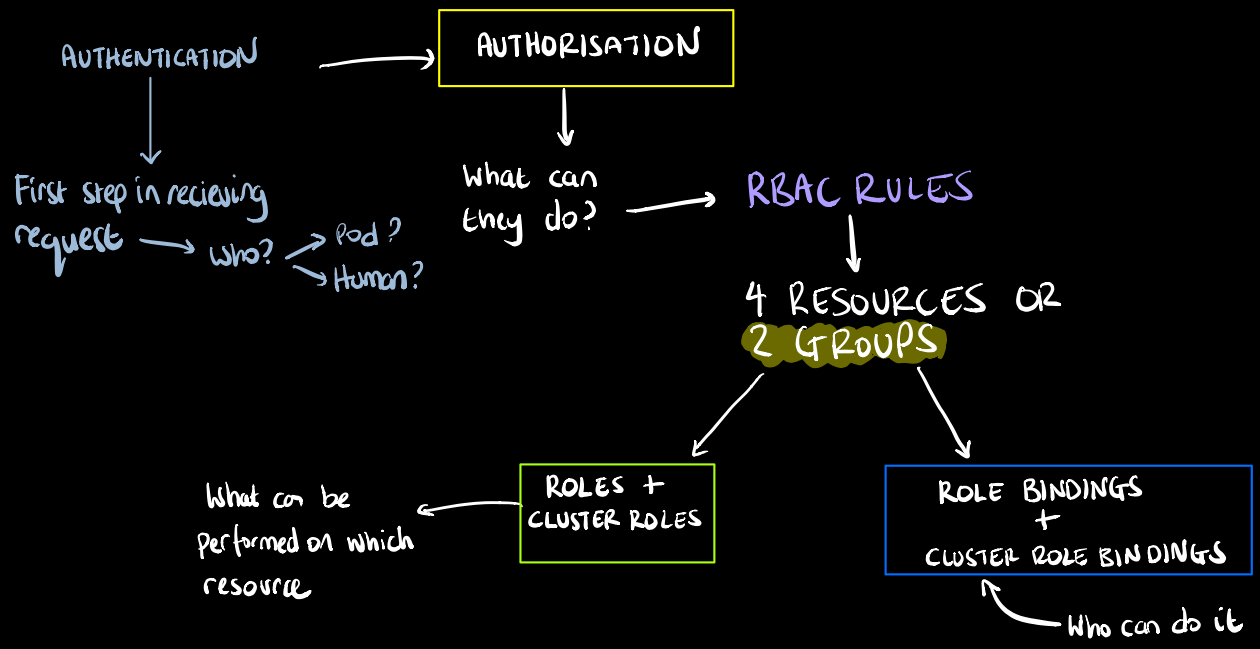
1. Create storage class object
2. Create PVC object
3. Create deployment
4. Rollout deployment
5. Check pods
6. Create file on mark
7. List contents.

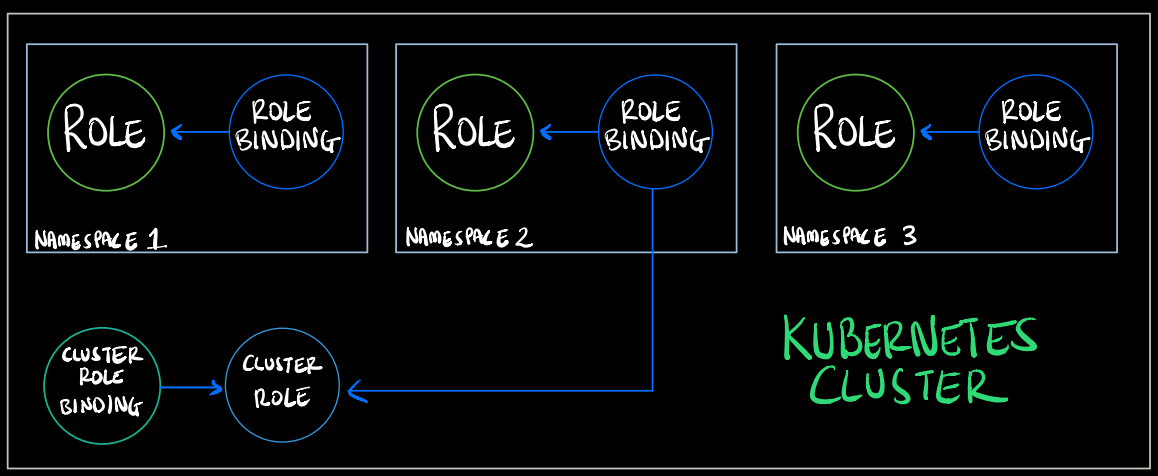

SECURING
THE
KUBERNETES
CLUSTER

Service Accounts and Users

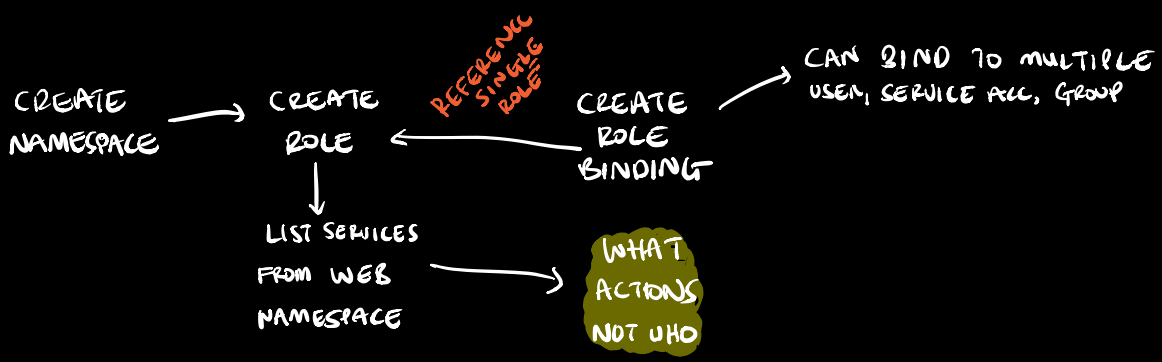


Cluster Authentication and Authorisation

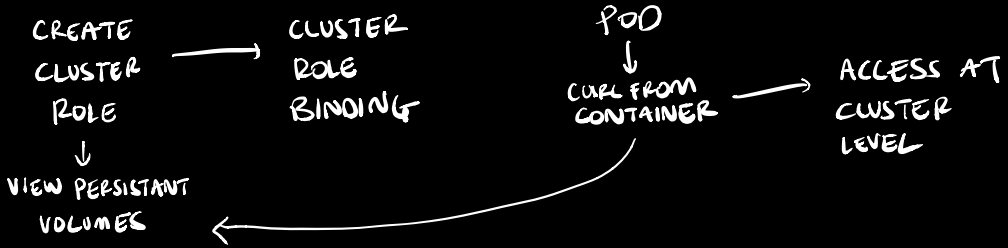




ROLE

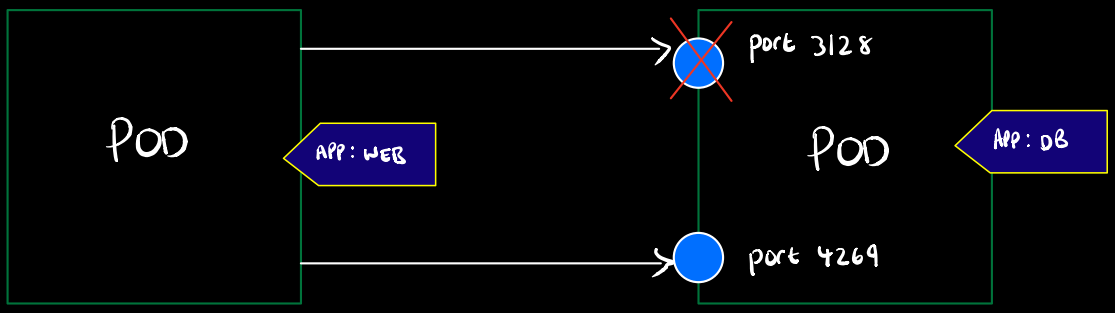


CLUSTER ROLE



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 ✓

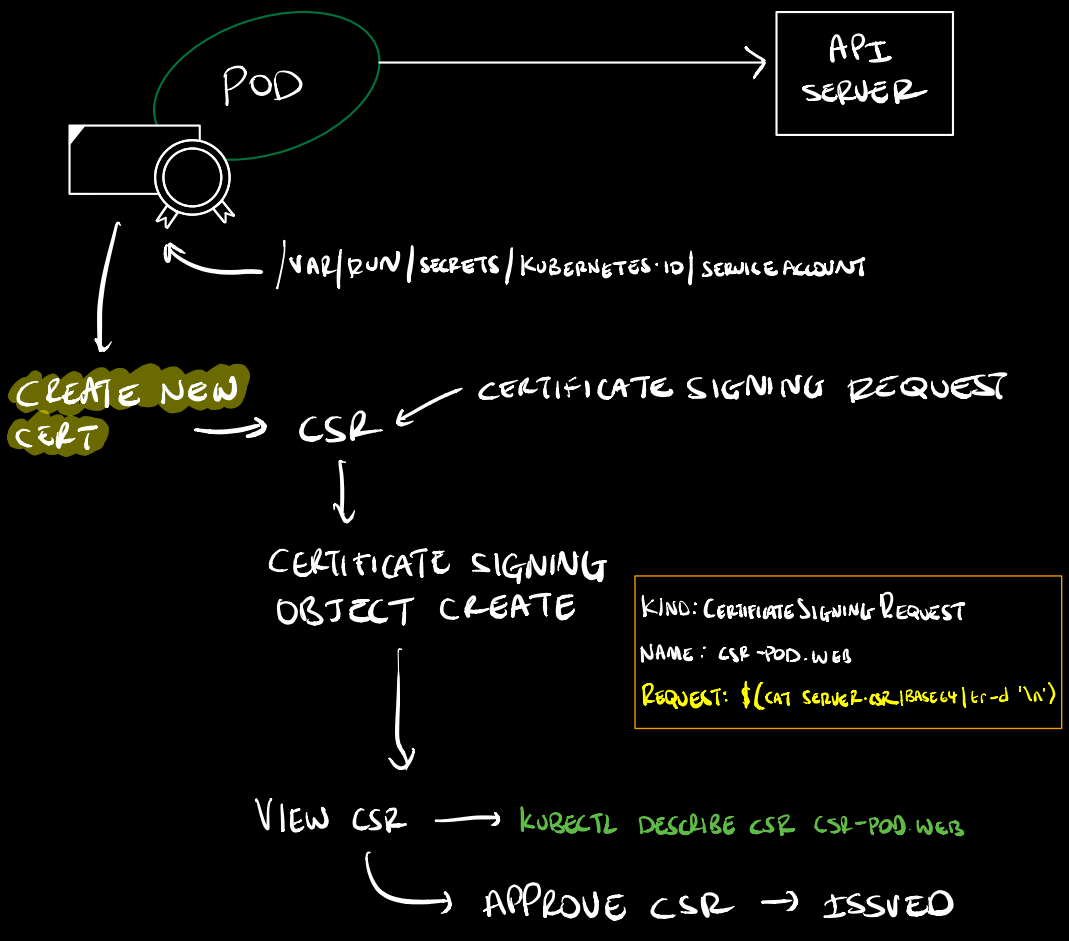
Create Network Policy → Label Pods

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

Label Pods → Allow comms between pods and specified port

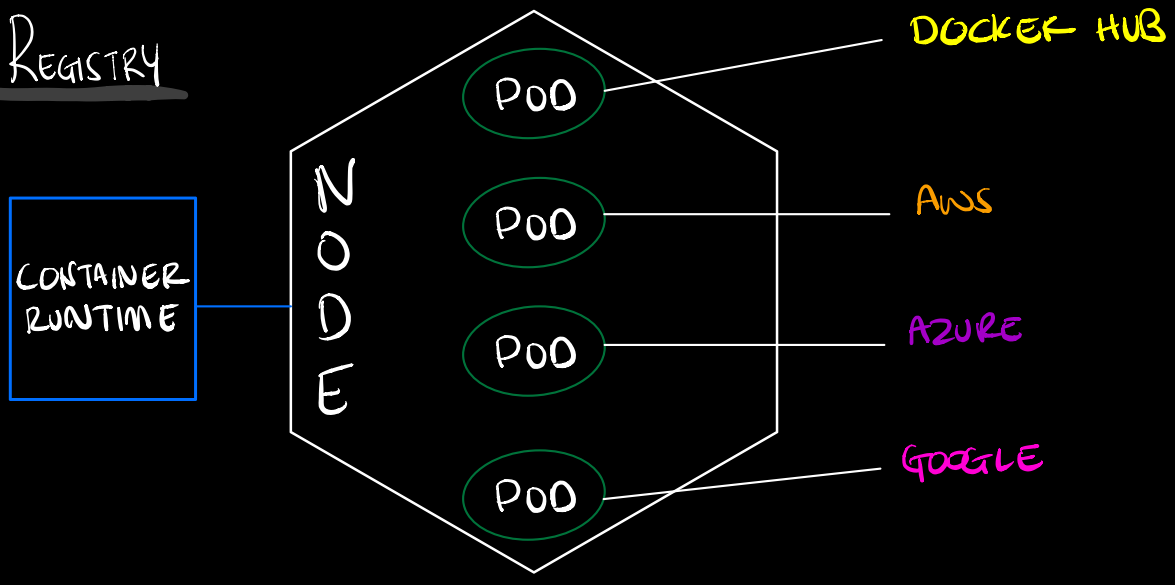
Creating TLS Certificates

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?

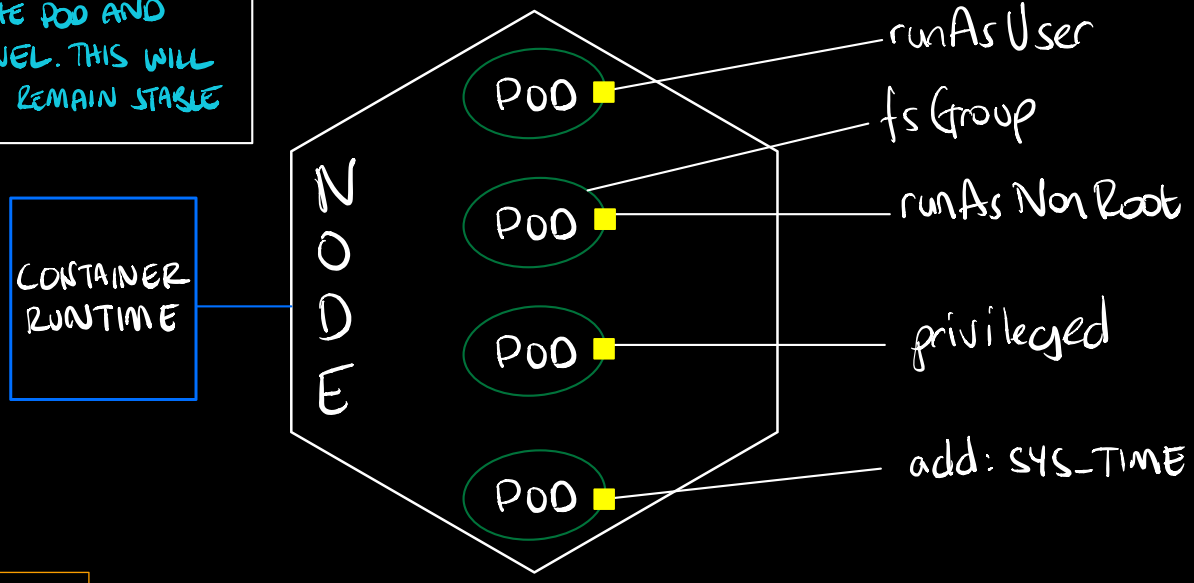
KUBERNETES CREATE SECRET → DOCKER REGISTRY TYPE
- → docker-server
- → docker-username
- → docker-password

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
```

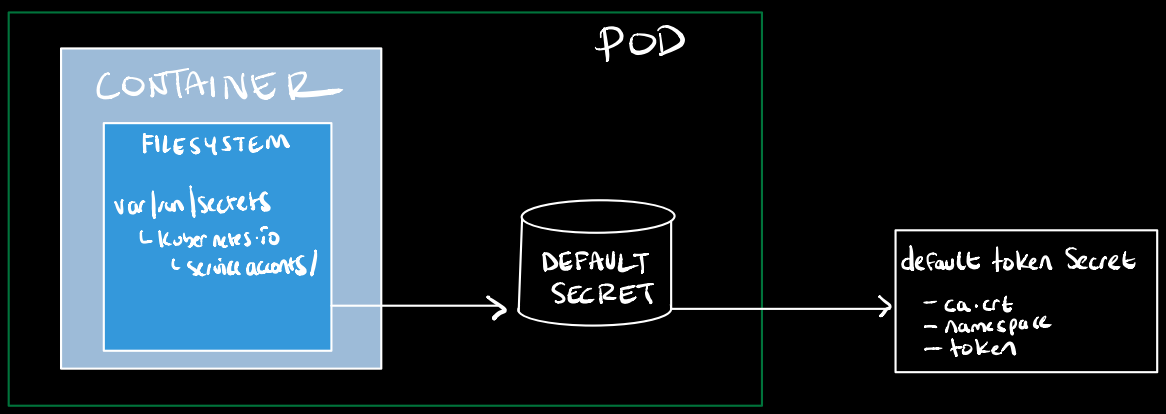
Securing Persistent Key/Value Store

> 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

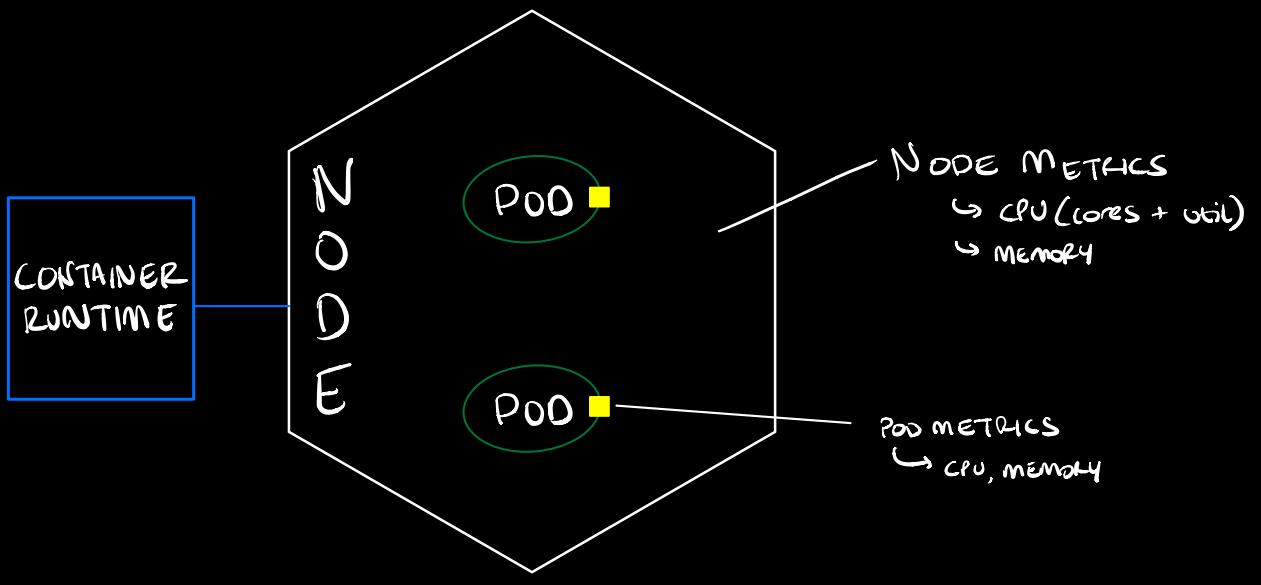
Mount in pod → use secret

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

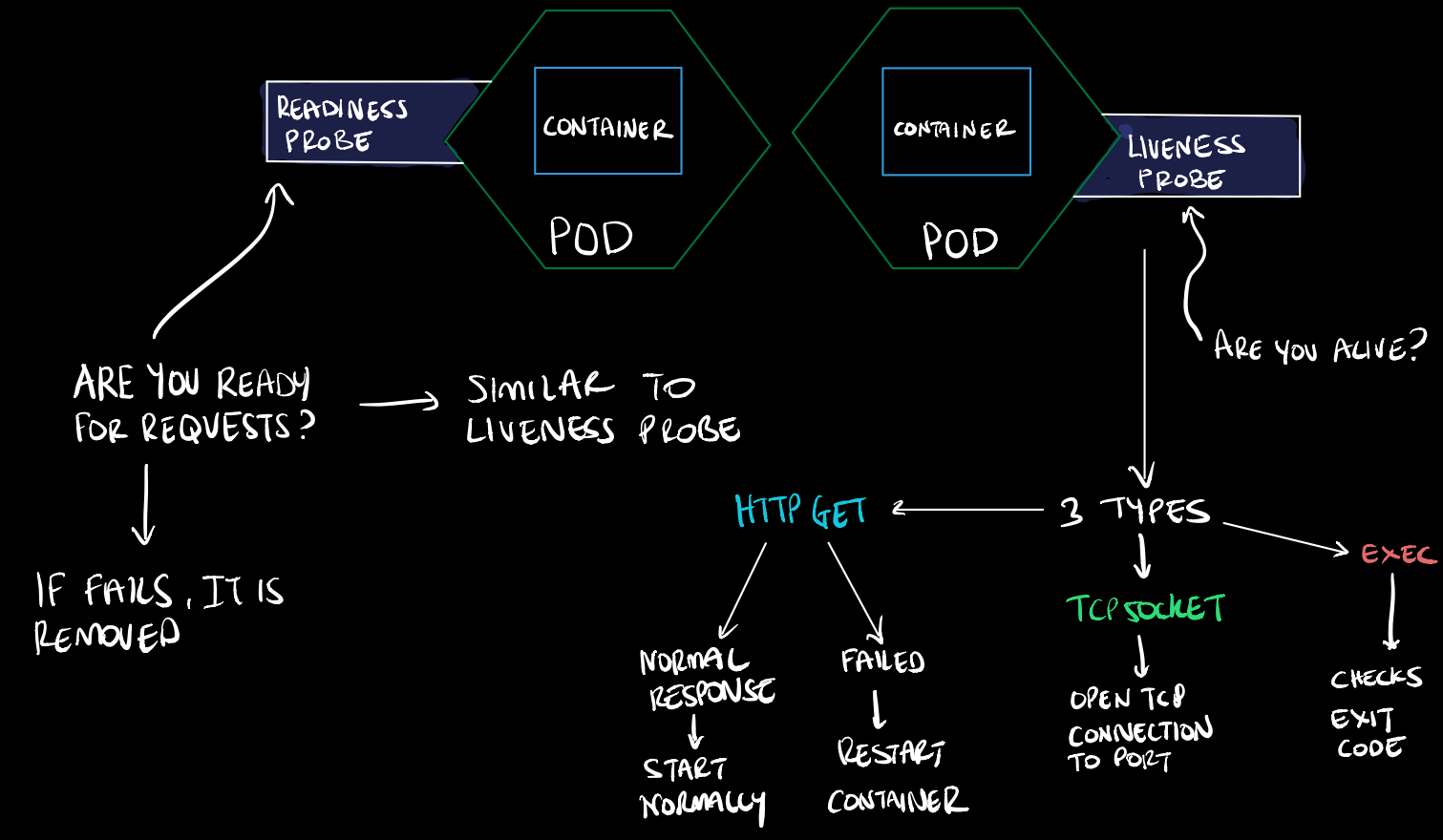


Install metric server →

- `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



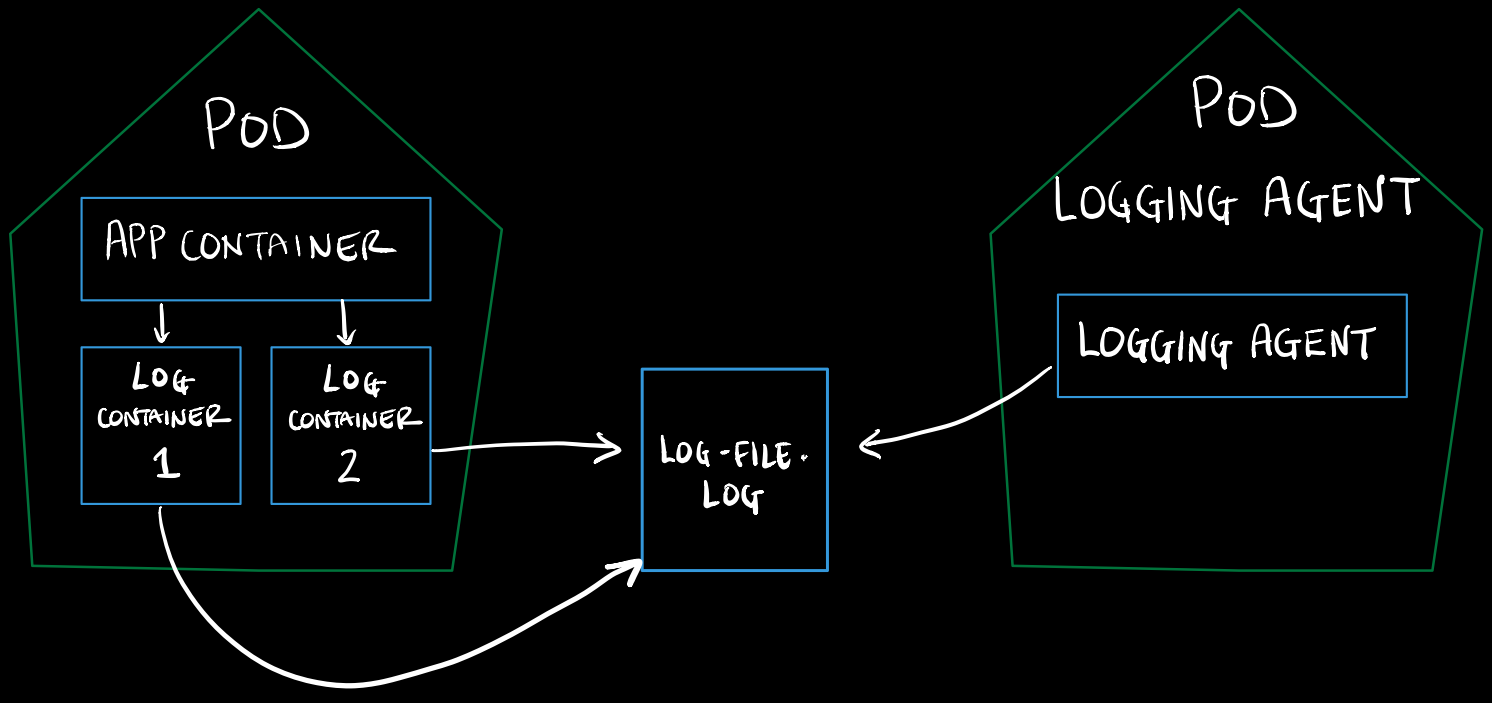
Managing Cluster Component Logs

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?

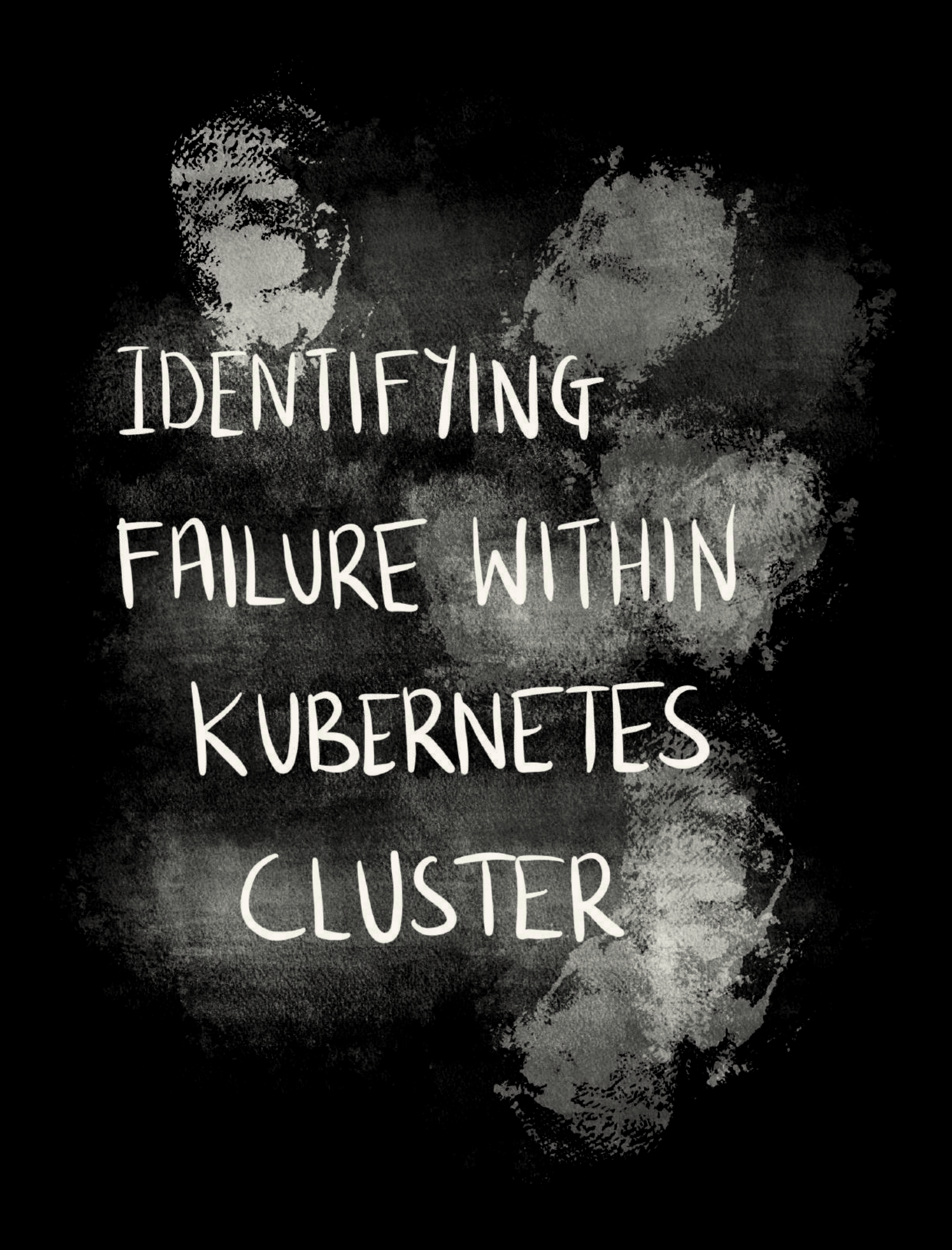
Log directory → /var/log/containers

↑ 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 →

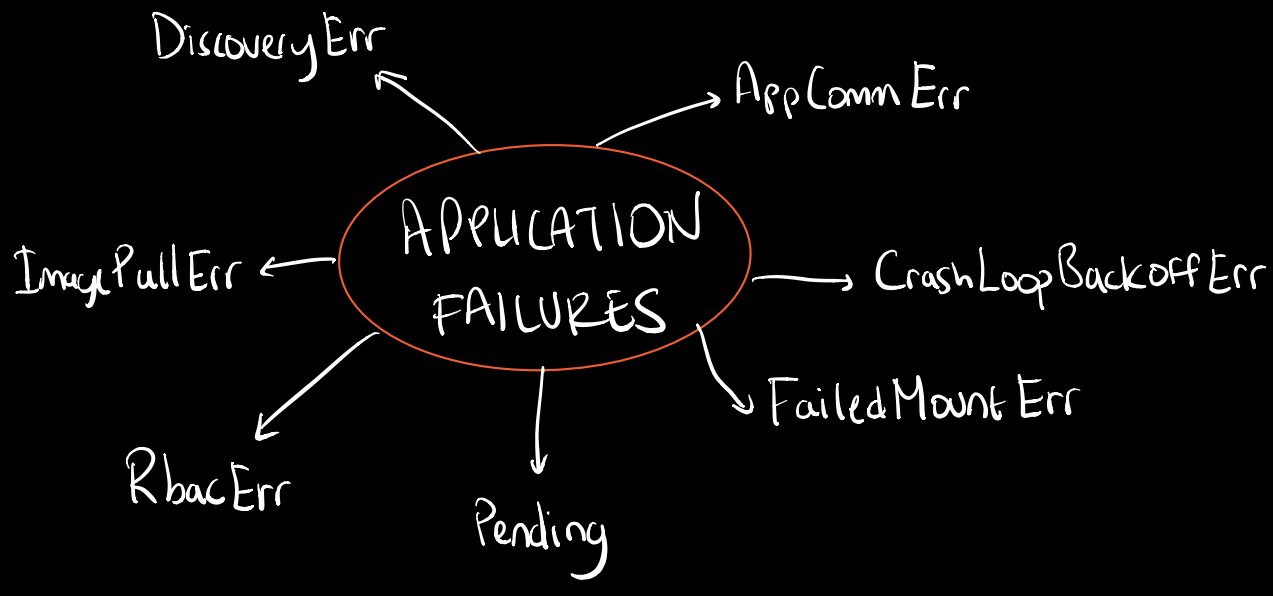
pod1.yaml

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

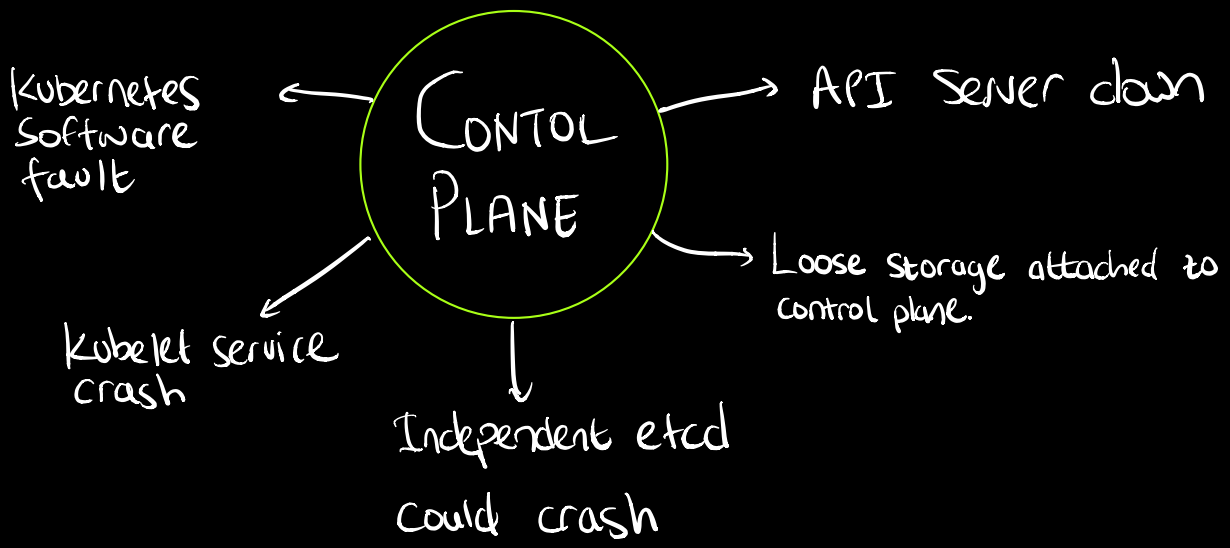
→ kubectl describe ↓ will show error message

→ Only particular fields can be changed i.e Image

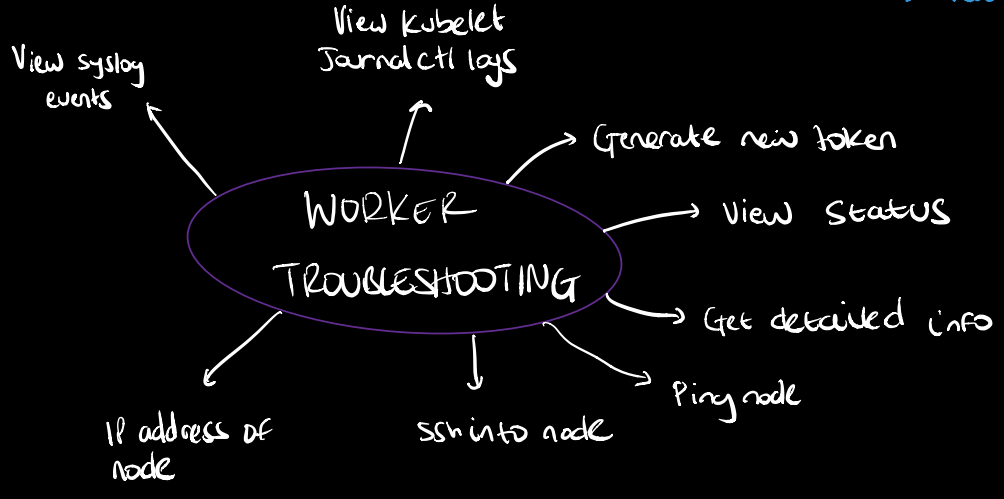
→ To change other fields of failured pod ↓ export configuration → kubectl get po mem2 -o yaml --export ↓ modify yaml ie change memory request

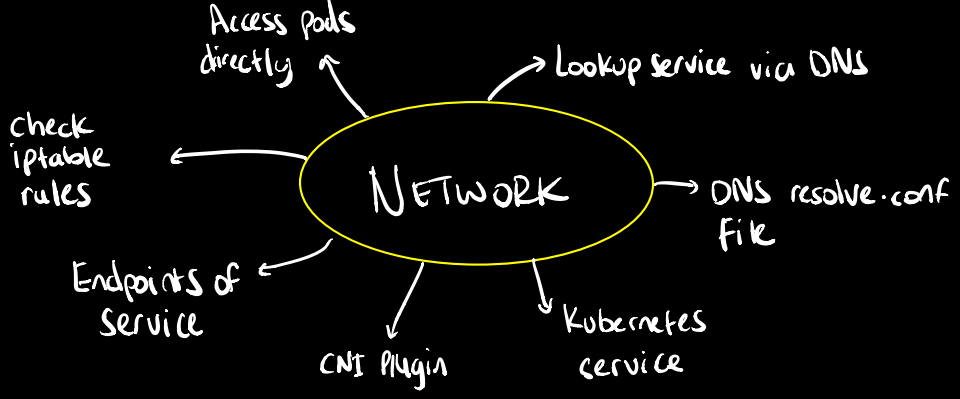


Troubleshoot Failures



- → 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