Contents

Overview

Understanding the Kubernetes Architecture

Building the Kubernetes Cluster

Network Cluster Communications

Pod Scheduling

Deploying Applications

Managing Data

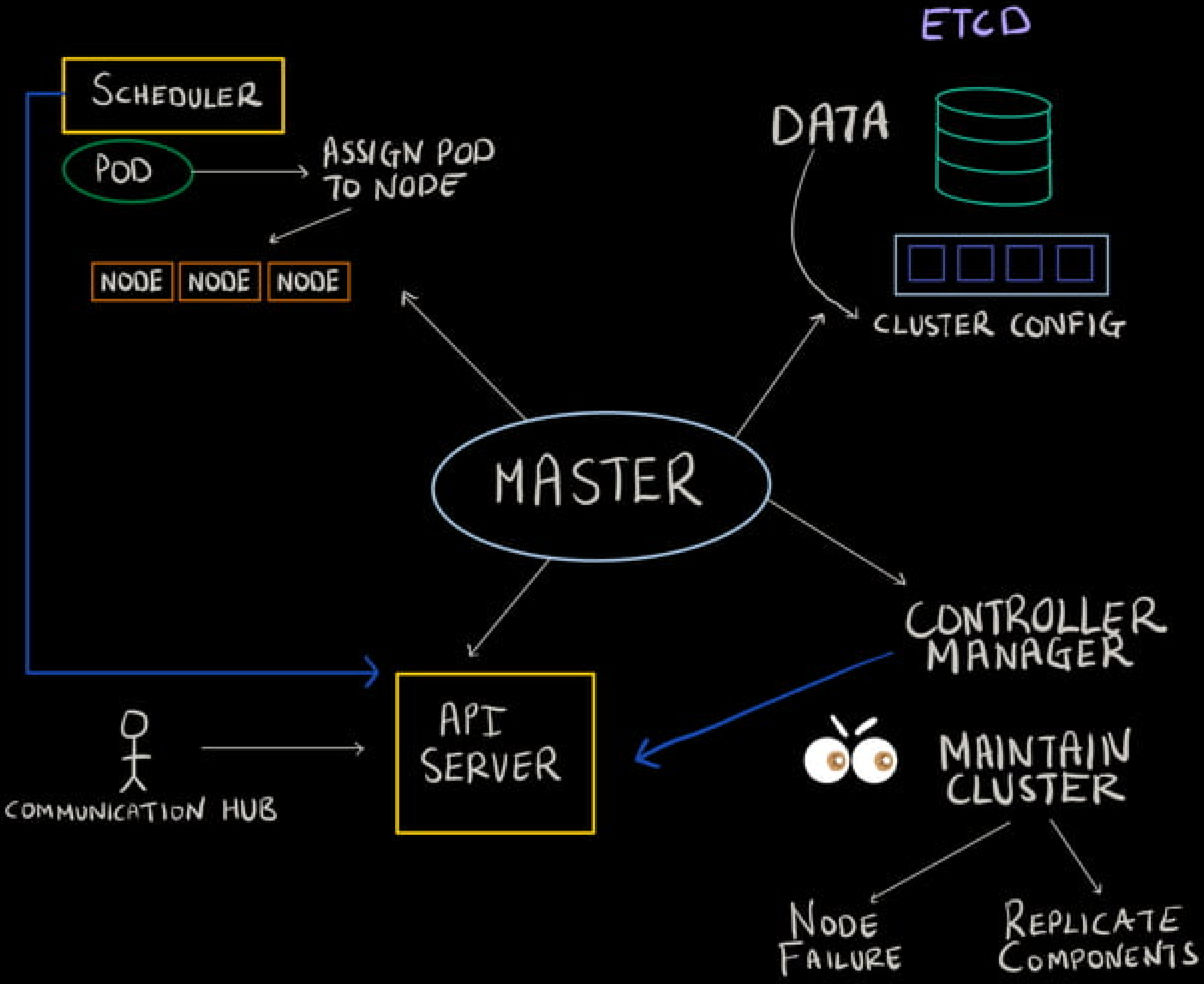
Securing the Kubernetes Cluster

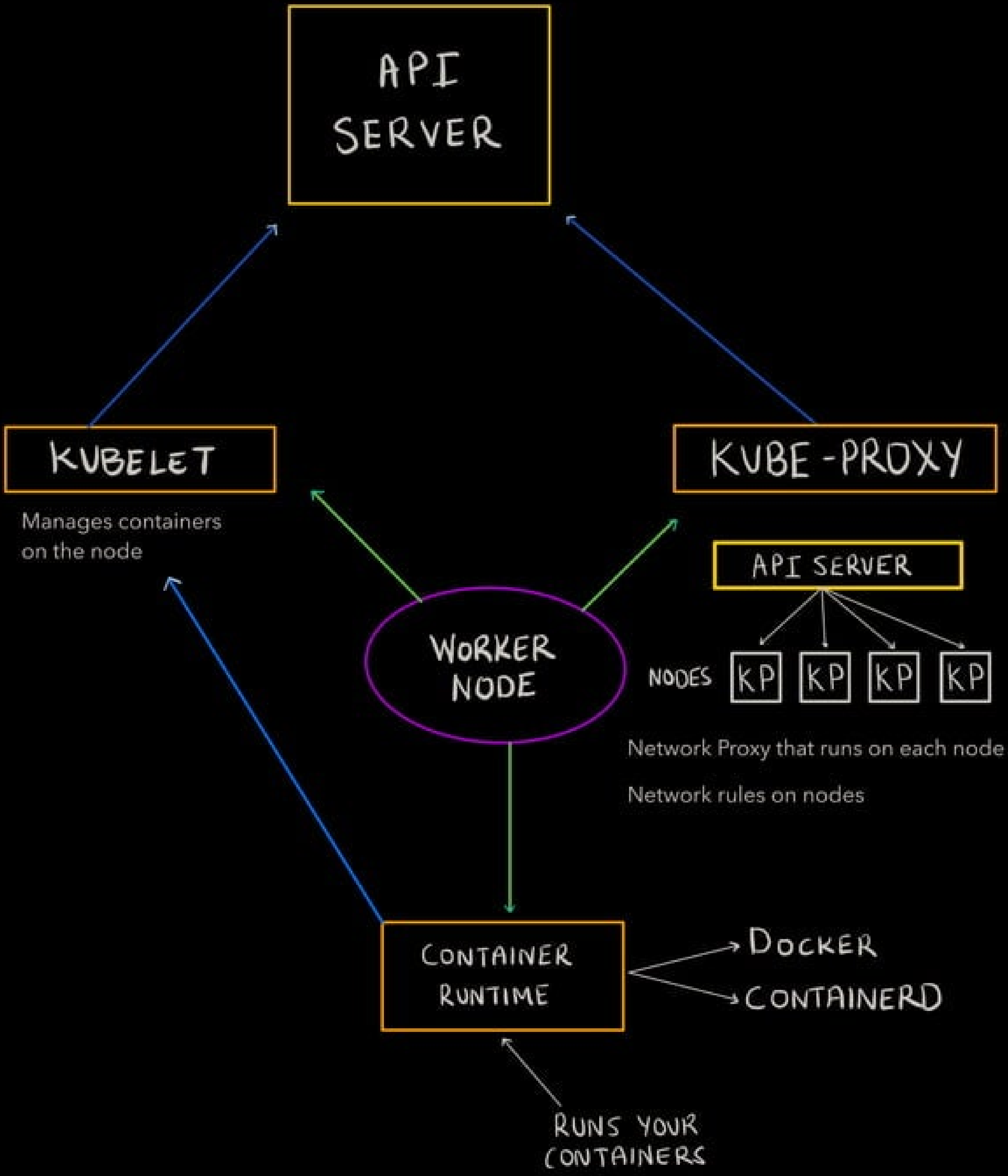
Monitoring Cluster Components

Identifying Failures

Understanding
the
Kubernetes Architecture

Cluster Architecture



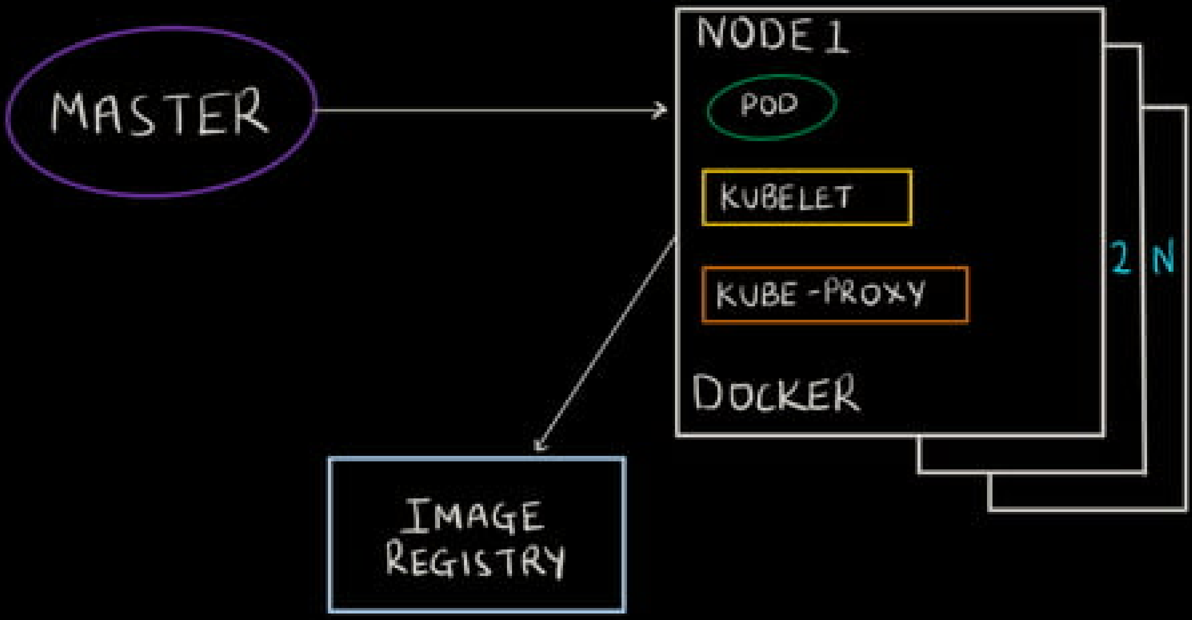


Manages containers on the node

Network Proxy that runs on each node

Network rules on nodes

Application running on Kubernetes

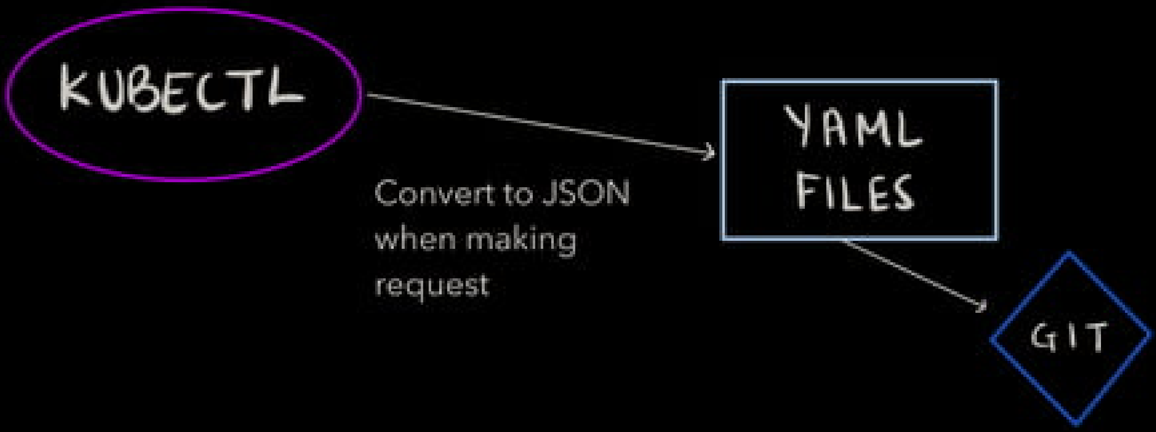


API Primitives

API server is the only one that communicates with Etcd

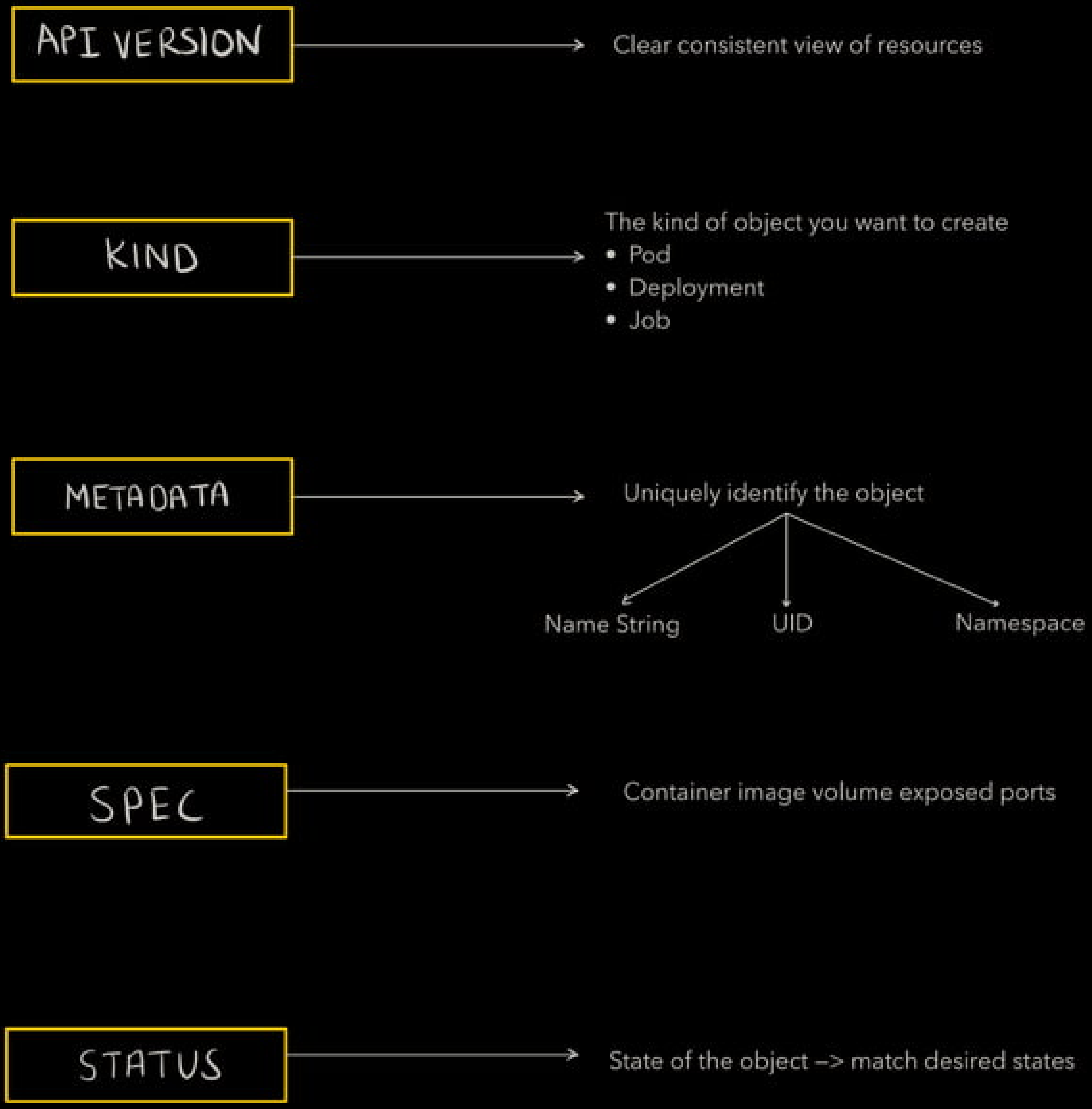
Every component communicates with the API server only and not directly with one another.

Objects like pods and services are declarative intents.



Convert to JSON when making request

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

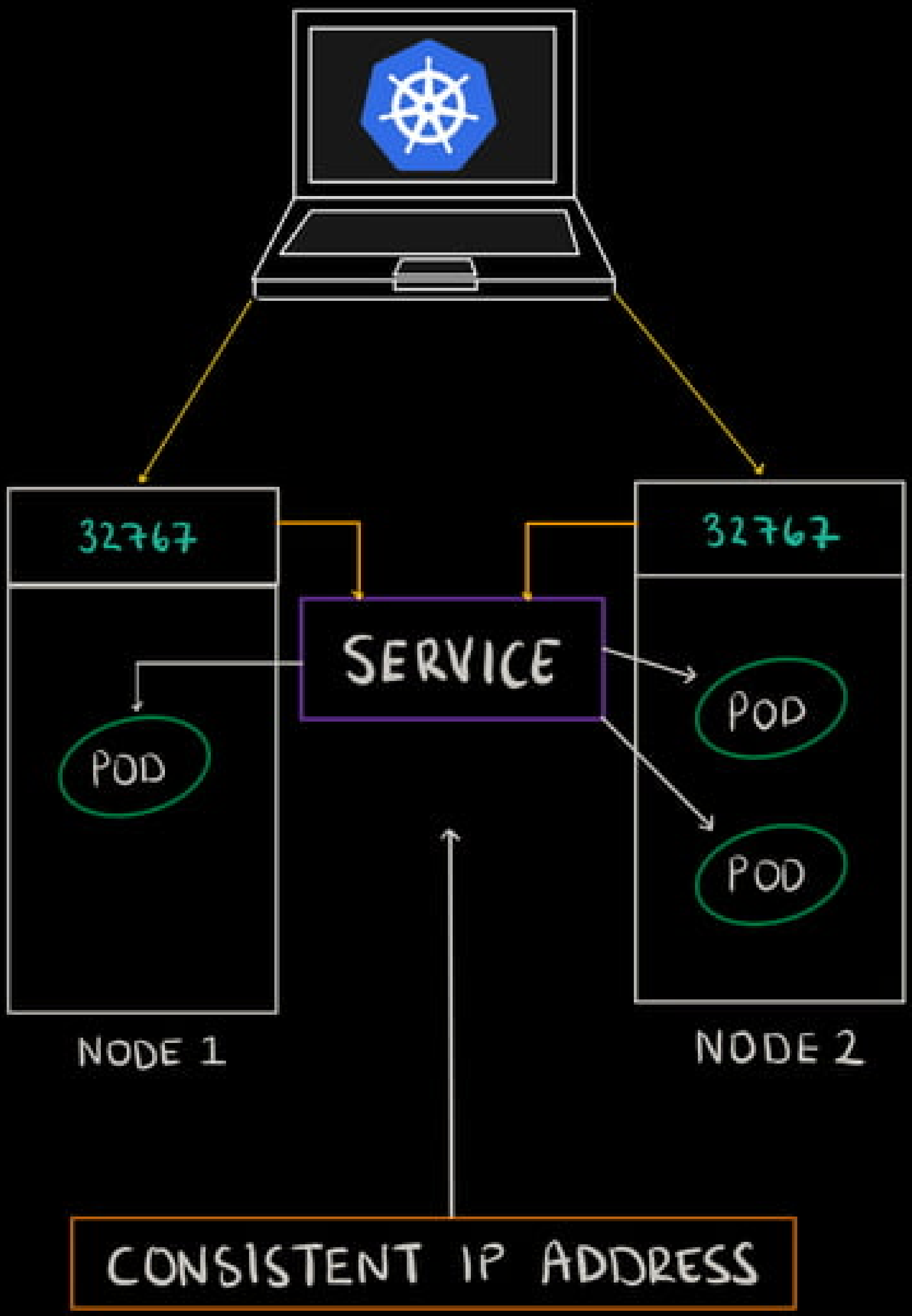
Namespace

SPEC → Container image volume exposed ports

STATUS → State of the object –> match desired states

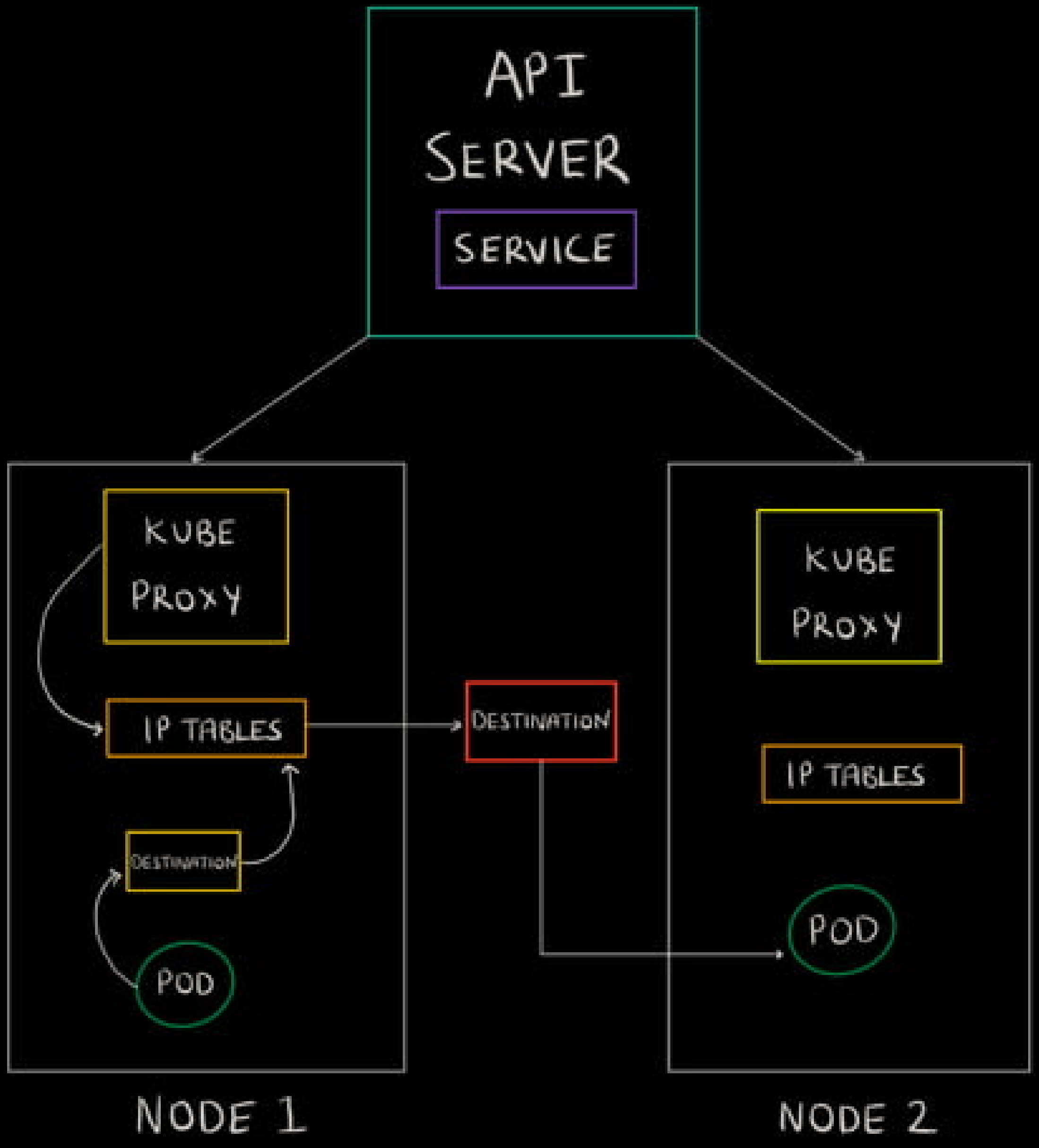
Services and Network Primitives

Services allow you to dynamically access a group of replica pods



Kube-Proxy

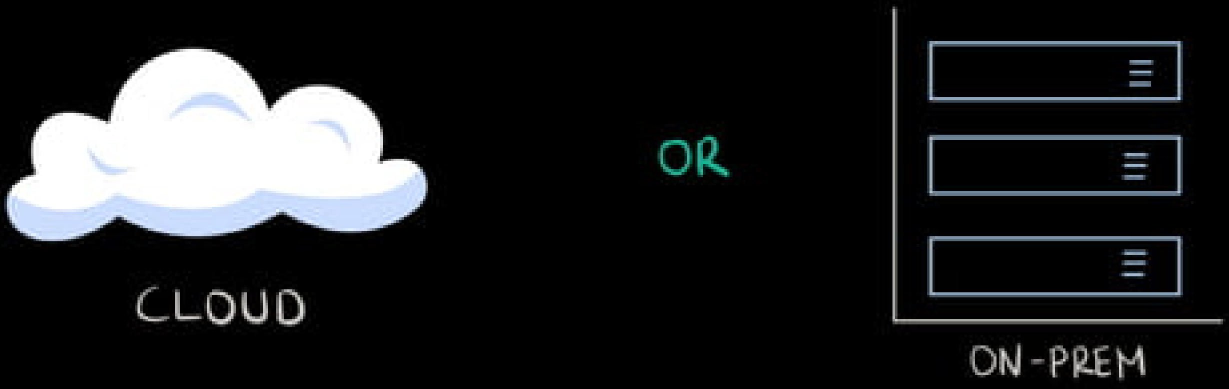
Kube-Proxy handles the traffic associated with a service by creating IP table rules



Building the Kubernetes Cluster

Release Binaries, Provisioning and Types of Clusters

PICKING THE RIGHT SOLUTION



CUSTOM

- Install manually
- Configure your own network fabric
- Locate the release binaries
- Build your own Images
- Secure cluster comms

OR

PRE-BUILT

- Minikube
- Minishift
- Micro K8S
- Ubuntu on LXD
- AWS, Azure and GCP

Installing kubernetes master and nodes

MASTER + WORKERS

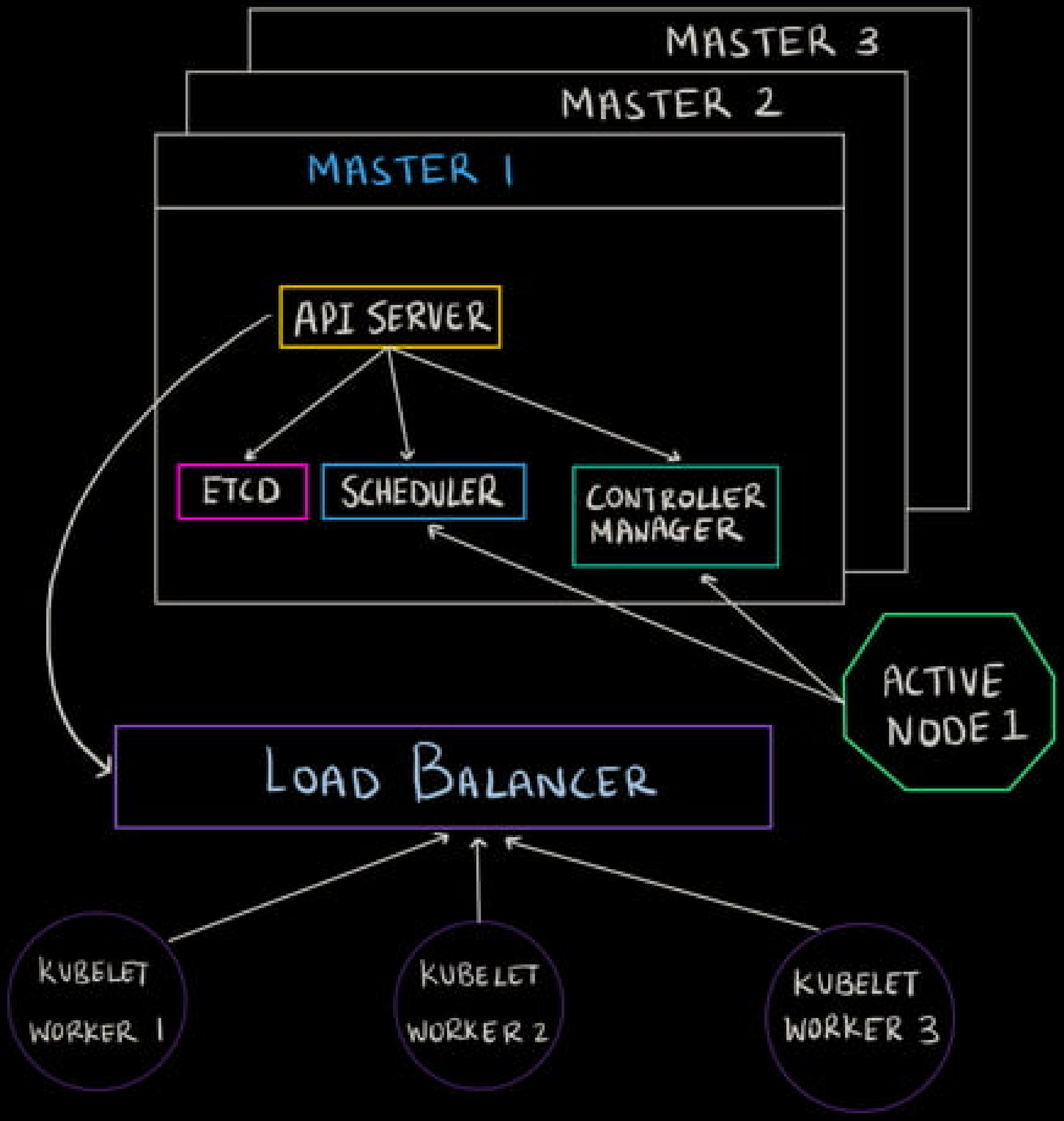
1. DOCKER + KUBERNETES
   - GPG KEY
   - ADD REPOS
2. UPDATE PACKAGES
3. INSTALL DOCKER, KUBELET, KUBEADM, KUBECTL
4. MODIFY BRIDGE ADAPTER SETTINGS

MASTER ONLY

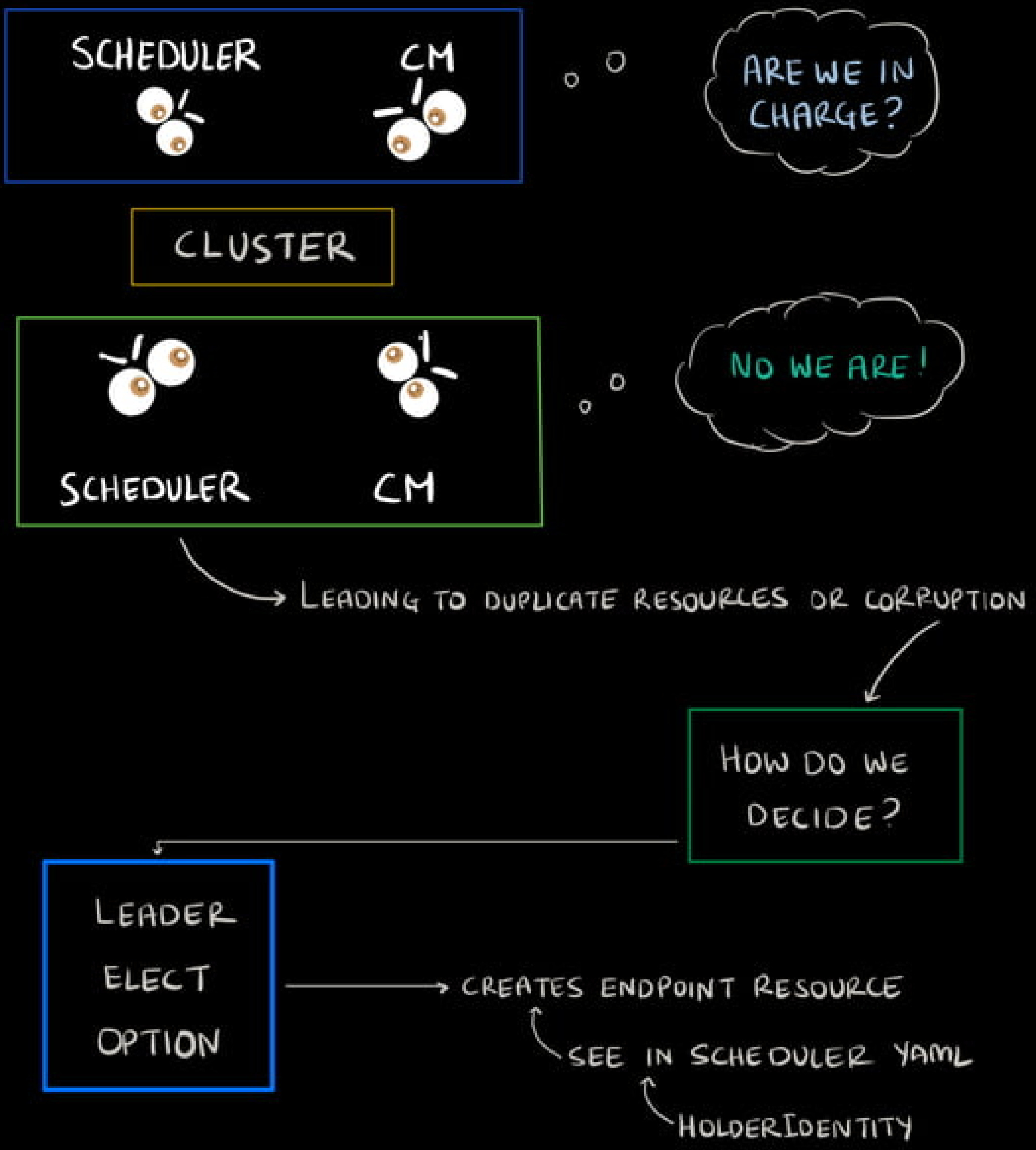
1. INITIALISE CLUSTER
2. MAKE DIRECTORY FOR K8S
3. COPY KUBE CONFIG
4. CHANGE OWERSHIP OF CONFIG
5. APPLY FLANNEL CNI

Building highly available cluster

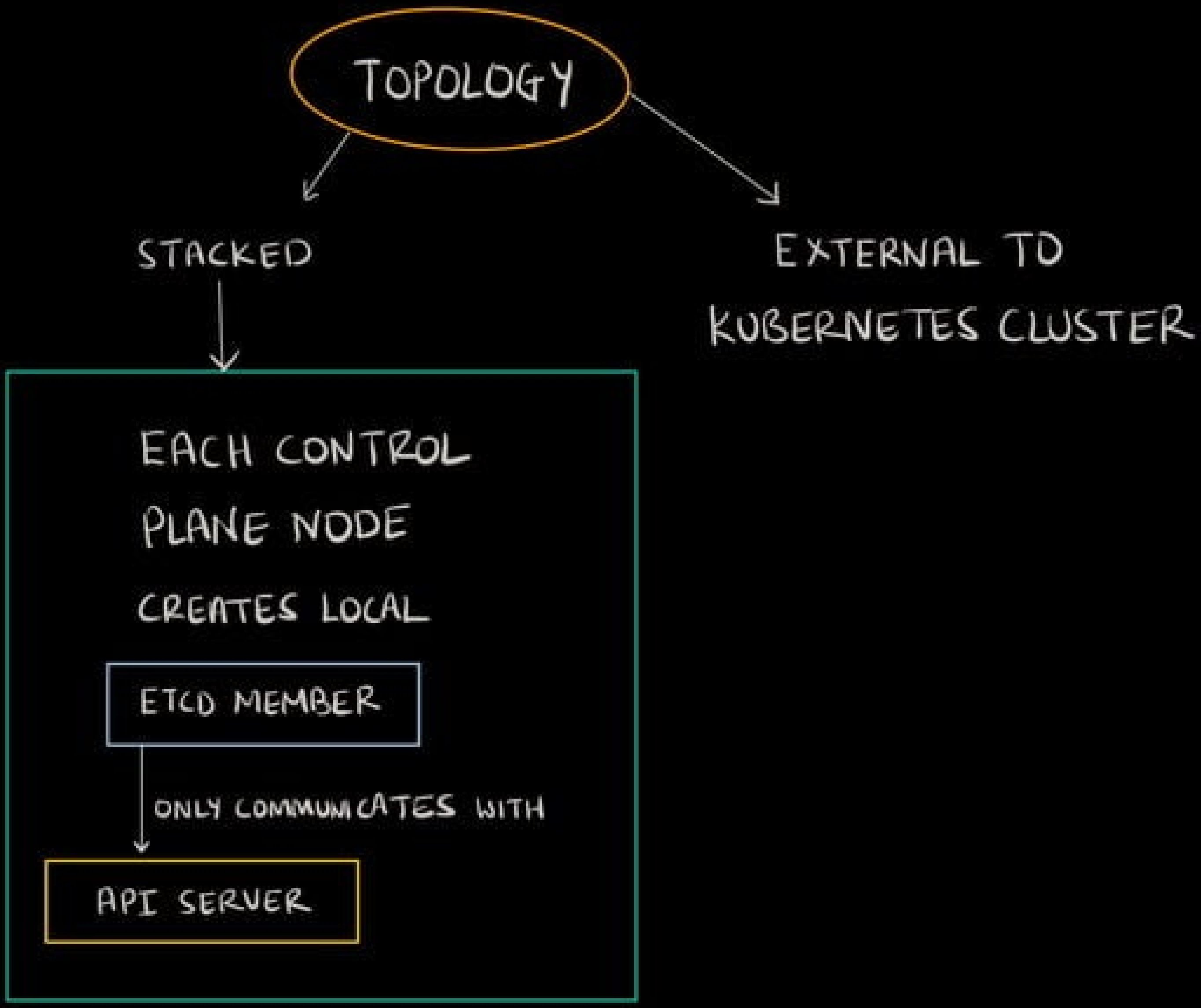
All components can be replicated, but only certain ones can operate simultaneously



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



Replicating etcd



TOPOLOGY

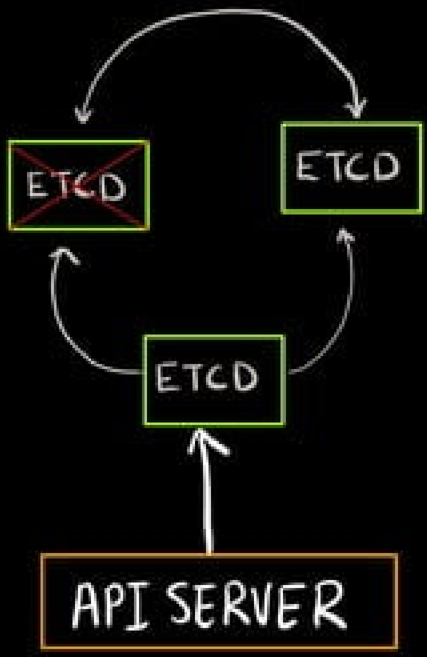
STACKED

EXTERNAL TO KUBERNETES CLUSTER

EACH CONTROL PLANE NODE CREATES LOCAL ETCD MEMBER

ONLY COMMUMICATES WITH

API SERVER



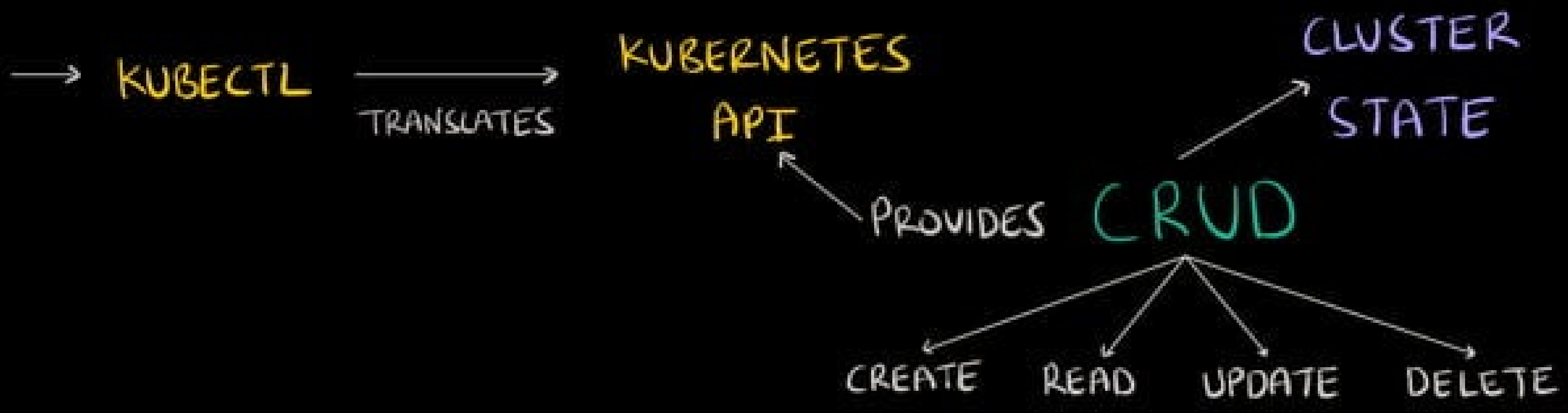
RAFT CONSENSUS ALGORITHM

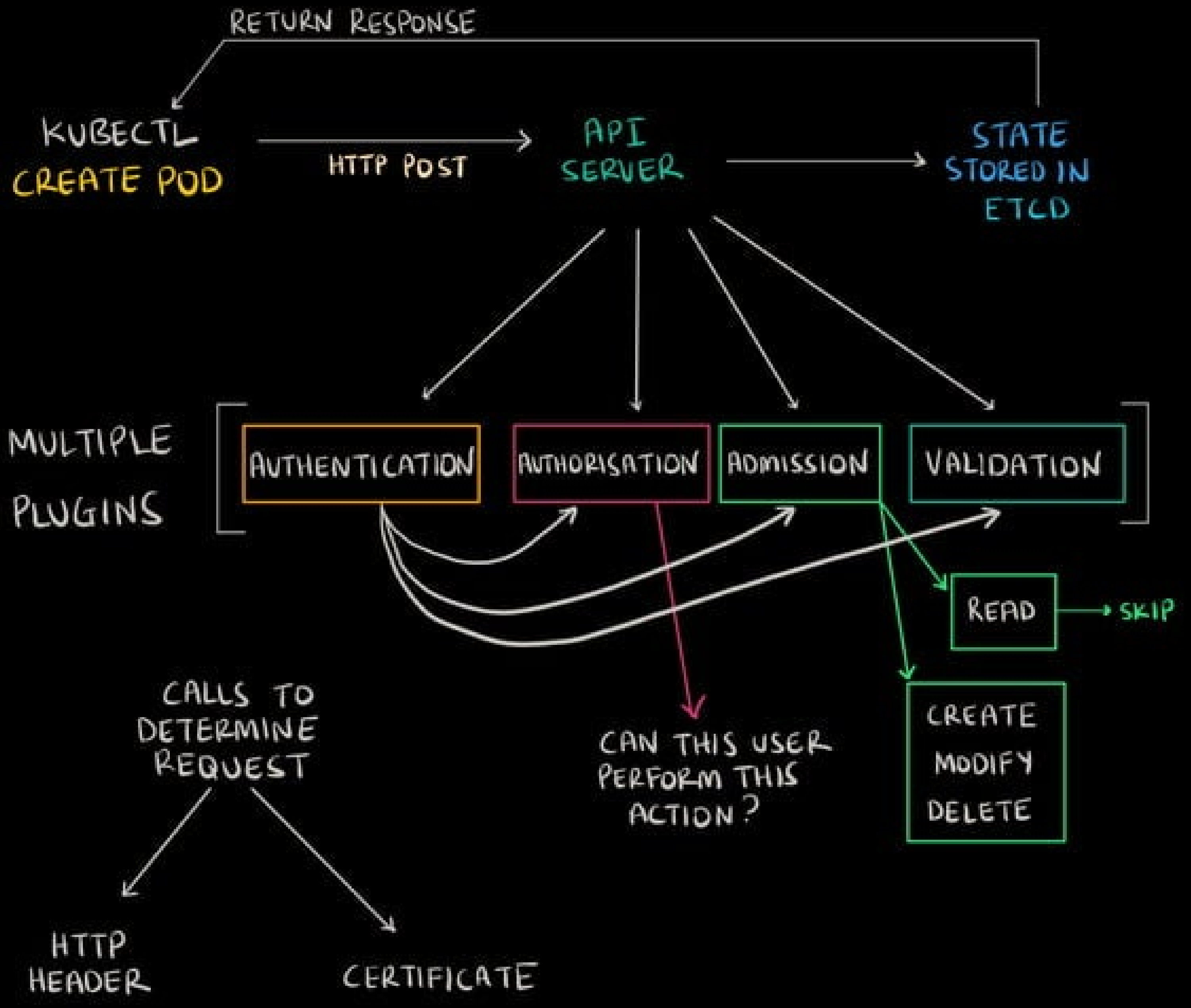
REQUIRES MAJORITY

THERE MUST BE MORE THAN HALF TAKING PLACE IN THE STATE CHANGE AND THEREFORE THERE MUST BE ODD NUMBER OF NODES

Configuring Secure Cluster Communications

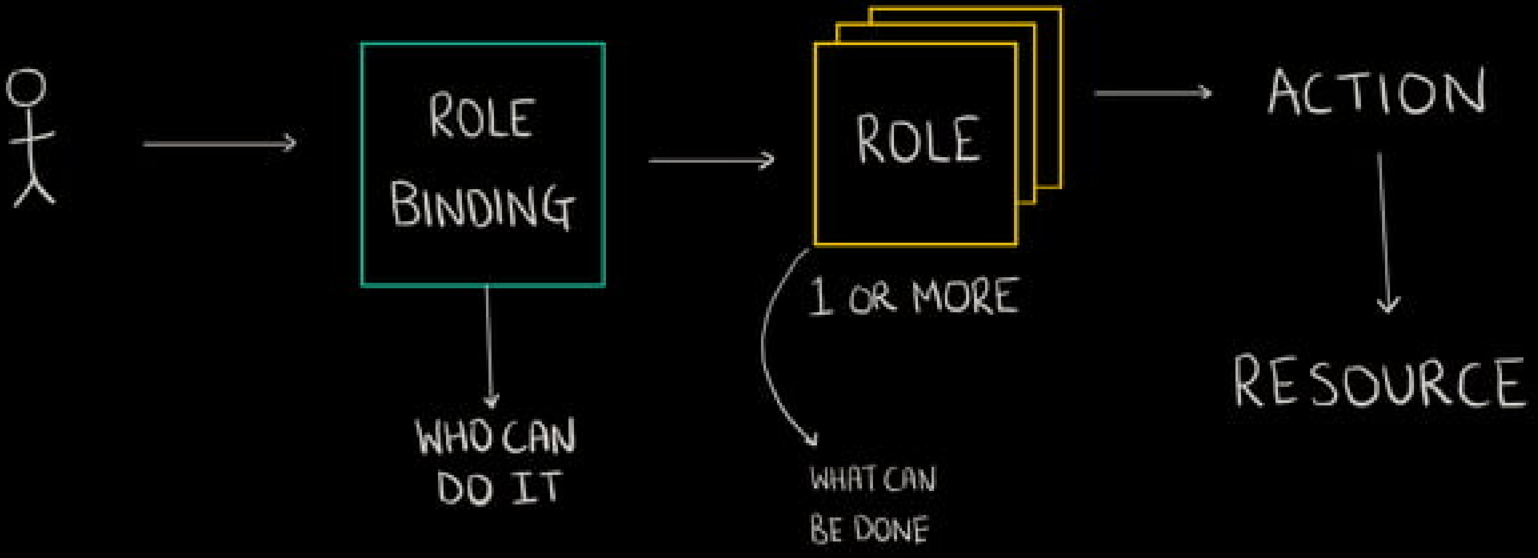
→ ALL COMMUNICATION VIA HTTPS

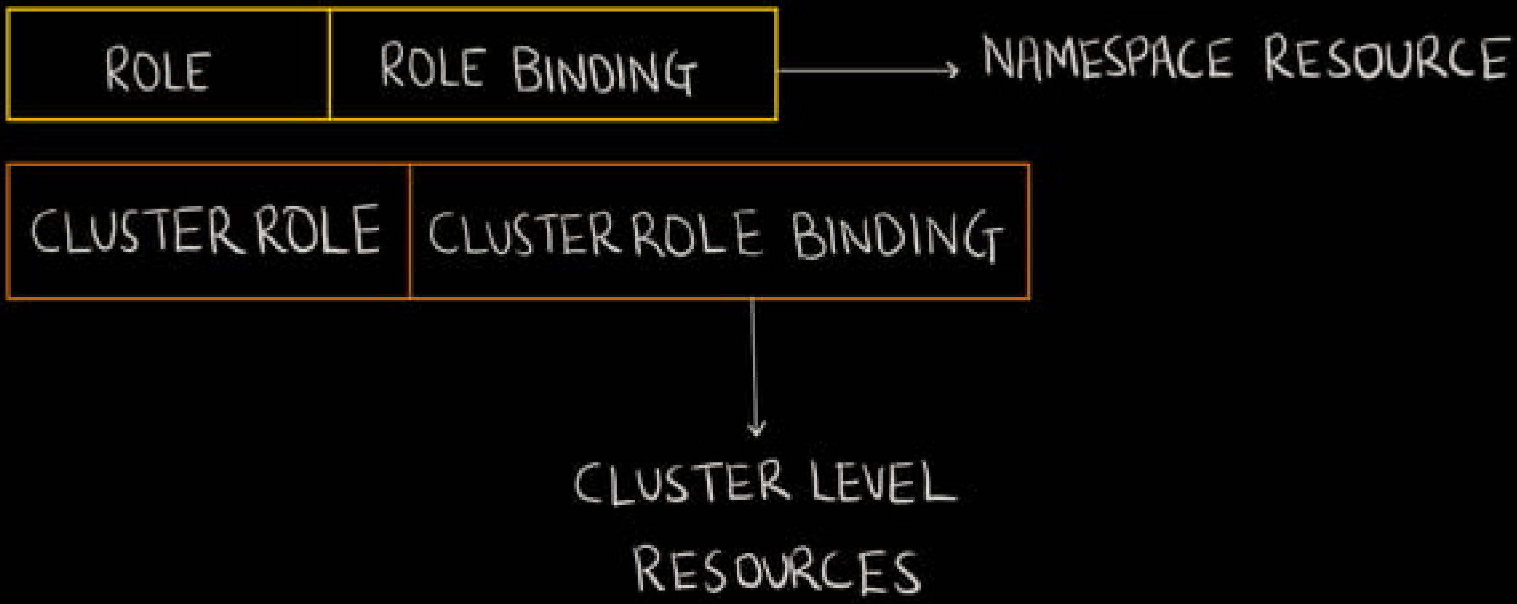




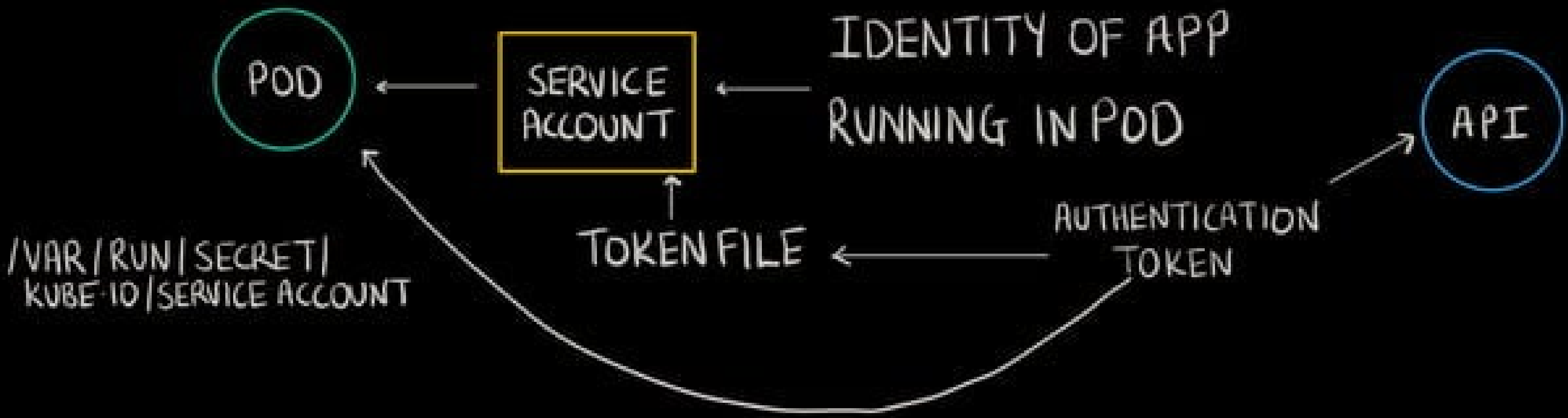
Building Highly Available Cluster

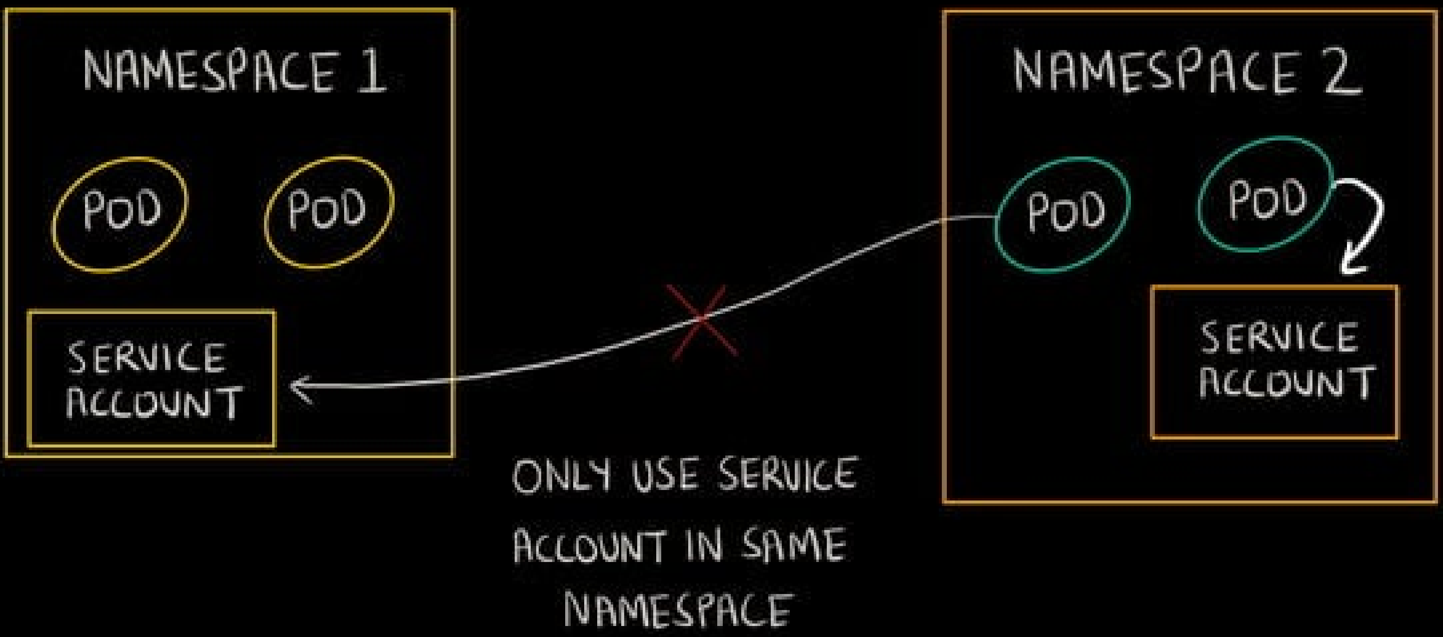
RBAC is used to prevent unauthorised users from modifying the cluster state



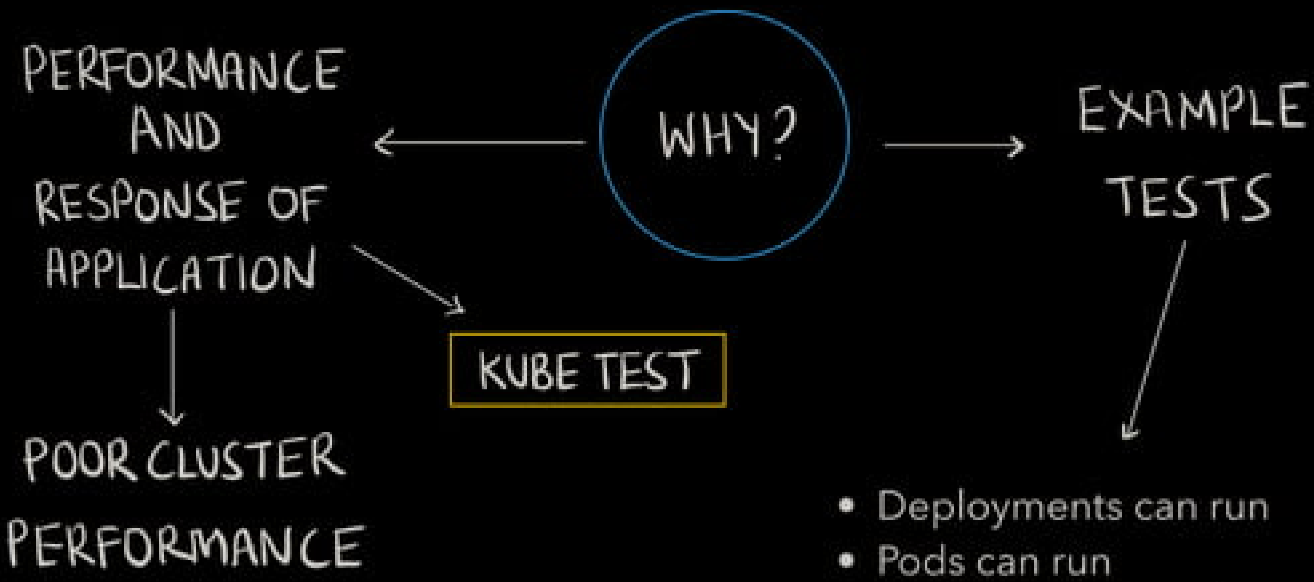


Service Account


POD
SERVICE ACCOUNT
IDENTITY OF APP RUNNING IN POD
API
TOKEN FILE
AUTHENTICATION TOKEN
/VAR/RUN/SECRET/ KUBE.IO/SERVICE ACCOUNT

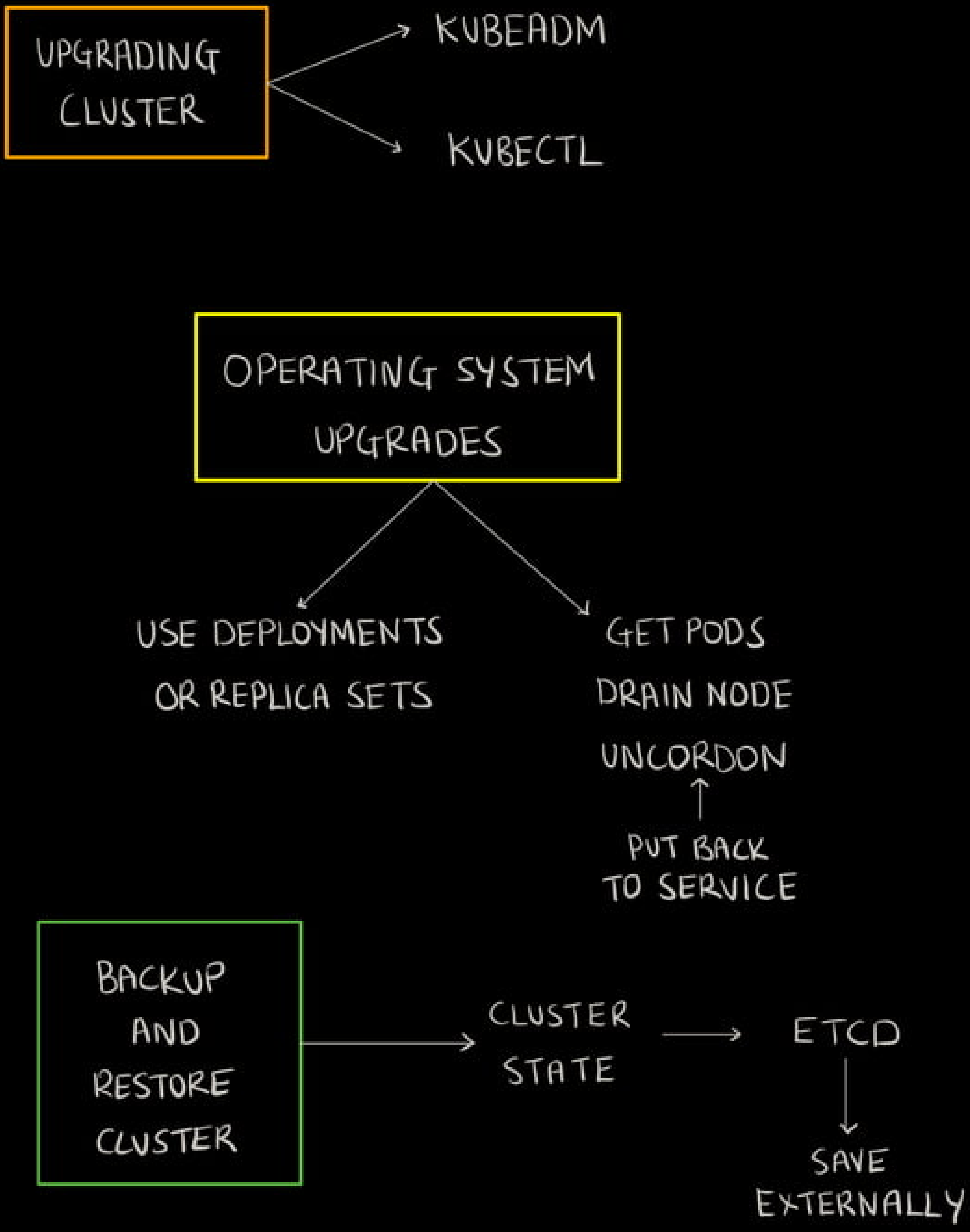


Running end to end tests on cluster



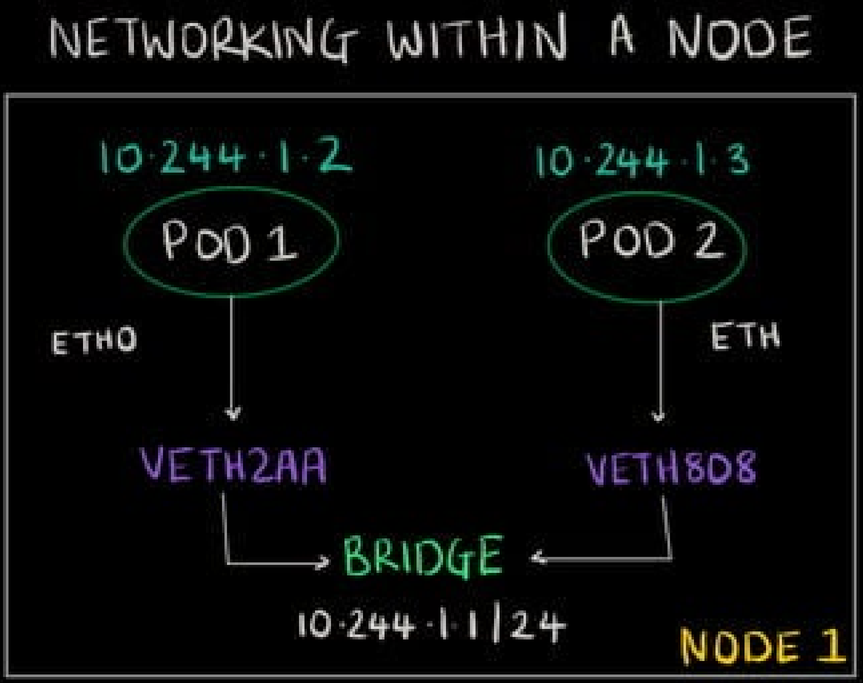
- Deployments can run
- Pods can run
- Pods can be directly accessed
- Logs can be collected
- Commands run from pods
- Services can provide access
- Nodes are healthy
- Pods are healthy

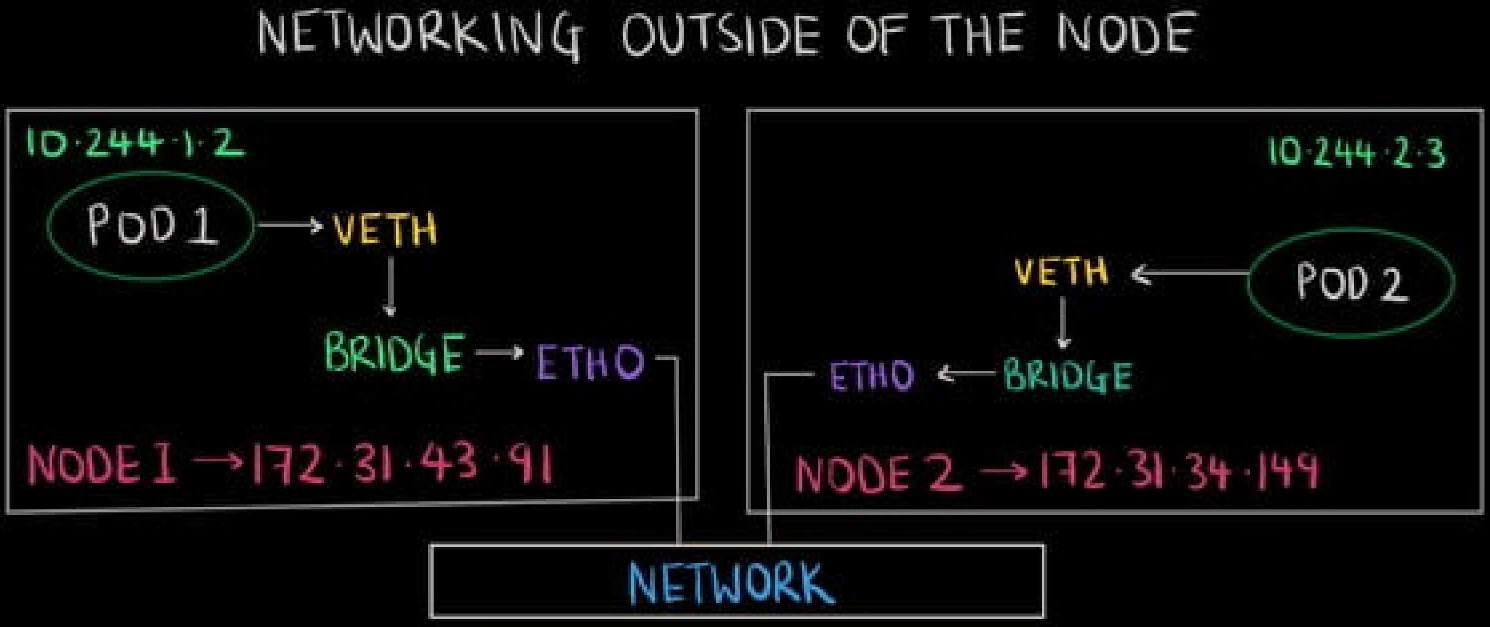
Managing Cluster



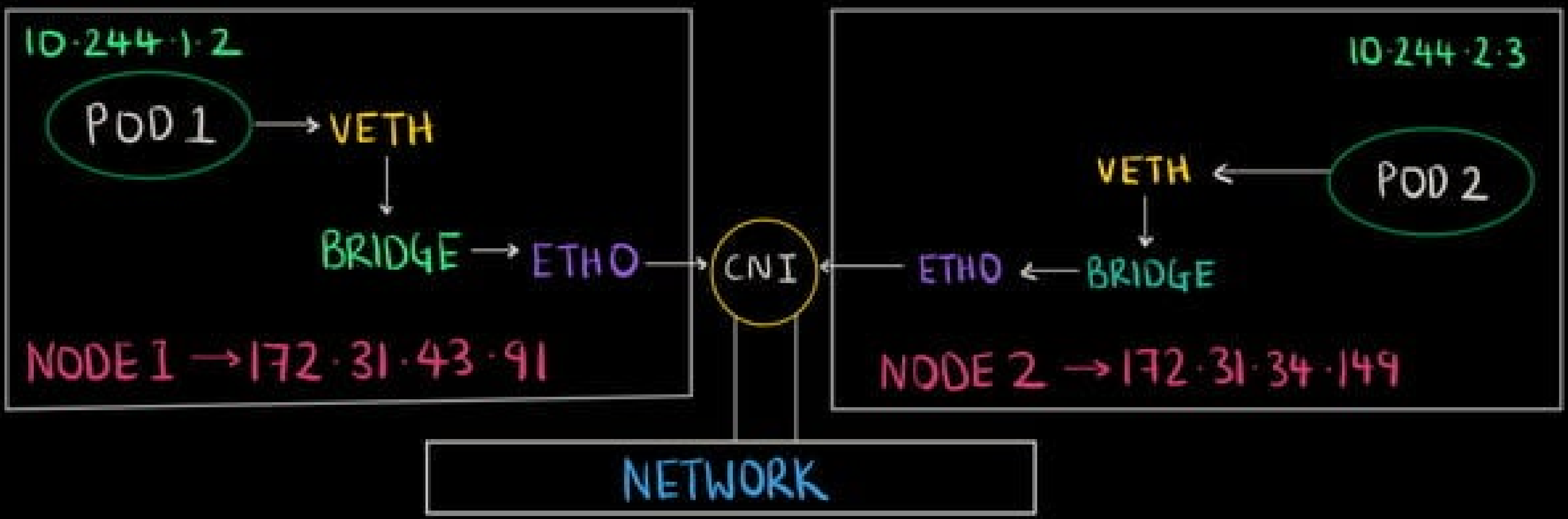
Network Cluster Communication

Pod and Node Networking





Container Network Interface



CNI IS A NETWORK OVERLAY

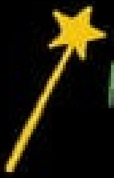
- ALLOWS BUILDING TUNNEL BETWEEN NODES
  - SITS ON TOP OF EXISTING NETWORKS
    - ENCAPSULATES PACKETS

HEADER	DATA	TRAILER

      ADD
      - CHANGES SOURCE/DEST

HOW DOES CNI DO THIS? → THERE IS A MAPPING ASSOCIATED IN USER SPACE

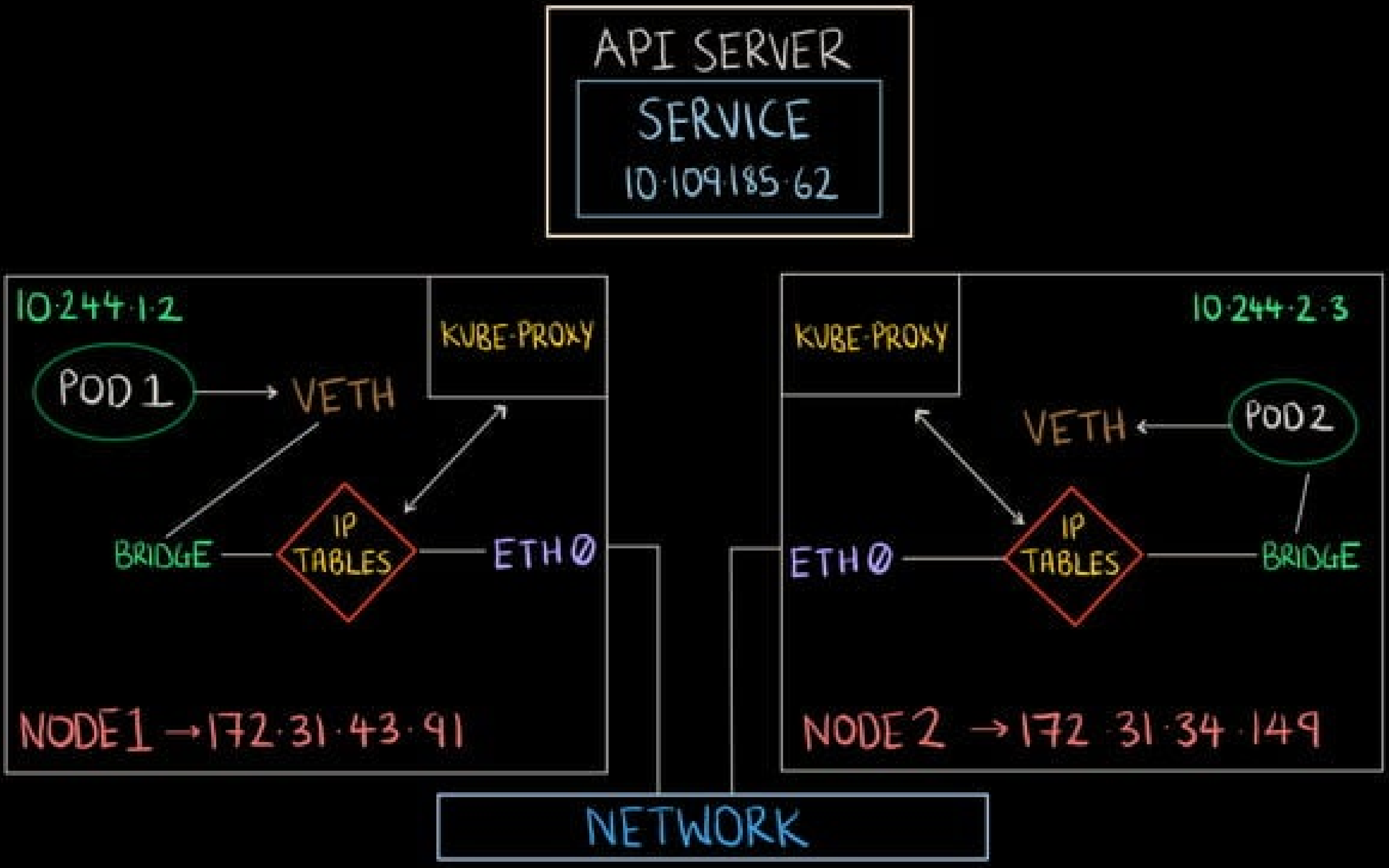
- PROGRAM ALL POD IP ADDRESS'S TO NODE IP, WHEN REACH OTHER NODE, DE-ENCAPSULATE PACKET AND GIVE TO BRIDGE
  - APPEARS LOCAL TO NODE!



EXAMPLE CNI

- CALICO
- FLANNEL

Service Networking



→ PODS COME AND GO

HOW DOES THE CLUSTER KEEP TRACK?

↳ SERVICES

↳ PROVIDES VIRTUAL INTERFACE → AUTO ASSIGNED TO PODS BEHIND INTERFACE

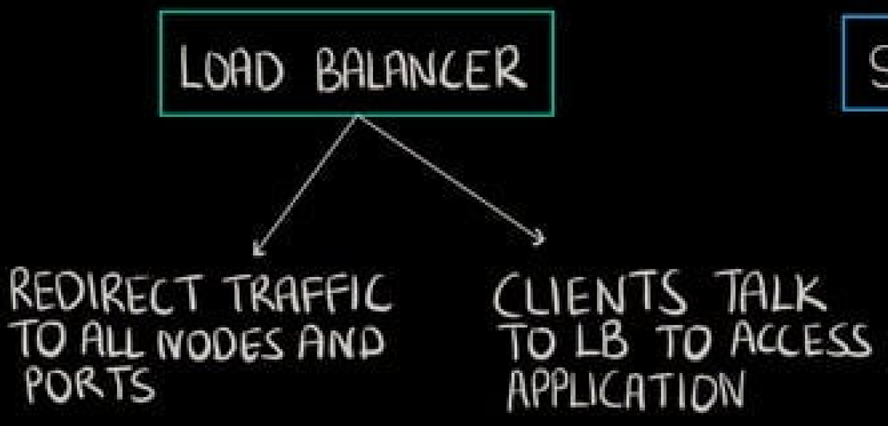
EXAMPLE

CLUSTER IP SERVICE → 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

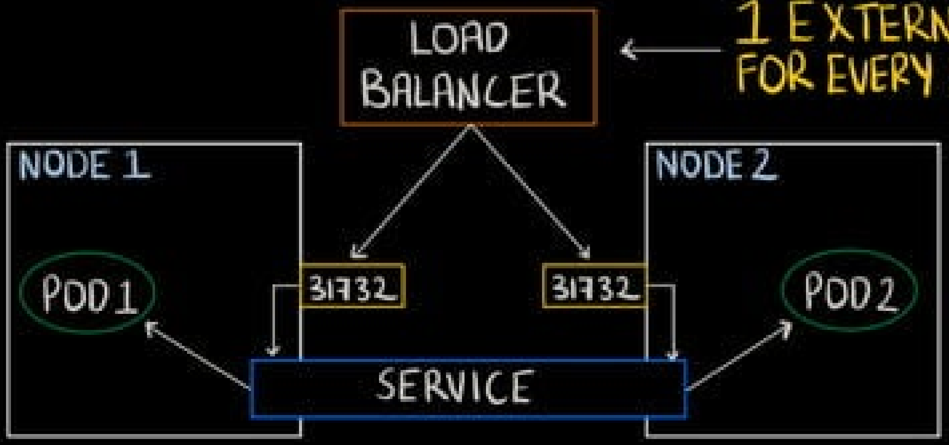
REDIRECT TRAFFIC TO ALL NODES AND PORTS

CLIENTS TALK TO LB TO ACCESS APPLICATION

SERVICE → TALKS TO 1 NODE AT A TIME

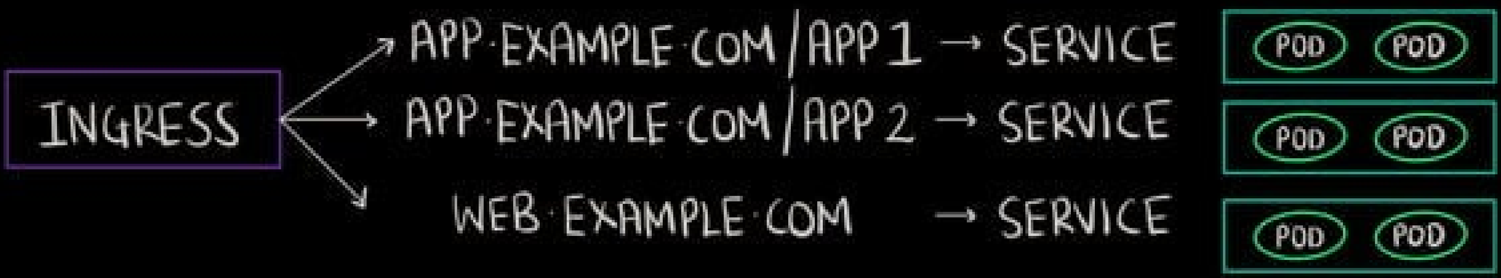
CANNOT SPLIT TRAFFIC LIKE A LOAD BALANCER

ONLY ACCESSIBLE INTERNALLY → DOES NOT HAVE EXTERNAL IP



1 EXTERNAL IP ADDRESS FOR EVERY SERVICE

Ingress



APP.EXAMPLE.COM/APP1 → SERVICE

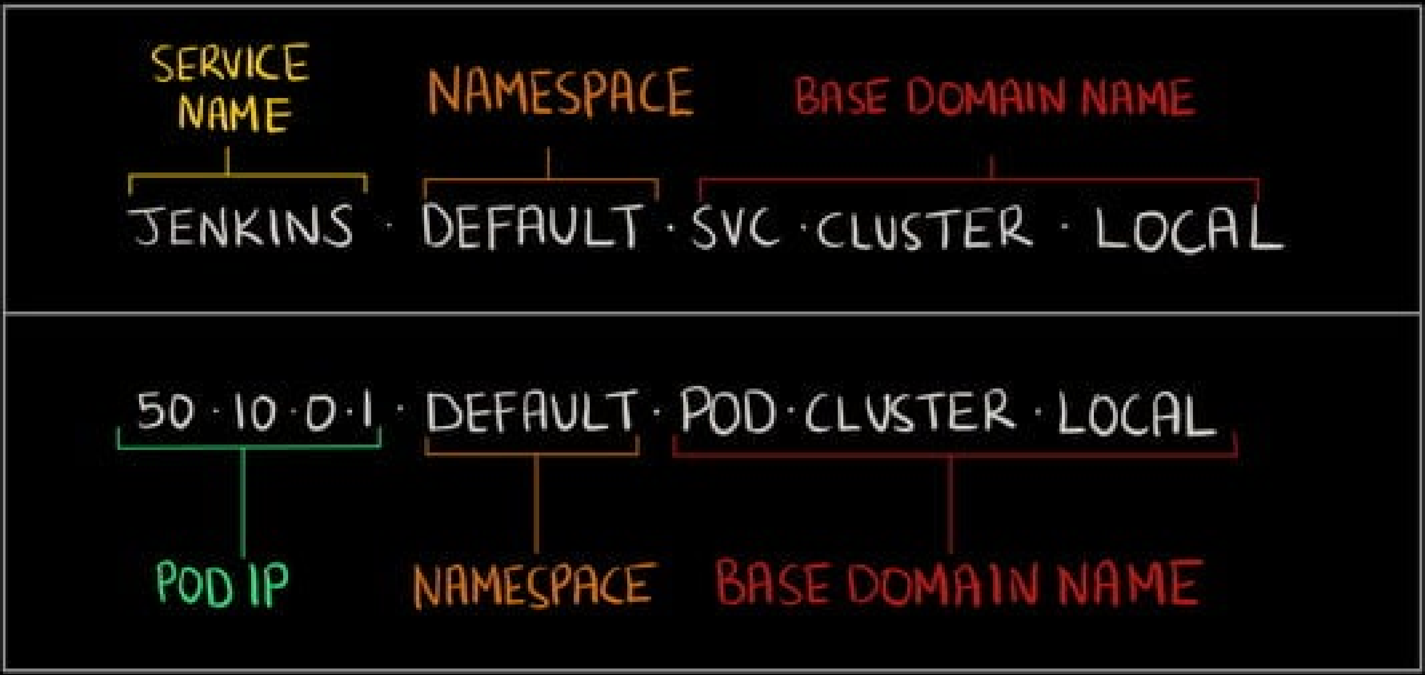
APP.EXAMPLE.COM/APP2 → SERVICE

WEB.EXAMPLE.COM → SERVICE

ACCESS MULTIPLE SERVICES WITH SINGLE IP ADDRESS

Cluster DNS

EVERY SERVICE DEFINED IN THE CLUSTER IS ASSIGNED A DNS NAME


SERVICE NAME
NAMESPACE
BASE DOMAIN NAME
JENKINS · DEFAULT · SVC · CLUSTER · LOCAL
50 · 10 · 0 · 1 · DEFAULT · POD · CLUSTER · LOCAL
POD IP
NAMESPACE
BASE DOMAIN NAME

A PODS DNS SEARCH WILL INCLUDE THE PODS OWN NAMESPACE AND THE CLUSTERS DEFAULT DOMAIN

Pod Scheduling

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?

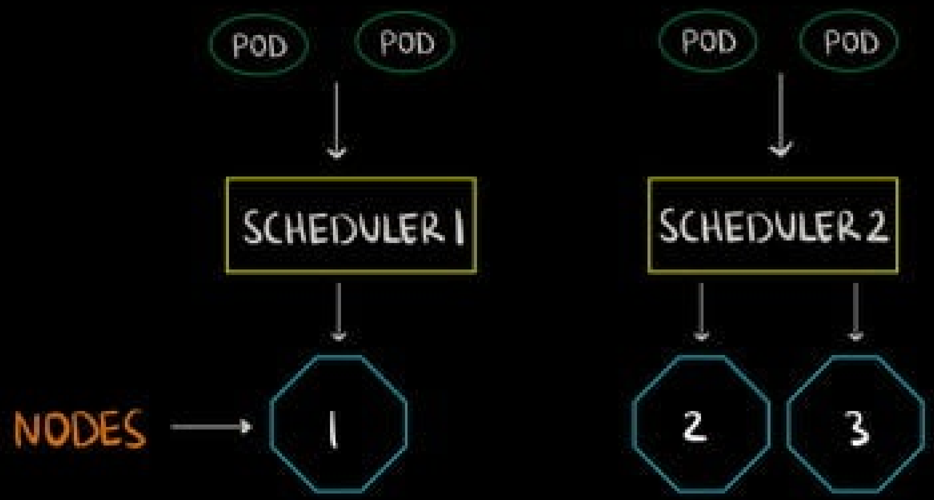
- 💰 SAME NODE TO SAVE COSTS
- WORKER NODES HAVE DIFFERENT DISKS

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 REQUEST 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 NOD OR POD AFFINITY?

Running multiple schedulers for multiple Pods

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



Scheduling 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 → POD MAY NOT BE USING ALL REQUESTED RESOURCE AT A GIVEN TIME

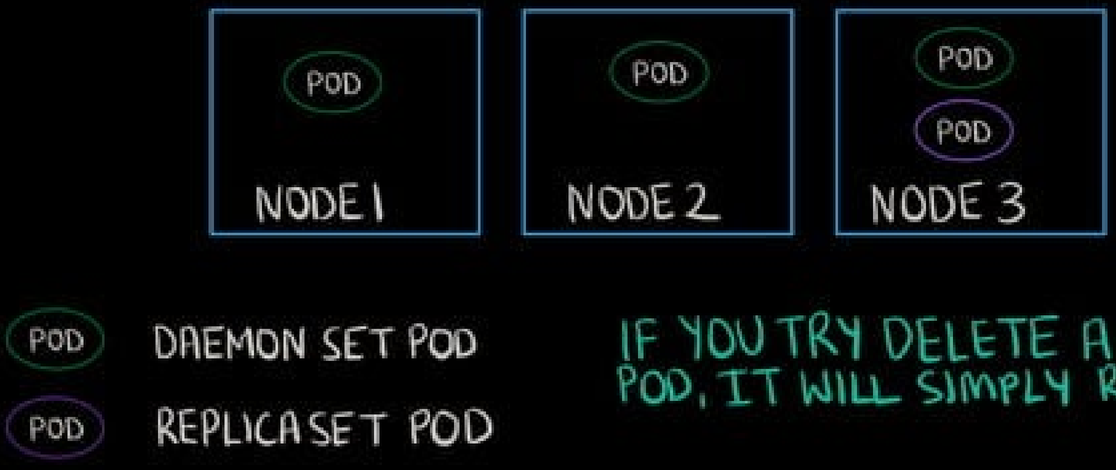
CPU | MEMORY → SCHEDULER LOOKS AT THE SUM OF RESOURCES REQUESTED BY EXISTING PODS

SCHEDULER → POST 8 STEPS → MOST REQUESTED PRIORITY / LEAST REQUESTED PRIORITY

MOST REQUESTED PRIORITY ← WHY? ← CLOUD ENVIRONMENTS YOU ARE PAYING FOR ALL RESOURCES

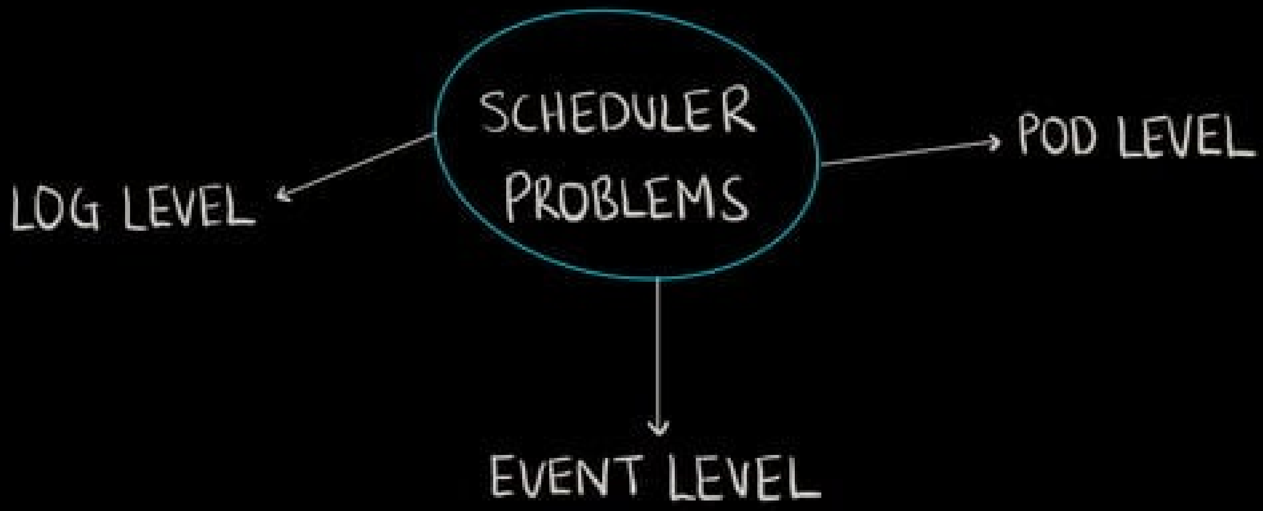
DaemonSets

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



IF YOU TRY DELETE A DAEMONSET POD, IT WILL SIMPLY RECREATE IT

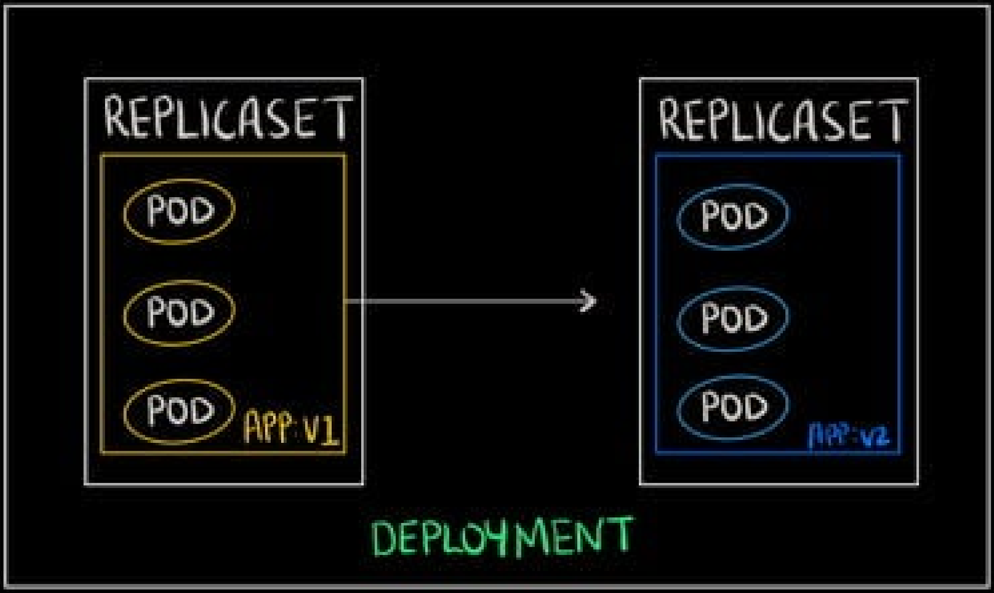
Display Scheduler Events



Deploying Applications

Deploying an Application, Rolling Updates and Rollbacks

DEPLOYMENTS → HIGH LEVEL RESOURCE FOR DEPLOYING AND UPDATING APPS



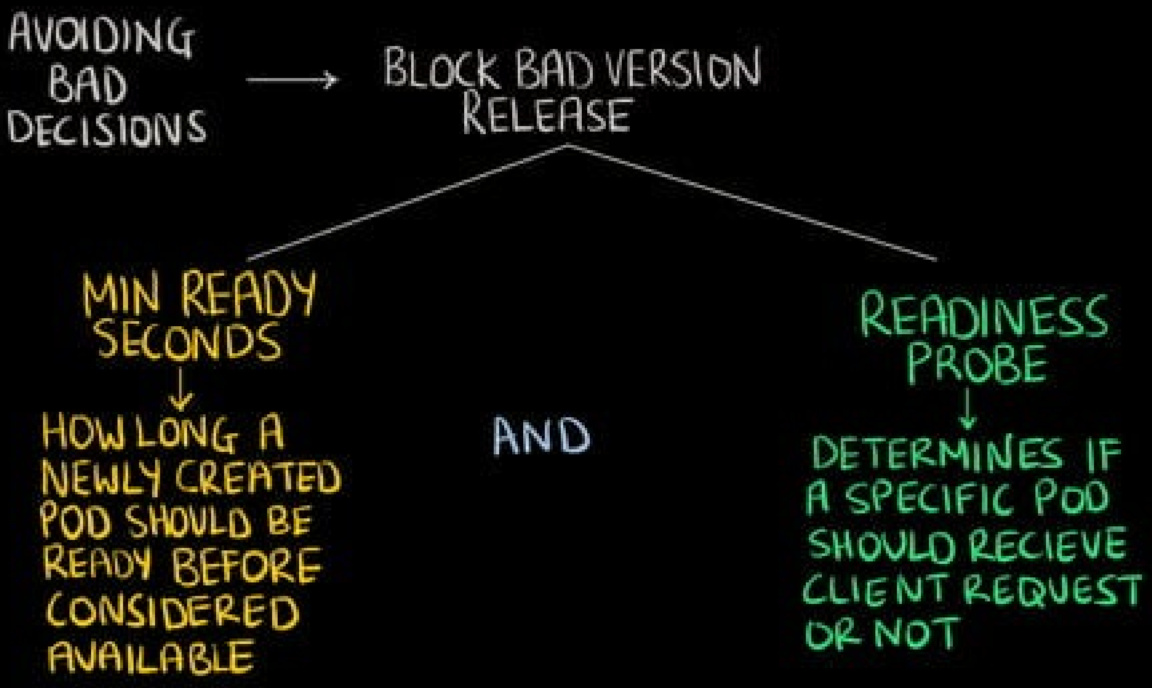
KUBECTL APPLY → MODIFY OBJECTS TO EXISTING YAML AND IF DEPLOYMENT NOT CREATED → ALSO CREATE

KUBECTL REPLACE → REPLACES OLD WITH NEW AND OBJECT MUST EXIST

ROLLING UPDATE → PREFERRED WAY → SERVICE NOT INTERRUPTED
→ FASTEST WAY

KUBECTL ROLLOUT → ROLLBACK PREVIOUS VERSION

Configuring an App for HA and Scale

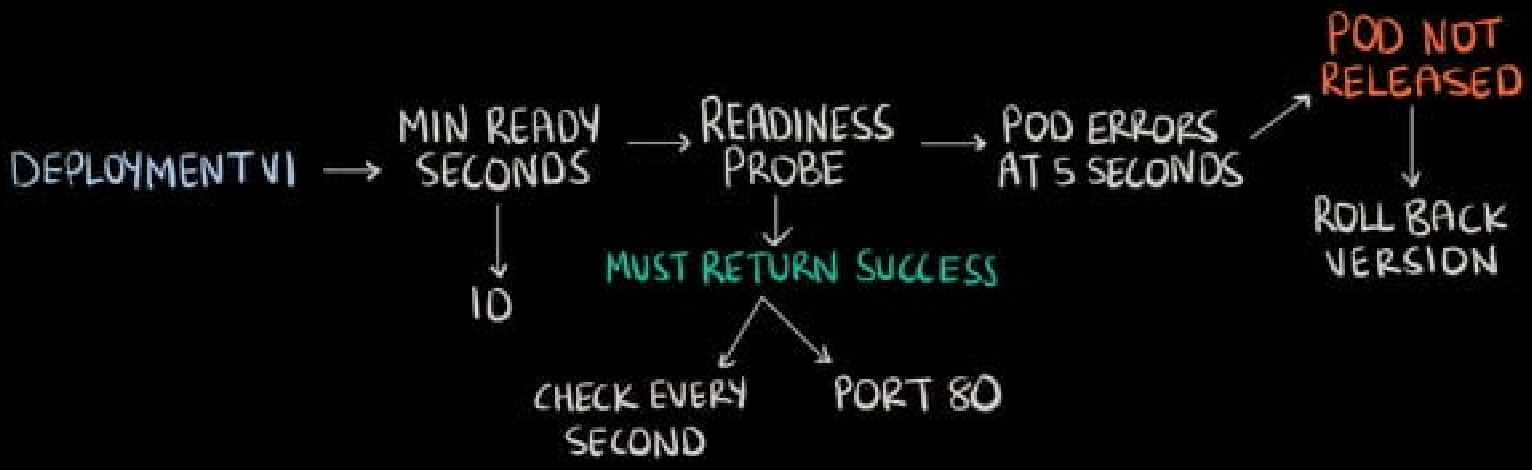


AVOIDING BAD DECISIONS → BLOCK BAD VERSION RELEASE

MIN READY SECONDS → HOW LONG A NEWLY CREATED POD SHOULD BE READY BEFORE CONSIDERED AVAILABLE

AND

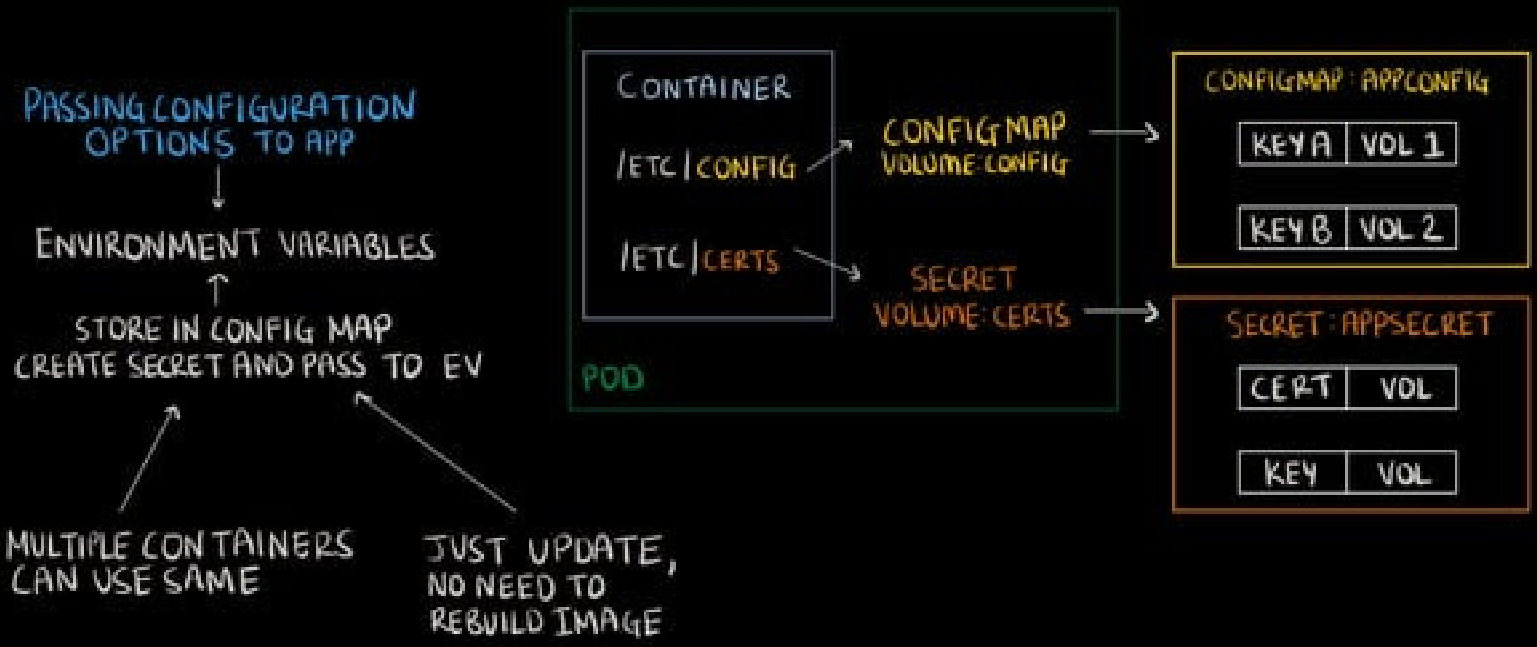
READINESS PROBE → DETERMINES IF A SPECIFIC POD SHOULD RECIEVE CLIENT REQUEST OR NOT



DEPLOYMENT V1 → MIN READY SECONDS → READINESS PROBE → POD ERRORS AT 5 SECONDS → POD NOT RELEASED → ROLL BACK VERSION

MIN READY SECONDS → 10

READINESS PROBE → MUST RETURN SUCCESS → CHECK EVERY SECOND / PORT 80



PASSING CONFIGURATION OPTIONS TO APP → ENVIRONMENT VARIABLES

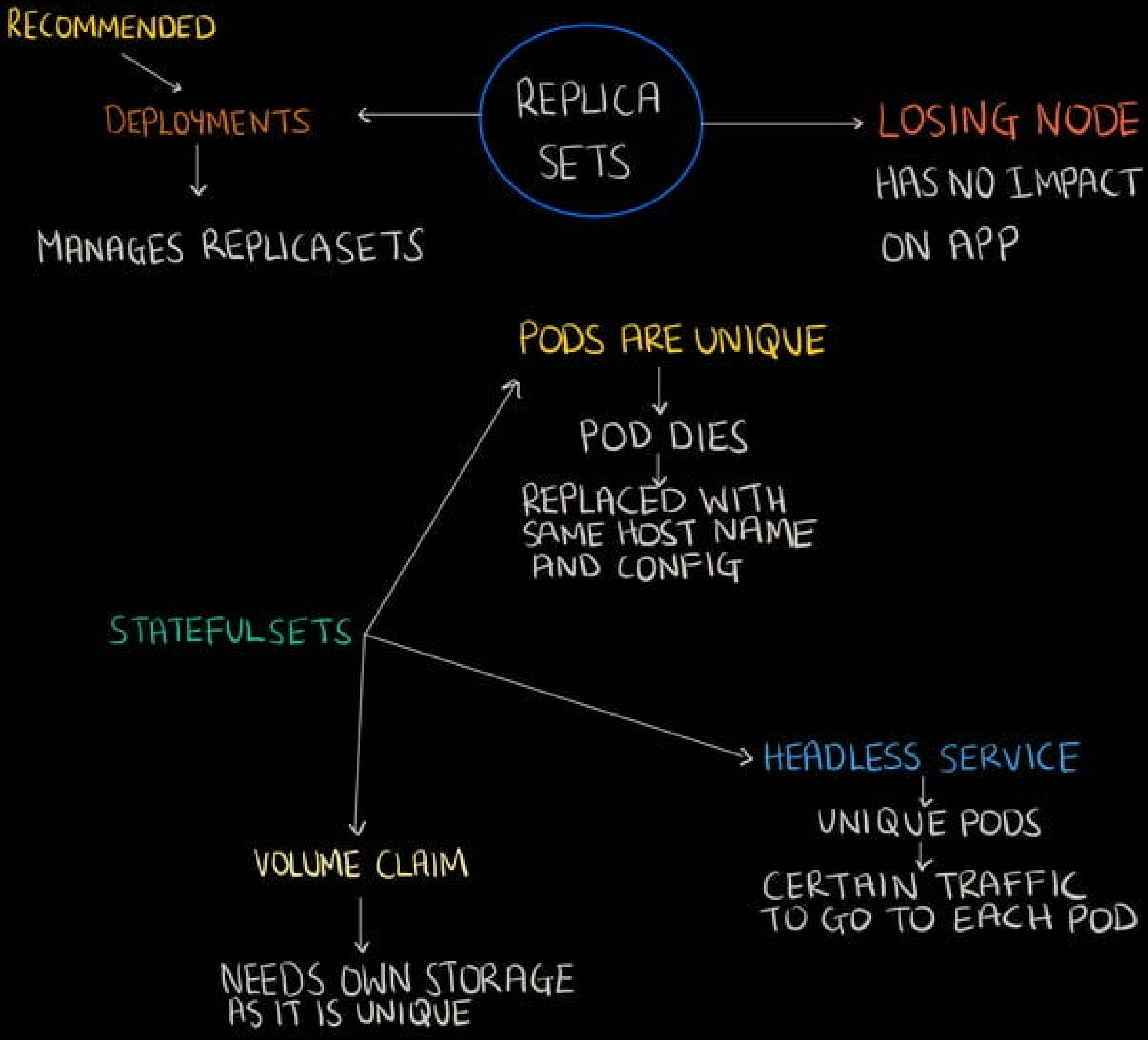
STORE IN CONFIG MAP
CREATE SECRET AND PASS TO EV

MULTIPLE CONTAINERS CAN USE SAME

JUST UPDATE, NO NEED TO REBUILD IMAGE

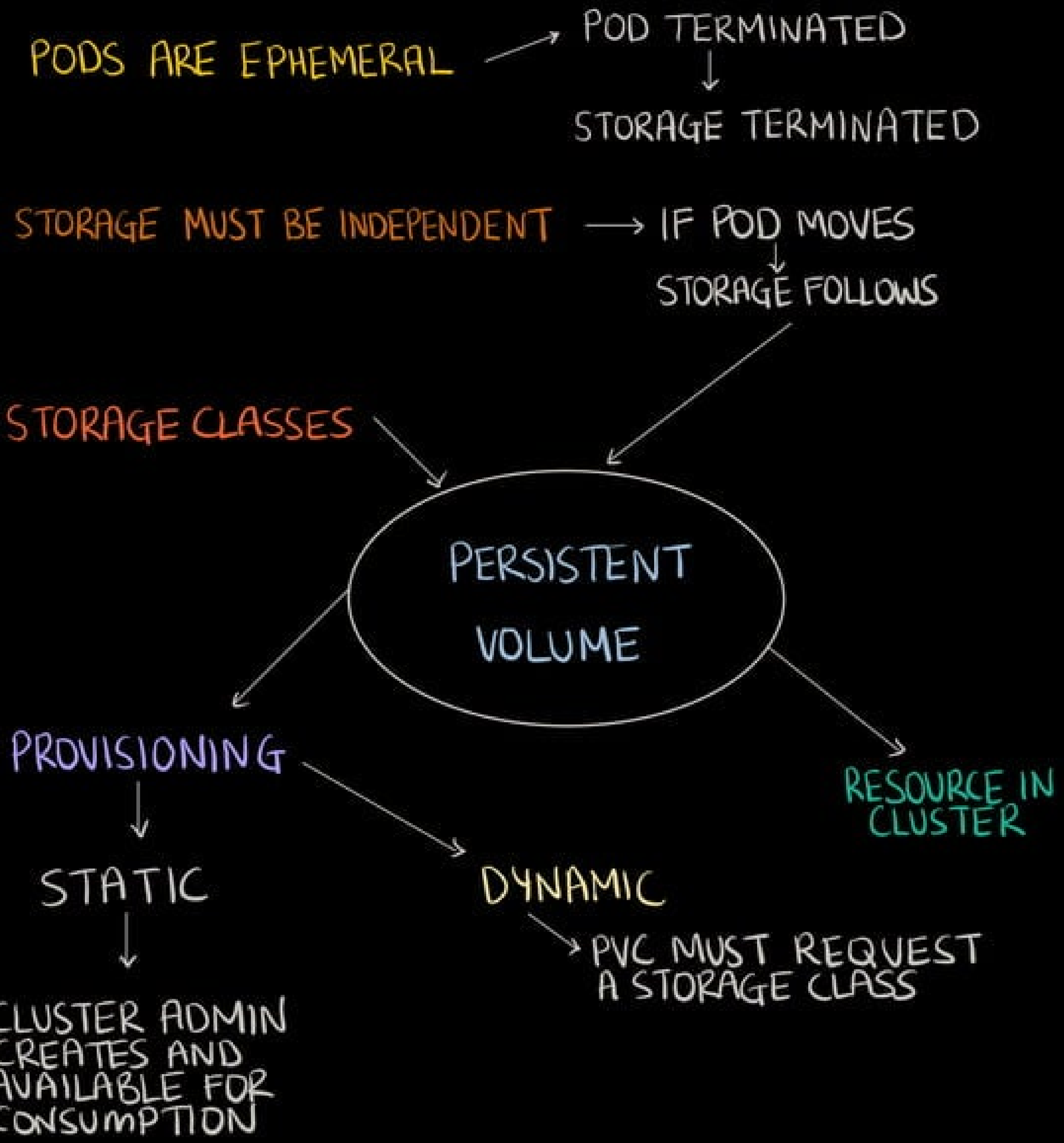
Creating a self-healing app

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



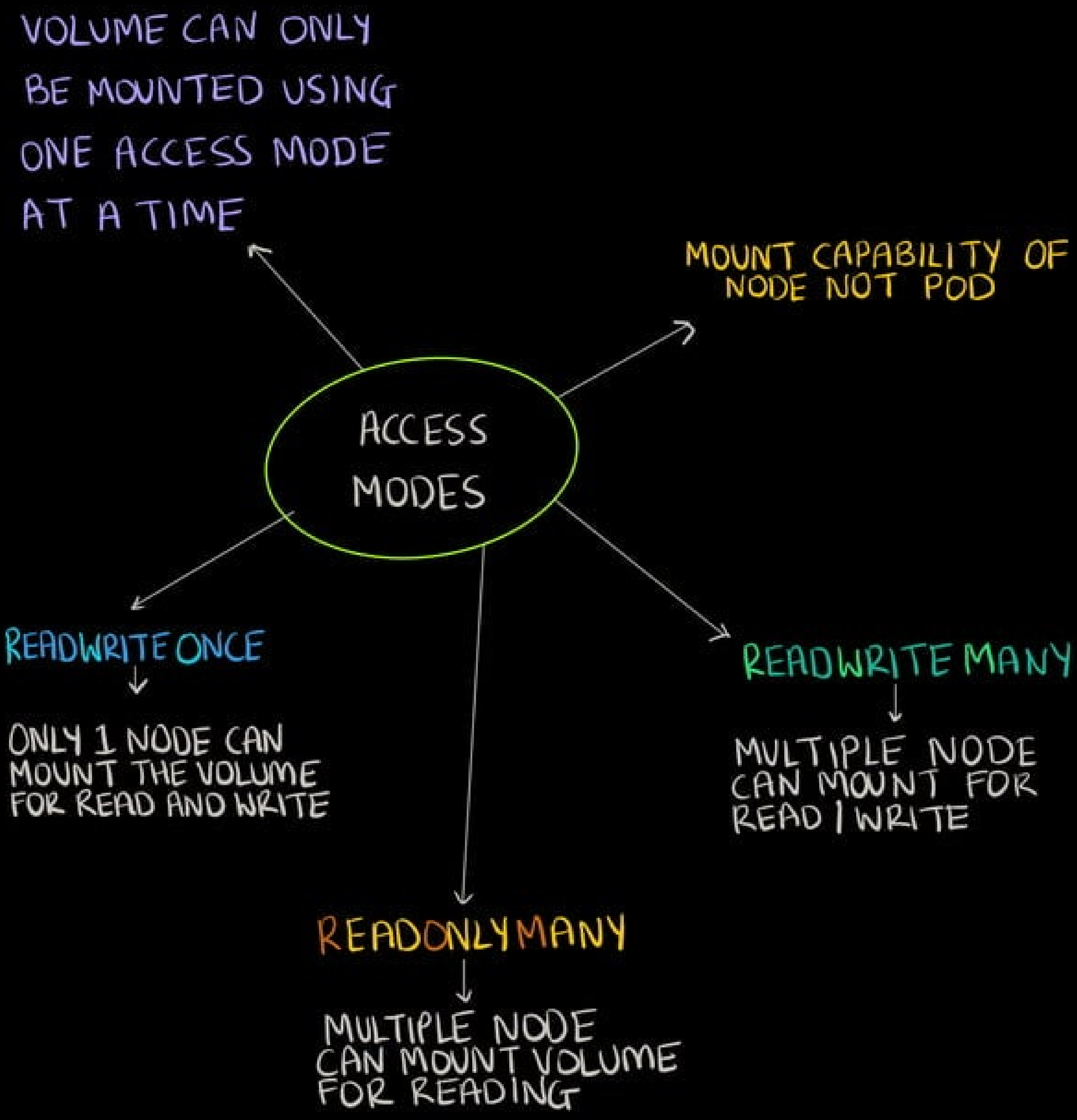
Managing Data

Persistent Volumes



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



ACCESS MODES

VOLUME CAN ONLY BE MOUNTED USING ONE ACCESS MODE AT A TIME

MOUNT CAPABILITY OF NODE NOT POD

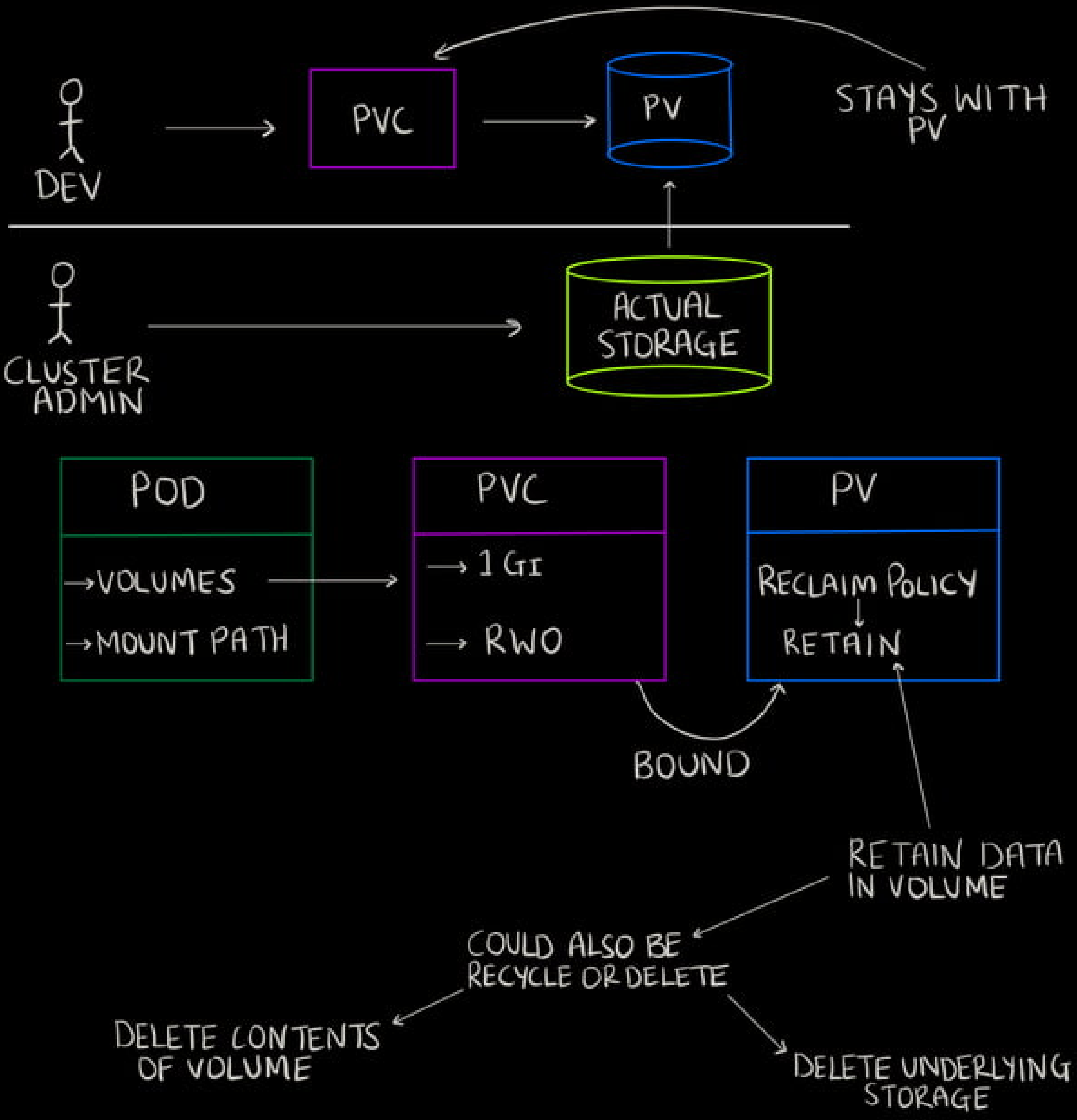
READWRITEONCE → ONLY 1 NODE CAN MOUNT THE VOLUME FOR READ AND WRITE

READWRITEMANY → MULTIPLE NODE CAN MOUNT FOR READ / WRITE

READONLYMANY → MULTIPLE NODE CAN MOUNT VOLUME FOR READING

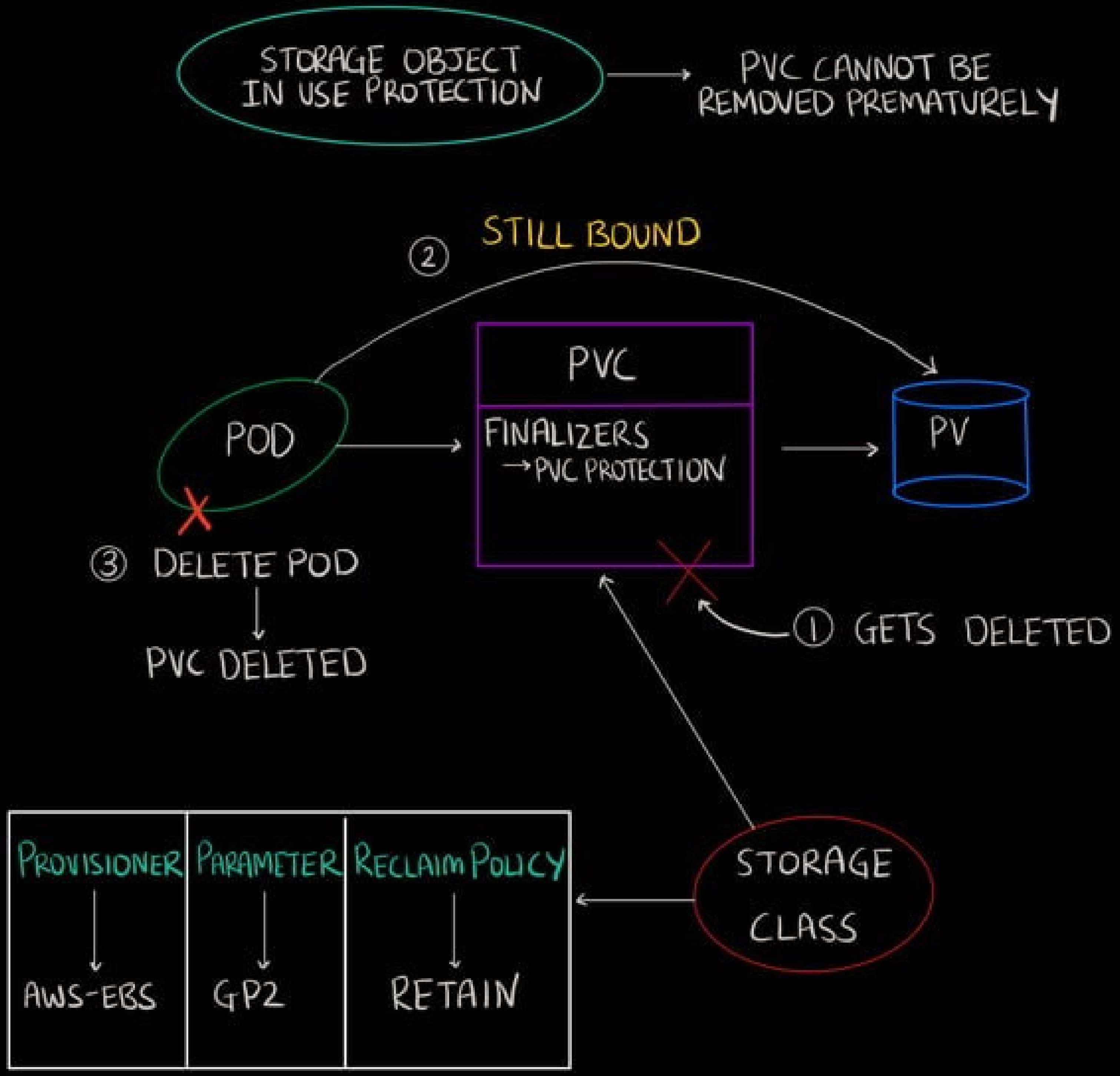
Persistent Volume Claims (PVC)

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



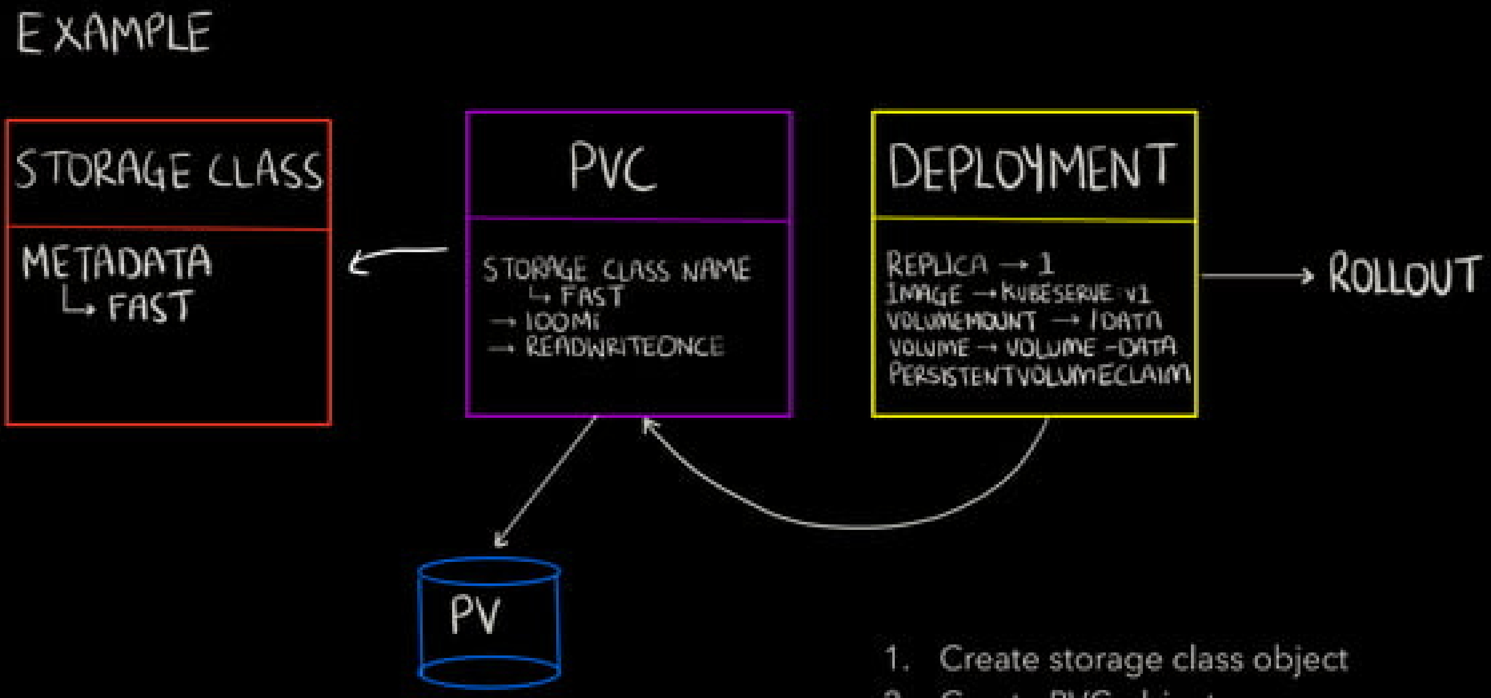
Storage Objects

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 the volume from the pod.



PROVISIONER	PARAMETER	RECLAIM POLICY
↓	↓	↓
AWS-EBS	GP2	RETAIN

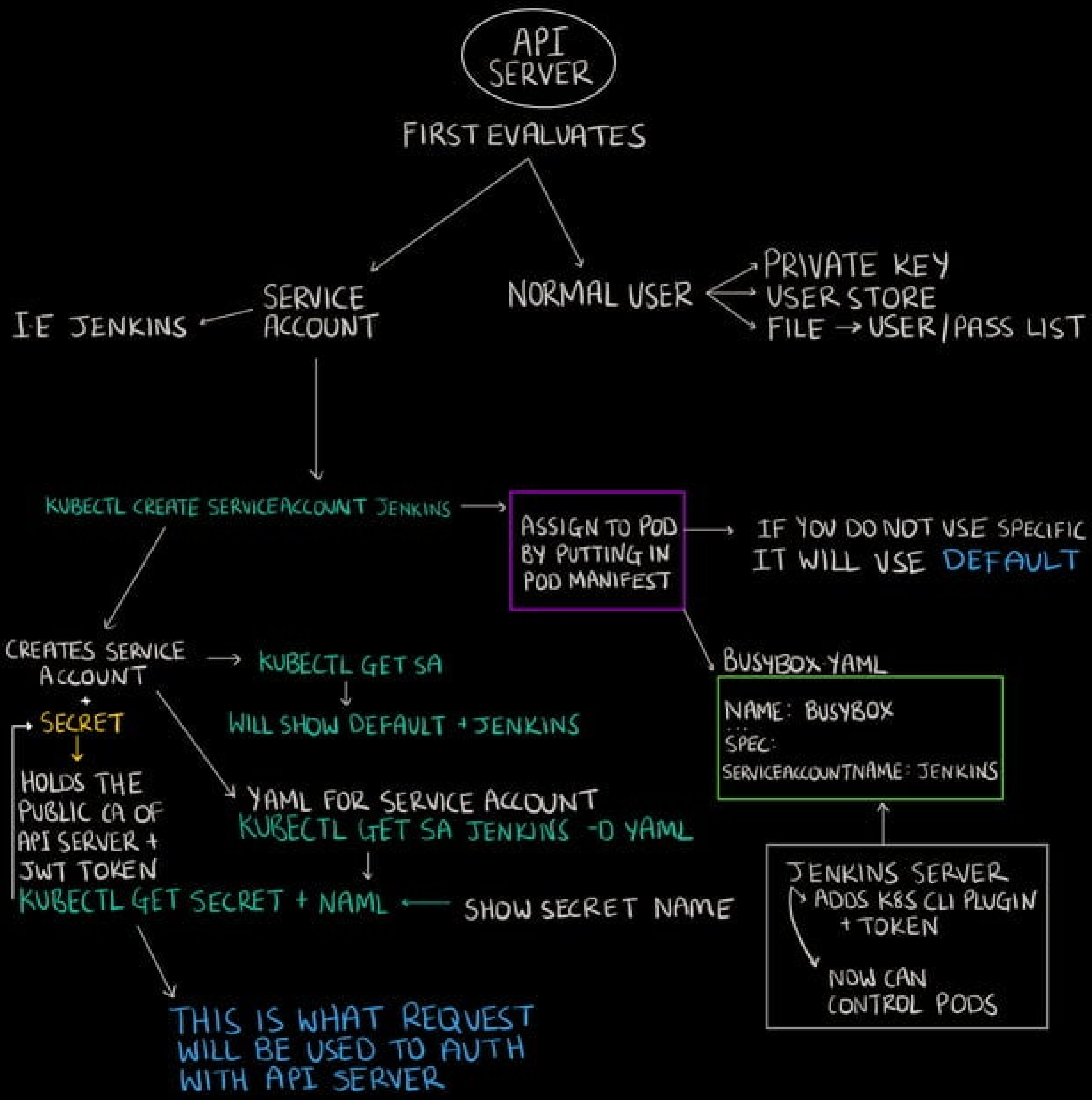
Applications with Persistent Storage



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

Securing the Kubernetes Cluster

Service accounts and users



API SERVER

FIRST EVALUATES

SERVICE ACCOUNT

I.E JENKINS

NORMAL USER

PRIVATE KEY

USER STORE

FILE → USER/PASS LIST

KUBECTL CREATE SERVICEACCOUNT JENKINS →

ASSIGN TO POD BY PUTTING IN POD MANIFEST → IF YOU DO NOT USE SPECIFIC IT WILL USE DEFAULT

BUSYBOX.YAML

```
NAME: BUSYBOX
...
SPEC:
SERVICEACCOUNTNAME: JENKINS
```

JENKINS SERVER

ADDS K8S CLI PLUGIN + TOKEN

NOW CAN CONTROL PODS

CREATES SERVICE ACCOUNT + SECRET → KUBECTL GET SA

WILL SHOW DEFAULT + JENKINS

SECRET

HOLDS THE PUBLIC CA OF API SERVER + JWT TOKEN

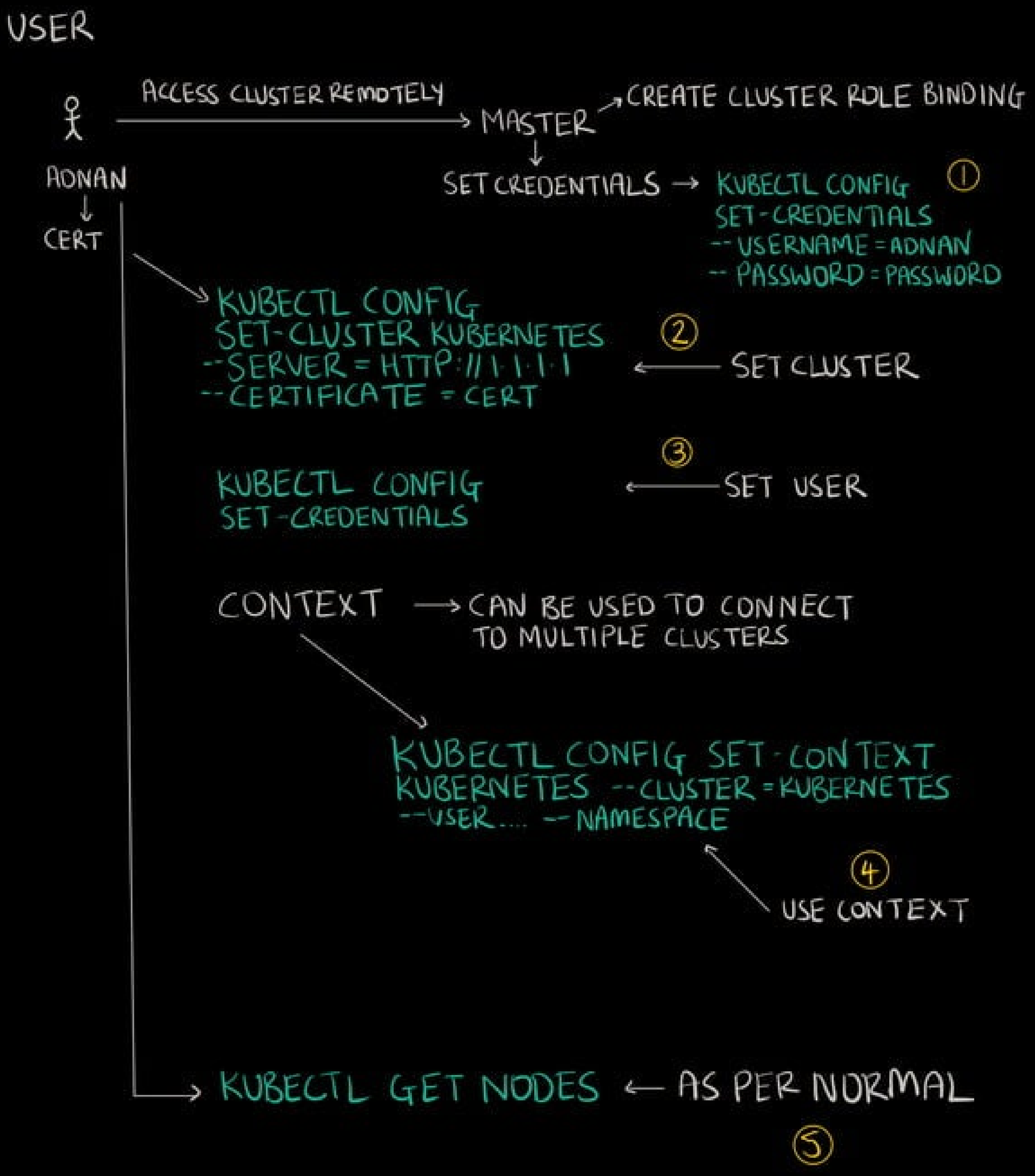
YAML FOR SERVICE ACCOUNT

KUBECTL GET SA JENKINS -O YAML

KUBECTL GET SECRET + NAML ← SHOW SECRET NAME

THIS IS WHAT REQUEST WILL BE USED TO AUTH WITH API SERVER

Service accounts and users



USER

ADNAN

CERT

ACCESS CLUSTER REMOTELY → MASTER → CREATE CLUSTER ROLE BINDING

MASTER ↓ SET CREDENTIALS →

① KUBECTL CONFIG SET-CREDENTIALS --USERNAME=ADNAN --PASSWORD=PASSWORD

② KUBECTL CONFIG SET-CLUSTER KUBERNETES --SERVER=HTTP://1.1.1.1 --CERTIFICATE=CERT ← SET CLUSTER

③ KUBECTL CONFIG SET-CREDENTIALS ← SET USER

CONTEXT → CAN BE USED TO CONNECT TO MULTIPLE CLUSTERS

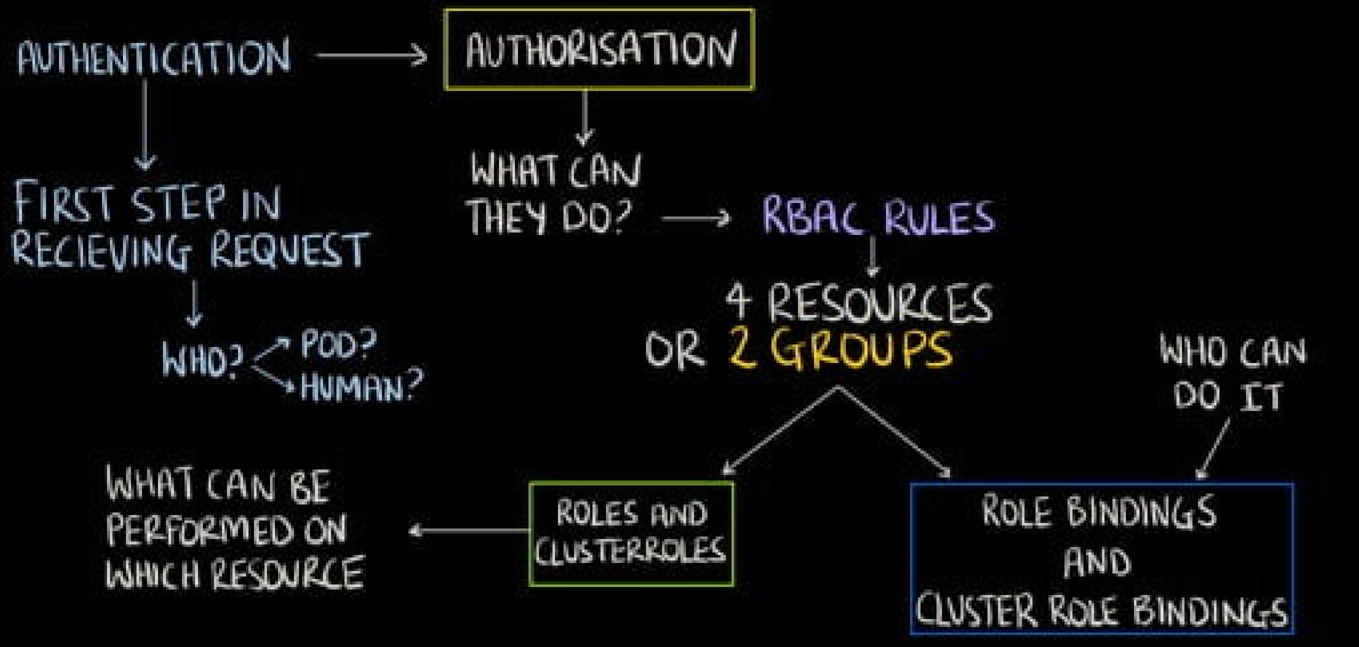
KUBECTL CONFIG SET-CONTEXT KUBERNETES --CLUSTER=KUBERNETES --USER.... --NAMESPACE

④ USE CONTEXT

KUBECTL GET NODES ← AS PER NORMAL

⑤

Cluster Authemtication and Authorisation



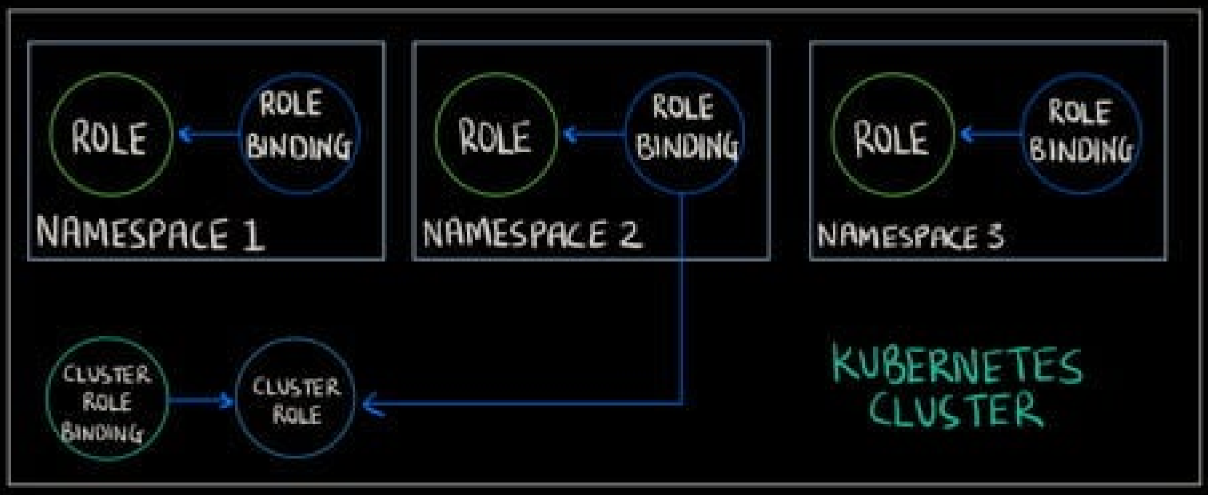
AUTHENTICATION → AUTHORISATION

AUTHENTICATION ↓ FIRST STEP IN RECIEVING REQUEST ↓ WHO? POD? HUMAN?

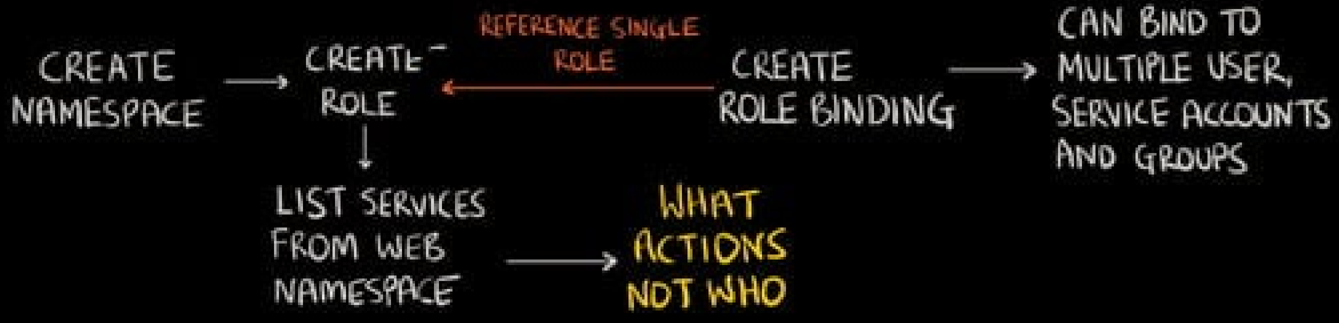
AUTHORISATION ↓ WHAT CAN THEY DO? → RBAC RULES ↓ 4 RESOURCES OR 2 GROUPS

ROLES AND CLUSTERROLES → WHAT CAN BE PERFORMED ON WHICH RESOURCE

ROLE BINDINGS AND CLUSTER ROLE BINDINGS ← WHO CAN DO IT



ROLE



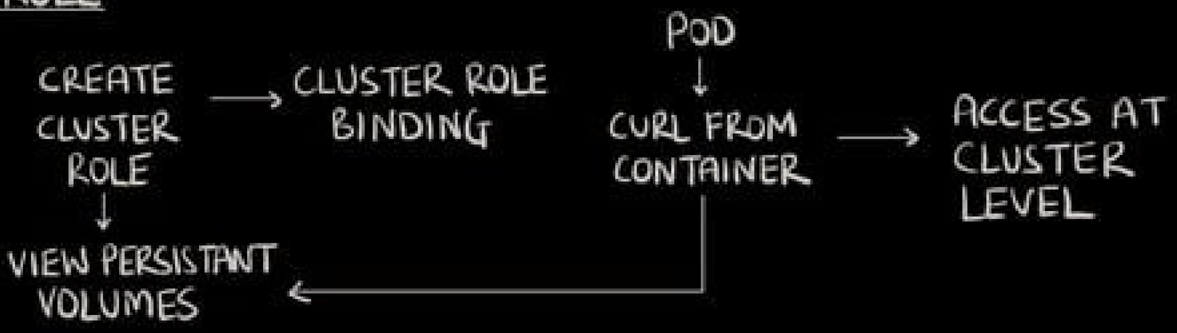
CREATE NAMESPACE → CREATE ROLE

CREATE ROLE ↓ LIST SERVICES FROM WEB NAMESPACE → WHAT ACTIONS NOT WHO

CREATE ROLE BINDING → REFERENCE SINGLE ROLE → CREATE ROLE

CREATE ROLE BINDING → CAN BIND TO MULTIPLE USER, SERVICE ACCOUNTS AND GROUPS

CLUSTER ROLE



CREATE CLUSTER ROLE → CLUSTER ROLE BINDING

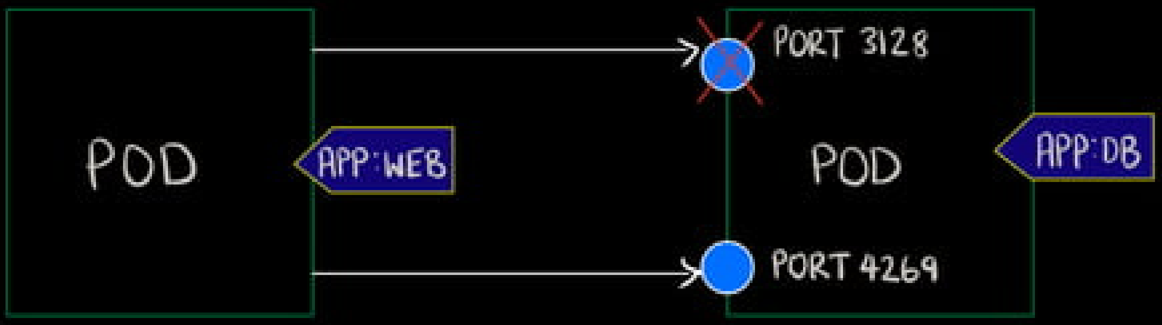
CREATE CLUSTER ROLE ↓ VIEW PERSISTANT VOLUMES

POD ↓ CURL FROM CONTAINER → ACCESS AT CLUSTER LEVEL

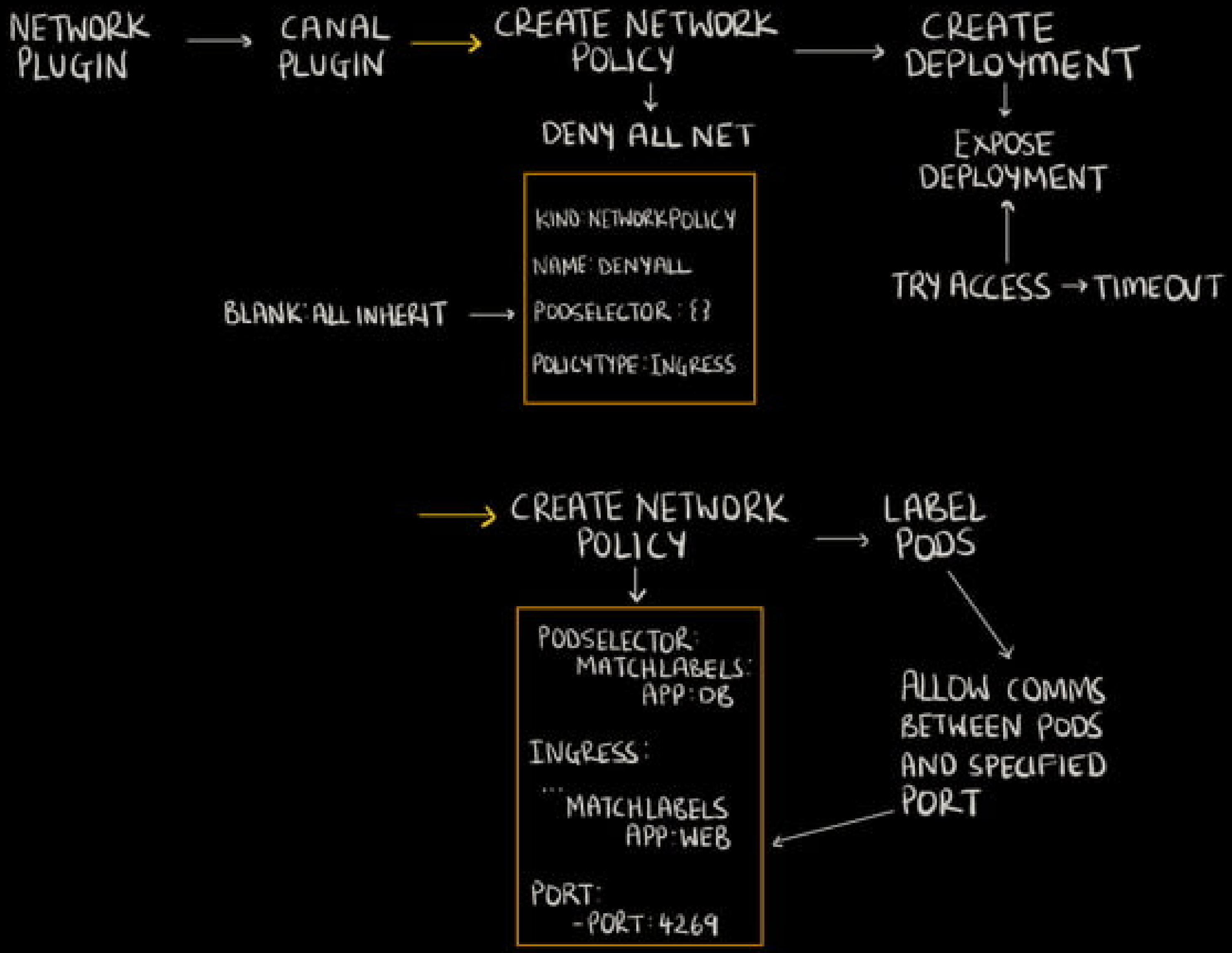
CURL FROM CONTAINER → VIEW PERSISTANT VOLUMES

Configuring Network

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



HOW?

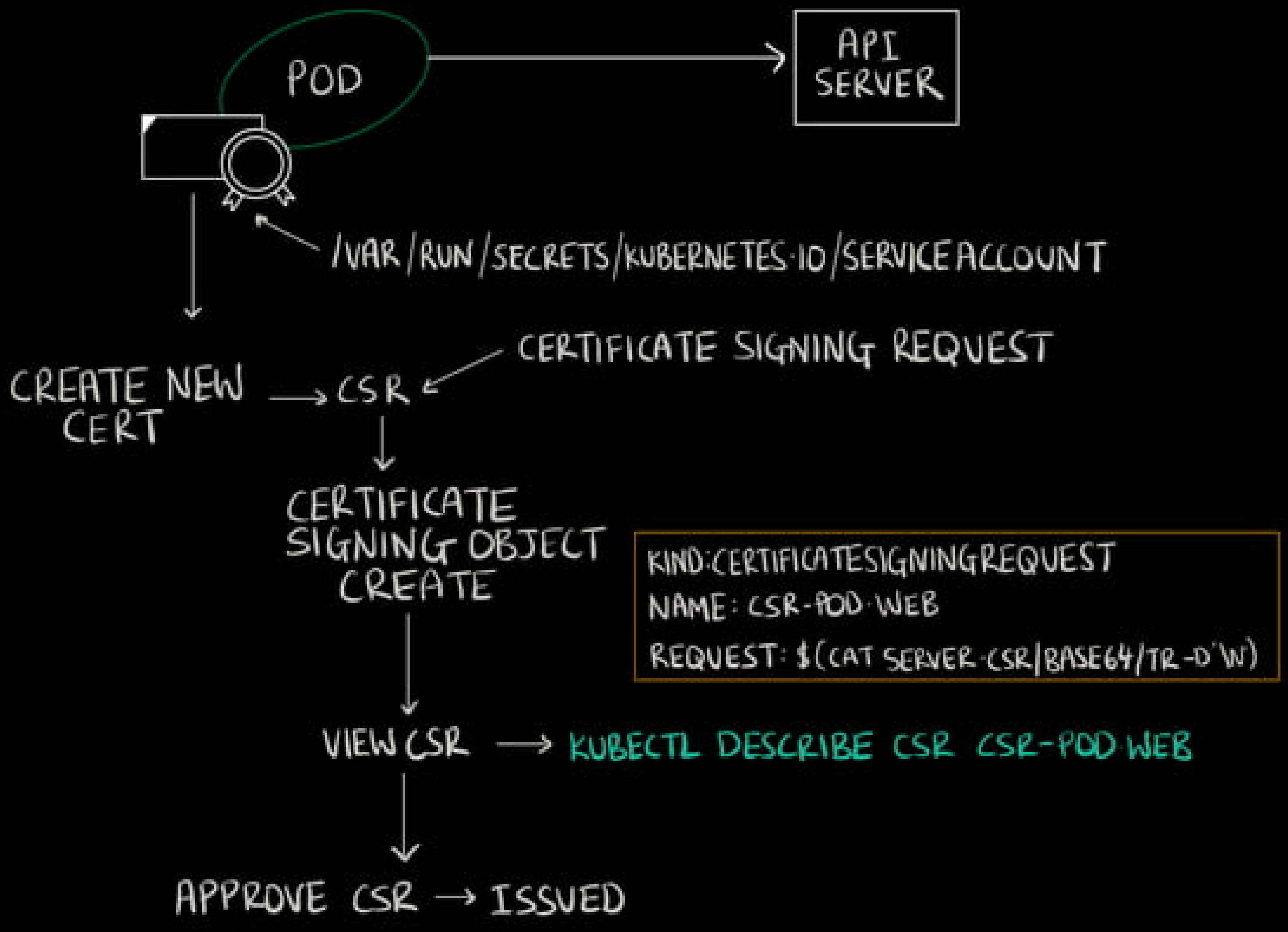


```
KIND: NETWORKPOLICY
NAME: DENYALL
PODSELECTOR: {}
POLICYTYPE: INGRESS
```

```
PODSELECTOR:
    MATCHLABELS:
        APP:DB
INGRESS:
  ...MATCHLABELS
        APP:WEB
PORT:
   -PORT:4269
```

Creating TLS certificates

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

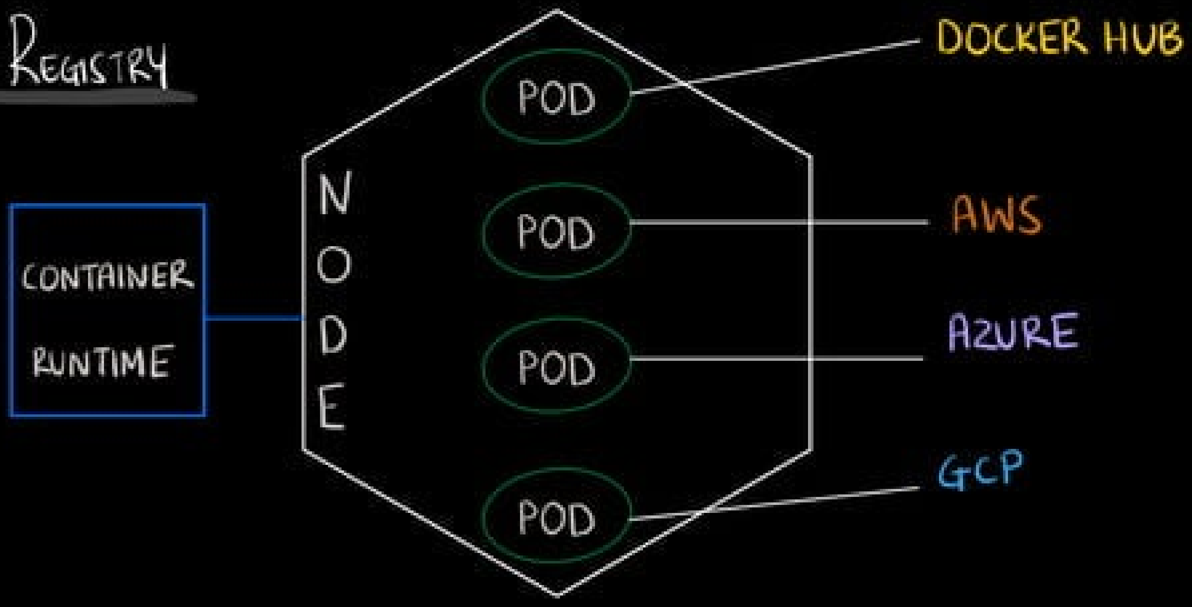

POD
API SERVER
/VAR/RUN/SECRETS/KUBERNETES.IO/SERVICEACCOUNT
CREATE NEW CERT
CSR
CERTIFICATE SIGNING REQUEST
CERTIFICATE SIGNING OBJECT CREATE
VIEW CSR
APPROVE CSR
ISSUED

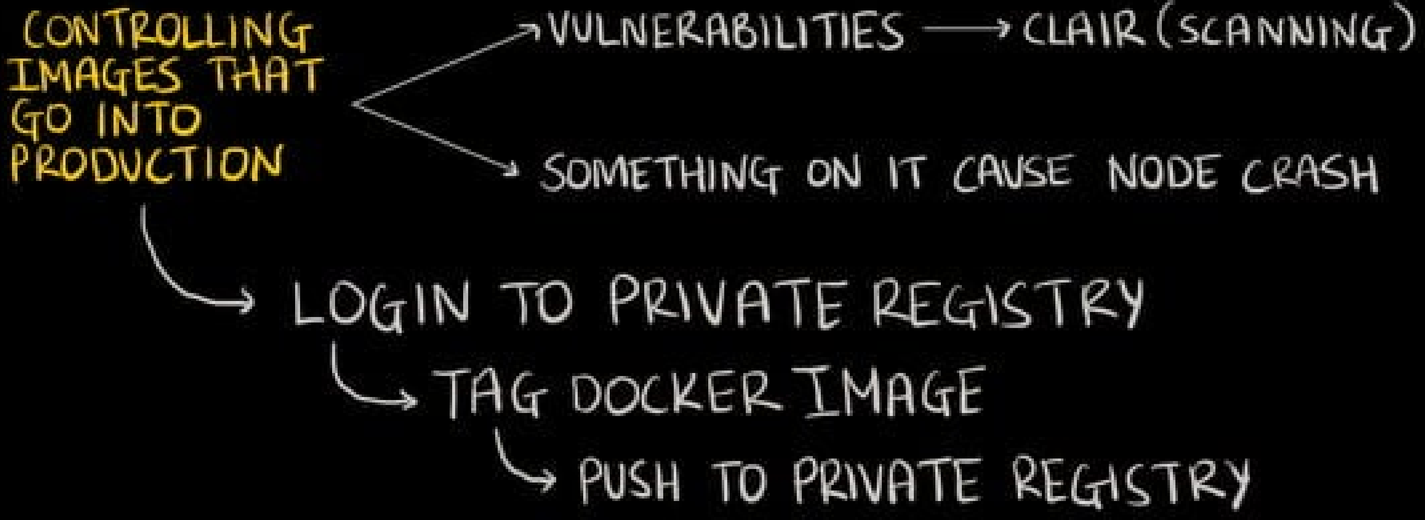
```
KIND:CERTIFICATESIGNINGREQUEST
NAME: CSR-POD.WEB
REQUEST: $(CAT SERVER.CSR|BASE64/TR-D'W)
```

```
KUBECTL DESCRIBE CSR CSR-POD.WEB
```

Secure Images

PRIVATE REGISTRY





CONTROLLING IMAGES THAT GO INTO PRODUCTION → VULNERABILITIES → CLAIR (SCANNING)

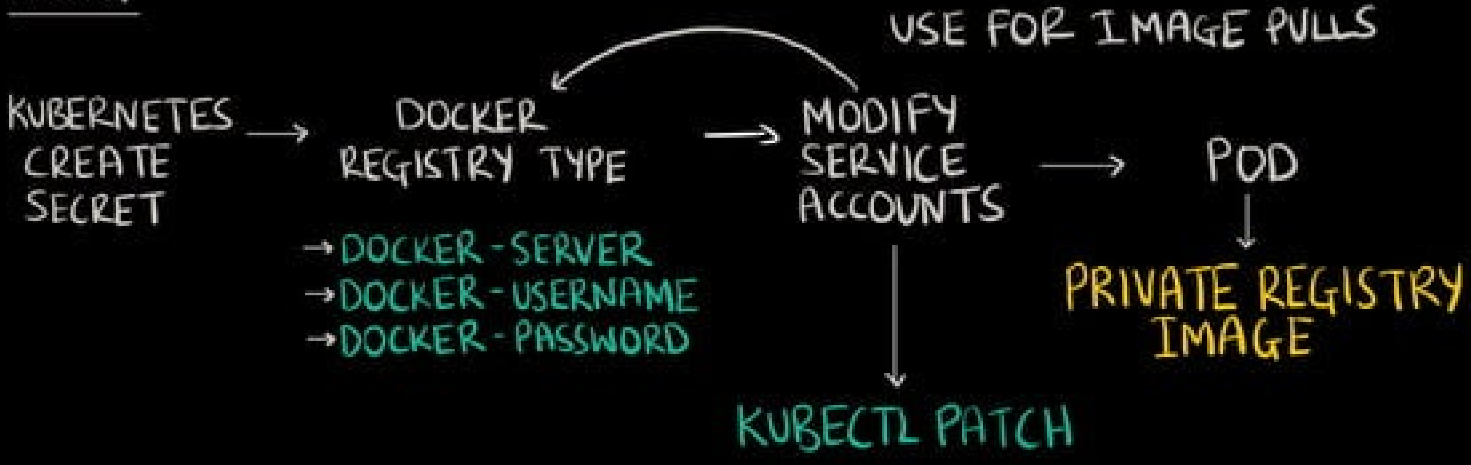
→ SOMETHING ON IT CAUSE NODE CRASH

→ LOGIN TO PRIVATE REGISTRY

→ TAG DOCKER IMAGE

→ PUSH TO PRIVATE REGISTRY

HOW?



KUBERNETES CREATE SECRET → DOCKER REGISTRY TYPE → MODIFY SERVICE ACCOUNTS → POD

- → DOCKER-SERVER
- → DOCKER-USERNAME
- → DOCKER-PASSWORD

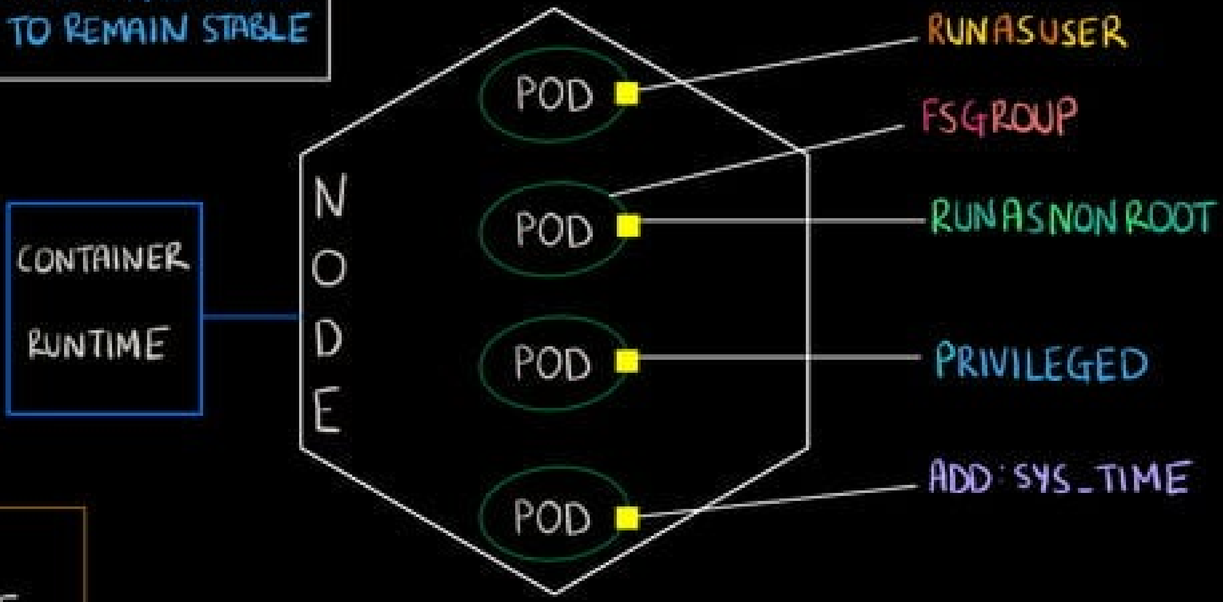
USE FOR IMAGE PULLS

MODIFY SERVICE ACCOUNTS → KUBECTL PATCH

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 'RUNASROOT'

→ ABILITY TO RUN AS PRIVILEGED → 'PRIVILEGED: TRUE'

CONTAINER LEVEL

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

ADD →

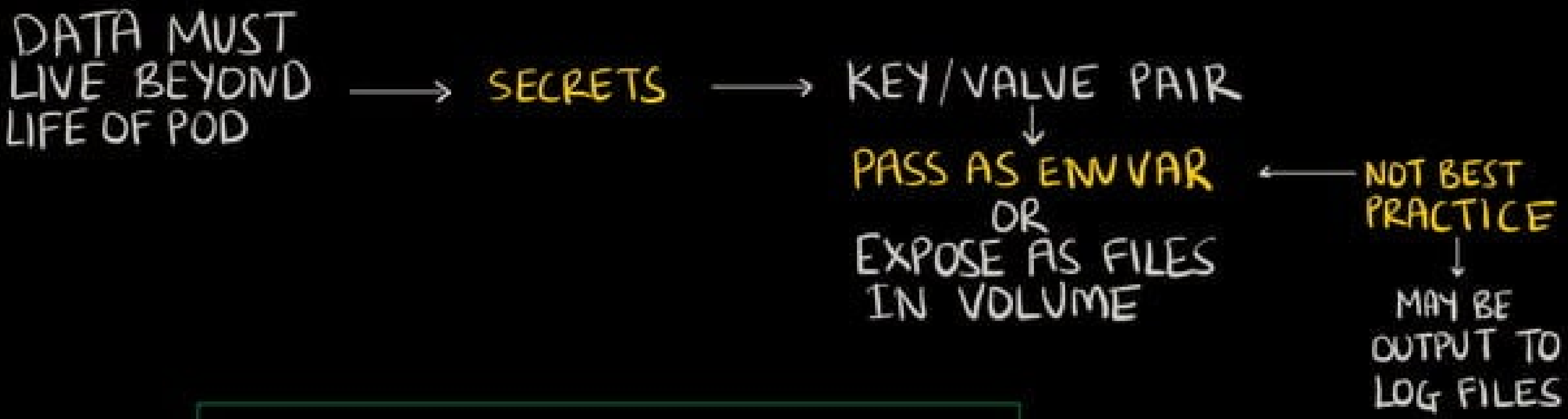
```
SECURITYCONTEXT:
  ADD:
    -SYS_TIME
    -NET_ADMIN
```

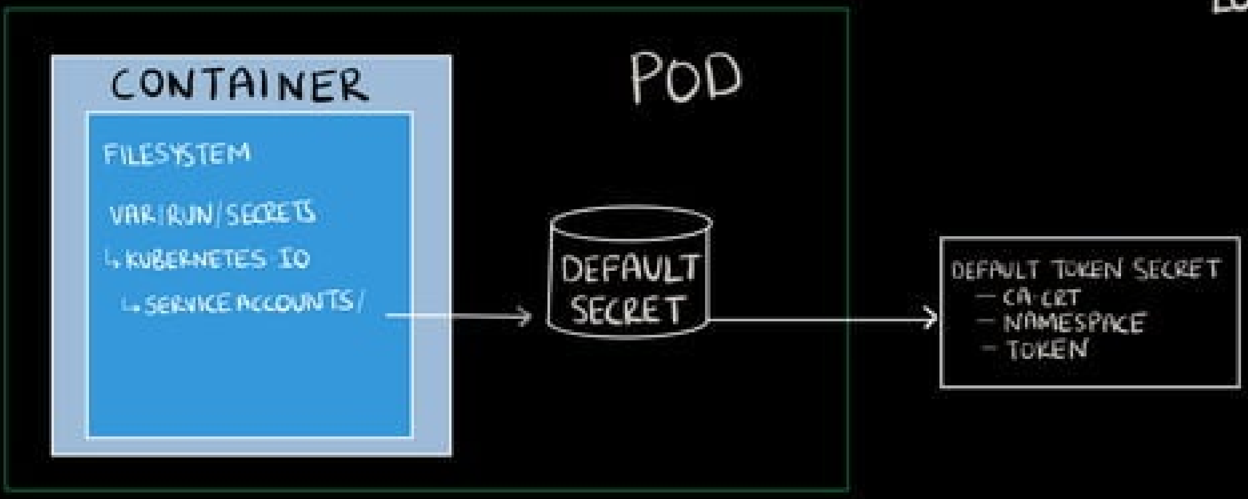
REMOVE →

```
SECURITYCONTEXT:
  DROP:
    CHOWN
```

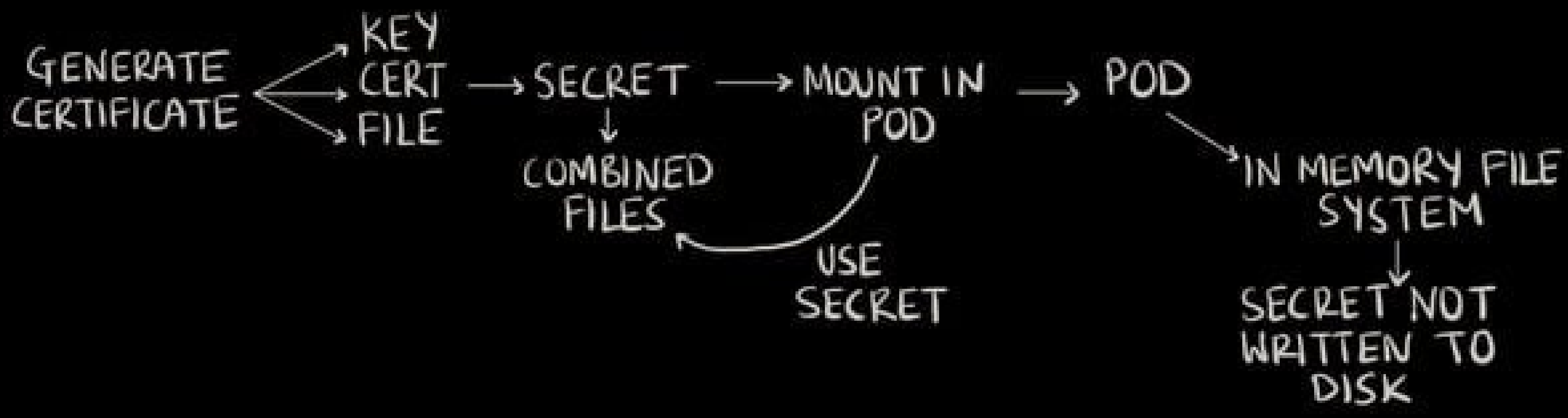
Securing persistent key/value store

Secrets allow you to expose entries as files in a volume
keeping this data secure is crucial to cluster security


DATA MUST LIVE BEYOND LIFE OF POD
SECRETS
KEY/VALUE PAIR
PASS AS ENV VAR
OR
EXPOSE AS FILES IN VOLUME
NOT BEST PRACTICE
MAY BE OUTPUT TO LOG FILES



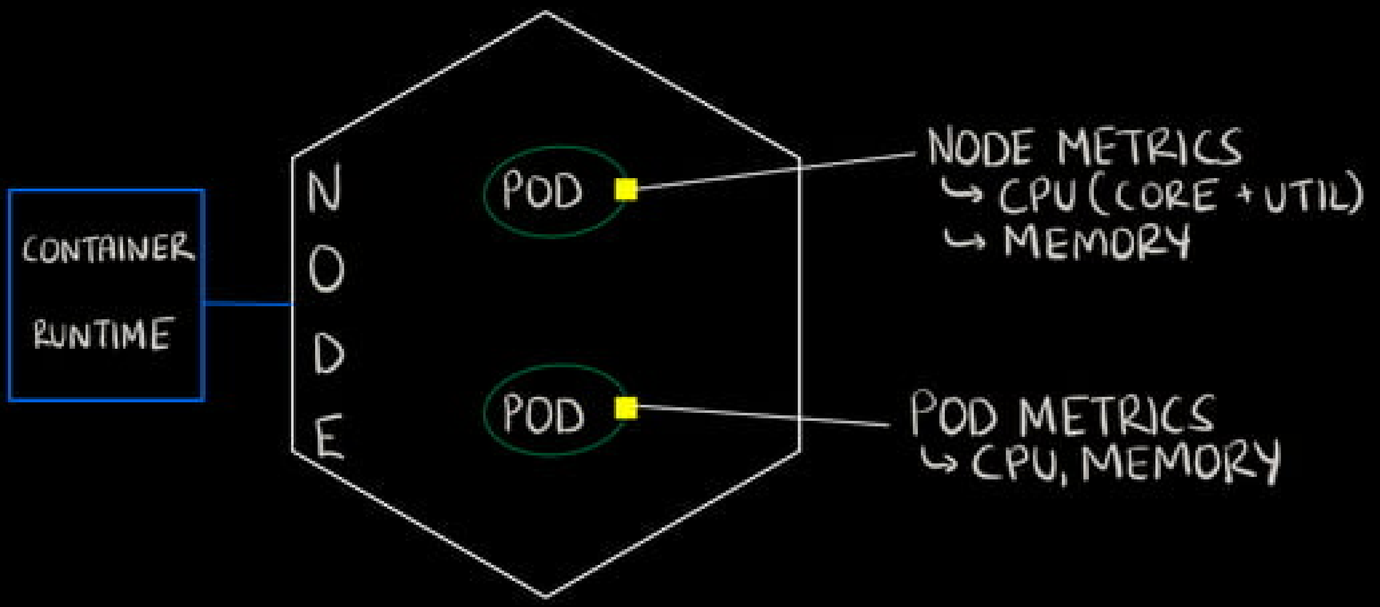
HTTPS TO WEBSITE



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-NAMESPACE → ALL NAMESPACE

KUBECTL TOP POD -N KUBE-SYSTEM → KUBE-SYSTEM NAMESPACE

KUBECTL TOP GROUP-CONTEXT --CONTAINERS → POD CONTAINERS

Managing cluster component logs

APP AND SYSTEM LOGS

→ LOGS TO CONTAINER STREAMED TO STDOUT ← USE KUBECTL LOGS

→ UNDERSTAND WHAT IS HAPPENING INSIDE CLUSTER

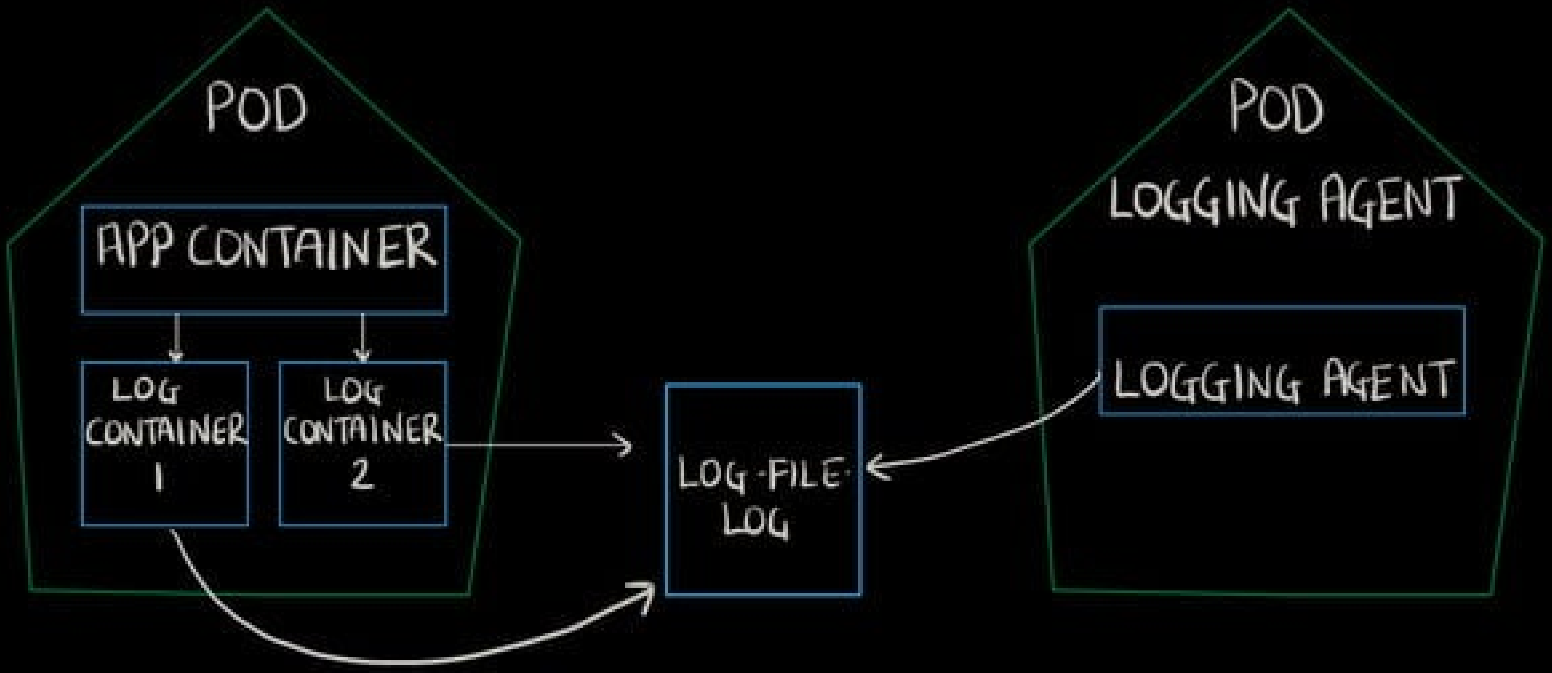
→ DEBUGGING

→ LOGS ACCUMULATE OVER TIME

→ IF MANY MICROSERVICE → WHICH LOGS ARE WHICH?

LOG DIRECTORY → /VAR/LOG/CONTAINERS

↑ ACCUMULATES LOGS

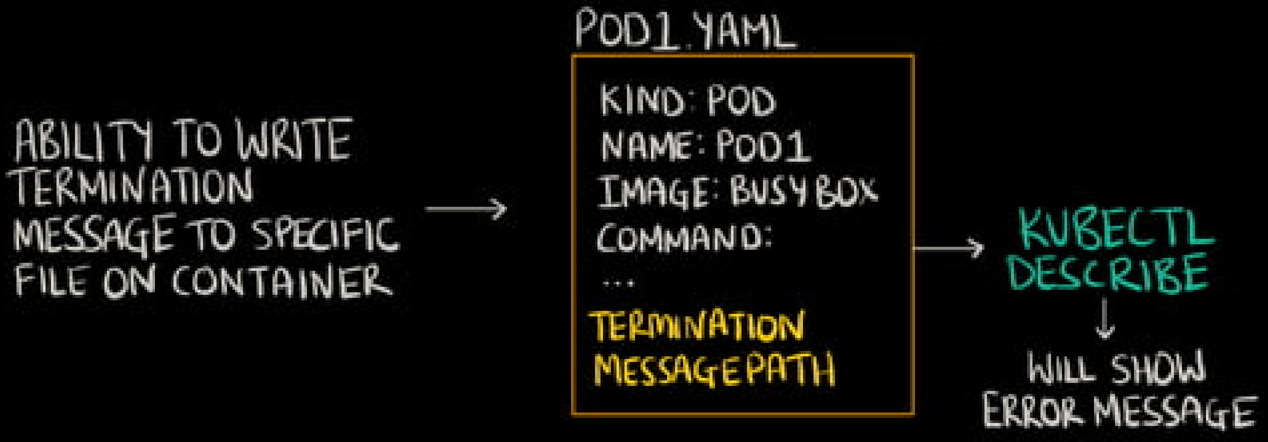


→ HAVE SIDECAR CONTAINER TO DO LOGGING SO YOU CAN ACCESS SPECIFIC LOGS

→ ABLE TO ROTATE LOGS USING OTHER TOOLING → NO NATIVE

Identifying Failures

Troubleshooting Application Failure



ABILITY TO WRITE TERMINATION MESSAGE TO SPECIFIC FILE ON CONTAINER →

POD1.YAML

```
KIND: POD
NAME: POD1
IMAGE: BUSYBOX
COMMAND:
...
TERMINATION
MESSAGEPATH
```

→ KUBECTL DESCRIBE

↓

WILL SHOW ERROR MESSAGE

→ ONLY PARTICULAR FIELDS CAN BE CHANGED I.E IMAGE

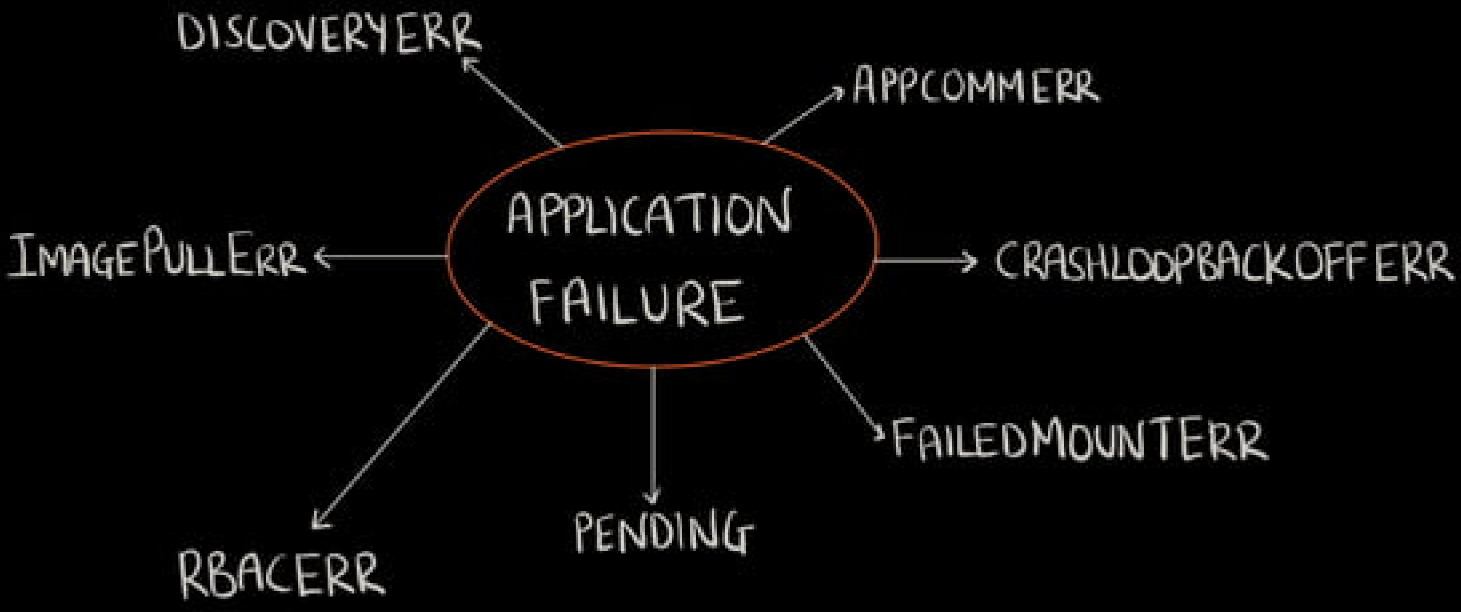
→ TO CHANGE OTHER FIELDS OF FAILED POD

↓

EXPORT CONFIGURATION

↓

MODIFY YAML I.E. CHANGE MEMORY REQUEST



APPLICATION FAILURE

DISCOVERYERR

APPCOMMERR

CRASHLOOPBACKOFFERR

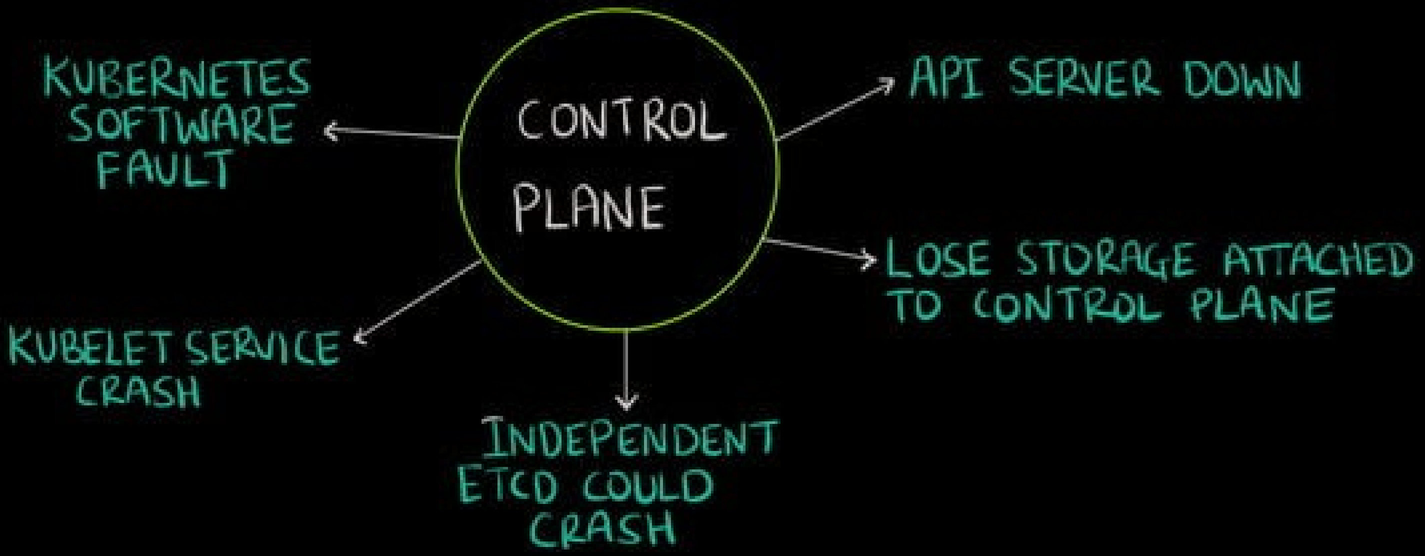
FAILEDMOUNTERR

PENDING

RBACERR

IMAGEPULLERR

Troubleshoot failures


CONTROL PLANE
API SERVER DOWN
KUBERNETES SOFTWARE FAULT
LOSE STORAGE ATTACHED TO CONTROL PLANE
KUBELET SERVICE CRASH
INDEPENDENT ETCD COULD CRASH

→ VIEW THE EVENTS FROM CONTROL PLANE COMPONENTS

→ VIEW LOGS FOR CONTROL PLANE PODS

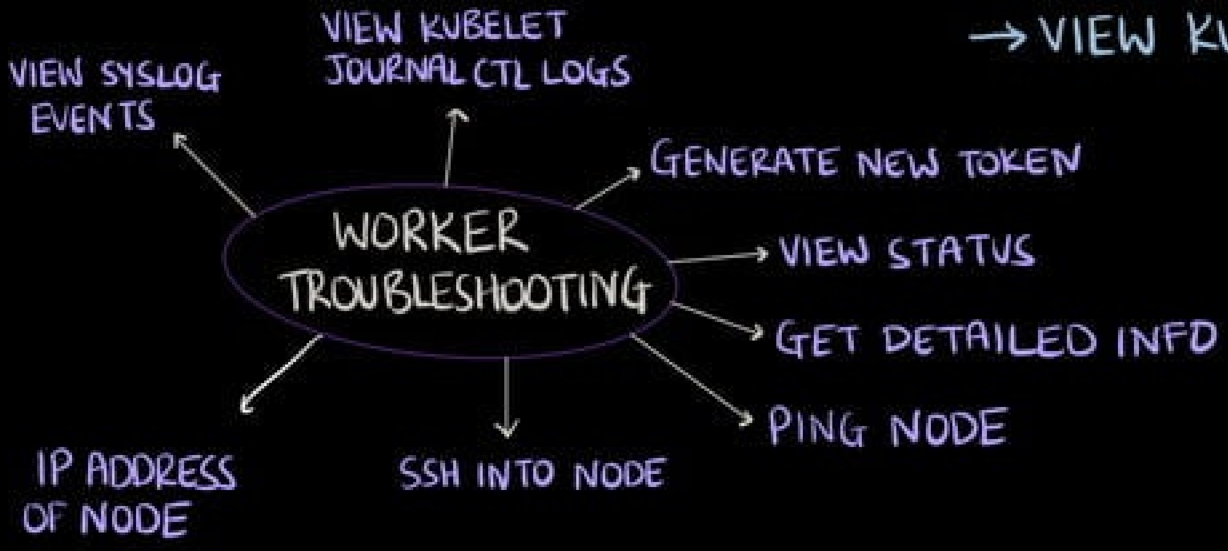
→ CHECK STATUS OF DOCKER SERVICE

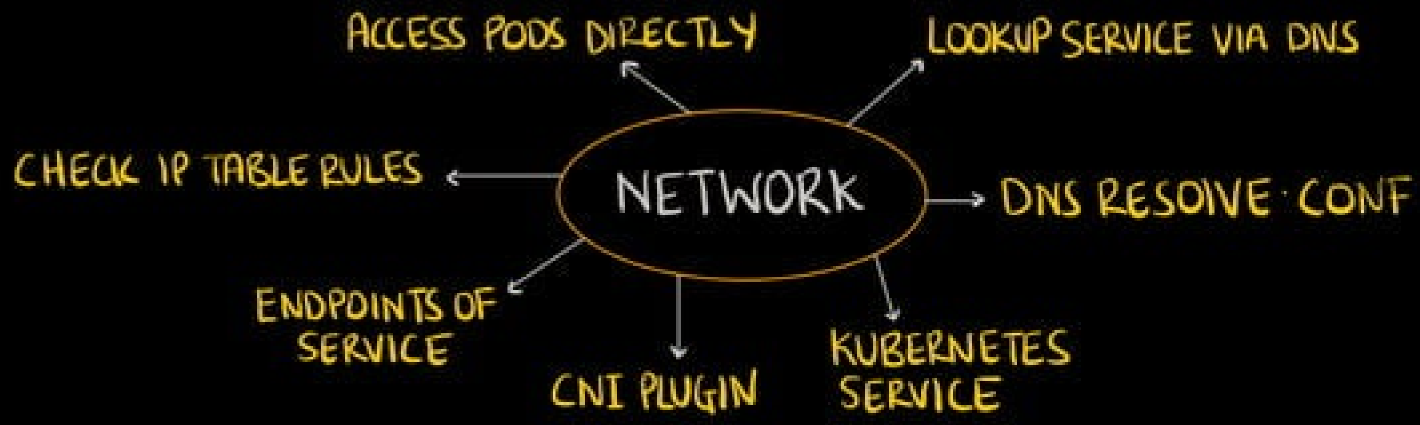
→ CHECK STATUS OF KUBELET SERVICE

→ DISABLE SWAP

→ CHECK FIREWALLD SERVICE

→ VIEW KUBE CONFIG


VIEW SYSLOG EVENTS
VIEW KUBELET JOURNAL CTL LOGS
GENERATE NEW TOKEN
WORKER TROUBLESHOOTING
VIEW STATUS
GET DETAILED INFO
PING NODE
IP ADDRESS OF NODE
SSH INTO NODE


ACCESS PODS DIRECTLY
LOOKUP SERVICE VIA DNS
CHECK IP TABLE RULES
NETWORK
DNS RESOIVE · CONF
ENDPOINTS OF SERVICE
CNI PLUGIN
KUBERNETES SERVICE