KUBERNETES

Date

→ Open source, container orchestration tool
→ Helps you manage containerized applications in different environment

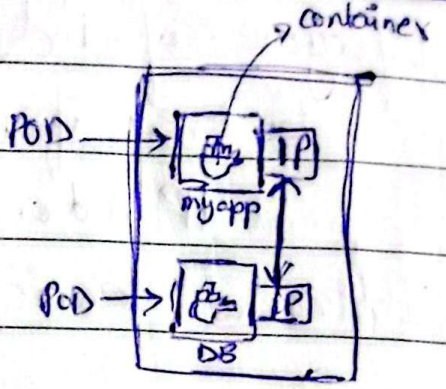
- The need for a container orchestration tool
→ Trend from Monolithic to microservice architecture
→ Increase usage of containers
→ Demand for a proper way of managing those hundreds of containers

- Features
↳ High availability / No downtime
↳ Scalability or high performance
↳ Disaster recovery - backup and restore

→ Through Kubernetes we can run our app on any cloud provider. It provides an abstraction layer for us.
→ Immutable Infra → For making any change to server → Create Copy of server from base image → changes made → replace old with new one

Date

▷ Main Components of Kubernetes



- Pod
  - ↳ Smallest unit of K8s
  - ↳ Abstraction over container
  - ↳ It contains one or more application containers that are tightly coupled
  - ↳ But mostly we should run one-container per Pod
  - ↳ It creates a layer of running env on top of container
    - ↳ K8s has made this abstraction so that we are not dependent on ~~contai~~ a specific containerization technology.
  - ↳ Each Pod gets it's own IP Addr (not the container!)
    - ↳ Pods can communicate ~~using~~ with each other using IP
  - ↳ Pods can die easily, so when they created again they get new IP. So if two Pods communicate each other then we should update the IP.
    - ↳ if we make a DB pod then if we make new DB, it will get new IP and we will have to change the IP in our application

- Service
  - ↳ For this reason service is used
  - ↳ It is basically a permanent IP Addr that can be attached to each Pod-

→ Service is also a load balancer for Pod. like we have two Pods of AddToCart Service. So the service sends request to the Ideal Pod.

Date

↳ Lifecycle of Pod & Service not connected.
↳ Even if Pod dies it will not loose it's permanent IP Adr.

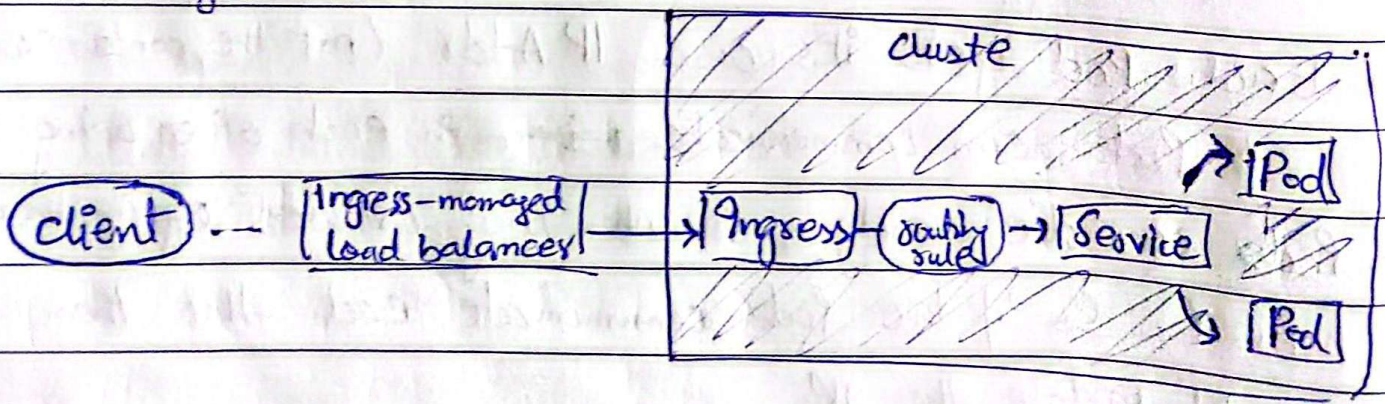
→ External Service (like we need to run an app on browser (easily accessible))

→ Internal Service (like DBs)

- Ingress

→ Exposes HTTP & HTTPS routes from outside the cluster to services within the cluster.

→ Traffic routing is controlled by rules defined on the ingress resource.



Date

- ConfigMap
  - ↳ Used to store non-confidential data in key value pairs
  - ↳ Pods can use them as env variables.
  - ↳ ConfigMap allows you to decouple env-specific configuration from your containes images
  - ↳ ConfigMap can also be mounted as data volumes

- Secret
  - ↳ It contains small amount of sensitive data such as a password / a token or a key!
  - ↳ they are similar to ConfigMap.
  - ↳ they store data in base64 encoding format

- Volumes
  - ↳ By default there is no data persistence on restarting Pod.
  - ↳ but can be done thru volume
  - ↳ This volume is seperate from cluster.
  - ↳ It can be the part of the local machine or remote

ReplicaSet: is a Kubernetes controller that ensures a specified no. of identical Pods are always running

Pod Blueprint → It is somewhat similar to AWS ASG

Date

- Deployment & Statefulness

↳ A deployment manages a set of Pods to run an application workload

↳ Usecase of Deployment

↳ It helps creating new replicas of existing pods

↳ If we change a pod then it helps to scale up the new one & gradually scale down the older

↳ If a pod crashes, it rolls back to stable state

↳ Helps in auto scaling (balances load automatically)

↳ Cleans up the older state

→ Deployment is an abstraction over Pods.

→ We work with Deployment

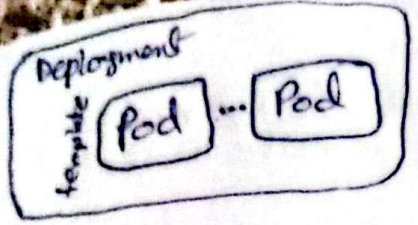
→ We cant replicate DBs using Deployment because it would cause data inconsistencies. (all DB pods sharing same Database)

↳ This can be done thru StatefullSets

↳ It makes sure that DB readwrites are synchronized

↳ But it is not easy to work with Statefullsets.

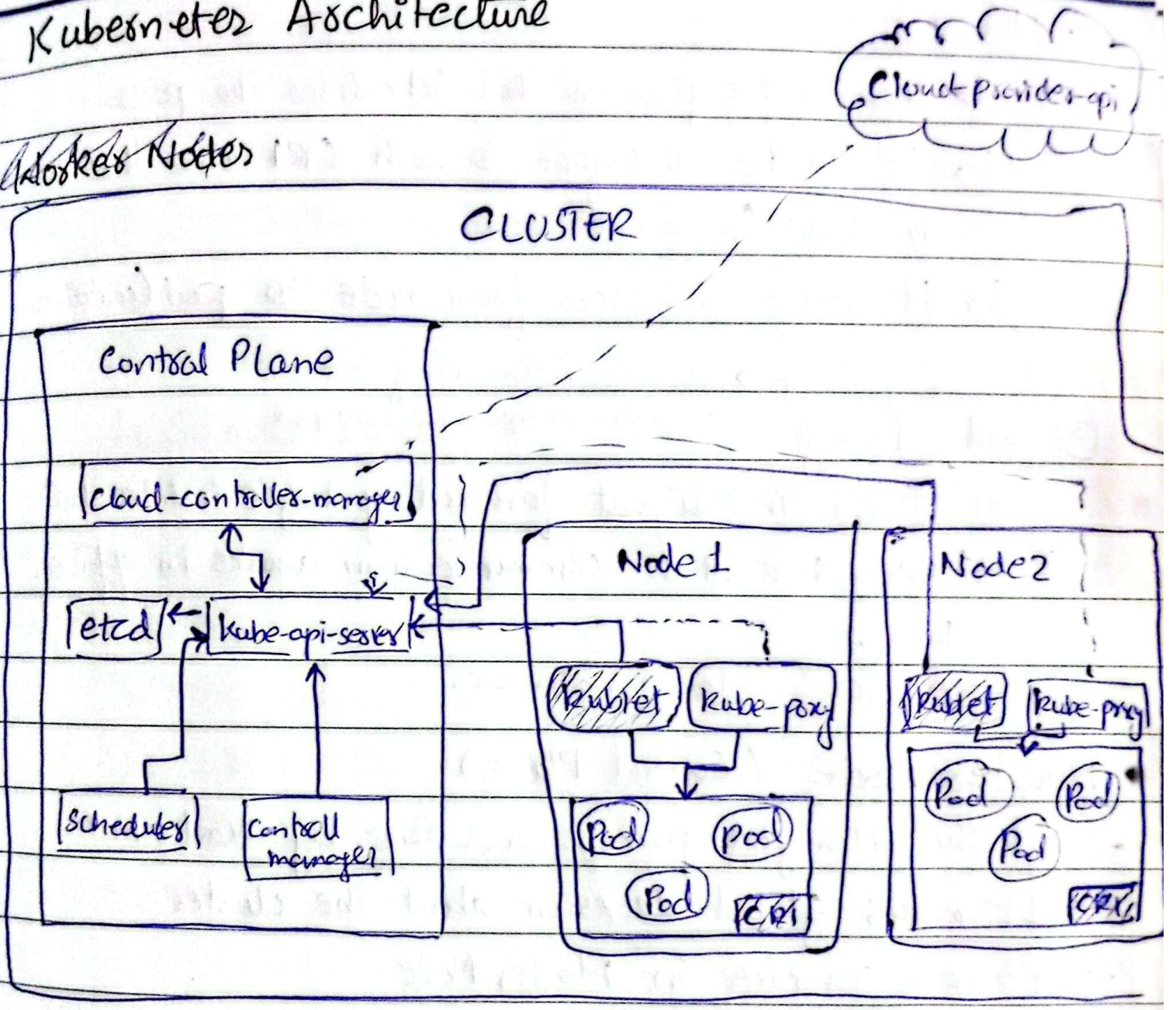
↳ recommended to use Statefull apps outside K8s cluster



Date

- Kubernetes Architecture

- ~~Kubernetes~~ Nodes



- Kubernetes Node
  - ↳ One of the main component
  - ↳ Each node have multiple Pods running on it.
  - ↳ Each node have 3 process
    - (1) Container Runtime (Docker Engine) → A fundamental component that empowers Kubernetes to run containers
      - → A software that runs container
      - → Eg: containerd (default CRI)

An agent that runs on each node
It takes a set of PodSpecs that are provided through various mechanism and ensures that the containers describe in that spec is running & healthy

Date

② Kublet
↳ This is the process that schedules the pods
↳ It is the interface to both CRI and the machine (node)
↳ It assign resources from node to pod (containers)

③ Kube Proxy → manages networking
maintains network rules on pod
↳ It has intelligent forwarding logic inside that makes sure that communication works in efficient way.
↳ Act as a load balancer

• Master Node (Control Plane)
→ All the Manging process are done by Master Node
→ It makes global decesion about the cluster
→ Process running in Master Node
① Api-Server → Exposes Kubernetes API
↳ It is like a cluster gateway
↳ It is the fontend for the Kubernetes control plane
↳ The client interacts with Api-server.
↳ ~~It gets the initial update~~
↳ Act as a gatekeeper for authentication
↳ It is designed to scale horizontally

Page No.

↳ a component that watches for newly created pods with no assigned node

Date

(2) Scheduler

↳ If we need to add a new pod Api-server sends req to Schedules.

↳ It intelligently identifies on which node to create the Pod

↳ Schedules on least busy Node.

↳ Schedules only decide on which node to create

↳ the actual creation is done on Node by Kublet

↳ Component that runs controller processes

(3) Controller Manager

↳ If a pod die on a Node so to manage & reschedule it it is done by this process

↳ Detects cluster state changes

↳ It request the schedules to create new pods

(4) Etcd

↳ Key value store of cluster

↳ Used as Kubernetes backing store for all cluster data

↳ Etcd is the cluster brain

↳ Application data is not stored (only cluster state info is stored)

↳ Stores data in a distributed manner

Date

→ Master Node is replicated and Api-server is load balanced.

→ Master Nodes req less resources than Worker Nodes

→ Types Of Controllers

① Deployment Controllers:
↳ Ensures the desired no. of Pods are running via ReplicaSet and manages rolling updates & rollback.
↳ Only specifies the no. of pods in a ReplicaSet.

② ReplicaSet Controller
↳ It creates or deletes pods based on the actual and desired no. of pods in ReplicaSet.

③ Node Controller
↳ Monitors node health and marks node as NotReady or removes Pods when nodes fail

④ Job Controller
↳ Ensures that a specified no. of Pods successfully complete a one-time task

Date

⑤ Service Controller

↳ Creates and manages & cloud LB for & Services of type LoadBalancer

⑥ ~~Updater~~ Endpoint Controller

↳ Update Endpoints or EndpointSlices to map Services to the corret Pod IPs

⑦ Service Account Controller

↳ Automatically creates a default ServiceAccount for every namespace and ensures Pods can authenticate with API Server.

"Controller continuously watch the API server and take corrective actions to ensure the actual cluster state always matches the desired state in etcd."

KUBERNETES

Date

- **Workload Objects**
  - ↳ Workload objects tell Kubernetes "how" and "where" to run you the application containers.
    - ↳ How many copies to run?
    - ↳ On which Nodes?
    - ↳ Whether it runs once or continously?

→ Types of Workload Objects

① **Deployment**

- ↳ A deployment is used to run applications that should always be running and can be replaced easily
- ↳ Like a web app manager
- ↳ Runs multiple copies of your app
- ↳ Replaces failed pods
- ↳ Support rolling updates & rollback
- ↳ Used in : Web servers, APIs, Microservices

↳ In real life we donot make pods we make workload objects and specify all containers inside that then K8s does rest

→ Difference b/w stateful set & deployment.

- In deployment pods have random hashes whereas in stateful set each pod has stable unique ordinal index
- In dep pods deleted/created in random order. Pods deleted and scaled in predictable order

Date

② Stateful Set

↳ For apps that need fixed identity and storage

↳ like a database manager

↳ Features

↳ Fixed pod names

↳ stable network identity

↳ Dedicated storage per pod

↳ Identity of pod is preserved when replaced by another

③ DaemonSet

↳ A DaemonSet ensures one copy of a Pod runs on every node

↳ Automatically runs on a new node

↳ No need to define replica count

↳ Used for log collector or monitoring agent

④ Job

↳ A Job runs a task until it finishes successfully

↳ What it does

↳ Runs to completion

↳ Retries on failure

↳ Stops when done

Date

⑤ CronJob

↳ A CronJob runs Jobs on a schedule

↳ Used cases: Nightly backups, Cleanup Jobs, Report Generation

- **Namespace**

↳ A logical isolation mechanism in Kubernetes that allows multiple teams or env to share the same cluster while keeping resources organized, secure and manageable.

↳ In traditional setups we have diff servers for diff env like dev, test, pat, prod etc. So in order to achieve this in kubernetes we use "namespace".

↳ Each name space runs its own version of the application, database configurations and ingress routing.

- **Volume**

↳ An abstraction that let us persist data

↳ Necessary bcz pods are ephemral - meaning data is deleted when pod is deleted

Date

- Benefits Of Kubernetes

① Scale the application as needed from command line or UI

② Automated rollouts and rollbacks

③ Containers get their own IP so you can put a set of containers behind a single DNS name for loadbalancing

④ Automatically mount local or public cloud or a network storage

⑤ Create and update secrets & configs without image rebuilding

⑥ Self Healing
   - ↳ Restarting failed containers
   - ↳ Replacing & Rescheduling containers as nodes die
   - ↳ Killing containers that don't respond

⑦ Automatic ~~Schedu~~ resource scheduling based on requirement & constraints.

Date

- **Kubectl**

↳ Command line tool used to:

- Talk to the Kubernetes API Server
- Create, update, delete and inspect Kubernetes objects

① Check cluster connectivity

» Kubectl cluster-info

② Get Commands

» Kubectl get <resource>

↳ resource can be nodes, pods, services, deployments, namespaces

↳ get detailed output

» Kubectl get pods -o wide

③ Describe commands.

↳ Shows detail information

» kubectl describe <resource> <resource_name>

pod my-pod

Date

↳ Used when pod is crashing, image is not pulling, healthChecks fail

④ Create and Apply Commands

↳ Create from Yaml

>> kubectl apply -f my-deployment.yml

↳ Creating Imperatively

>> kubectl create deployment nginx --image=nginx

(nginx → name of deployment)

⑤ Delete Commands

>> kubectl delete pod my-pod

>> kubectl delete -f my-deployment.yaml

>> kubectl delete all --all -n dev

↳ delete everything in namespace

⑥ Logs & Exec (Debugging)

>> kubectl logs my-pod

>> kubectl logs my-pod -c my-containes

(-c → log from specific containes)

Date

↳ Get shell inside container

>> kubectl exec -it my-pod -- /bin/bash

⑦ Namespace

>> kubectl get ns → list all namespaces

↳ Create namespace

>> kubectl ~~get~~ create ns dev

⑧ Scalling

>> kubectl scale deployment myapp --replicas=5

⑨ Edit Deployment

>> kubectl edit deployment my-deployment

Date

• KUBERNETES CONFIGURATION FILE (YAML)

→ Deployment File

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
```

→ defines the object we are creating like Service, StatefulSet, Daemon...

used for identification. label selection. "name" must be unique in a namespace

→ write the specification of deployment

→ no. of pods to run

deployment manages pods with this label
Must match pod template label exactly

Date

```
template:
  metadata:
    labels:                    → must match the selector label
      app: myapp
  spec:                        → specs for a pod
    containers:                → container is an array therefore we can multiple containers
    - name: myapp-container
      image: nginx:1.25

    ports:
      - containerPort: 8080

    env:
      - name: ENV
        value: "dev"
```

Date

→ Service File

```
apiVersion: v1
kind: Service

metadata:
  name: myapp-service

spec:
  type: ClusterIP
  selector:
    app: myapp
  ports:
    - port: 80
      targetPort: 8080
```

spec: ⟶ Can be NodePort ⇒ Expose via Node IP
LoadBalancer ⇒ Cloud LB (AWS, GCP)
ExternalName ⇒ DNS mapping
ClusterIP ⇒ Internal only (default)

selector: app: myapp → ~~else~~ should match the pod label

port: ⟶ port on which client request

targetPort: 8080 ↳ Port on which service will request to pod

should match "containerPort" in pod specification.

optional, if not define default value = port is

client → 80 → myapp-service → 8080 → Pod

Page No.

Date

→ ConfigMap File

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap

data:
  BACKEND_URL: http://myapp.com/api/:5000
  DB_HOST: my-sql-service
  DB_PORT: "3306"
```

→ Secret File

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-appsecrte

type: Opaque
data:                          ↗ base 64 encoded
  DB_USER: abge1d2zt=   # admin
  DB_PASSWORD: geqlm34=   # pass123
```

Date

- Usage:
  ↳ Used in Pod's specification

```
spec:
  containers:
    - name: mysql
      image: mysql

      env:
        - name: MY_SQL_USER
          valueFrom:
            secretKeyRef:
              name: ~~DB_PASSWORD~~ myapp-secret
              Key: DB_PASSWOR

        - name: MYSQL_PASSWORD
        :
```

configMap use krna hoga to bas "secretKeyRef" → "configMapKeyRef".

Date

→ StatefulSet File

↳ Everything is similar to deployment file only we add "volume" here.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: my-sql-service → required
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
```

Date

```
spec:
  containers:
    - name: mysql-container
      image: mysql

      env:
        - name: MYSQL_ROOT_PSWD
          valueFrom:
            secretKeyRef:
              name: mysql-secret
              key: db_password

        - name: MYSQL_USER

      ports:
        - containerPort: 3306

      volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
```

Date

!!

```
volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
```

→ Headless Service

↳ Stateful set requires headless service

```
apiVersion: v1
kind: ~~StatefulSet~~ Service
metadata:
  name: mysql-service

spec:
  clusterIP: None        # Headless
  selector:
    app: mysql
  ports:
    - port: 3306
```

Page No.