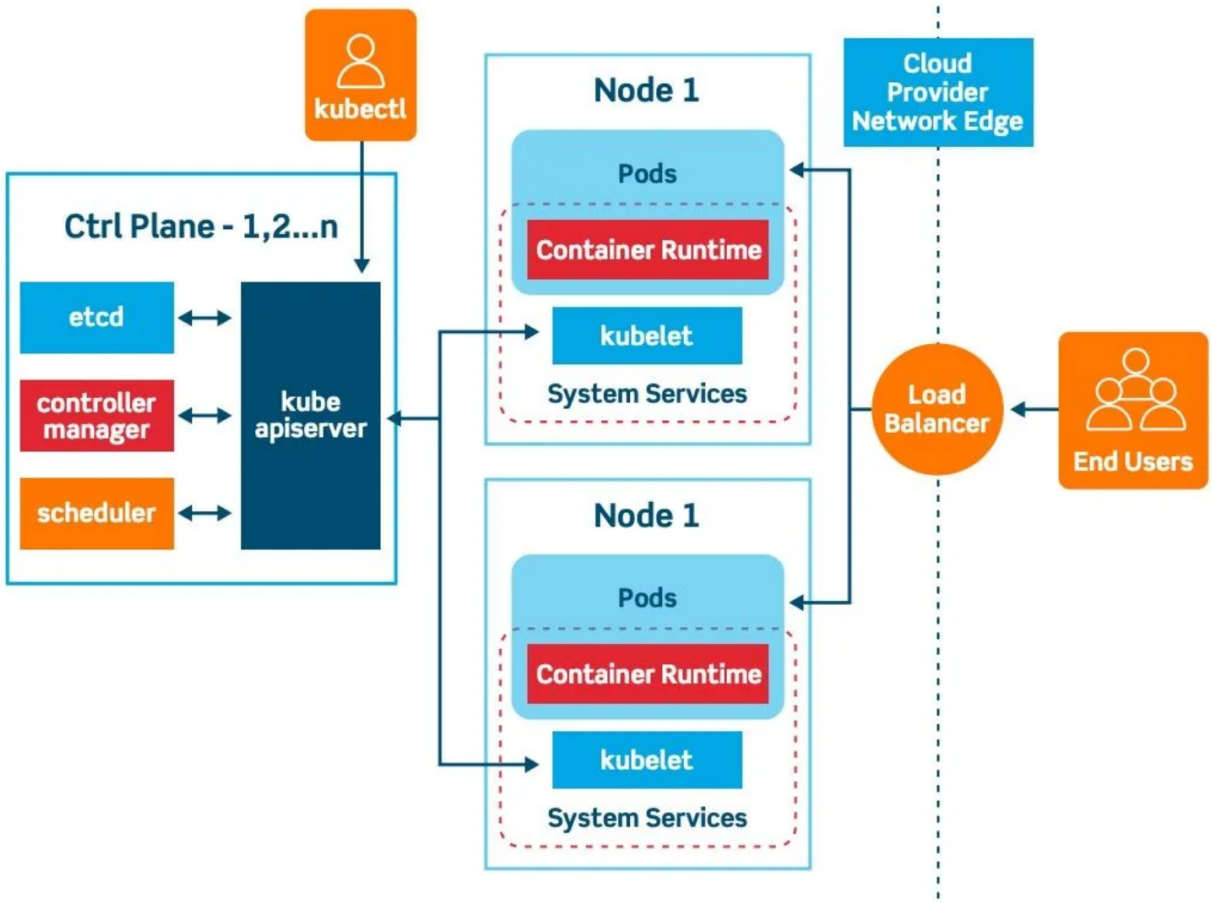
Kubernetes Simplified – Part 1

Handwritten By:
Vaibhav Matkar

kubernetes → container orchestration tool

kubernetes Architecture



- kubernetes cluster → set of nodes → Master, Worker
- control plane → s/w component → mgmt of state of cluster
- Master Node - contains control plane → mgmt of cluster
- Worker Node - Machines where containers actually run

components of master node

① API server → Act as frontend to cluster
↳ Nerves of kubernetes → responsible for communication

② ETCD → stores cluster data in key-value store
↳ Brain of the kubernetes.

③ controll Manager → Manages different controllers
① Node controller → Notice + respond if down
② Replication controller → ensures specified number of pod relica's are running or not
③ Endpoints controller → connects service & pod using labels.
④ serviceAccount Token controller → Watches serviceAccounts → creates secret containing token → Mount token into pods running under that service account.

④ kube-scheduler → Assign each pod to a suitable node

components of node

① kubelet → Make sures that containers are running in pod with help of podspecs
↳ Agent
determines how pod runs

- It does not manages containers not created by k8's

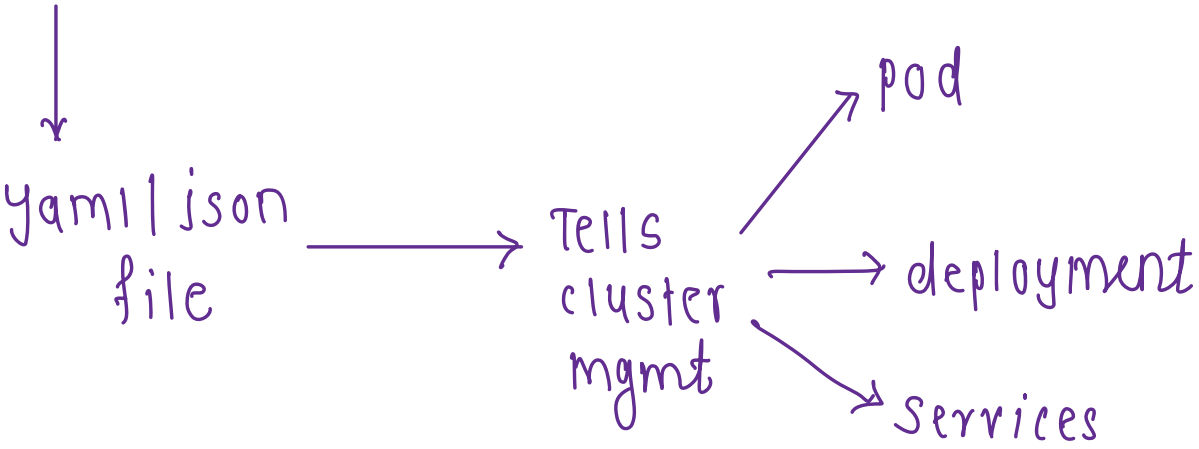
② kube-proxy → Maintains network rules on nodes
ie. Loadbalancing between pods

③ container-runtime → s/w responsible for running containers

④ pods → smallest deployable unit in kubernetes.

kubectl → cmd line tool to interact with k8 cluster.

manifest file in kubernetes



components of manifest file

① apiversion — Usable kubernetes API version

② kind — type of kubernetes resource/object

③ metadata - identifying data about resource

i> name → unique name given to resource

ii) namespace → opt

iii> Labels → key-value pairs

iv) Annotations → Non-identifying meta-data

④ Spec - gives kubernetes desired state of the resource/object

⑤ Status - gives current state of the resource

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: web-application
spec:
    containers:
        - name: web-app
          image: web-app-devansh
          env:
            - name: APP-COLOR
              valueFrom:
                configMapKeyRef:
                    name: app-config
                    key: APP-COLOR
```

Node Affinity → decides on which node pod can run

pod Affinity → decides pod-pod relationship
i.e which pod should run with which another pod

Taint → Reserve node for special workload otherwise repel pods

Toleration → property of pods that allows them to run on node if their capacity matches taint.

Ingress controller
↓
s/w component that decides rules for how external traffic reach services inside cluster

Namespace → Allows to seperate resources without conflict eg. dev/QA/prod

context → parameters / shortcut that tells kubectl which cluster, user & namespace to use

Lifecycle of a pod

① pending → pods created | containers not running | scheduler finding Node to place pods

② Running → pod scheduled on node | containers are created & running

③ succeeded → All containers exited successfully | pod does not restart again

④ failed → Atleast one container terminated with failure | pods won't restart auto.

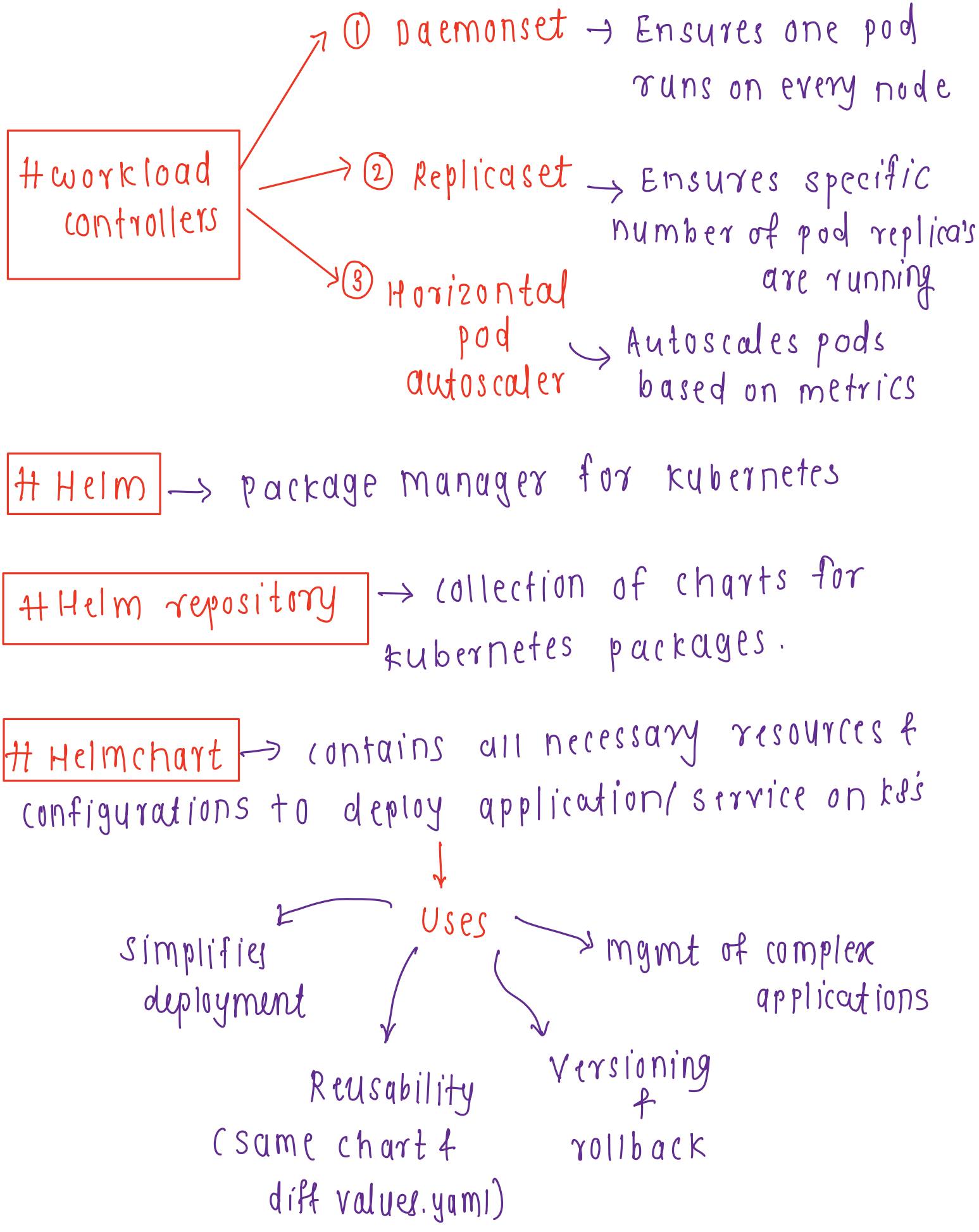
⑤ Unknown → Kubernetes can't communicate with node (Network error)

pod Healthcheck

① Liveness probe → check for alive if fail → restart

② Readiness probe → check pod is ready to serve traffic
if fails → k8 removes service endpoints → No traffic to pod

③ startup Probe → checks container started successfully else pause liveness & readiness probe


#workload controllers
① Daemonset → Ensures one pod runs on every node
② Replicaset → Ensures specific number of pod replica's are running
③ Horizontal pod autoscaler → Autoscales pods based on metrics
Helm → package manager for kubernetes
Helm repository → collection of charts for kubernetes packages.
Helmchart → contains all necessary resources & configurations to deploy application/service on k8's
Uses
simplifies deployment
mgmt of complex applications
Reusability (same chart & diff values.yaml)
Versioning & rollback

Kubernetes object → persistent entities within kubernetes that defines desired state of cluster.

eg. pods / deployment / Replicaset / statefulset.

Deployment in Kubernetes

↳ Gives declarative updates to pods & replicasets

↳ common way to run applications

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
```

Replicaset in kubernetes

↳ Ensures specified number of pods are running at all times.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-rs-pod
  matchExpressions:
  - key: env
  operator: In
  values:
  - dev
template:
  metadata:
    labels:
      app: nginx-rs-pod
      env: dev
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
     - containerPort: 80
```

statefulset in kubernetes → for stateful applications having stable network identities & persistent storage.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: registry.k8s.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

Daemonset in Kubernetes

↳ Ensures one pod runs on every node.

↳ commonly used for logging / monitoring / network agent

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: example-daemonset
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: example
  template:
    metadata:
      labels:
        app.kubernetes.io/name: example
    spec:
      containers:
      - name: pause
        image: registry.k8s.io/pause
      initContainers:
      - name: log-machine-id
        image: busybox:1.37
        command: ['sh', '-c', 'cat /etc/machine-id > /var/log/machine-
        volumeMounts:
        - name: machine-id
          mountPath: /etc/machine-id
          readOnly: true
        - name: log-dir
          mountPath: /var/log
      volumes:
      - name: machine-id
        hostPath:
          path: /etc/machine-id
          type: File
      - name: log-dir
        hostPath:
          path: /var/log
```

job in kubernetes

↳ Runs the pod till completion / for batch processing

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

cronjob in kubernetes

↳ Runs job on schedule. eg. for backup script

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

configmap working

① define configmap → Yaml
→ CLI

② use it in pod by 3 ways:

① Environment variable → Inject configmap value as Ev. in pod

② Command-line arguments → Give reference during container command

③ mounted files → mount configmap values as file in container

Deploy pod in kubernetes

① using YAML manifest file

② using kubectl command directly

scaling a deployment

① manual scaling (kubectl scale command)

② Declarative scaling
Edit replicas in deployment.yml

③ Autoscaling (HPA - based on cpu/memory)

Accounts in Kubernetes

① User account → for users → To gain access to cluster

② service account → gives identity to pod to talk to k8 API

↓

Namespace has defaud service account

creating a own service account:

① create a service account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cicd-deployer
  namespace: deployment
---
```

② create Role containing permissions.

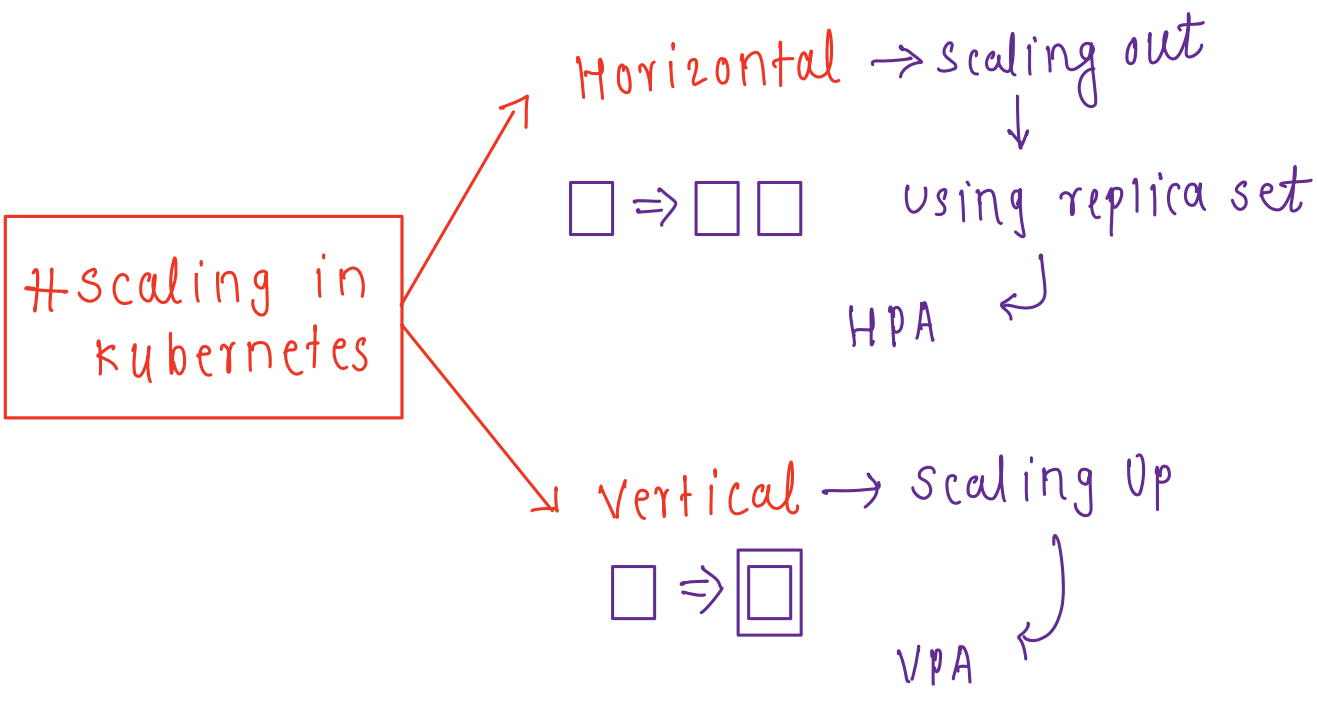
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-manager
  namespace: deployment
rules:
- apiGroups: ["apps", ""]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
---
```

③ Bind role to service Accounts:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cicd-deployer-binding
  namespace: deployment
subjects:
- kind: ServiceAccount
  name: cicd-deployer
  namespace: deployment
roleRef:
  kind: Role
  name: deployment-manager
  apiGroup: rbac.authorization.k8s.io
```

④ Attach it to pod:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: example-container
    image: example/image
  serviceAccountName: example-serviceaccount
```



#Kubernetes deployment → High level resource that manages replica sets.

↓

provides → Rollback

↓

Rolling update

Updates

Kubernetes	Docker swarm
① complicated setup & strong cluster	① simple setup & weak cluster
② Kubernetes dashboard GUI	② No GUI
③ Auto scaling	③ No auto-scaling
④ shares storage volume with other containers in same pod	④ shares storage volume with any other containers
⑤ Build in tool for logging & monitoring	⑤ Uses 3rd party tools like ELK
⑥ Load balancing with manual intervention	⑥ Automated load balancing

Steps to deploy highly available application using kubernetes

i) create kubernetes cluster with 'n' nodes across multiple Az/ regions

ii) create k8 deployment → specifies replica

iii) create k8 service → provides stable IP & DNS to access applicn

iv) configure ingress to load-balance traffic across replica's.

Self-healing in k8's → If 2 replica → 1 failed → pod / container / Node → Automatically same is created → Ensures no of replica's are maintainned.

Types of clusters in kubernetes
- single Node cluster
- Multi-Node cluster

kubernetes Namespace

Logical clusters inside kubernetes cluster

- system Namespace
  - ① default
  - ② kube-system
  - ③ kube-public
  - ④ kube-node-lease
- custom Namespace
  - created
    - ① dev
    - ② QA
    - ③ prod

storage management in kubernetes.

① volume → Way to share data between containers in pod via filesystem

- (i) configmap → inject config data into pods (read only)
- (ii) emptyDir → created when pod is assigned to node & deleted when pod is deleted. (Temp)
  - → created on file system of a node.
  - → This directory is shared across all containers in pod

(iii) hostpath → file is mounted in pod from node file system.

(iv) persistentVolumeclaim [PVC] → request for storage made by pod

(v) secret → TO pass sensitive information to pod (readonly)

(2) persistent volume → provided by administrator at cluster level

(3) storage classes → for dynamic provisioning of storage
→ possible to add retain policy

create command → create k8 resource → based on yaml/json → Error! if resrc already exists

apply command → checks if resrc already exists
↳ update → Based on yaml/json
→ create new resrc ✗

service discovery → how one service found other service internally using DNS. [As pod ip changes]

kube-state-metrics → simple service that listens k8 API server → generate metrics about states of the objects.

Service mesh → communication between micro-services
eg. Istio & Linkerd

pod priority and preemption

Assigns priority to pod

Removing low priority pod to give space to high priority pod

Pod preset → Inject environment variables secrets/ volume mounts at pod creation time

pod scheduler → Decides where pod should run → give more control over pods scheduling

Cluster Federation → kubefed → mgmt of a multiple k8's cluster as a single entity.

Resource Quotas → Limits resource utilisation
↳ prevents consumption of all cluster resources by any one application

Labels → key value pairs attached to kubernetes resource eg. env = prod , app = frontend

#Selectors → Used to select k8 object based on labels

#Annotations → stores metadata about objects

pod disruption budget [PDB] → k8 object ensures your application remains highly available during voluntary disruptions by limiting how many pods can be down at same time.

- minAvailable → min pods that must stay up
- maxUnavailable → max pods that can be disrupted at once.

#minikube → runs single-node kubernetes cluster inside VM.

#skaffold → command line tool → Automate → Build → push → deploy in kubernetes

#kompose → migration tool to convert docker compose file into kubernetes manifest. i.e. from local deployments to kubernetes deployment

#kubeadm → Tool for quickly setting up kubernetes cluster using best practices. like installer.

#kernel tunning → adjusting linux parameters on nodes / pods to optimize system performance.

- Network
- Memory
- Security

#importance of container image security

i> Avoid vulnerable libraries / Base images

ii> Eliminate hidden malware

iii> Secure Embedded secrets (tokens/ SSH keys)

iv> For image integrity and authenticity

v> Reduce attack surface for images

#configuration of local storage in K8's / LPV's

① create PV & apply → ② Deploy PVC → ③ Bind PVC's to pods

CRI [container Runtime Interface]

↓

Interface required for the communication between kubernetes & container runtimes.

CNI [container Network interface]

- calico
- Flannel
- weave

implement

↓

Interface required for the networking in kubernetes

- Assign ip addresses to pod
- mgmt of network connectivity
  - pod
  - Nodes
  - Ext network
- mgmt of network policies & Routing

CSI [container storage Interface]

- AWS EBS
- GCP persistent disk

↓

Interface used for the mgmt of persistent storage in k8's

- provisioning
- Attachment
- Mount

Kubernetes Commands & Usages:

1- Basic Cluster Information:

- `kubectl cluster-info`: Display information about the cluster.
- `kubectl version`: Display version info.

2- Working with Nodes and Cluster:

- `kubectl get nodes`: List nodes in a cluster.
- `kubectl describe node <node-name>`: Show details of a specific node.

3- Working with Pods:

- `kubectl get pods`: List all pods in all namespaces.
- `kubectl run <name> --image=<image>`: Deploy a new Pod with a given image.
- `kubectl describe pod <pod-name>`: Describe a specific pod.
- `kubectl logs <pod-name>`: Fetch the logs from a pod.
- `kubectl delete pod <pod-name>`: Delete a specific pod.

4- Working with Deployments:

- `kubectl create deployment <name> --image=<image>`: Create a new deployment.
- `kubectl get deployments`: List all deployments.
- `kubectl describe deployment <deployment-name>`: Describe a specific deployment.
- `kubectl scale deployment <deployment-name> --replicas=<num>`: Scale up/down a deployment.

5- Working with Services:

- `kubectl expose deployment <name> --type=LoadBalancer --port=8080`: Expose a deployment as a service.
- `kubectl get services`: List all services.
- `kubectl describe service <service-name>`: Describe a specific service.

6- Config and Storage:

- `kubectl get configmaps`: List all config maps.
- `kubectl create configmap <name> --from-file=<path>`: Create a config map from a file.
- `kubectl get secrets`: List all secrets.
- `kubectl create secret`: Create a secret.
- `kubectl get pv`: List all persistent volumes.
- `kubectl get pvc`: List all persistent volume claims.

7- Namespaces and Context:

- `kubectl get namespaces`: List all namespaces.
- `kubectl config get-contexts`: Show all contexts.
- `kubectl config use-context <context-name>`: Switch to a different context.

8- Others:

- `kubectl apply -f <filename>`: Apply a configuration from a file.
- `kubectl delete -f <filename>`: Delete resources defined in a file.
- `kubectl exec -it <pod-name> -- /bin/sh`: Execute a command inside a running pod.
- `kubectl port-forward <pod-name> <local-port>:<pod-port>`: Forward a port from a running pod to a local port.

9- Advanced:

- `kubectl get all`: List all resources.
- `kubectl rollout status deployment/<deployment-name>`: View the rollout status of a deployment.
- `kubectl rollout history deployment/<deployment-name>`: View the history of a deployment.
- `kubectl rollout undo deployment/<deployment-name>`: Rollback to a previous version of a deployment.

10- Monitoring & Logging:

- `kubectl top nodes`: Display resource (CPU/Memory/Storage) usage of nodes.
- `kubectl top pods`: Display resource (CPU/Memory/Storage) usage of pods.

11- Autoscaling:

- `kubectl autoscale deployment <deployment-name> --min=<min-pods> --max=<max-pods> --cpu-percent=<cpu-util-percentage>`: Auto scale a deployment based on CPU utilization.

12- Working with Helm:

- `helm list`: List releases.
- `helm install <chart>`: Install a helm chart.
- `helm uninstall <release-name>`: Uninstall a helm release.

13- Using Network Policies:

- `kubectl get networkpolicies`: List all network policies.
- `kubectl describe networkpolicy <policy-name>`: Describe a specific network policy.

14- Using CronJobs and Jobs:

- `kubectl get cronjobs`: List all cronjobs.
- `kubectl get jobs`: List all jobs.
- `kubectl logs job/<job-name>`: Fetch logs from a job.

15- Working with RBAC:

- `kubectl get roles`: List all roles in the current namespace.
- `kubectl get clusterroles`: List all cluster roles.

16- Using kubectl Plugins:

- `kubectl krew search`: Search plugins available for kubectl.
- `kubectl krew install <plugin-name>`: Install a kubectl plugin.