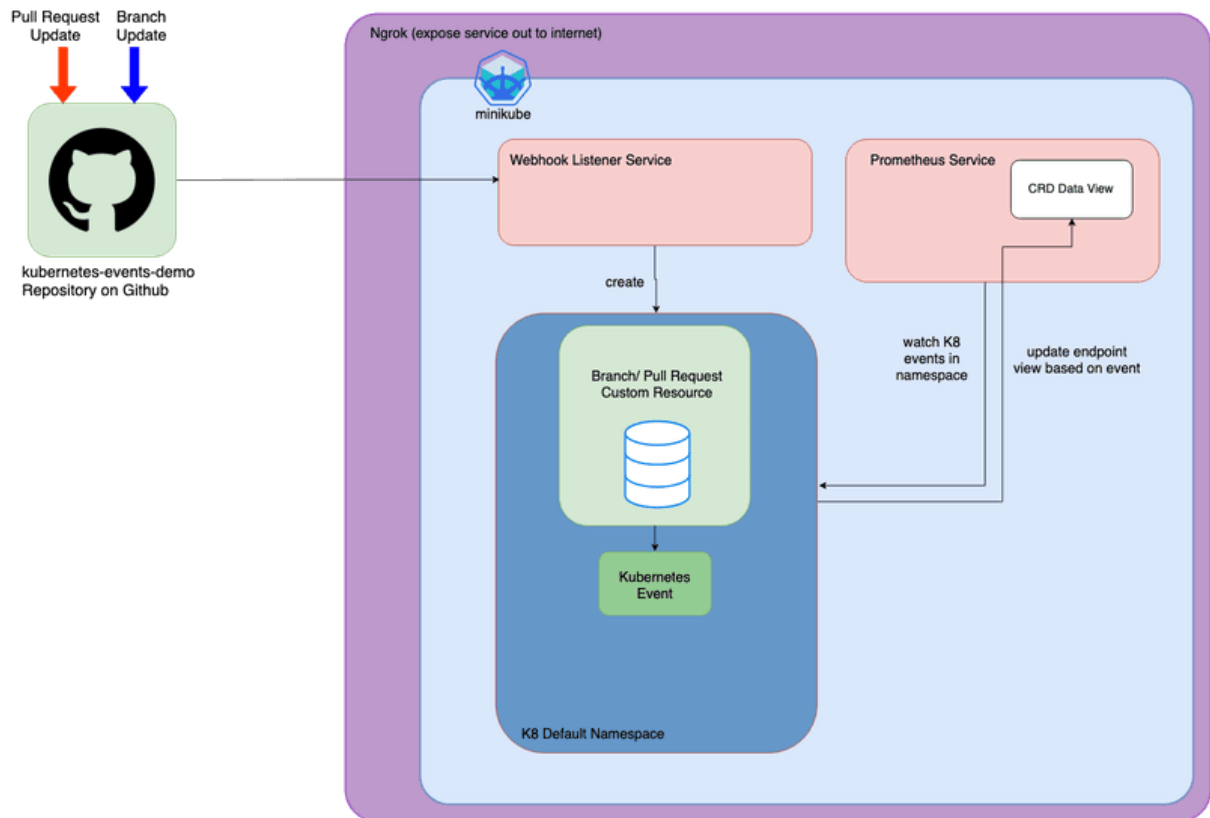
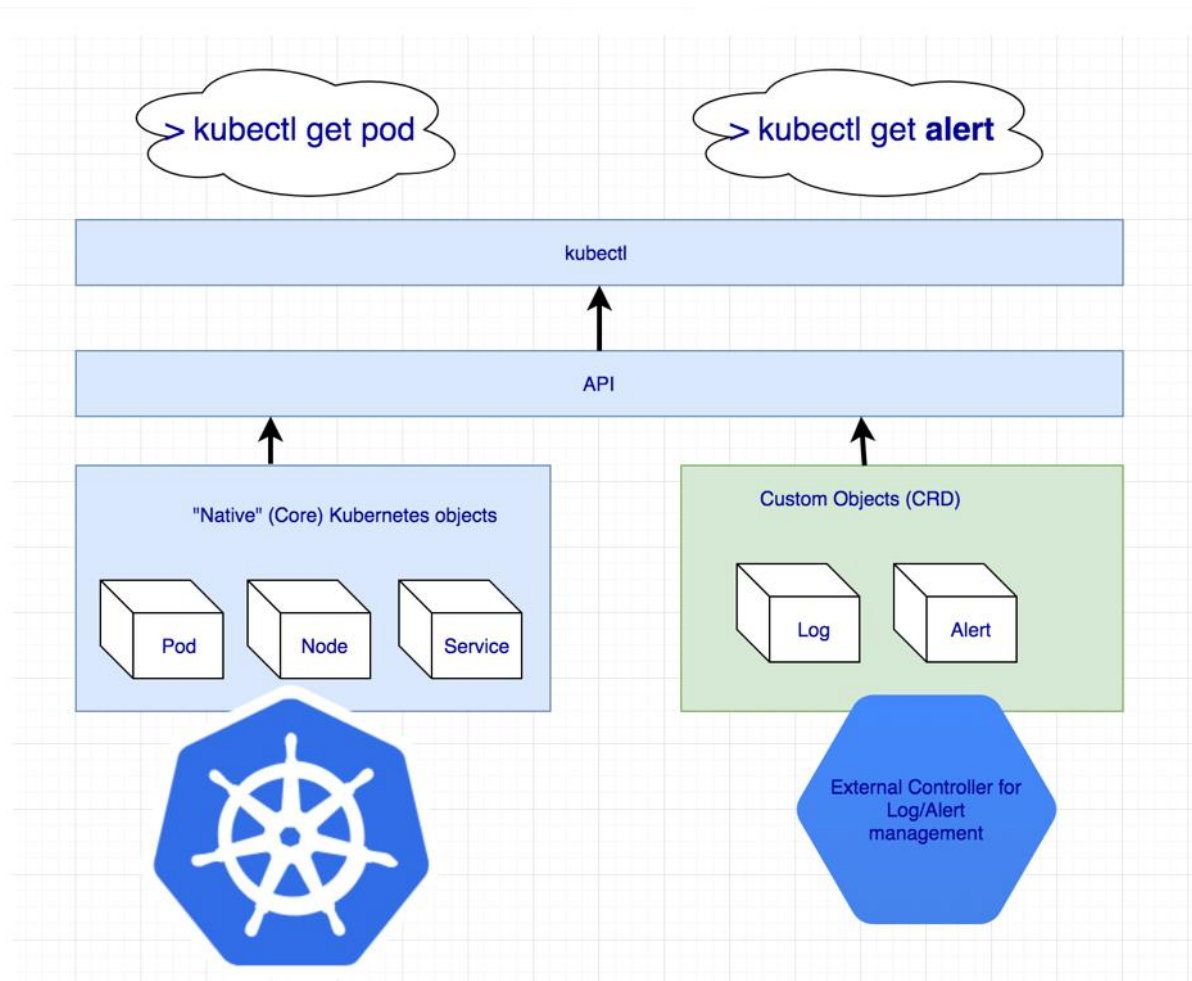


Module 2: Kubernetes Deep Dive — Detailed Notes for Participants

1. Custom Resource Definitions (CRDs)





1.1 What Are CRDs?

- **CRDs allow you to extend Kubernetes** with your own resource types.
- Kubernetes has built-in objects like Pods, Deployments, Services.
- Using CRDs, you create **custom objects** such as:
 - KafkaCluster
 - MySQLBackup
 - Certificate
 - PrometheusRule

These allow teams to define new APIs under:

apiVersion: <group>/<version>

kind: <CustomResource>

1.2 Use Cases

- Operators (Prometheus Operator, Kafka Operator, ArgoCD)
- Automating lifecycle management of applications

- Custom controllers watching CRDs and taking actions automatically

1.3 Example CRD

apiVersion: apiextensions.k8s.io/v1

kind: CustomResourceDefinition

metadata:

name: databases.mycompany.com

spec:

group: mycompany.com

versions:

- name: v1

served: true

storage: true

schema:

openAPIV3Schema:

type: object

properties:

size:

type: integer

version:

type: string

scope: Namespaced

names:

plural: databases

singular: database

kind: Database

1.4 Working With Custom Resources

kubectl get crd

kubectl get databases

kubectl describe database sample-db

kubectl apply -f my-database.yaml

1.5 How CRDs Work

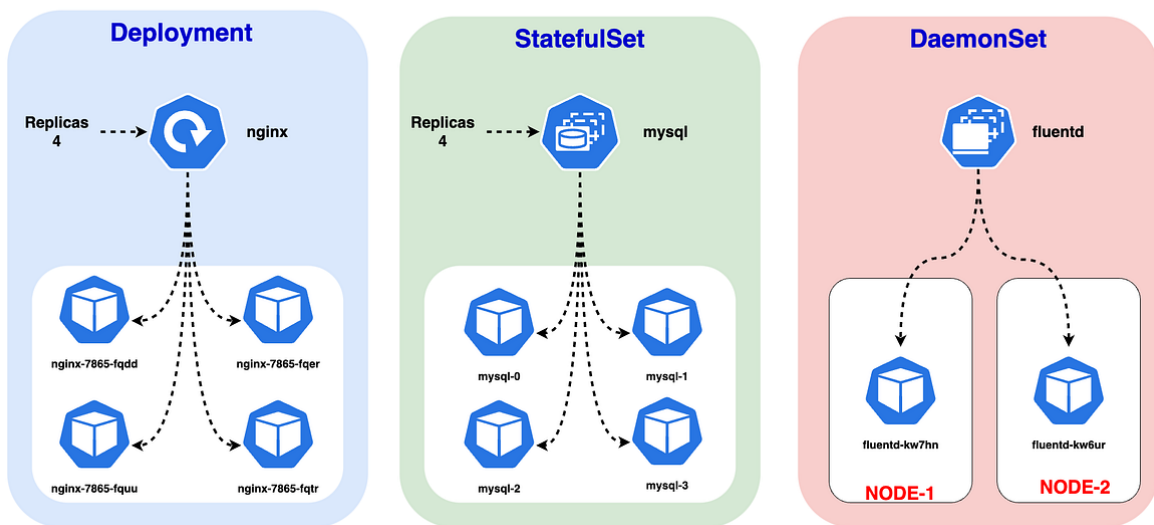
- A **Controller** detects changes in custom resources.
- Performs **reconciliation loops**.
- Ensures the real-world state matches the desired state.

2. StatefulSets, DaemonSets & Advanced Pod Scheduling

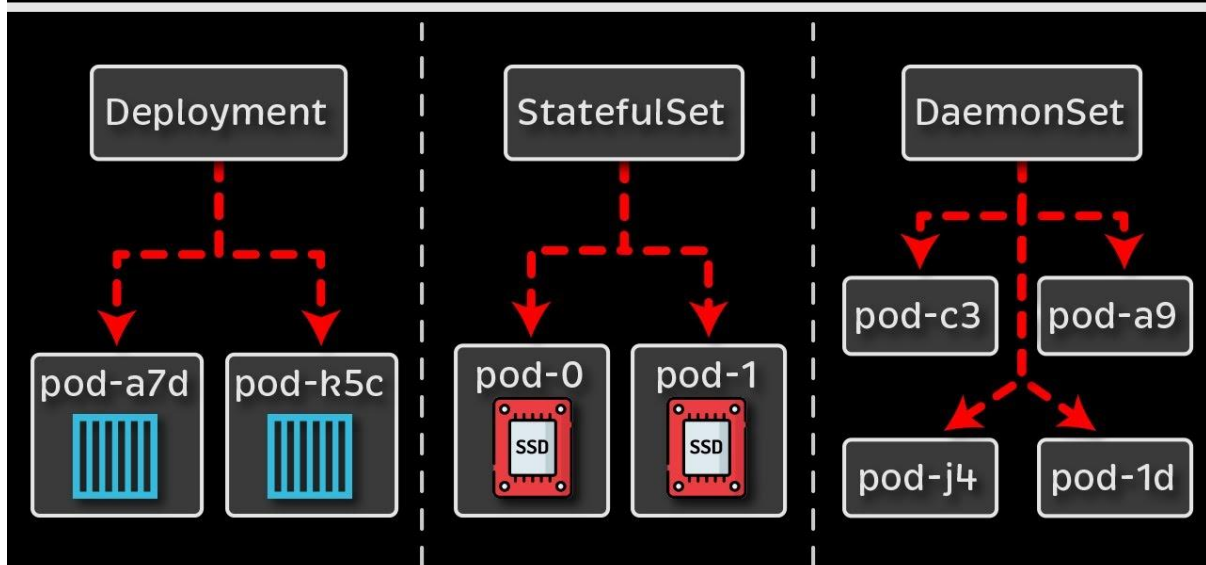


Deployments, StatefulSets, and DaemonSets

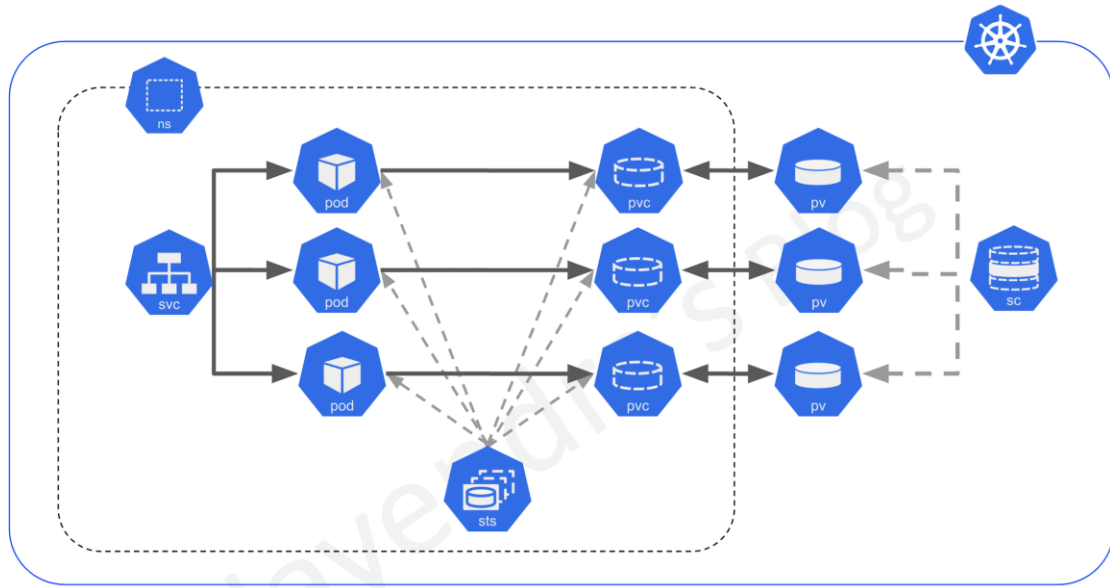
Anvesh Muppada



Deployment vs. StatefulSet vs. DaemonSet



StatefulSet Architecture



2.1 StatefulSets

Used for **stateful** workloads:

- Databases (MongoDB, Cassandra)
- Message queues (Kafka, RabbitMQ)
- Distributed systems (ElasticSearch)

Key features:

- Stable **network identity** (pod-0, pod-1, ...)
- Stable **persistent storage**
- Ordered **deployment, update, deletion**

Example:

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mongo

spec:

serviceName: "mongo"

replicas: 3

```
selector:
  matchLabels:
    app: mongo
template:
  metadata:
    labels:
      app: mongo
  spec:
    containers:
      - name: mongo
        image: mongo:7
        volumeMounts:
          - name: data
            mountPath: /data/db
    volumeClaimTemplates:
      - metadata:
          name: data
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 10Gi
```

2.2 DaemonSets

Used when you need **one pod per node**, such as:

- Log shippers (**Fluentd**, **Filebeat**)
- Monitoring agents (**Prometheus Node Exporter**)
- Security agents (**Falco**)
- Storage agents

Example:

apiVersion: apps/v1

```
kind: DaemonSet
metadata:
  name: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    metadata:
      labels:
        app: node-exporter
    spec:
      containers:
        - name: exporter
          image: prom/node-exporter
```

2.3 Advanced Pod Scheduling

Node Affinity

Schedule pods to preferred nodes.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: disktype
              operator: In
              values:
                - ssd
```

Taints & Tolerations

Prevent pods from running on specific nodes unless they tolerate.

Taint a node:

```
kubectl taint nodes node1 dedicated=db:NoSchedule
```

Toleration:

tolerations:

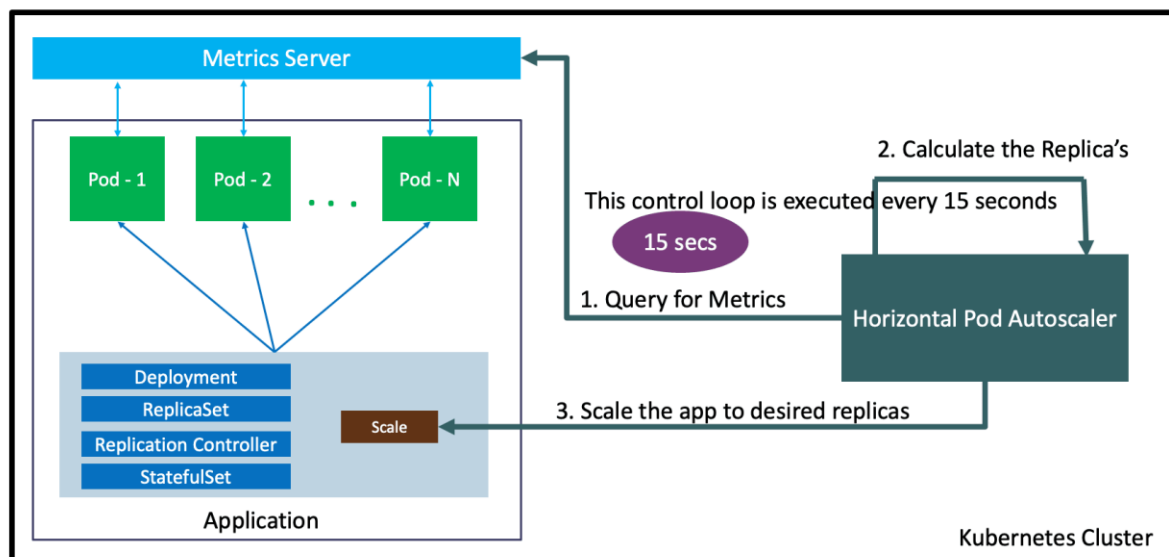
- key: "dedicated"
- operator: "Equal"
- value: "db"
- effect: "NoSchedule"

Pod Topology Spread Constraints

Ensures even distribution across nodes/availability zones.

3. Horizontal & Vertical Pod Autoscaling (HPA & VPA)

How HPA works?



3.1 Horizontal Pod Autoscaler (HPA)

HPA automatically increases or decreases number of pods based on:

- CPU
- Memory
- Custom metrics (Prometheus, Datadog)
- External metrics (queue length, requests/sec)

Example:

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: web-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: web-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

Commands:

kubectl get hpa

3.2 Vertical Pod Autoscaler (VPA)

VPA adjusts **CPU & memory requests/limits** for pods.

Use cases:

- Workloads with unpredictable resource usage
- ML systems
- Batch workloads

Modes:

- **Off**: only recommendations
- **Auto**: updates resources & restarts pods
- **Initial**: applies recommendations only at startup

Example:

apiVersion: autoscaling.k8s.io/v1

kind: VerticalPodAutoscaler

metadata:

name: webapp-vpa

spec:

targetRef:

apiVersion: "apps/v1"

kind: Deployment

name: webapp

updatePolicy:

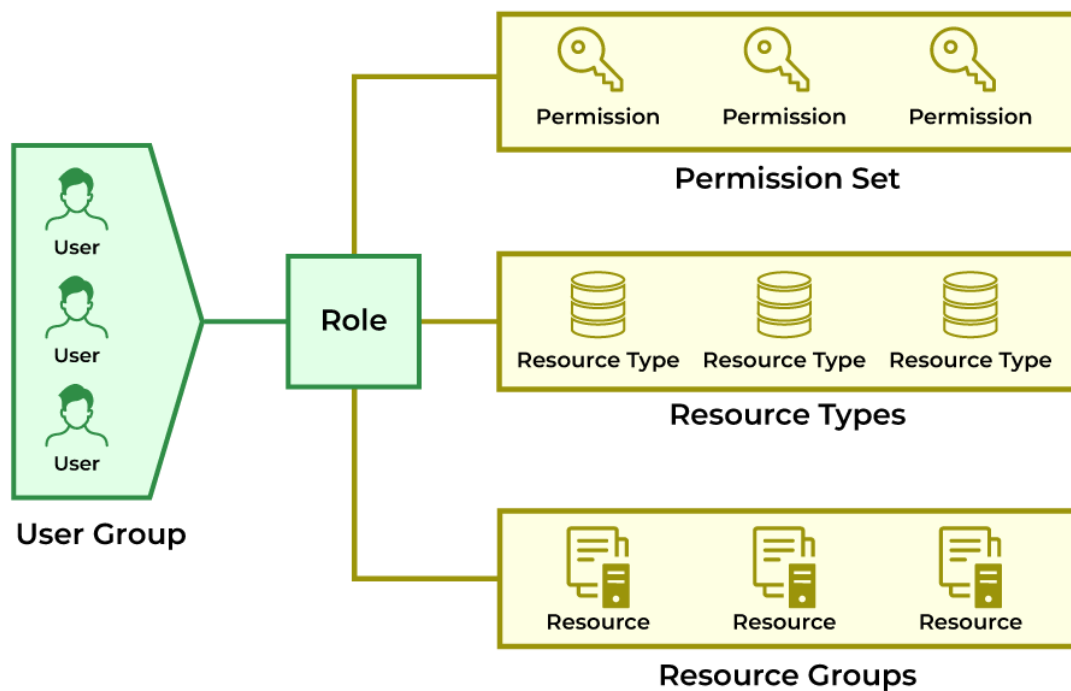
updateMode: "Auto"

3.3 Cluster Autoscaler

Not part of HPA/VPA but often used:

- Adds/removes worker nodes based on cluster demand
- Works with AWS EKS, GKE, AKS

4. Kubernetes Security: RBAC, Network Policies



4.1 RBAC (Role-Based Access Control)

RBAC controls **who can do what** in cluster.

4.1.1 Types of RBAC Objects

Object	Purpose
Role	Permissions within a namespace
ClusterRole	Permissions cluster-wide
RoleBinding	Attach Role to user/group/service account
ClusterRoleBinding	Attach ClusterRole

Example Role:

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

namespace: dev

name: pod-reader

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "watch", "list"]

Bind it:

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: read-pods

namespace: dev

subjects:

- kind: User

name: vivek

roleRef:

kind: Role
name: pod-reader
apiGroup: rbac.authorization.k8s.io

4.2 Network Policies

Restrict traffic **between pods**, like firewall rules.

Policies define:

- **Ingress** (incoming)
- **Egress** (outgoing)

Example: Allow traffic only from frontend to backend:

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: backend-policy

namespace: app

spec:

podSelector:

matchLabels:

role: backend

ingress:

- from:

- podSelector:

matchLabels:

role: frontend

Benefits:

- Block lateral movement by attackers
 - Improve zero-trust security
 - Reduce blast radius during compromise
-

Module 2 Summary (Quick Recap)

Topic	Summary
CRDs	Extend Kubernetes with custom APIs
StatefulSets	Persisted pods with stable identity
DaemonSets	One pod per node (monitoring, logging)
Scheduling	Affinity, anti-affinity, taints/tolerations
HPA/VPA	Autoscaling based on metrics
RBAC	Define permissions (who can do what)
Network Policies	Restrict pod-to-pod communication