KUBERNETES HANDBOOK

From Beginner to Expert

For Cloud & DevOps Engineers

By :

SANDIP DAS

LearnXOps.com

Table of Contents

ABOUT ME

Hi! I’m Sandip Das, a Senior Cloud, DevOps & MLOps Engineer with 12+ years of experience, primarily working on production-level Kubernetes projects. Around 70-80% of my daily work involves Kubernetes, focusing on automation, scaling, and optimizing workloads for clients across the US, UK, EU, and UAE.

LearnXOps.com

Introduction

What is Kubernetes ?

Kubernetes is an **open-source container orchestration platform** that automates deployment, scaling, and management of containerized applications. It enables efficient workload distribution, self-healing, and service discovery across a cluster of nodes.

Why use Kubernetes?

- **Scalability**: Automatically scales applications horizontally and vertically based on demand, ensuring efficient handling of dynamic workloads.
- **High Availability**: Ensures applications stay up with self-healing capabilities, automatic rescheduling, and failover mechanisms.
- **Efficient Resource Utilization**: Optimizes container scheduling and resource allocation, ensuring cost-effectiveness and performance.
- **Automation**: Handles deployment, updates, and rollbacks seamlessly, reducing manual intervention and human errors.
- **Portability**: Runs across different cloud providers and on-premises environments, ensuring flexibility and avoiding vendor lock-in.
- **Service Discovery & Load Balancing**: Distributes traffic efficiently to services, ensuring optimal performance and reliability.
- **Security & Isolation**: Supports Role-Based Access Control (RBAC), namespaces, and network policies for secure, multi-tenant environments.
- **Extensibility**: Easily integrates with CI/CD pipelines, monitoring tools, and custom controllers, enabling advanced automation and observability.
- **Microservices Management:** Efficiently orchestrates decentralized services with automated deployment, service discovery, and load balancing.
- **Cloud-Native Ecosystem**: Seamlessly integrates with cloud services, DevOps tools, and modern application development workflows.

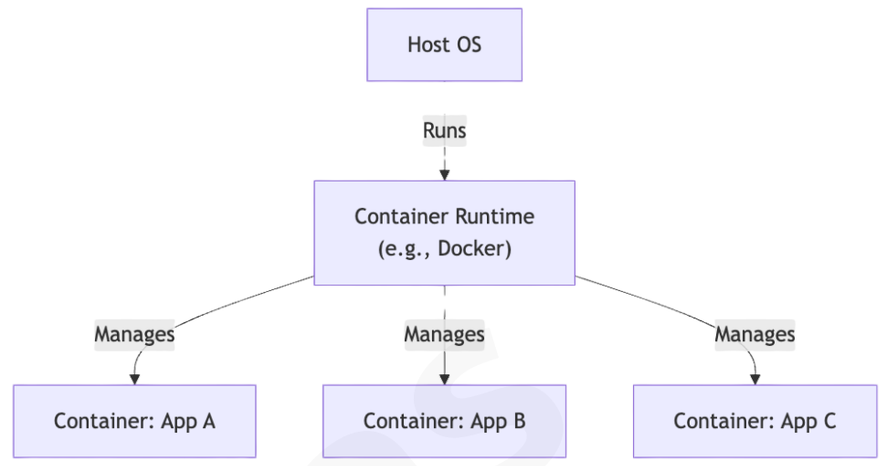
Introduction

Before Diving Deep into Kubernetes, UNDERSTAND this:

Containers:

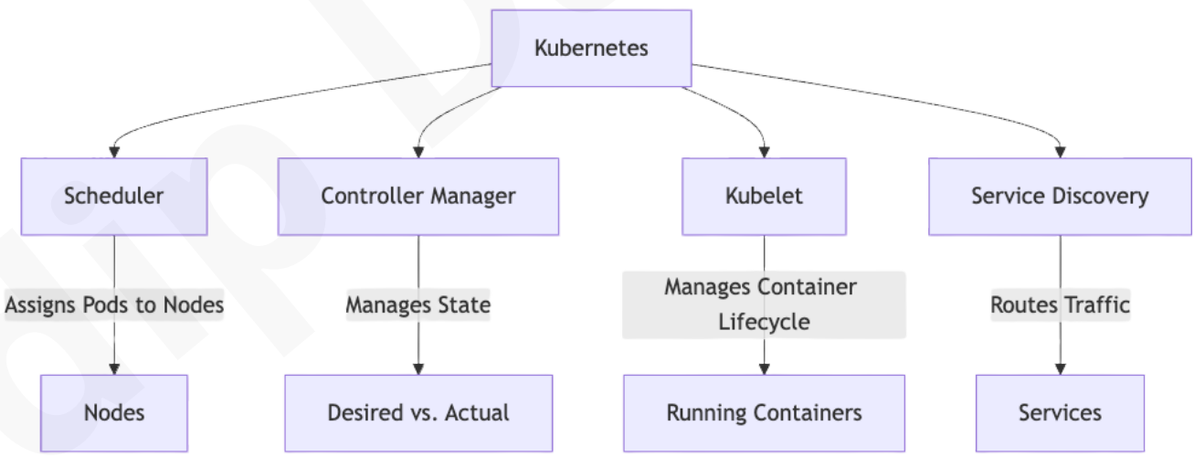
Containers are **lightweight**, **portable**, and **isolated** environments that **package** applications and their dependencies.

Kubernetes orchestrates multiple containers across different nodes in a cluster.



Orchestration:

Orchestration means the **automation** of **container lifecycle** e.g. **deploy**, **scale**, **heal**, **network**

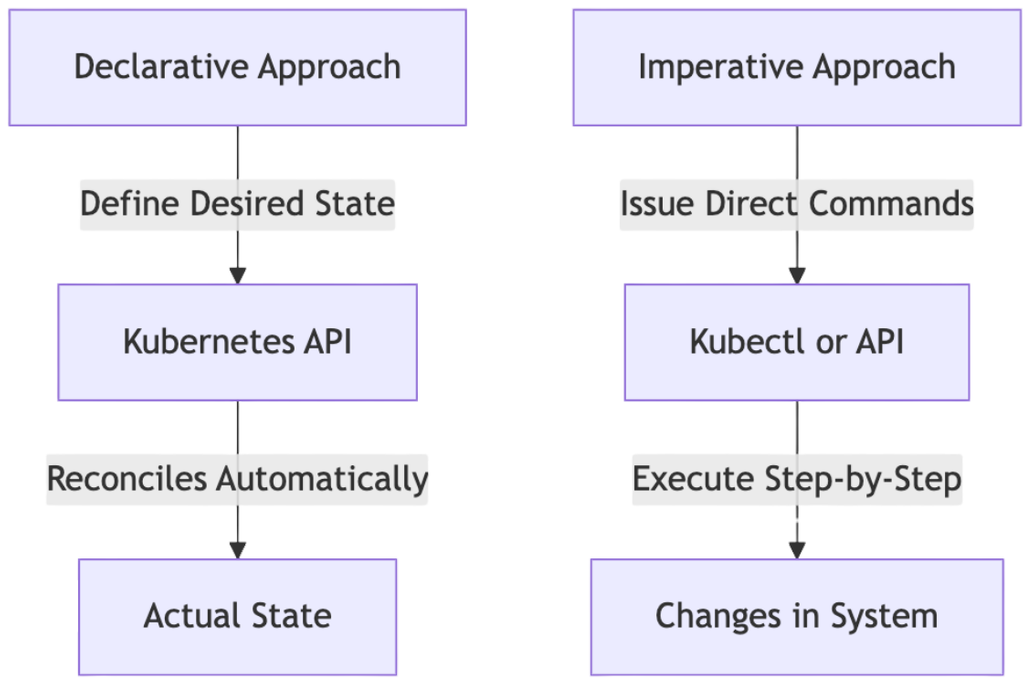


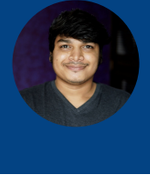
Orchestration in **Kubernetes** refers to the **automated deployment**, **scaling**, and **management** of containerized applications. Kubernetes ensures **high availability**, **self-healing**, and **efficient resource utilization.**

Declarative vs. Imperative Approaches:

Declarative: Define desired state (YAML/JSON) and Kubernetes ensures it is maintained.

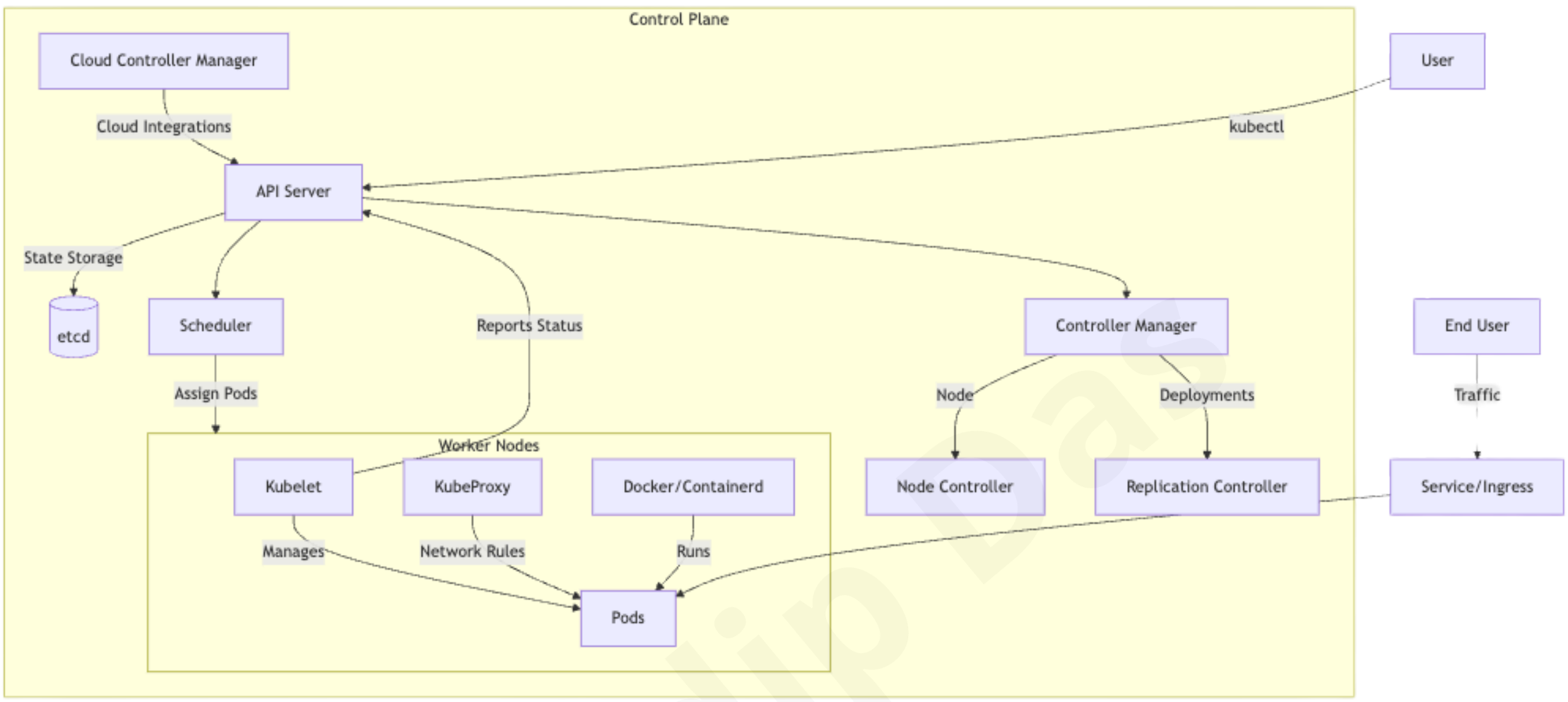
Imperative: Issue commands (e.g., kubectl create). Explicitly tell Kubernetes what to do at each step.





Introduction

Kubernetes Architecture Overview



Control Plane (Master Nodes)

The Control Plane is responsible for managing and orchestrating the Kubernetes

- **API Server**: Frontend for cluster interactions (REST API).
- **etcd**: Distributed key-value store for cluster state.
- **Scheduler**: Assigns pods to nodes based on resources.
- **Controller Manager**: Runs core control loops (e.g., Node, ReplicaSet).
- **Cloud Controller Manager**: Manages cloud-specific integrations (e.g., load balancers).

Worker Nodes

Worker nodes execute the actual application workloads by running containers inside Pods.

- **Kubelet:** Agent managing pods and reporting node status.
- **kube-proxy:** Maintains network rules for pod communication.
- **Container Runtime:** Runs containers (e.g., Docker, Containerd).

Kubernetes Networking & Load Balancing / Add-ons

Networking in Kubernetes ensures seamless Pod-to-Pod communication, service discovery, and traffic routing.

- **Ingress -** Routes external traffic to services.
- **Service -** Provides stable network identity for Pods.
- **CNI (Container Network Interface) -** Ensures networking between Pods (Flannel, Calico).

Introduction

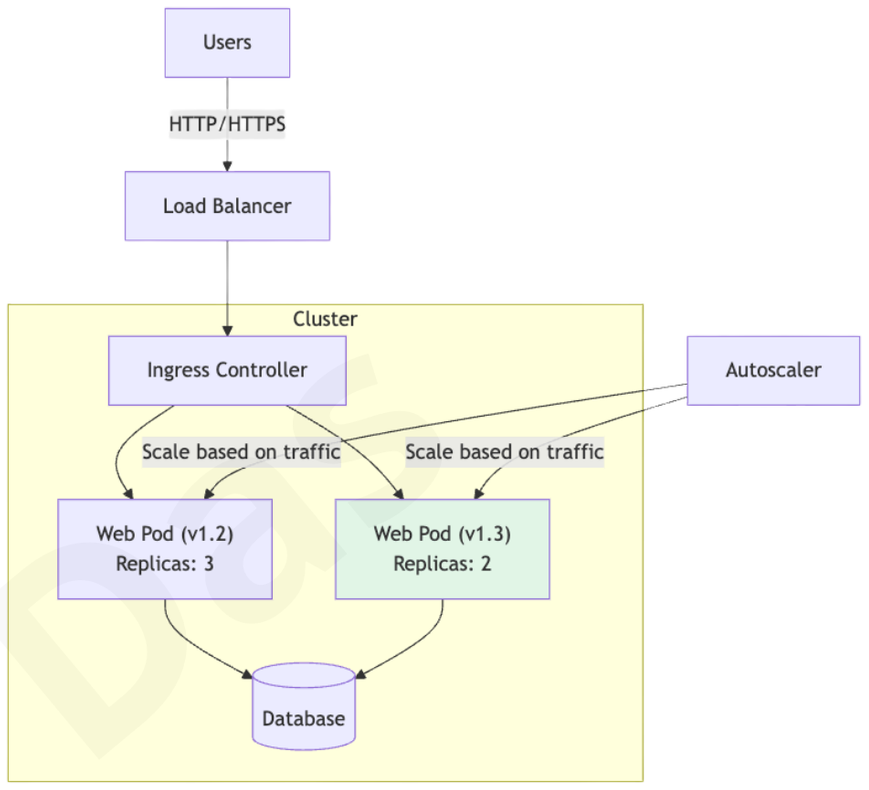
Now, let’s discuss the Kubernetes usecases:

Web Applications

Kubernetes efficiently manages scalable, resilient, and highly available web applications, automatically distributing traffic and handling failures.

Key Features:

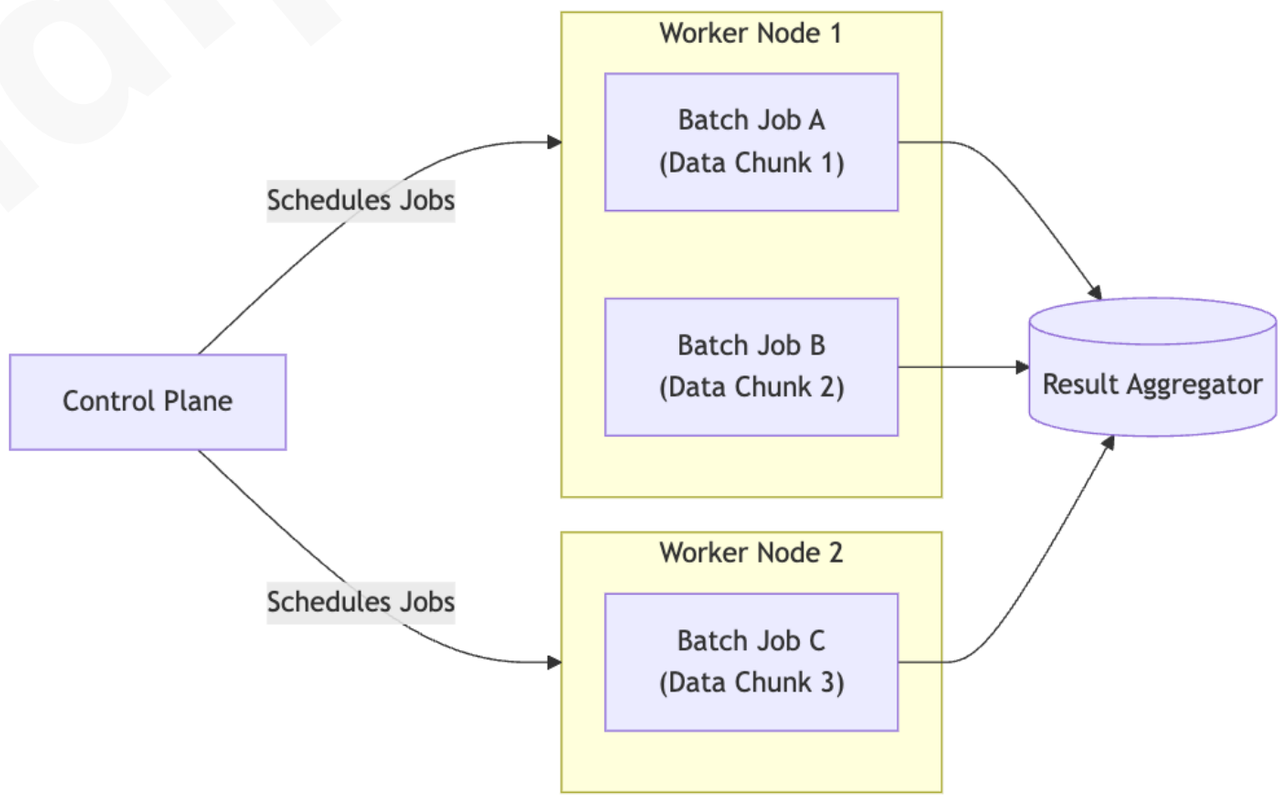
- **Load balancing** using Services and Ingress
- **Auto-scaling** with Horizontal Pod Autoscaler (HPA)
- **Database** connection with **Persistent Storage**



Batch Processing

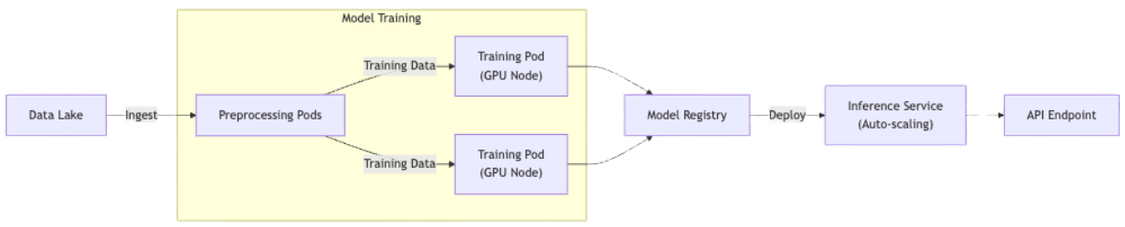
It’s all about job **scheduling**, **parallel tasks**, and **resource management**

Kubernetes is ideal for batch processing jobs, handling periodic tasks such as data processing, video encoding, and report generation.



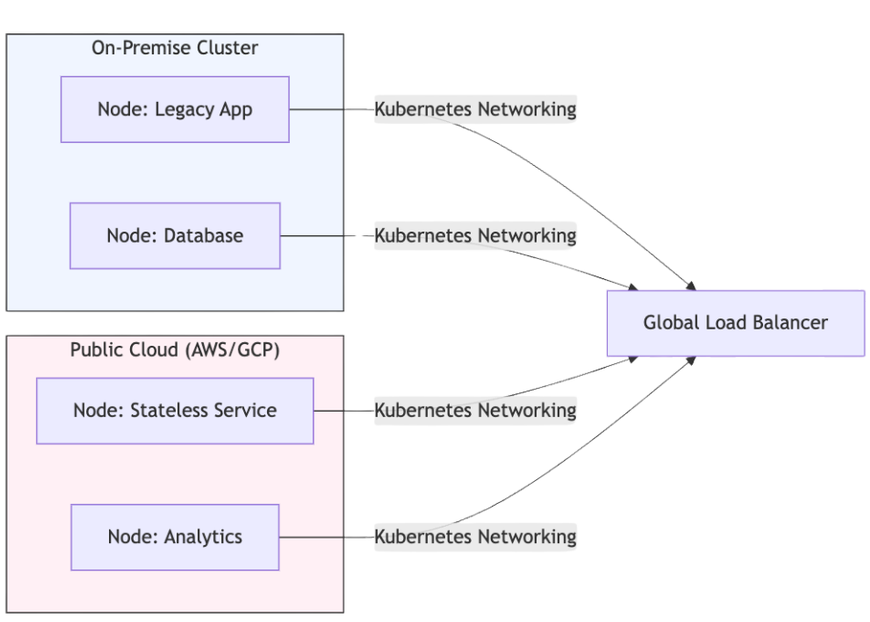
Machine Learning

Kubernetes supports ML workloads by managing training jobs, inference services, and distributed computing using tools like Kubeflow.



Hybrid Clouds

Kubernetes enables hybrid cloud architectures, allowing workloads to run across on-premises and cloud environments.



@Sandip Das

Set-up Environment

Local Clusters (For Development & Testing)

Ideal for learning, development, and testing applications locally before deploying to production.

- **Minikube**: Runs a single-node Kubernetes cluster locally.
- **Kind (Kubernetes in Docker):** Uses Docker containers to simulate a multi-node Kubernetes cluster.
- **Docker Desktop**: Comes with a built-in Kubernetes cluster for quick local testing

Production-Ready Self-Managed Clusters

For organizations that require full control over their Kubernetes infrastructure.

- **kubeadm:** Simplifies manual Kubernetes cluster setup.
- **kops:** Automates Kubernetes deployment on AWS.
- **Rancher:** Provides a user-friendly UI to manage Kubernetes clusters.

Managed Kubernetes Services (Cloud Providers)

Best for scalable, secure, and automated production workloads without operational overhead.

- **Amazon EKS (Elastic Kubernetes Service) –** AWS-managed Kubernetes**.**
- **Google GKE (Google Kubernetes Engine) –** Google Cloud's Kubernetes service.
- **Azure AKS (Azure Kubernetes Service) –** Microsoft Azure's Kubernetes service.

Cost Considerations:

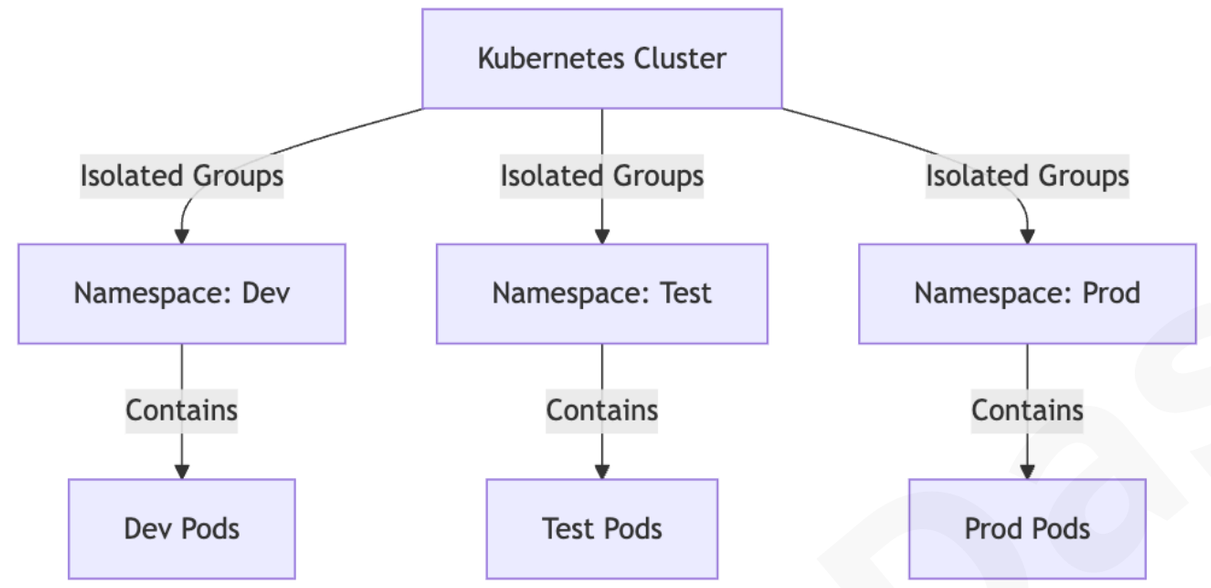
- **EKS**: Control plane cost + EC2/node groups.
- **GKE**: Free control plane, charges for nodes + autopilot premium.
- **AKS**: Free control plane, pay only for nodes/services.

Once again: When to Use What?

- **Local**: Learning, prototyping (minikube/kind).
- **Production Self-Managed:** Full control, compliance (kubeadm/kops).
- **Managed**: Speed, scalability, and reduced ops (EKS/GKE/AKS).

Core Concepts in Action

Kubernetes provides several core concepts to efficiently manage containerized applications. Let's break down each concepts.



Namespaces

A Namespace is a virtual cluster within a Kubernetes cluster.

Key Use Cases

1. **Resource Isolation**: Has its own set of resources (Pods, Services, ConfigMaps, etc.), Prevent teams/apps from interfering (e.g., dev vs prod).
2. **Access Control**: RBAC rules scoped to namespaces.
3. **Resource Quotas**: Limit CPU/memory per namespace.

Commands:

```
# List namespaces
kubectl get namespaces
# Create a namespace
kubectl create namespace staging
# Set default namespace for all commands
kubectl config set-context --current --namespace=staging
```

Common Default Namespaces

Namespace	Purpose
`default`	The default Namespace if none is specified.
`kube-system`	System-related Pods (DNS, controllers, etc.).
`kube-public`	Publicly readable resources (rarely used).
`kube-node-lease`	Stores heartbeats from nodes.
`custom-namespace`	User-defined Namespaces for applications.

@Sandip Das

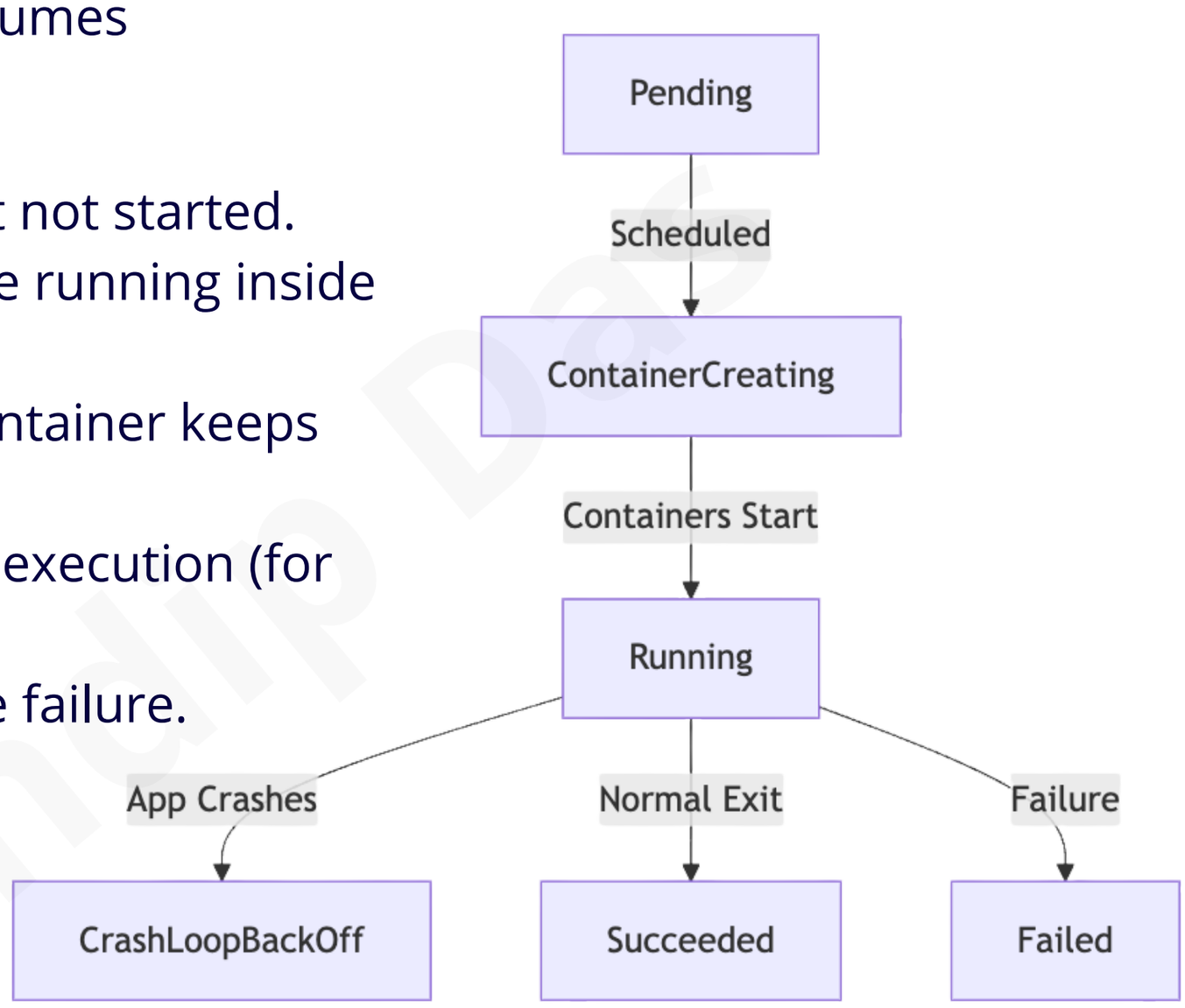
Core Concepts in Action

Pods: The Smallest Deployable Unit

A Pod encapsulates one or more containers, sharing the same network namespace and storage volumes

Pod Lifecycle:

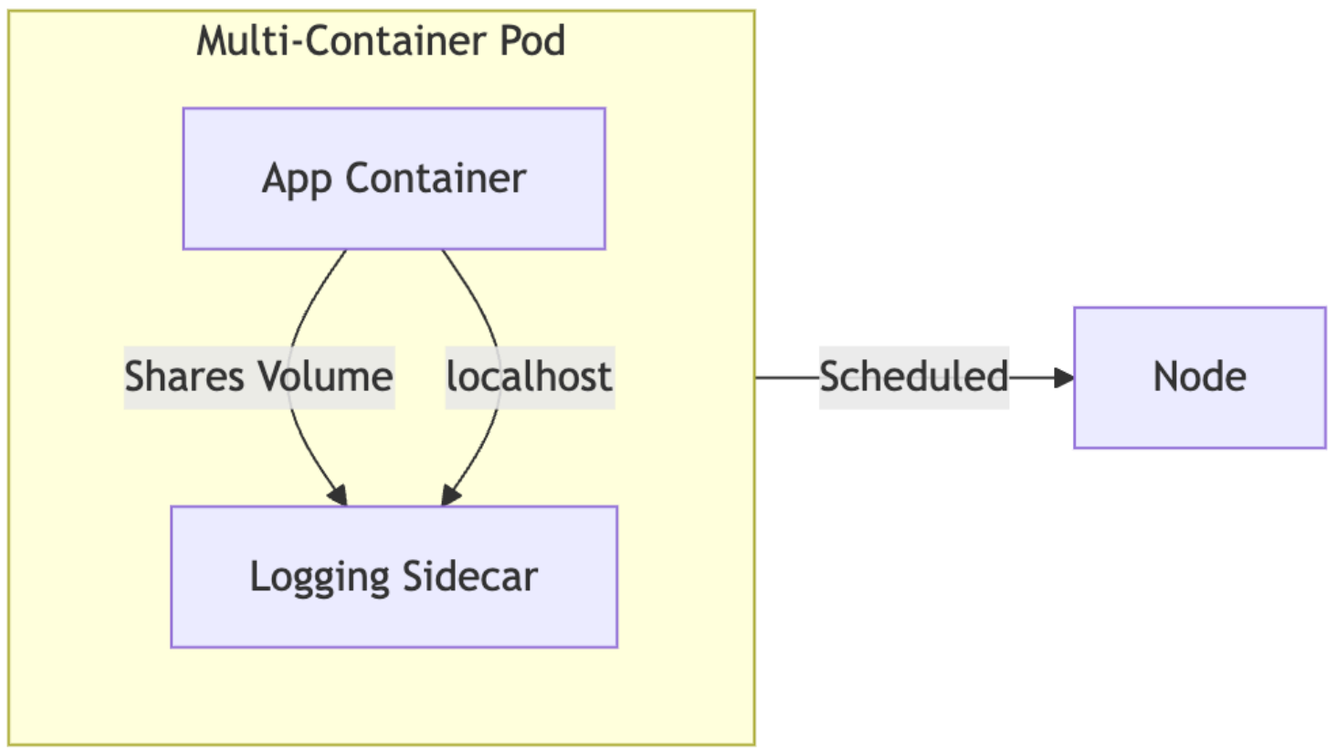
- **Pending**: Scheduled but not started.
- **Running**: Containers are running inside the Pod.
- **CrashLoopBackOff**: Container keeps failing and restarting.
- **Succeeded**: Completed execution (for jobs).
- **Failed**: Non-recoverable failure.



Multi-Container Pods & Sidecar Pattern

A Pod can have multiple containers that communicate using localhost.
The Sidecar pattern extends the primary container's functionality (e.g., logging, proxying). e.g.:

```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-pod
spec:
containers:
- name: app-container
  image: my-app:latest
- name: logging-sidecar
  image: fluentd
```



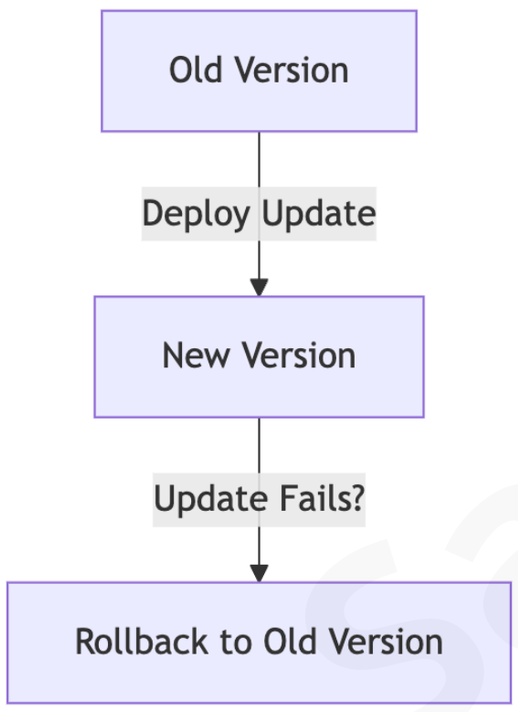
- The app container runs the main application.
- The sidecar container handles logging or metrics collection.

Core Concepts in Action

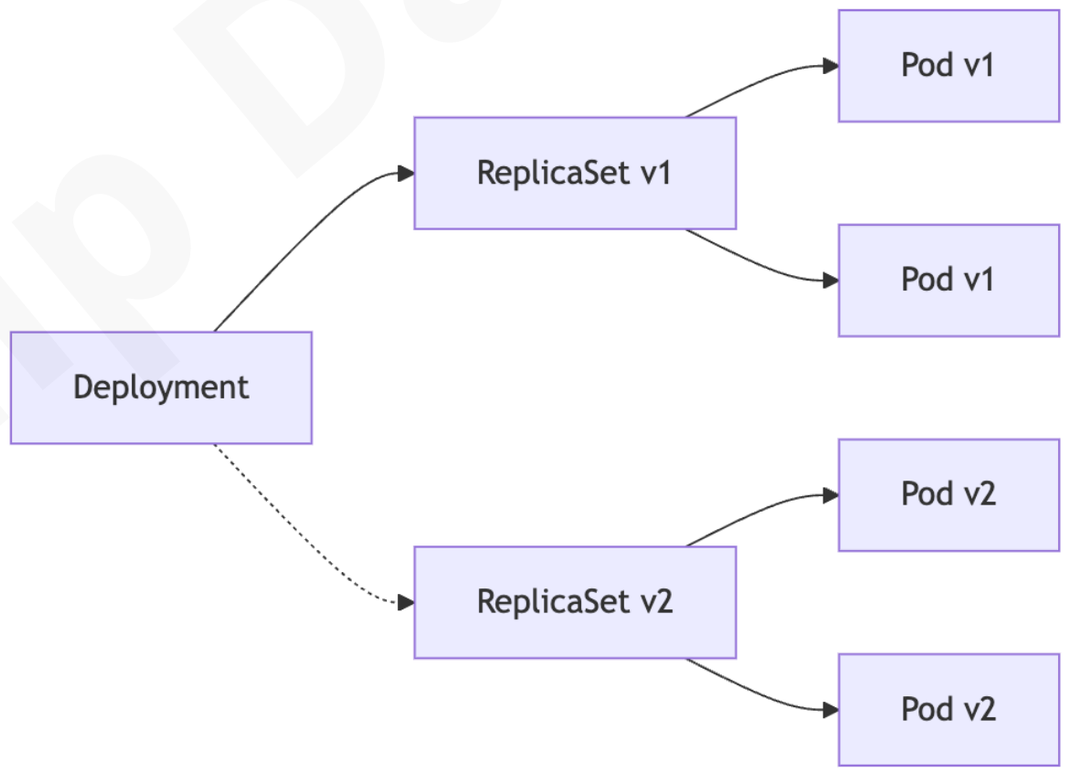
Deployments: Managing Stateless Workloads

Deployments manage **rolling updates, rollbacks**, and **replica scaling.**

Rolling Updates: Gradual replacement of old pods with new ones.
Rollbacks: Revert to a previous ReplicaSet version**.**
Replica Management: A Deployment ensures the desired number of replicas is running.



Rolling Updates & Rollbacks



Replica Management

Example: Via :Imperative commands

```
kubectl create deployment nginx --image=nginx:1.20
kubectl set image deployment/nginx nginx=nginx:1.21  # Rolling update
kubectl rollout undo deployment/nginx          # Rollback
```

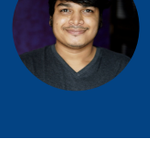
Example: Via :Declarative

We can do via Kubernetes manifest files , run:

```
kubectl apply -f nginx_deployment.yaml
```

nginx_deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```



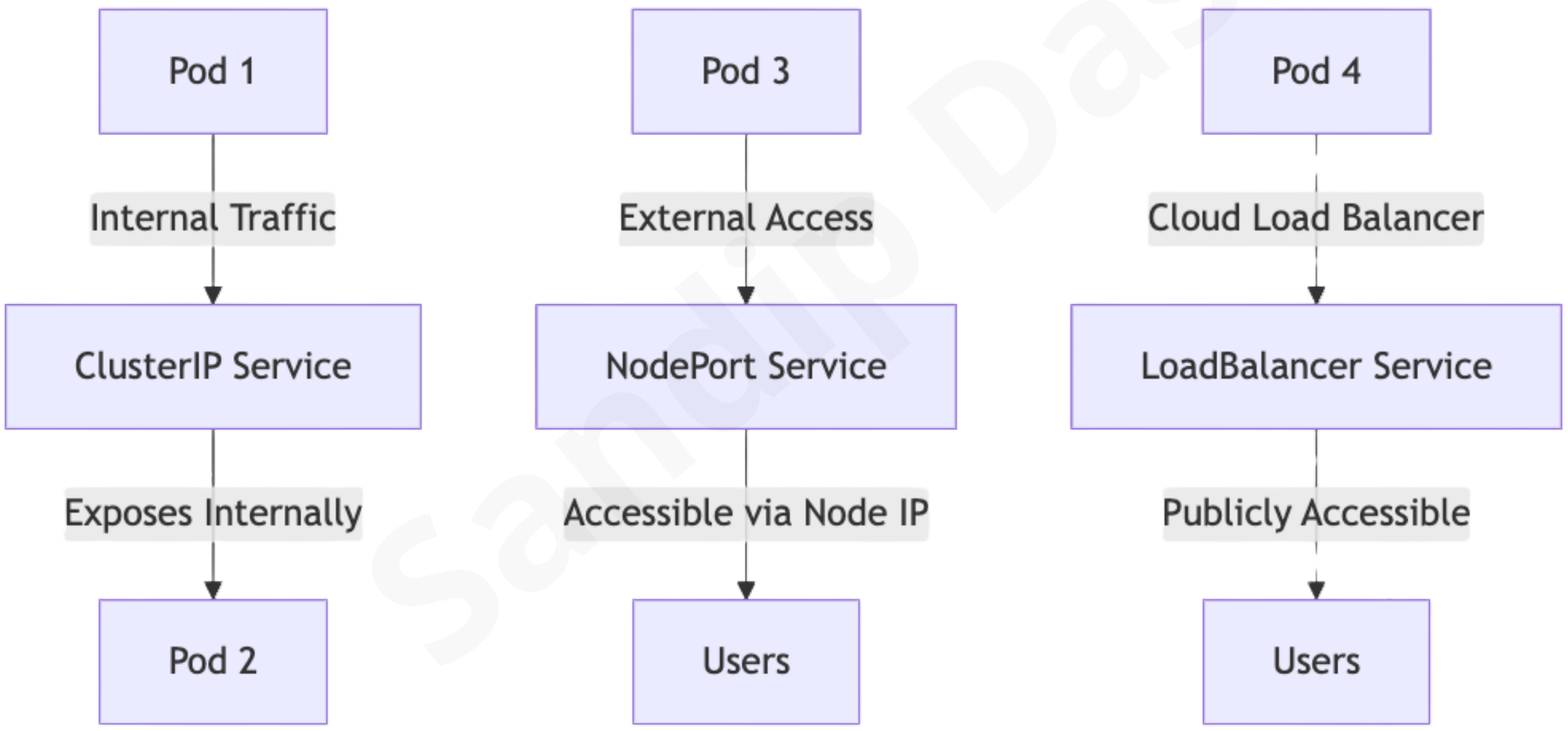
@Sandip Das

Core Concepts in Action

Services

A Service enables communication between Pods and external users.

- **ClusterIP:** Internal IP for intra-cluster communication.
- **NodePort:** Exposes a specific port on each node.
- **LoadBalancer:** Uses a cloud provider's load balancer.




```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30080
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Core Concepts in Action

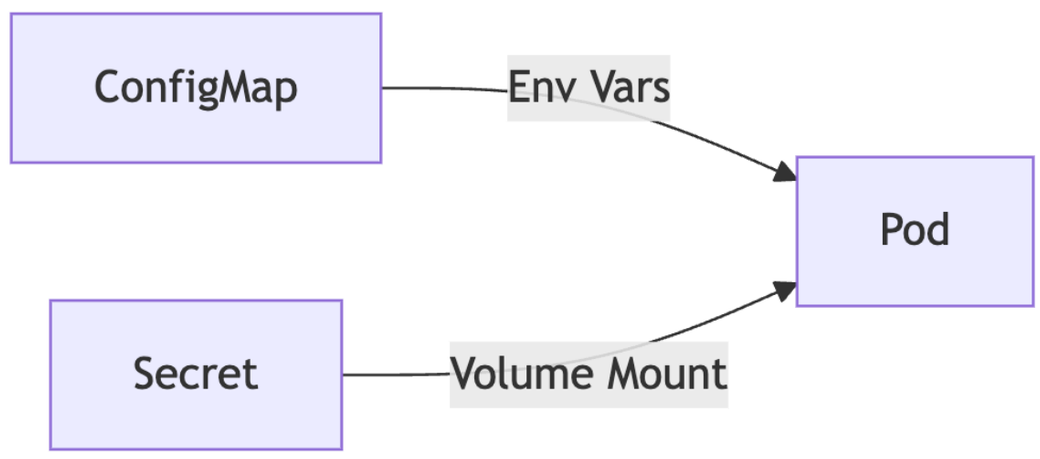
ConfigMaps & Secrets

ConfigMaps store non-sensitive configuration data, while **Secrets** handle sensitive data.

ConfigMap: **Non-sensitive configs** (e.g., environment variables).
Secret: **Sensitive data** (base64-encoded).

Example: ConfigMap (Environment Variables)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  DATABASE_URL: "mysql://db:3306"
```



Mount the ConfigMap as an environment variable:

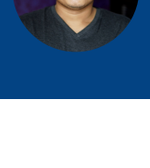
```
env:
- name: DATABASE_URL
  valueFrom:
    configMapKeyRef:
      name: app-config
      key: DATABASE_URL
```

Example: Secret (Base64 Encoded Data)

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
data:
  password: cGFzc3dvcmQ=  # Base64 encoded "password"
```

Mount the Secret as an environment variable:

```
env:
- name: MYSQL_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mysql-secret
      key: password
```



Advanced Workloads and Patterns

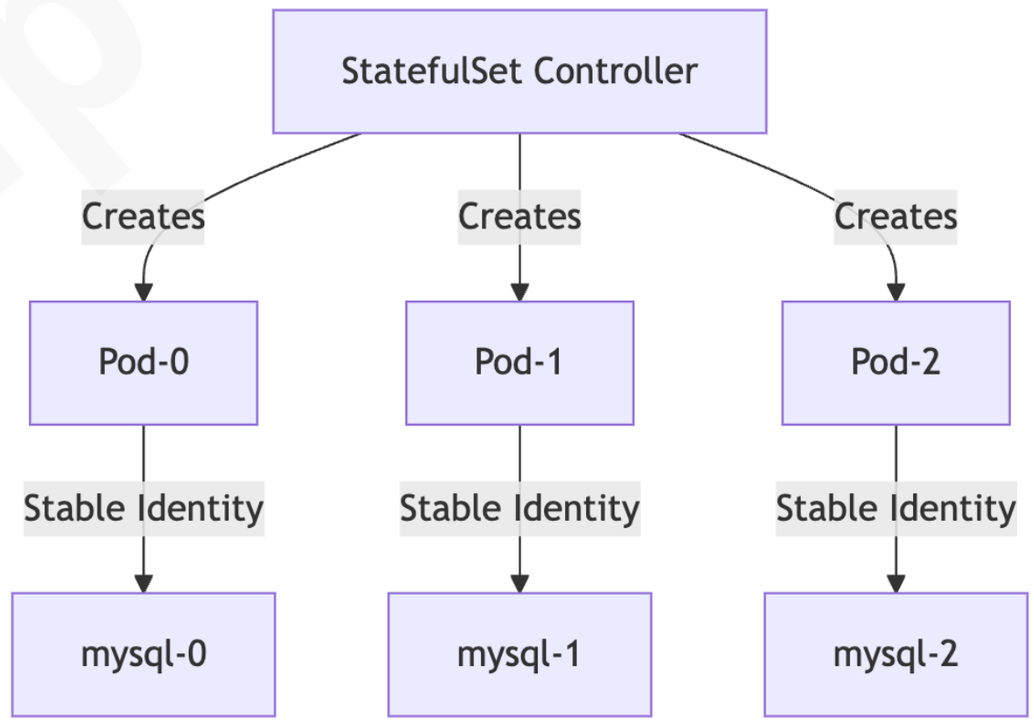
Kubernetes provides advanced workload controllers and patterns for managing stateful applications, node-wide services, batch processing, and custom automation. Let's dive into:

- **StatefulSets** (for databases)
- **DaemonSets** (for node monitoring)
- **Jobs & CronJobs** (for batch processing)
- **Operators & CRDs** (for automation)

StatefulSets

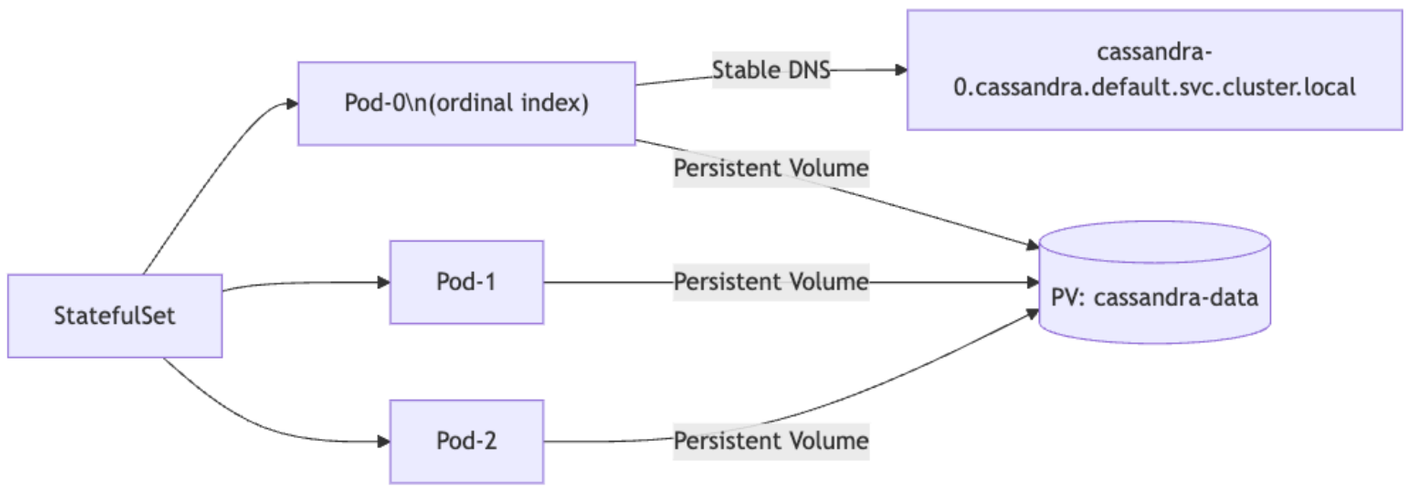
Unlike Deployments, **StatefulSets** maintain sticky identities for Pods, making them ideal for databases like Cassandra, MySQL, MongoDB.

Key Features:
✅ Unique, stable network identifiers (pod-0, pod-1, pod-2...).
✅ Persistent storage maintained across pod restarts.
✅ Ordered pod creation, scaling, and deletion.



Example: StatefulSet for Cassandra

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
spec:
  serviceName: "cassandra"
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
spec:
  containers:
  - name: cassandra
    image: cassandra:latest
    ports:
    - containerPort: 9042
```



Click here for more advance example

@Sandip Das

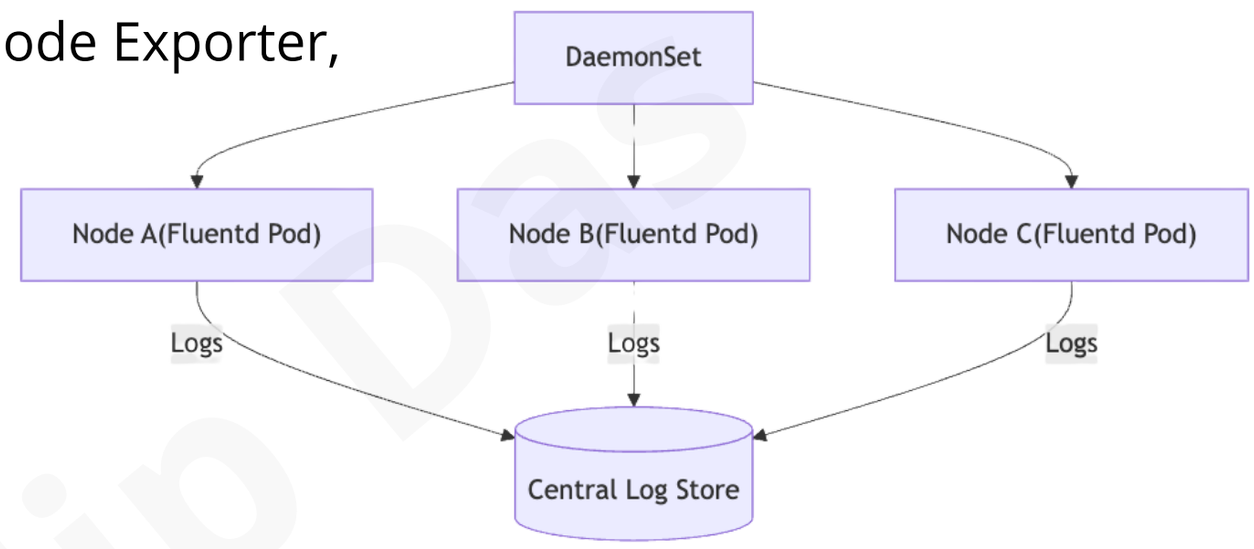
Advanced Workloads and Patterns

DaemonSets

A DaemonSet ensures that one copy of a Pod runs on every node in the cluster. Useful for node monitoring, logging, and security agents.

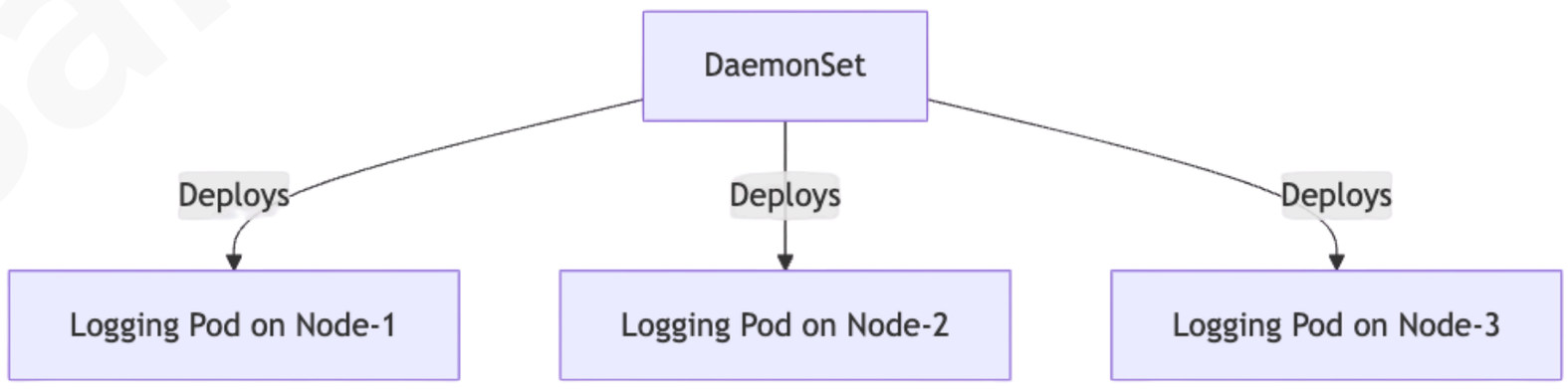
Common Use Cases:

✅ Log collectors (Fluentd, Filebeat).
✅ Node monitoring (Prometheus Node Exporter, Datadog).
✅ Security agents (Falco, Sysdig).



Example: Fluentd DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
      - name: fluentd
        image: fluent/fluentd-kubernetes-daemonset
        volumeMounts:
        - name: varlog
          mountPath: /var/log
volumes:
- name: varlog
  hostPath:
    path: /var/log
```



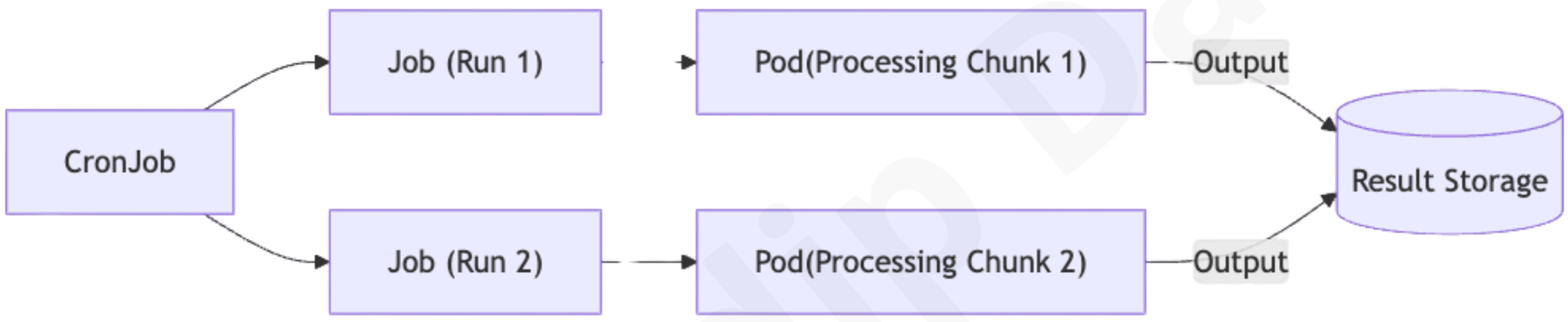
Advanced Workloads and Patterns

Jobs & CronJobs

A Job runs a task until completion, while a CronJob schedules recurring tasks. (e.g., data processing, backups).

Key Features:

✅ Runs to completion (no restart on failure).
✅ Ideal for ETL jobs, report generation, backups.
✅ Supports parallelism and retry policies.



Example (CronJob for daily cleanup):

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: daily-cleanup
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: cleanup
            image: alpine
            command: ["/bin/sh", "-c", "rm -rf
/tmp/old-files"]
          restartPolicy: OnFailure
```

Example: Job (One-Time Execution):

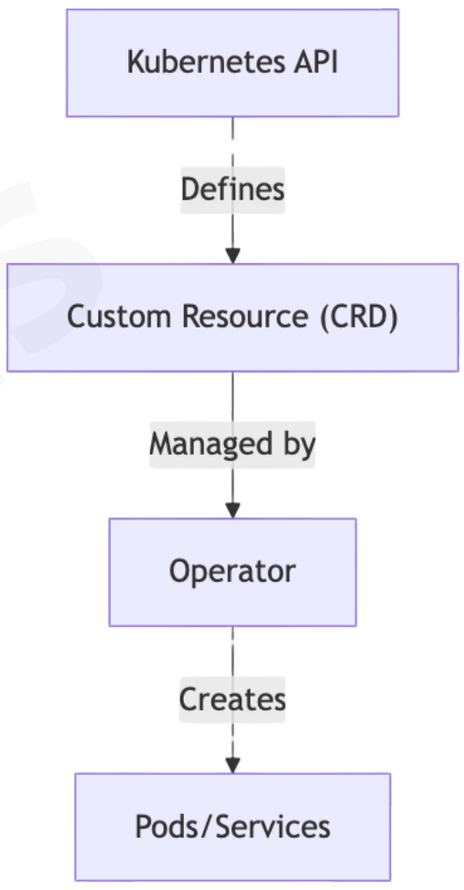
```
apiVersion: batch/v1
kind: Job
metadata:
  name: data-processing-job
spec:
  completions: 3
  template:
    spec:
      containers:
      - name: data-processor
        image: python:3.8
        command: ["python", "-c",
"print('Processing data...')"]
      restartPolicy: Never
```


Advanced Workloads and Patterns

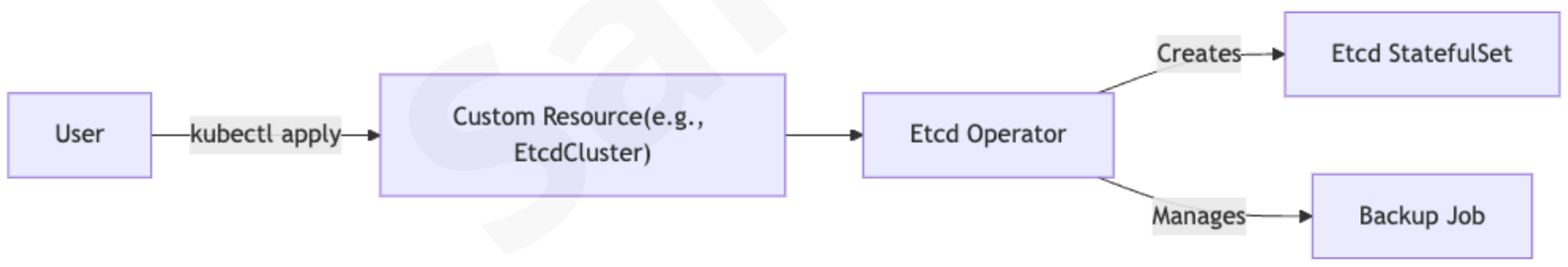
Operators & Custom Resource Definitions (CRDs)

Operators automate complex application lifecycle management by defining custom Kubernetes controllers (e.g., databases, middleware).

Key Features:
✅ Automates application deployment, scaling, backups, failover.
✅ Uses Custom Resource Definitions (CRDs) to extend Kubernetes API.
✅ Ideal for managing databases, ML workflows, monitoring tools.



Example (Etcd Operator):



Step 1: Define a CRD:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: etcdclusters.etcd.database.coreos.com
spec:
  group: etcd.database.coreos.com
  scope: Namespaced
  names:
    plural: etcdclusters
    singular: etcdcluster
    kind: EtcdCluster
  versions:
  - name: v1beta2
    served: true
    storage: true
```

Step 2: Deploy the Operator::

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: etcd-operator
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: etcd-operator
        image: quay.io/coreos/etcd-operator:v0.9.4
```

Step 3: Create a Custom Resource:

```
apiVersion: etcd.database.coreos.com/v1beta2
kind: EtcdCluster
metadata:
  name: example-etcd-cluster
spec:
  size: 3
  version: 3.4.9
```


Networking Deep Dive

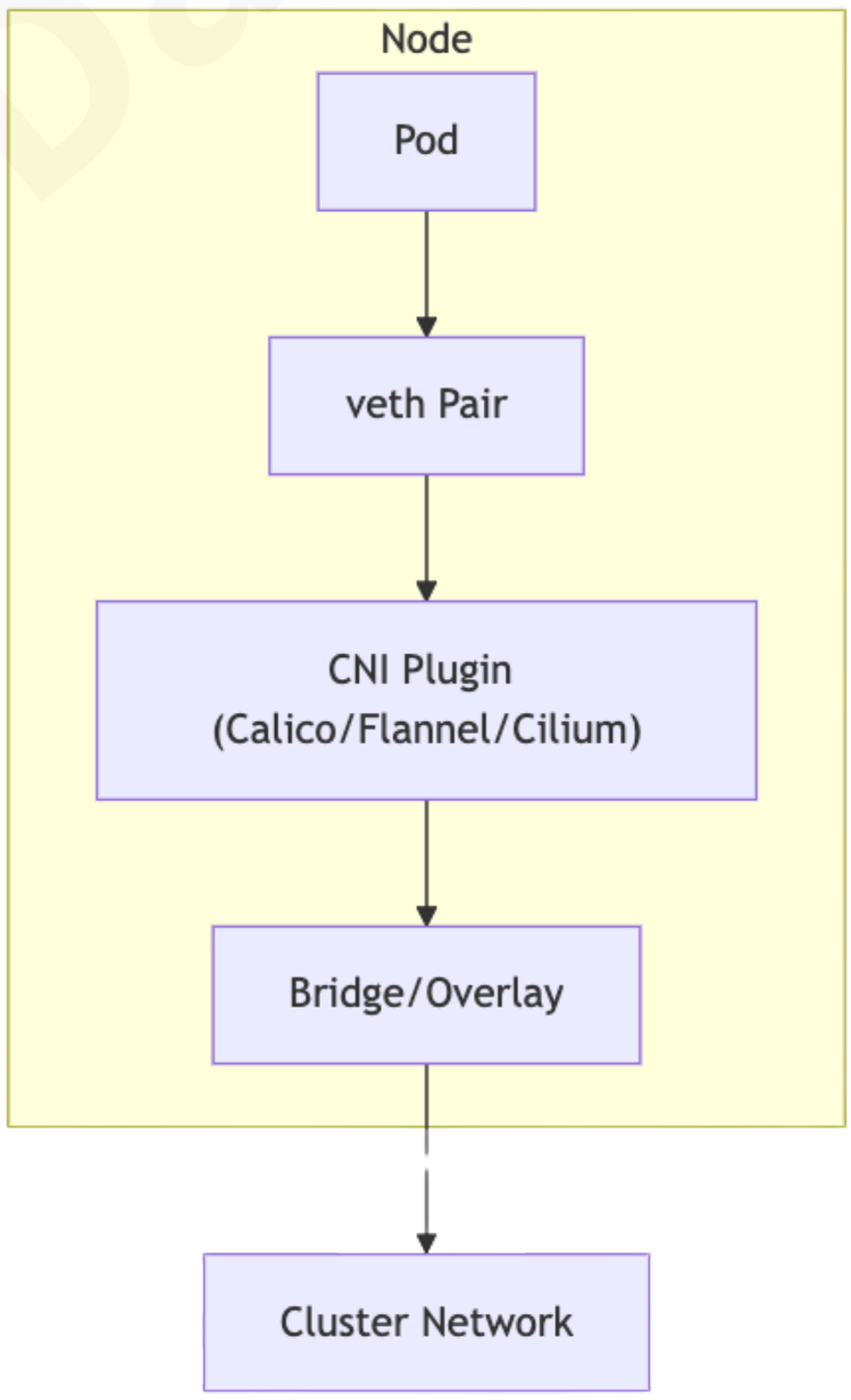
Kubernetes networking is a critical aspect of managing pod communication, service discovery, ingress traffic, and security. Let's dive into:

- CNI Plugins (Calico, Flannel, Cilium)
- Ingress Controllers (NGINX, Traefik)
- DNS & Service Discovery (CoreDNS, troubleshooting)
- Hands-on Lab (Expose a microservice via Ingress)

CNI Plugins: Enabling Pod-to-Pod Networking

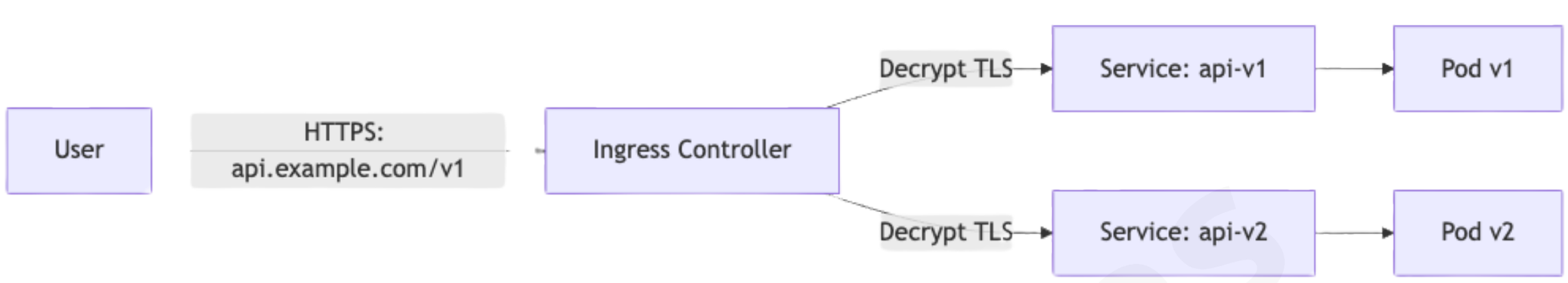
Kubernetes relies on Container Network Interface (CNI) plugins to manage networking between Pods.

Plugin	Key Features	Use Case
Calico	BGP routing, network policies, IPv6/IPv4 support.	Enterprise security/compliance.
Flannel	Simple overlay network (VXLAN or host-gw).	Basic networking for small clusters.
Cilium	eBPF-based networking, L7 policies, service mesh.	High-performance observability.



Networking Deep Dive

Ingress Controllers: Managing External Traffic



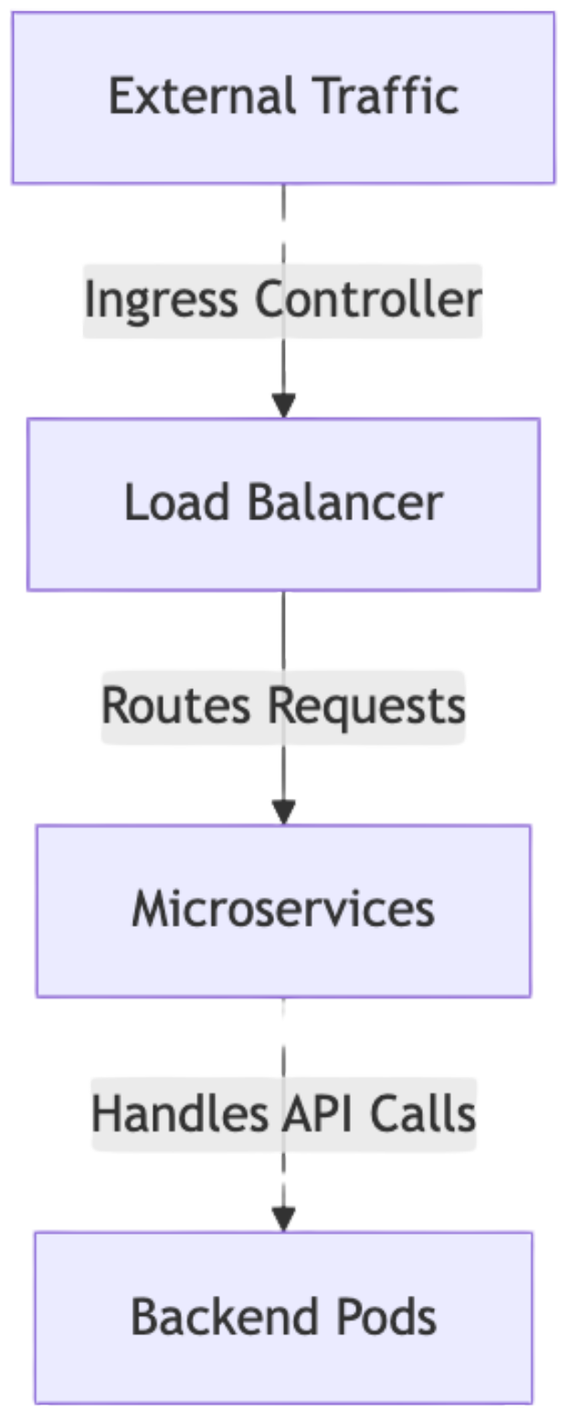
Ingress routes HTTP/HTTPS traffic to services inside the cluster.

Popular Ingress Controllers

- **NGINX Ingress**: Most widely used, supports path-based routing and TLS.
- **Traefik**: Lightweight, dynamic configuration, suitable for microservices.

Path-Based Routing with NGINX Ingress example:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 80
      - path: /dashboard
        pathType: Prefix
        backend:
          service:
            name: dashboard-service
            port:
              number: 80
```



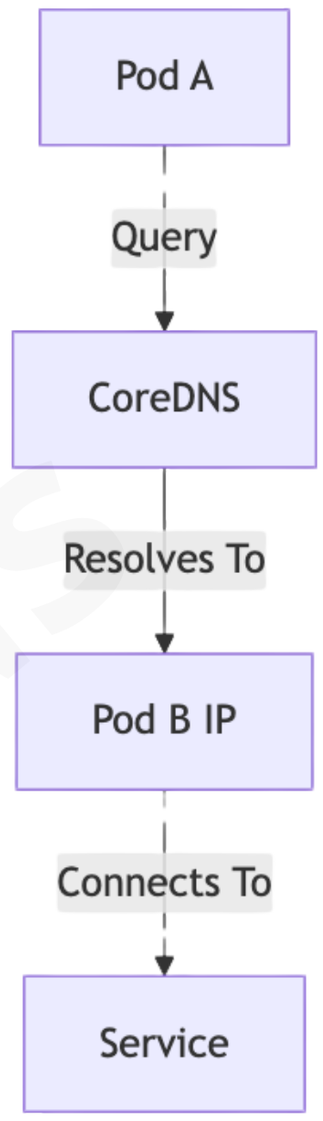
Networking Deep Dive

DNS & Service Discovery

Kubernetes uses CoreDNS for internal service discovery and DNS resolution.

How Service Discovery Works

- Pods talk to each other via **service-name.namespace.svc.cluster.local**.
- **CoreDNS** automatically maps Service names to Pod IPs.



<u>Troubleshooting Connectivity</u>

Check if CoreDNS is running:

kubectl get pods -n kube-system -l k8s-app=kube-dns

Test DNS resolution inside a Pod:

kubectl run busybox --image=busybox -it --restart=Never -- nslookup api-service.default.svc.cluster.local

Common Issues:
DNS not resolving? Restart CoreDNS:
kubectl rollout restart deployment coredns -n kube-system

Pod can’t reach a service? Check if the service exists:
kubectl get svc

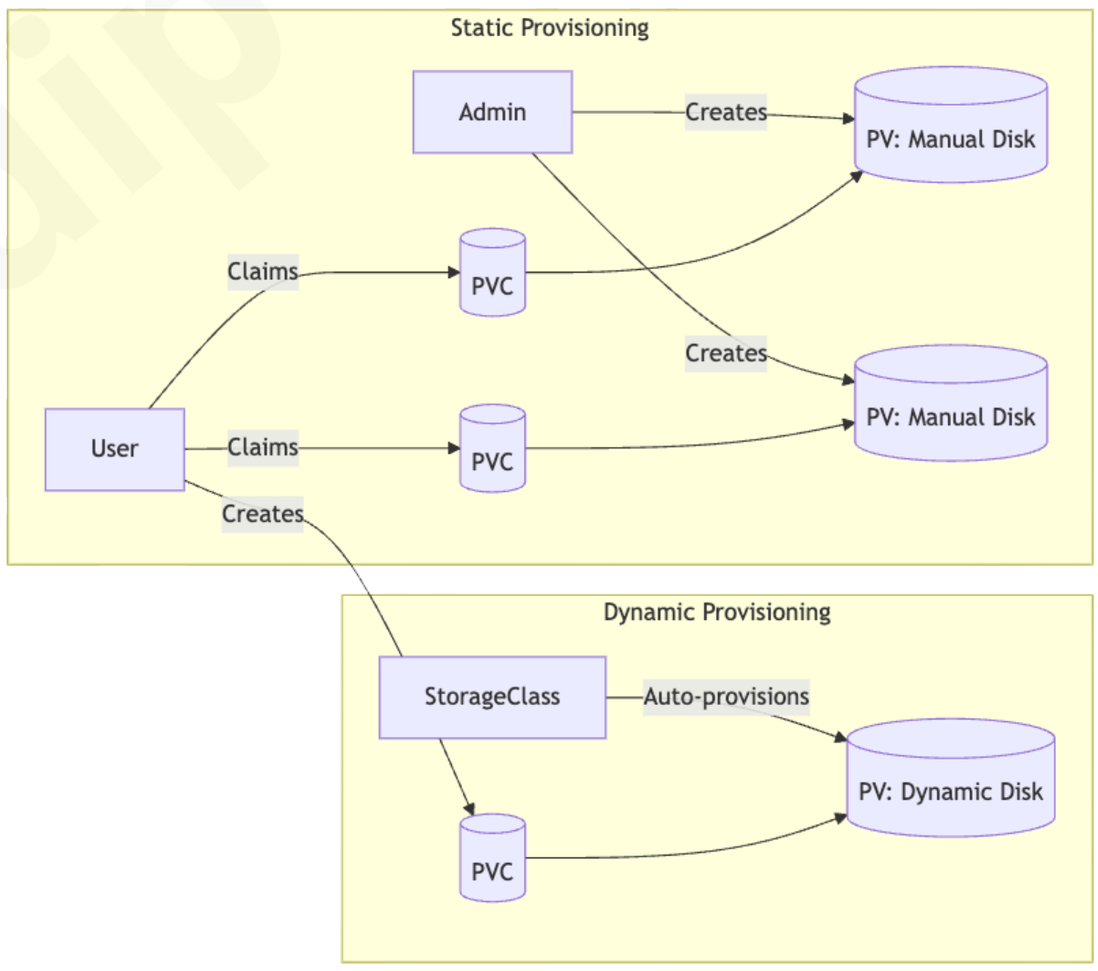
Storage Solutions

Kubernetes storage ensures data persistence for stateful applications.
This deep dive covers:

- PersistentVolumes (PVs): Static vs. Dynamic Provisioning
- StorageClasses (EBS, GCP Persistent Disk)
- Stateful Apps & VolumeClaimTemplates
- Gotcha: Avoiding Data Loss During Pod Rescheduling

PersistentVolumes (PV) & PersistentVolumeClaims (PVC)

In **Kubernetes**, PersistentVolumes (**PVs**) are **storage resources** independent of Pods, while PersistentVolumeClaims (**PVCs**) allow Pods to **request storage**.



<u>Static vs. Dynamic Provisioning</u>

Static: Admin manually creates PVs, and users claim them via PersistentVolumeClaims (PVCs).
Dynamic: Automatically provisions PVs on-demand using StorageClass.

Static Provisioning

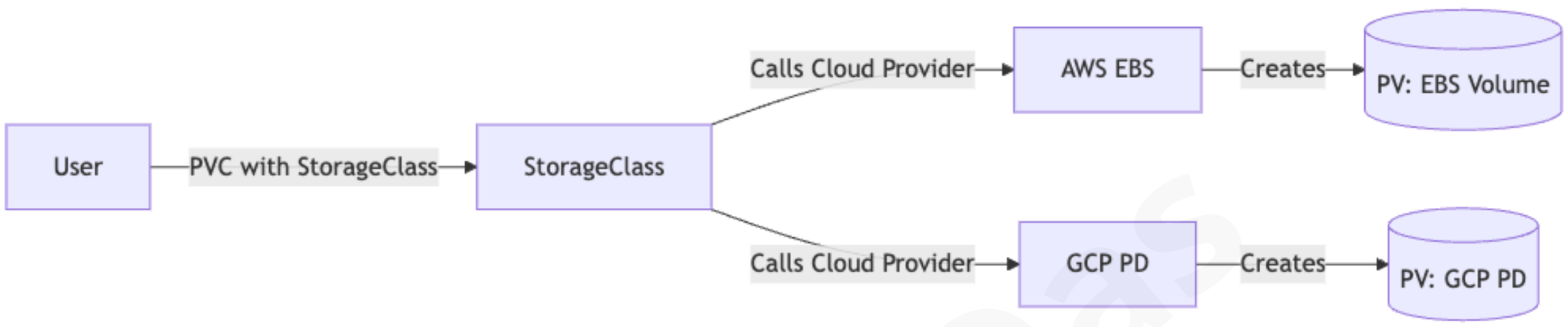
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: static-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Dynamic Provisioning

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: gp2
```


Storage Solutions

StorageClasses: Cloud Storage Solutions



StorageClasses enable dynamic provisioning for cloud storage:

✅ **Key Features:**

- **Provisioner**: Defines the backend storage provider.
- **Reclaim Policy**: (Retain, Delete, Recycle).
- **Access Modes**: (ReadWriteOnce, ReadWriteMany).

Cloud ProviderStorageClass Example

AWS (EBS) : provisioner: **ebs.csi.aws.com**
GCP (PD) : provisioner: **pd.csi.storage.gke.io**
Azure (Disk): provisioner: **disk.csi.azure.com**

AWS EBS StorageClass

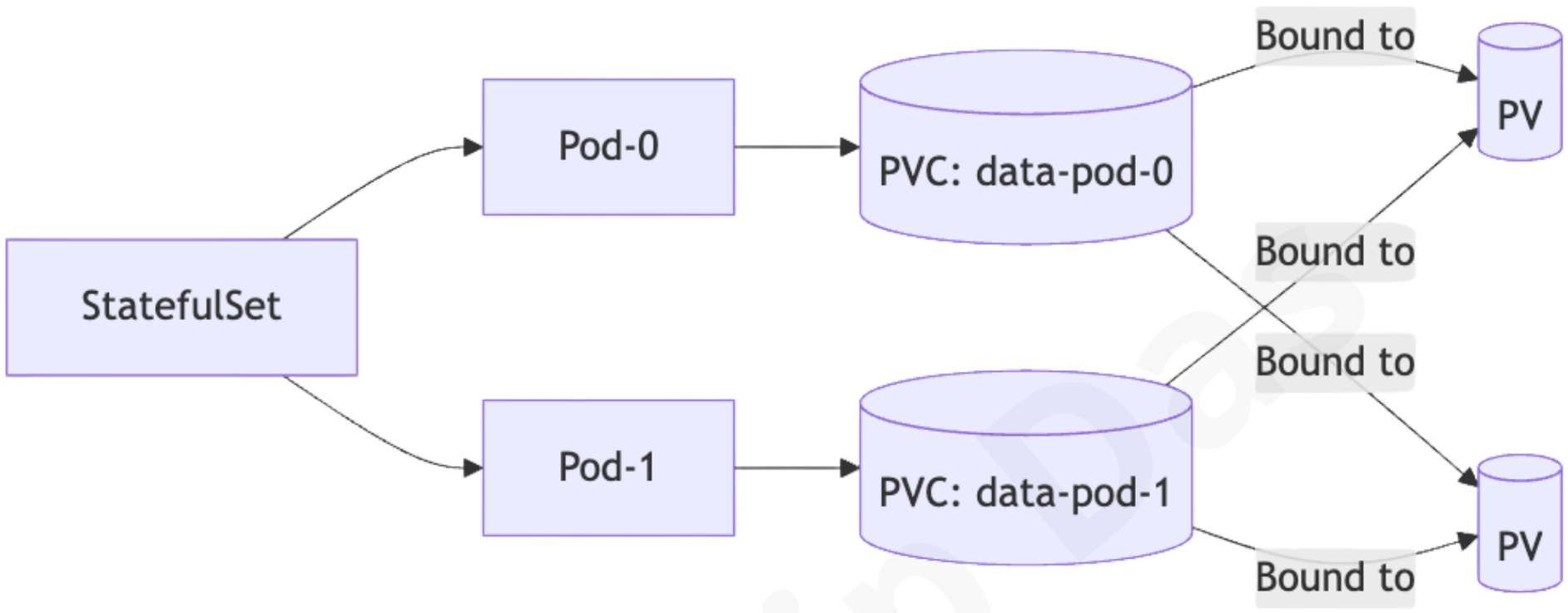
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp2
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
```

GCP Persistent Disk StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  fstype: ext4
```


Storage Solutions

Stateful Apps & VolumeClaimTemplates in StatefulSets



For stateful applications, Kubernetes ensures persistent storage across pod restarts and automatically creates unique PVCs for each pod in a StatefulSet (e.g., databases).

Why VolumeClaimTemplates?

- Each Pod in a StatefulSet gets a unique PVC.
- Prevents data corruption in distributed databases.

Example (MySQL StatefulSet):

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "ebs-sc"  # Dynamic provisioning
      resources:
        requests:
          storage: 20Gi
```

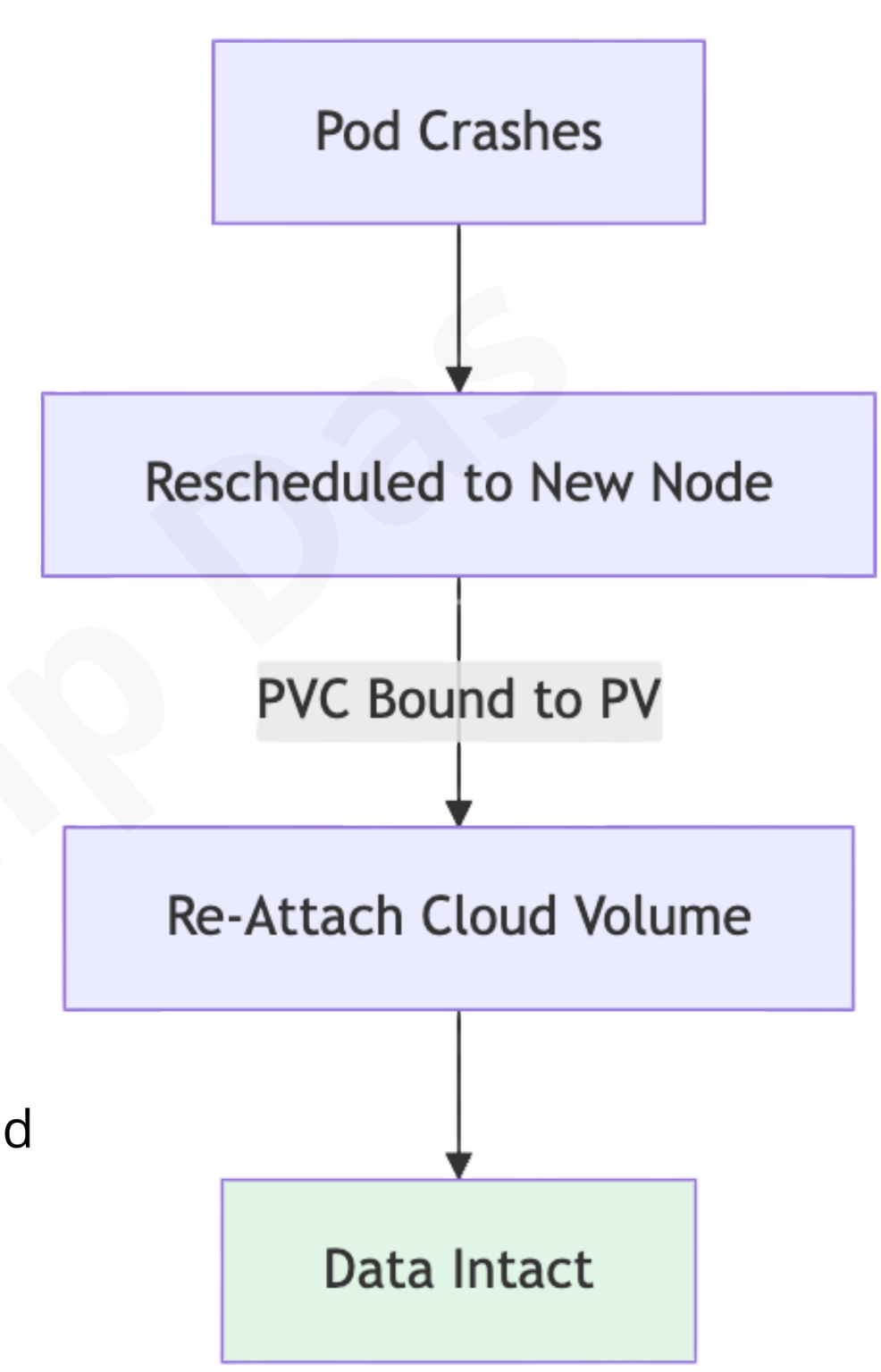
@Sandip Das

Storage Solutions

Gotcha: Avoiding Data Loss During Pod Rescheduling

Common Issues

- Pods get rescheduled to another node, losing attached storage.
- PVCs fail to reattach, causing application failures.



Key Strategies:

1. **Reclaim Policy:** Set PV's persistentVolumeReclaimPolicy to Retain (not Delete).
2. **Backups**: Use tools like Velero to backup PVs.
3. **Volume Binding**: Use WaitForFirstConsumer to delay PV binding until pod scheduling.
4. **ReadWriteMany (RWX)**: For shared volumes (e.g., NFS), but avoid for databases.

Troubleshooting Tips

- **PVC Pending**: Check StorageClass, resource quotas, or cloud provider limits.
- **Access Mode**s: Ensure pods don't request ReadWriteOnce volumes in multi-pod deployments.
- **Backup/Restore**: Use Velero to clone PVs across clusters.

Best Practices:

- **For AWS/GCP**: Use EBS/Persistent Disks with StorageClasses.
- **For NFS/CephFS**: Ensure multi-node attach support.

Example (Retain Policy):

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-retain
spec:
  persistentVolumeReclaimPolicy: Retain  # Critical!
  capacity:
    storage: 10Gi
# ... rest of PV config
```


Troubleshooting and Error Handling

Common Errors & Fixes

ImagePullBackOff

Cause: Kubernetes can't pull the image due to an incorrect image name, missing authentication, or registry issues.

🔍 **Troubleshooting Steps**

```
kubectl get pods
kubectl describe pod <pod-name>
```

✅ **Fixes**

⚠ **Wrong Image Name? Check the exact image name:**

```
 containers:
- name: my-app
  image: nginx:latest  # Ensure the correct version/tag
```

⚠ **Private Registry? Create a Secret and reference it in the Pod:**

```
kubectl create secret docker-registry regcred \
  --docker-server=<REGISTRY> --docker-username=<USER> \
  --docker-password=<PASSWORD>
```

In the kubernetes template

```
imagePullSecrets:
- name: regcred
```

CrashLoopBackOff

Cause: The container is crashing repeatedly due to liveness probe failures, Application crashes, missing dependencies, or resource constraints (**OOMKilled**).

🔍 **Troubleshooting Steps:**

```
kubectl describe pod <pod-name>
kubectl logs <pod-name> -c <container-name>
```

✅ **Fixes:**

Check Application Logs (kubectl logs) for errors.

Increase CPU/Memory Limits if running out of resources.

```
resources:
  requests:
    memory: "256Mi"
    cpu: "500m"
  limits:
    memory: "512Mi"
    cpu: "1"
```

Fix Liveness Probe Issues:

```
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
```

@Sandip Das

Troubleshooting and Error Handling

NetworkPluginNotReady

Cause: Kubernetes can't communicate between Pods due to CNI misconfiguration and / or Misconfigured kubelet or cni.conf**.**

🔍 Troubleshooting Steps:

```
kubectl get nodes -o wide
kubectl describe node <node-name>
kubectl get pods -n kube-system
```

✅ Fixes

Check if the CNI Plugin is Installed:

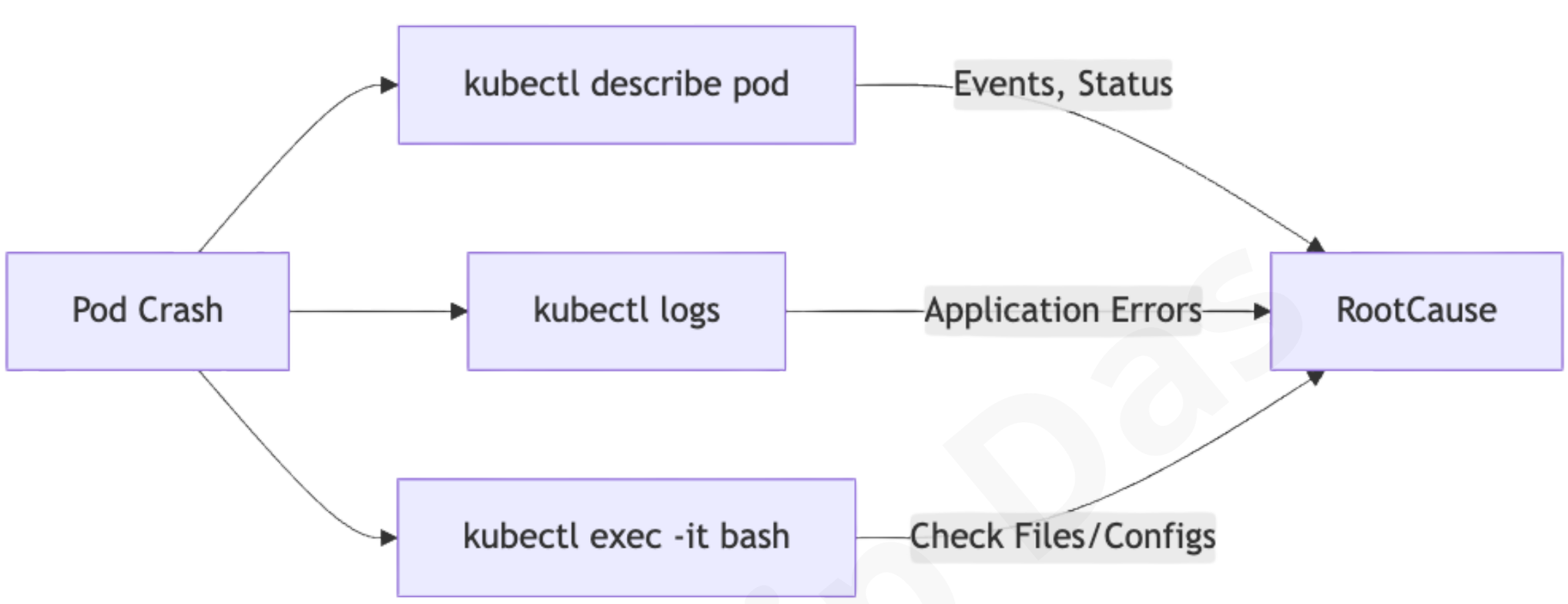
```
ls /etc/cni/net.d/
```

Reinstall Flannel/Calico CNI Plugin:

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```


Troubleshooting and Error Handling

Debugging Tools



🔍 1️⃣ kubectl describe (Detailed Pod/Node Info):

kubectl describe pod <pod-name>
kubectl describe node <node-name>

✅ Use it for:

- Event logs, failed mounts, and scheduling failures.

🔍 2️⃣ kubectl logs (Container Logs):

kubectl logs <pod-name> -c <container-name>

✅ Use it for:

- Checking application errors.
- Seeing crash reasons (e.g., missing dependencies).

🔍 3️⃣ kubectl exec (Debugging Inside a Pod):

kubectl exec -it <pod-name> -- /bin/sh

✅ Use it for:

- Checking files inside a container (ls, cat /var/log/app.log).
- Testing database connections (mysql -h db-service -u root -p).

Troubleshooting and Error Handling

Real-World Scenarios & Fixes

Scenario 1️⃣: Service Endpoints Missing (Label Mismatch)
Issue: A Pod is running, but the Service can't route traffic to it.

🔍 **Check if Endpoints Exist:**
kubectl get endpoints <service-name>

✅ Fix

- Ensure Pod Labels Match the Service Selector:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app  # Must match Pod labels
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  template:
    metadata:
      labels:
        app: my-app  # Ensure it matches Service selector
```

Scenario 2️⃣: PersistentVolume Stuck in "Pending" (Storage Class Issues)
Issue: A Pod requests storage, but the PersistentVolume is stuck in Pending.

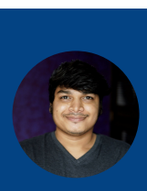
🔍 **Check PVC Status:**
kubectl get pvc
kubectl describe pvc <pvc-name>

✅ Fix
Ensure a StorageClass is Defined:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-storage
provisioner: kubernetes.io/aws-ebs
```

Ensure the PV & PVC Match:

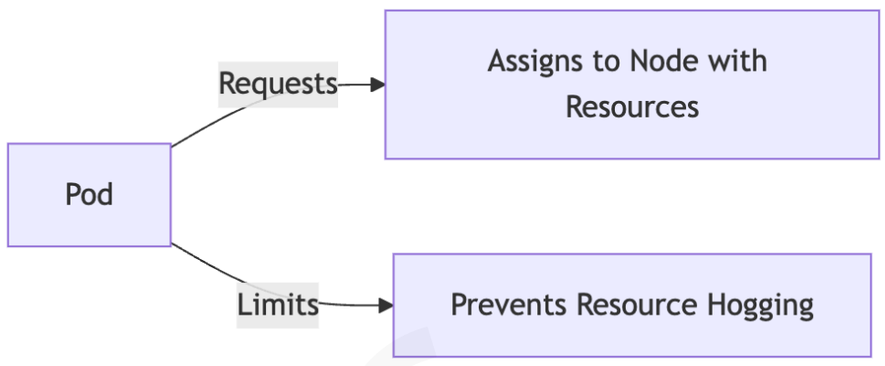
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  storageClassName: fast-storage
```

Optimization & Cost Management

Resource Requests & Limits:

Purpose: Avoid CPU throttling and OOM (Out-Of-Memory) kills by reserving and capping resources.



What Are Resource Requests & Limits?

- **Requests**: Minimum guaranteed CPU/memory for a Pod.
- **Limits**: Maximum CPU/memory a Pod can use.

```
apiVersion: v1
kind: Pod
metadata:
  name: optimized-app
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:  # Guaranteed minimum
        cpu: "100m"
        memory: "256Mi"
      limits:    # Maximum allowed
        cpu: "500m"
        memory: "512Mi"
```

Issue	Cause	Fix
CPU Throttling	CPU requests are too low	Increase CPU requests
OOM Kills	Memory limit exceeded	Set proper memory limits
Node Overcommit	Requests exceed node capacity	Adjust pod placement

Key Tips:

1. Always set requests to match your app's baseline needs.
2. Use limits to prevent runaway processes from starving other pods.
3. Monitor with kubectl top pods or tools like Prometheus.

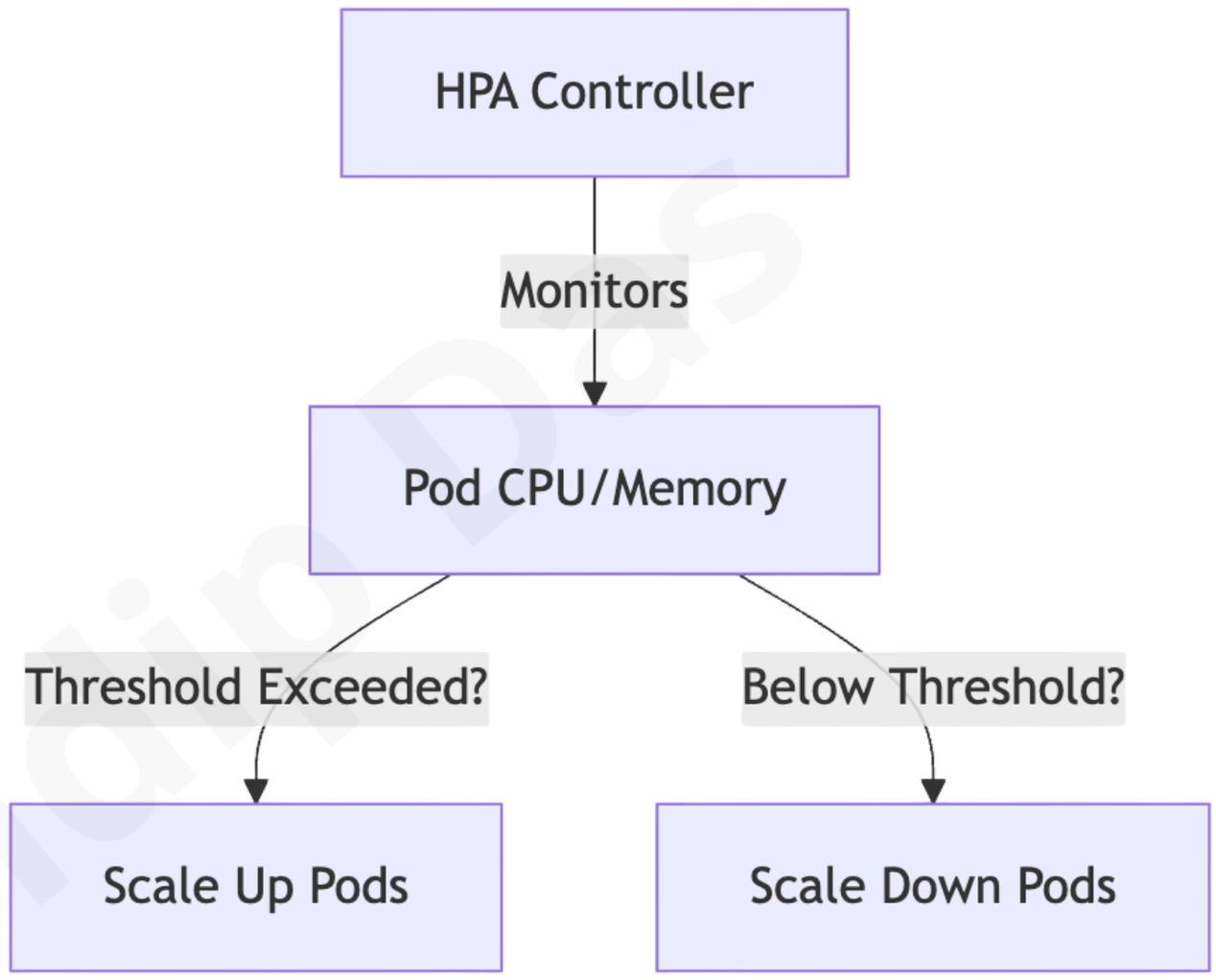
Optimization & Cost Management

Horizontal Pod Autoscaler (HPA):

HPA automatically scales Pods based on CPU, memory, or custom metrics.

Example: Scale Pods Based on CPU

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

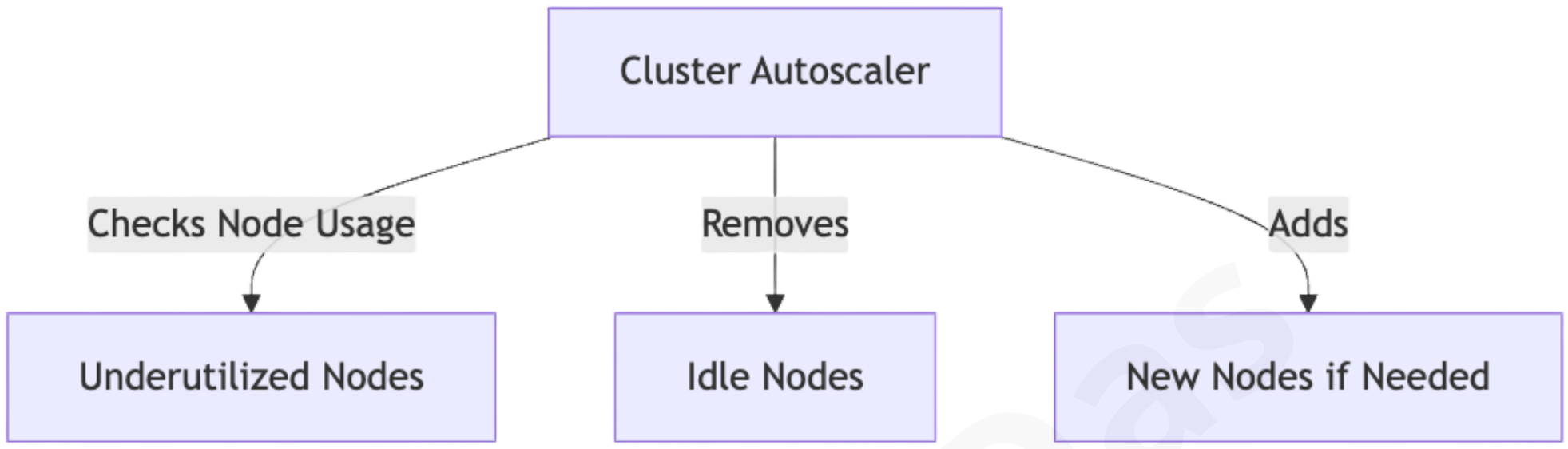


Key Tips:

1. **Use custom metrics (e.g., requests per second) for precise scaling.**
2. **Combine with Cluster Autoscaler to scale nodes when needed.**

Optimization & Cost Management

Cluster Autoscaler (CA)



Purpose: Automatically **resize (add/remove nodes or VMs)** node pools based on pod scheduling demands.

Enable Cluster Autoscaler (EKS/GKE/AKS):
kubectl apply -f cluster-autoscaler.yaml

cluster-autoscaler.yaml

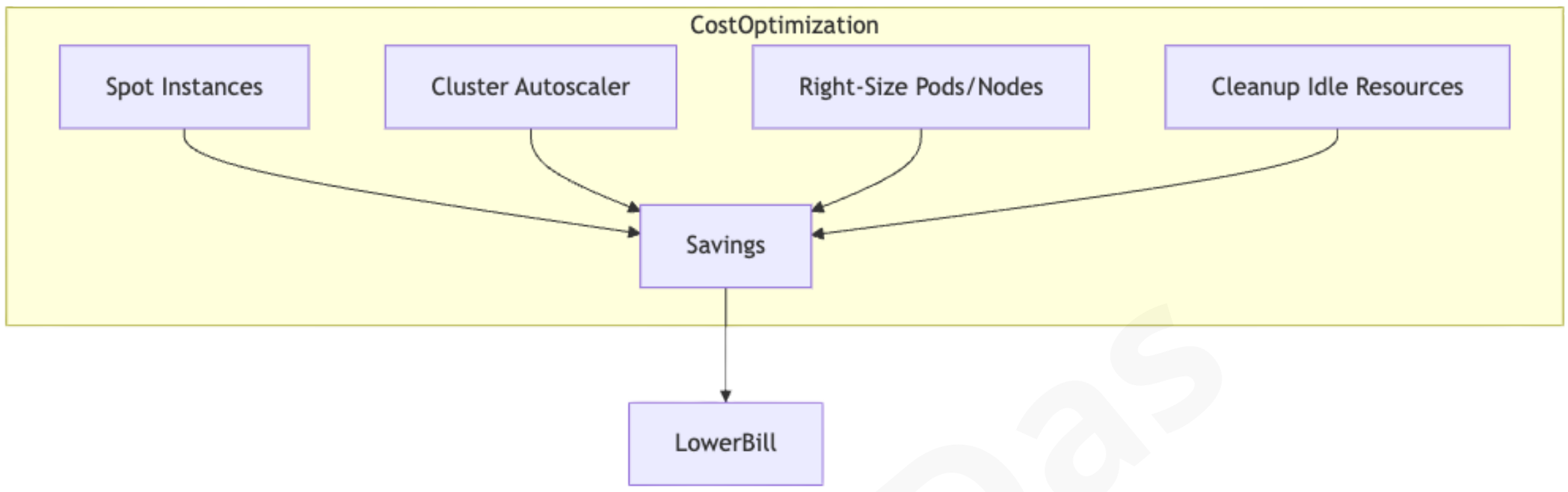
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler
  namespace: kube-system
data:
  scale-down-enabled: "true"
  scale-down-unneeded-time: "10m"
  scale-down-utilization-threshold: "0.5"
```

Key Tips:

1. **Configure node groups with mixed instance types (e.g., spot + on-demand).**
2. **Use pod priority to prioritize critical workloads during scaling.**

Optimization & Cost Management

Cost Saving Tips



Spot Instances (AWS/GCP/Azure)

- Use for stateless, fault-tolerant workloads (e.g., batch jobs).
- Save 60–90% compared to on-demand instances.

Avoid Overprovisioning

- Right-size node pools (avoid oversized VMs).
- Use Vertical Pod Autoscaler (VPA) to adjust requests/limits automatically.

Optimize Pod Density

- Pack more pods per node (balance with resource limits).
- Use Karpenter (AWS) for cost-aware node provisioning.

Cleanup Idle Resources

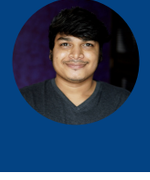
- Delete unused pods, services, and PVs.
- Schedule preemptible/spot nodes for non-critical workloads.

Real-World Example: Cost-Optimized Workflow

1. **Stateless Apps:** Deploy on spot instances with HPA.
2. **Stateful Apps:** Use on-demand + regular snapshots.
3. **Batch Jobs:** Run on spot instances with retries.

Tools for Monitoring Costs

- **Kubecost**: Track cluster spend by namespace/deployment.
- **Cloud Provider Tools:** AWS Cost Explorer, GCP Cost Management.



Kubernetes Gotchas and Anti-Patterns

While Kubernetes is powerful, certain gotchas and anti-patterns can lead to downtime, security vulnerabilities, and cost inefficiencies. Now we will cover:

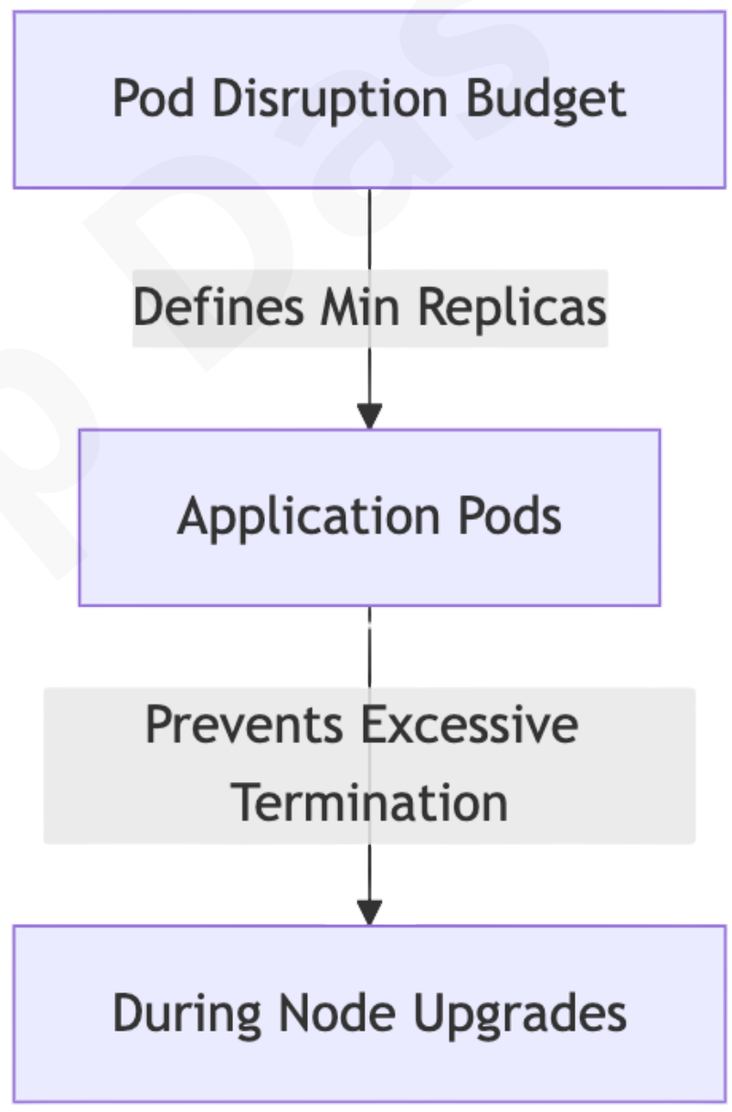
✅ Pod Disruption Budgets (PDBs): Avoiding unnecessary downtime during upgrades
✅ HostPath Volumes: Security risks and safer alternatives
✅ Zombie Namespaces: Orphaned resources silently consuming costs
✅ Version Skew: Control plane and worker node version mismatches

Pod Disruption Budgets (PDBs)

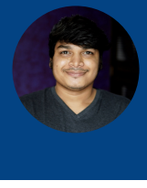
Problem: Upgrades/rollouts can cause unintended downtime if PDBs are not set to protect critical pods.

Example: PDB for High Availability

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-app-pdb
spec:
  minAvailable: 2  # At least 2 pods must be running
  selector:
    matchLabels:
      app: my-app
```



💡 **Pro Tip:**
Always define PDBs for stateful or high-availability workloads (e.g., databases, API gateways).



@Sandip Das

Kubernetes Gotchas and Anti-Patterns

HostPath Volumes: Security Risks

hostPath volumes mount directories from the host node directly into a Pod.

Problem: This bypasses Kubernetes' abstraction and can introduce security risks as it mounts expose host filesystems, risking privilege escalation or data corruption.

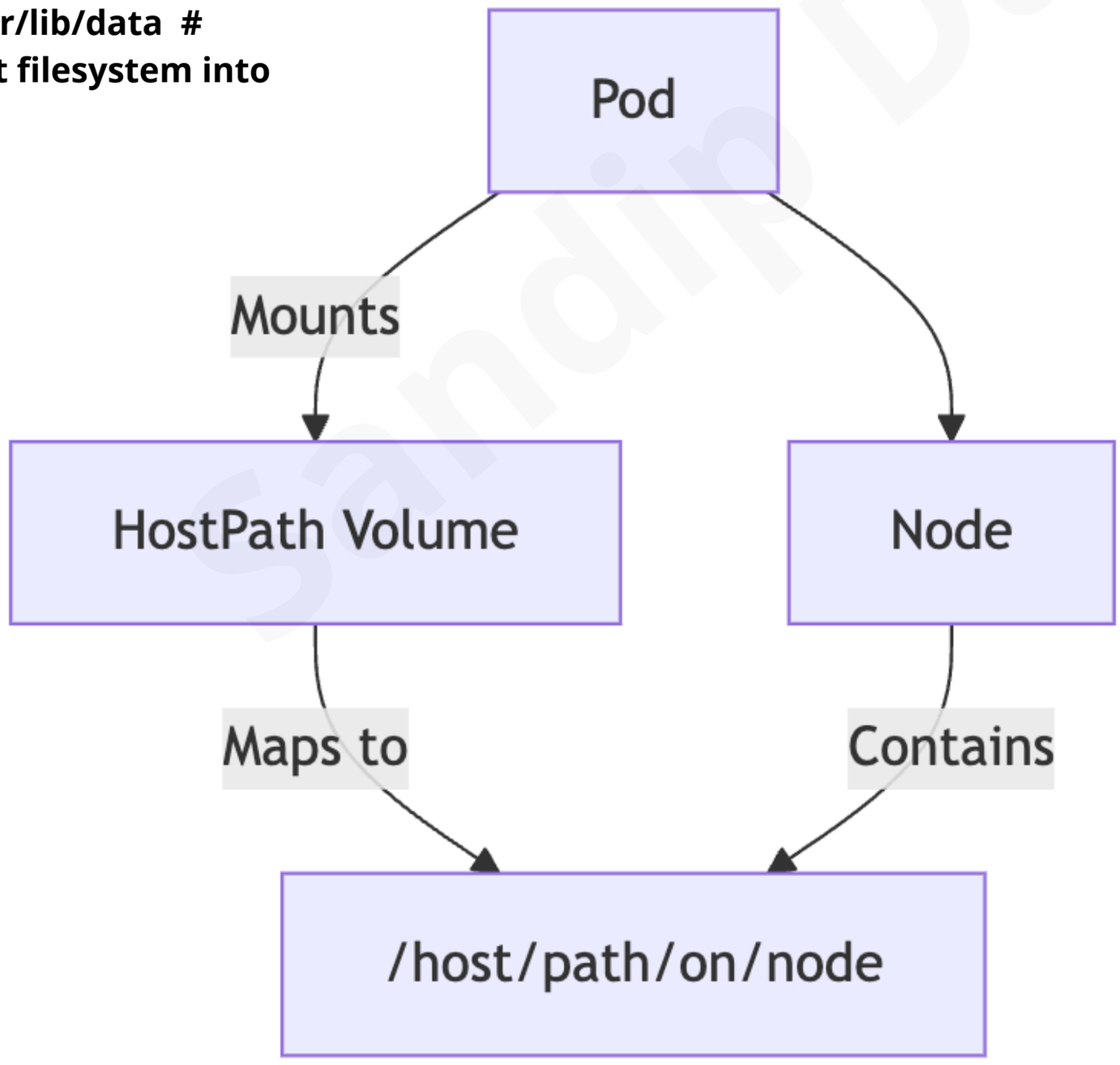
Anti-Pattern: (BAD)

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-pod
spec:
  containers:
  - name: my-container
    image: nginx
    volumeMounts:
    - mountPath: /data
      name: host-data
  volumes:
  - name: host-data
    hostPath:
      path: /var/lib/data  # Mounts host filesystem into container!
```

Solution: (Good)

- Use PersistentVolumes (PV) or CSI drivers for cloud storage.
- If unavoidable, restrict hostPath to read-only and specific directories:

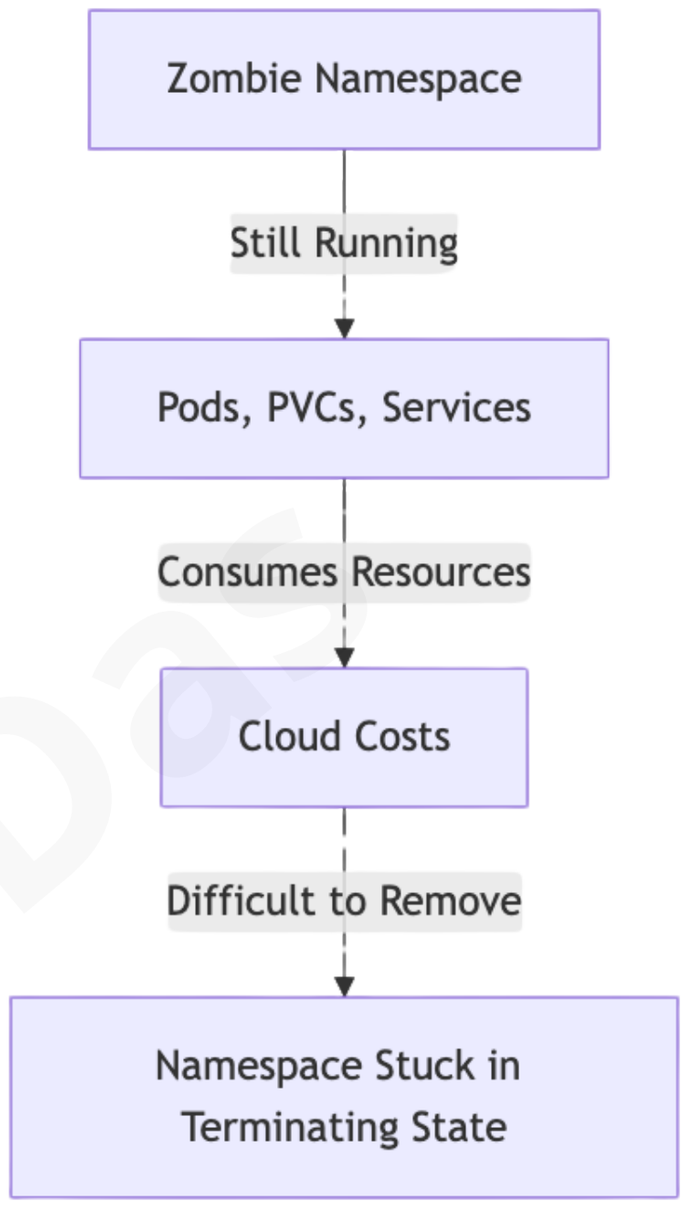
```
volumes:
- name: logs
  hostPath:
  path: /var/logs
  type: DirectoryOrCreate
  readOnly: true
```



Kubernetes Gotchas and Anti-Patterns

Zombie Namespaces

A zombie namespace is a stuck or orphaned namespace that still contains resources but isn't actively used.



Problem: Deleting a namespace doesn't always clean up resources (e.g., cloud load balancers, PVs), leading to lingering costs.

How to Identify:

```
# List resources in "Terminating" namespaces
kubectl get ns --field-selector status.phase=Terminating

# Check for orphaned cloud resources (e.g., AWS EBS volumes, GCP disks).
```

✅ **Fix:**
Force-delete the namespace:

```
kubectl get namespace <name> -o json > tmp.json
# Remove `"kubernetes"` from finalizers array in tmp.json
kubectl replace --raw "/api/v1/namespaces/<name>/finalize" -f tmp.json
```

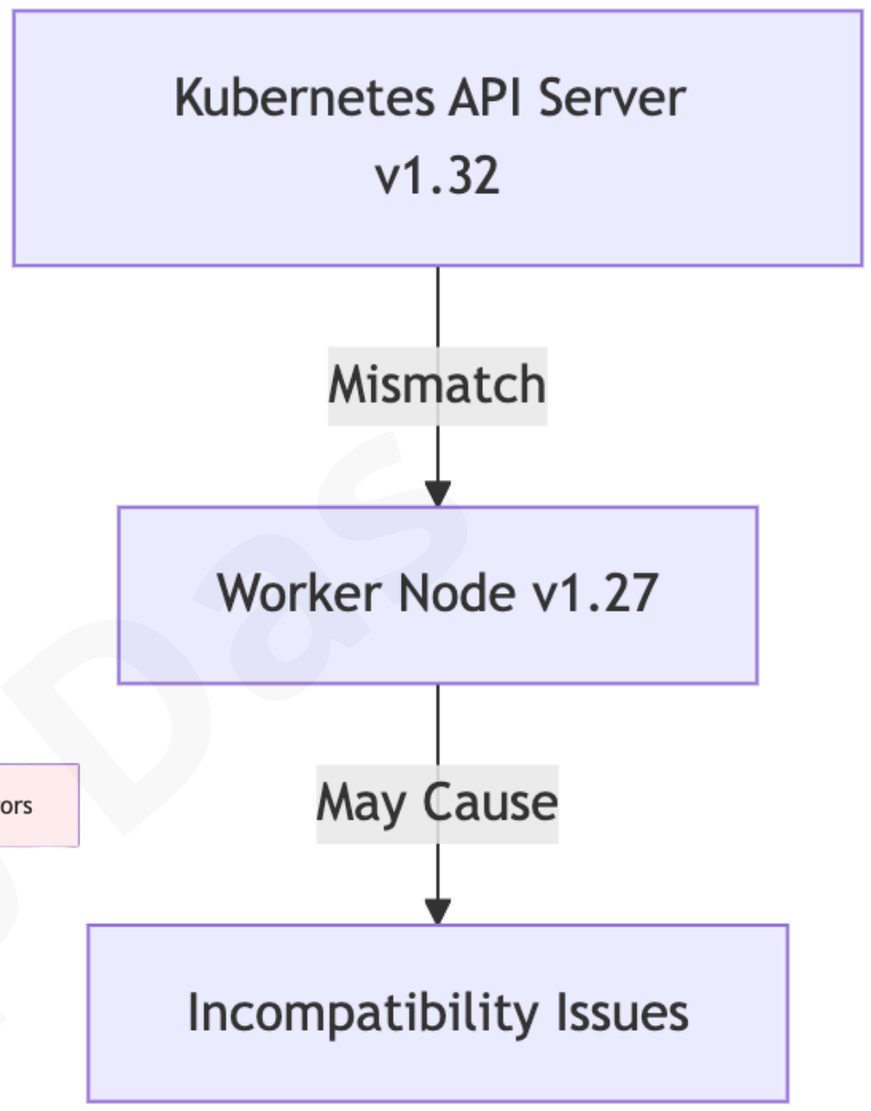
💡**Pro Tip: Use tools like Kyverno to enforce cleanup policies.**

@Sandip Das

Kubernetes Gotchas and Anti-Patterns

Version Skew

Kubernetes releases frequent updates, and sometimes control plane and worker nodes end up running different versions, it's called **Version Skew.**



Problem: API deprecations or unsupported features when components are too far apart.

✅ **Fix:**
Check version and upgrade incrementally

```
# Check versions
kubectl version --short
kubelet --version

# Upgrade nodes incrementally (e.g., 1.25 → 1.26 → 1.27....1.32).
```


@Sandip Das

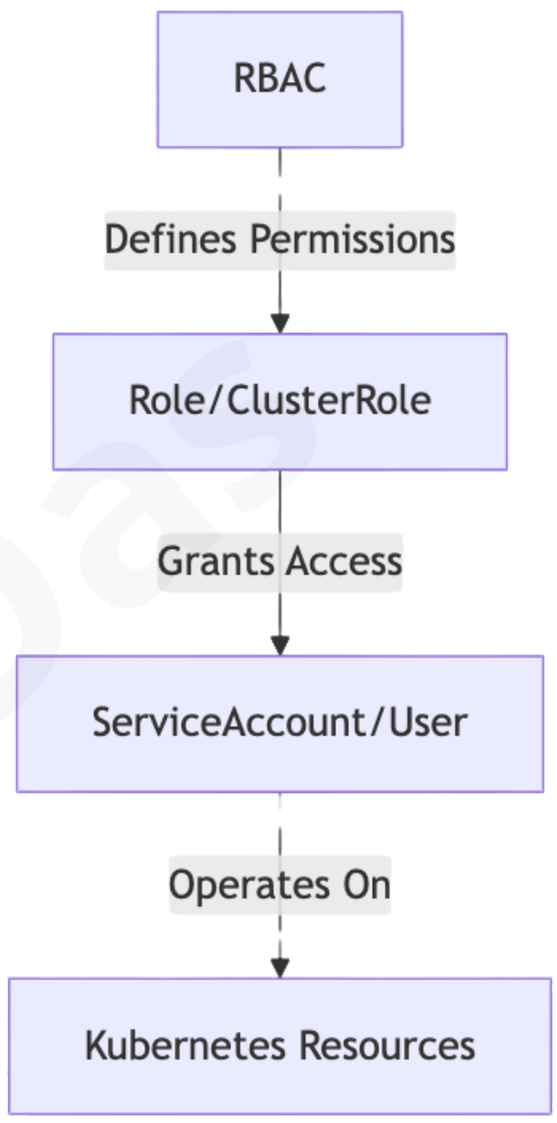
Kubernetes Security Best Practices

Security in Kubernetes is critical to protect your cluster from unauthorized access, malicious workloads, and network attacks. Now we will cover:

✅ RBAC (Role-Based Access Control): Controlling user and service permissions
✅ Pod Security Admission (PSA) : Restricting privileged pods
✅ Network Policies: Isolating namespaces and securing traffic
✅ Image Security: Scanning images and avoiding vulnerabilities

Role-Based Access Control (RBAC)

RBAC restricts access based on Roles, ClusterRoles, and ServiceAccounts.



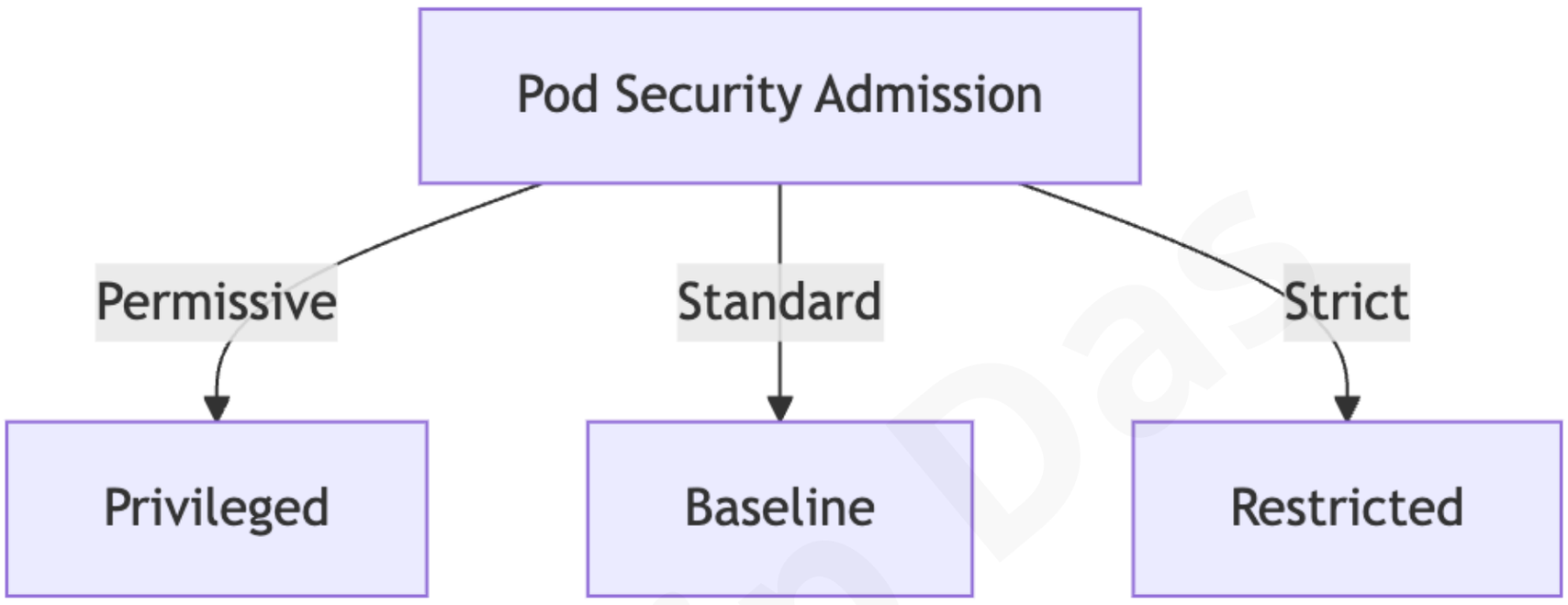
Example:

```
# Role: Read-only access to pods in "monitoring" namespace
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: monitoring
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]

# RoleBinding: Assign to a ServiceAccount
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: monitoring
subjects:
- kind: ServiceAccount
  name: prometheus-sa
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

💡 **Best Practices:**
Avoid wildcard (*) permissions.
Use dedicated ServiceAccounts (not default).

Kubernetes Security Best Practices

Pod Security Policies (PSPs)



Pod Security Admission (PSA) is the replacement for PodSecurityPolicies (PSP), which was deprecated in Kubernetes 1.21 and removed in Kubernetes 1.25. PSA enforces security policies at the namespace level using predefined security standards.

Enforcing PSA at Namespace Level
You assign PSA levels using labels on namespaces.

kubectl label namespace dev pod-security.kubernetes.io/enforce=baseline
kubectl label namespace prod pod-security.kubernetes.io/enforce=restricted

PSA can be applied in three modes:
Enforce : Rejects non-compliant Pods before creation
Audit : Logs warnings for non-compliant Pods
Warn: Displays a warning but still allows Pod creation
e.g.
kubectl label namespace dev pod-security.kubernetes.io/enforce=baseline
kubectl label namespace dev pod-security.kubernetes.io/audit=restricted
kubectl label namespace dev pod-security.kubernetes.io/warn=restricted
Example Privilage pod:

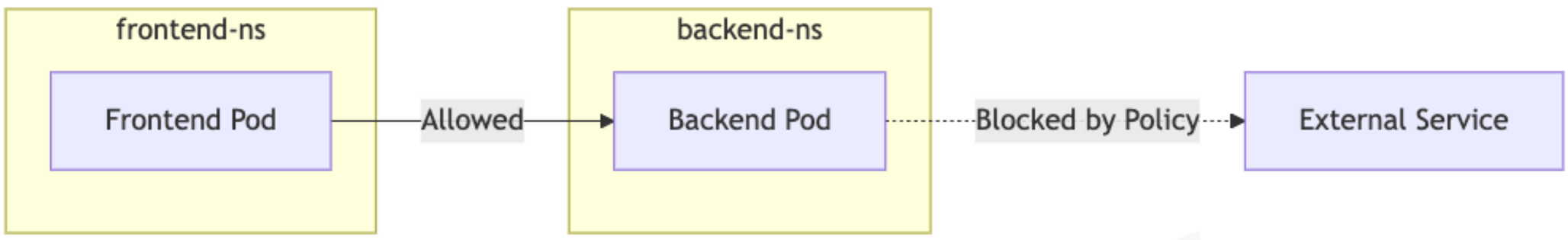
```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
  namespace: prod
spec:
  containers:
  - name: nginx
    image: nginx
    securityContext:
      privileged: true
```

✅ **Expected Outcome:**
Pod creation fails due to PSA enforcing restricted.
Kubernetes denies privileged containers in the prod namespace.

@Sandip Das

Kubernetes Security Best Practices

Network Policies



By default, Kubernetes allows all Pods to talk to each other across namespaces.
Principle: Deny all traffic by default, then allow specific communication.

Example (Deny All + Allow Namespace Traffic):

```
# Deny all ingress/egress by default
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress

# Allow traffic from "frontend" namespace
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend
spec:
  podSelector:
    matchLabels:
      app: backend
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: frontend
```


Kubernetes Security Best Practices

Image Security

Using **insecure images** can expose your cluster to attacks.
Principle: Scan images for vulnerabilities and avoid mutable tags like :latest.

Best Practices:

- Use immutable tags (e.g., sha256:abc123 or v1.2.3).
- Integrate tools like Trivy or Clair into CI/CD.
- Block unsigned images with Cosign/Notary.

Example (ImagePullPolicy):

```
containers:- name: app
  image: my-registry/app:v1.2.3
  imagePullPolicy: IfNotPresent  # Avoid "Always" for mutable tags
```


@Sandip Das

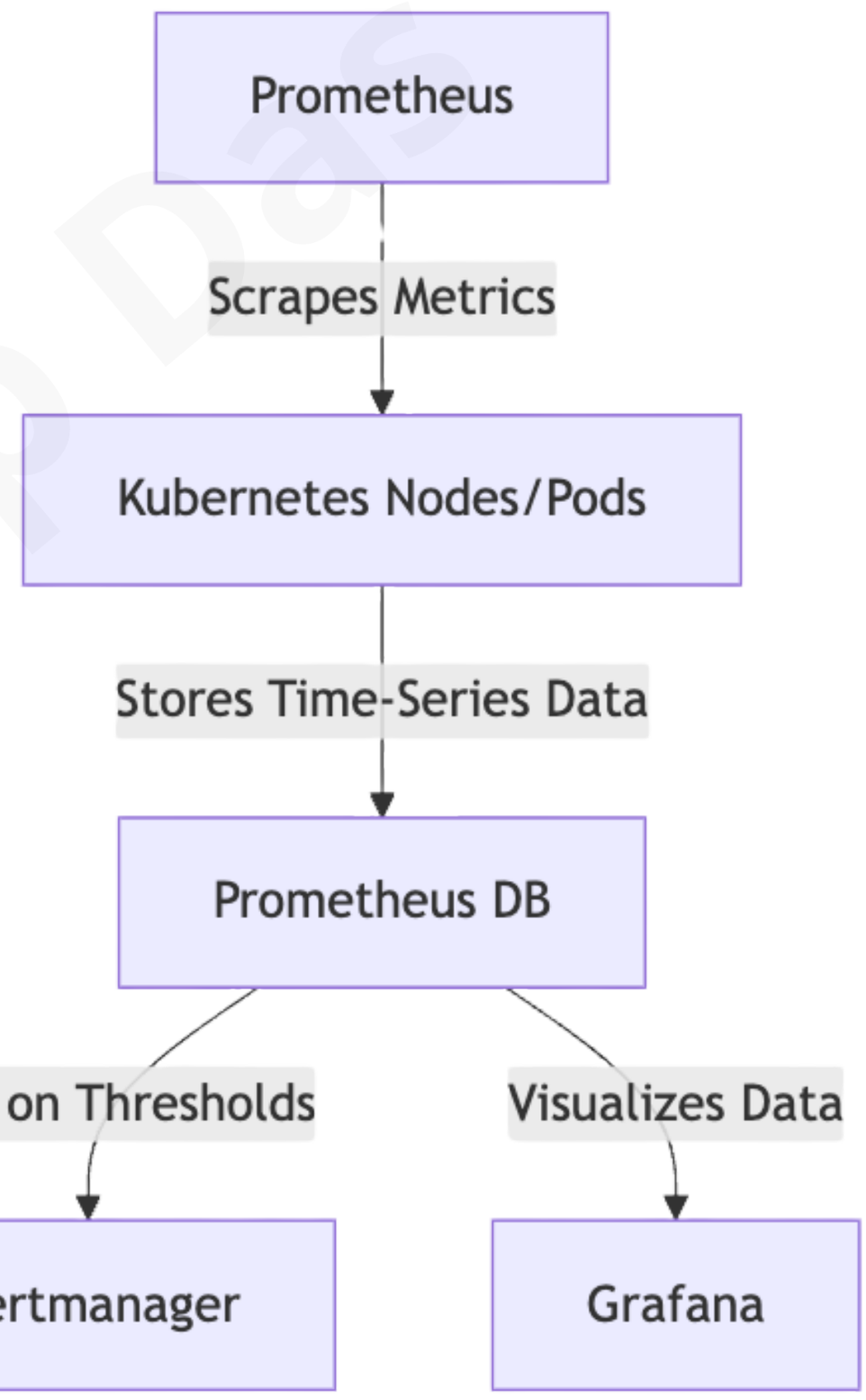
Monitoring, Logging, and Observability

Observability in Kubernetes consists of metrics, logging, and tracing to gain insights into application performance and health. now we will cover:

✅ Prometheus/Grafana → Metrics collection & alerting
✅ EFK Stack (Elasticsearch, Fluentd, Kibana) → Centralized logging
✅ Distributed Tracing (Jaeger, OpenTelemetry) → Request flow visualization

Prometheus/Grafana

Prometheus collects time-series data, and **Grafana** visualizes it.



Install Prometheus via Helm:

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm install prometheus prometheus-community/kube-prometheus-stack

Define a Prometheus Alert Rule

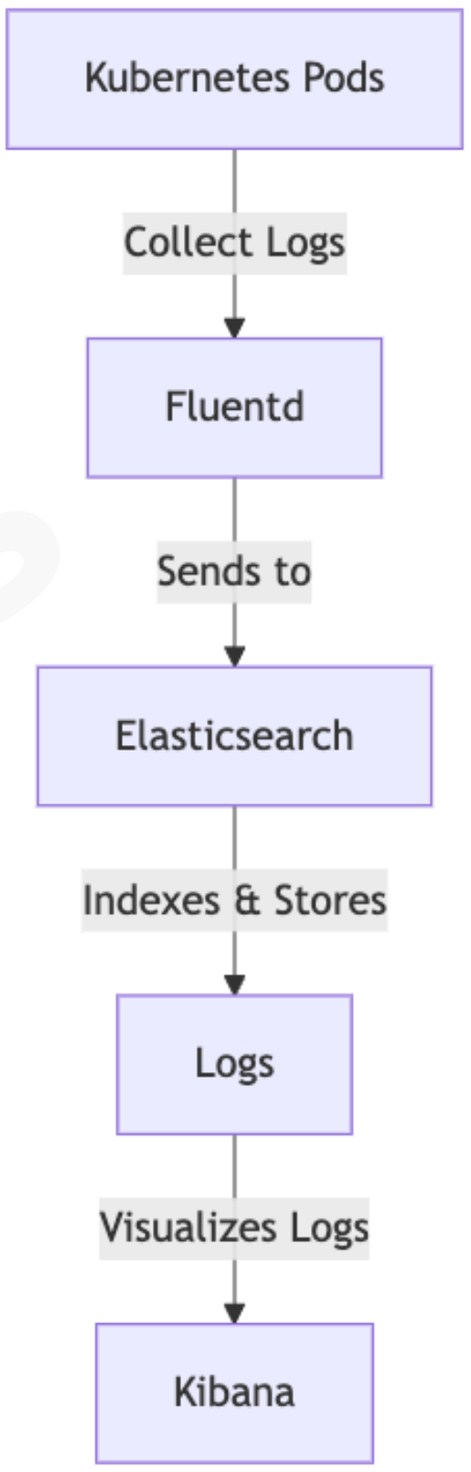
```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: high-cpu-usage
spec:
  groups:
  - name: node-rules
    rules:
    - alert: HighCPUUsage
      expr: node_cpu_seconds_total > 80
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "High CPU usage detected"
```

✅ Benefits

- Real-time monitoring of CPU, memory, and requests.
- Alerts via Alertmanager (Slack, Email, PagerDuty).
- Grafana dashboards for visualization.

Monitoring, Logging, and Observability

EFK Stack: Centralized Logging



EFK (Elasticsearch, Fluentd, Kibana) enables log collection, storage, and visualization.

Install EFK via Helm:

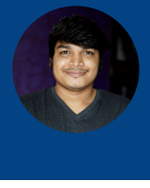
```
helm repo add elastic https://helm.elastic.co
helm install elasticsearch elastic/elasticsearch
helm install kibana elastic/kibana
helm install fluentd stable/fluentd
```

Fluentd Configuration (Logging Pods)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentd-config
data:
  fluent.conf: |
    <source>
      @type tail
      path /var/log/containers/*.log
      pos_file /var/log/td-agent/containers.log.pos
      tag kubernetes.*
      format json
    </source>

<match kubernetes.**>
  @type elasticsearch
  host elasticsearch
  port 9200
  logstash_format true
</match>
```

✅ **Benefits**

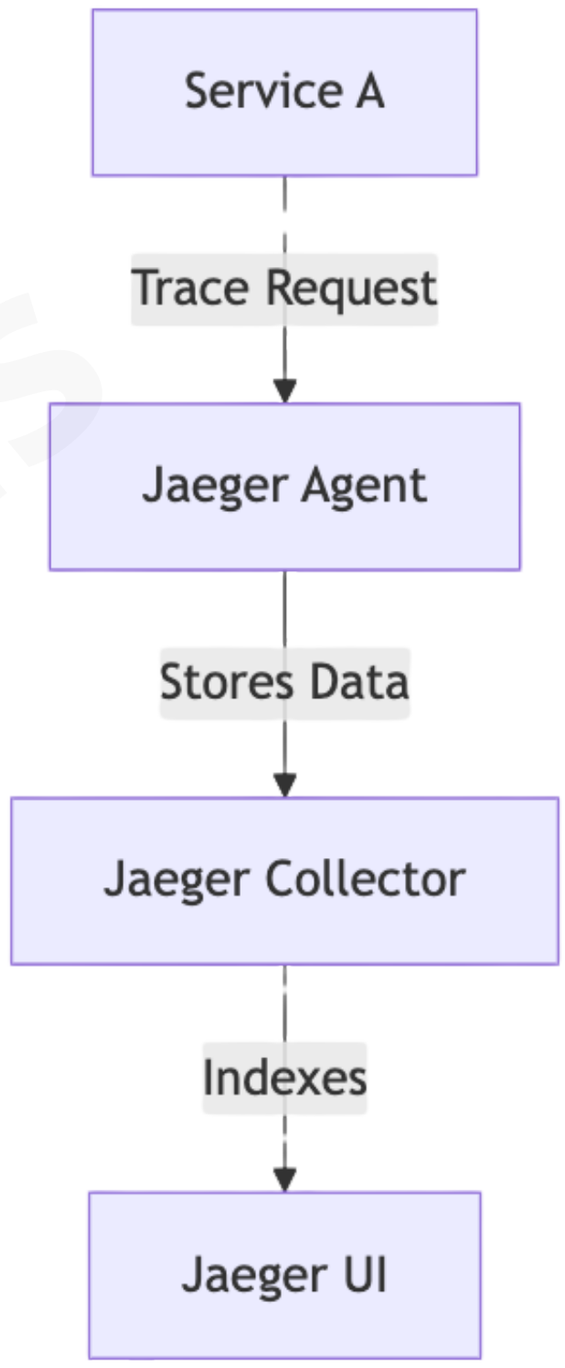
- **Aggregates logs from all Kubernetes Pods.**
- **Elasticsearch stores logs, enabling fast searches.**
- **Kibana visualizes logs, helping with debugging.**



Monitoring, Logging, and Observability

Distributed Tracing: Jaeger & OpenTelemetry

Tracing helps track requests as they move through microservices.



Install Jaeger via Helm:

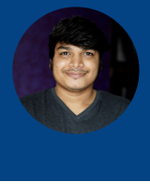
```
helm repo add jaegertracing
https://jaegertracing.github.io/helm-charts
helm install jaeger jaegertracing/jaeger
```

Tracing Requests in Python (OpenTelemetry)

```
from opentelemetry import trace
from opentelemetry.exporter.jaeger.thrift import JaegerExporter
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import SimpleSpanProcessor

# Setup Tracer
trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)

# Export to Jaeger
jaeger_exporter = JaegerExporter(agent_host_name="jaeger", agent_port=6831)
trace.get_tracer_provider().add_span_processor(SimpleSpanProcessor(jaeger_exporter))

# Create a span
with tracer.start_as_current_span("operation"):
    print("Tracing request in Jaeger")
```

✅ **Benefits**

- **Identifies latency issues in microservices.**
- **Shows dependencies between services.**
- **Helps with debugging slow requests.**



CI/CD Pipelines with Kubernetes

CI/CD (Continuous Integration and Continuous Deployment) automates application building, testing, and deployment in Kubernetes. Now we will cover:

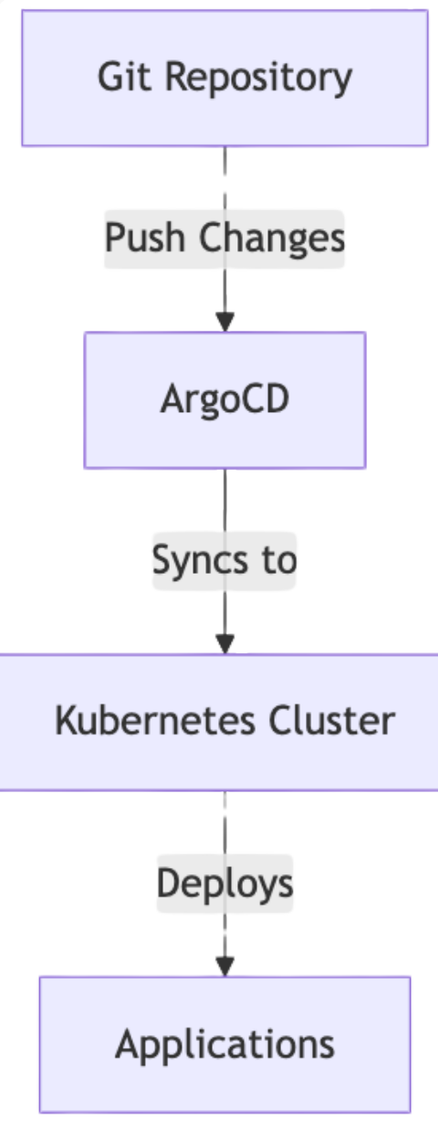
✅ GitOps with Argo CD → Declarative Kubernetes deployments
✅ Jenkins Pipelines → Automating build, test, and deployment
✅ Helm Charts → Packaging and managing Kubernetes releases

GitOps with Argo CD: Declarative Deployments

GitOps uses Git as the single source of truth for Kubernetes configurations, ensuring declarative deployments.

Install Argo CD:

```
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```



Expose Argo CD Dashboard

```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

Deploy an Application with Argo CD

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  destination:
    namespace: my-app
    server: https://kubernetes.default.svc
  source:
    repoURL: https://github.com/my-org/my-repo.git
    targetRevision: main
    path: my-app
syncPolicy:
  automated:
    prune: true
    selfHeal: true
```

@Sandip Das

CI/CD Pipelines with Kubernetes

Jenkins Pipelines

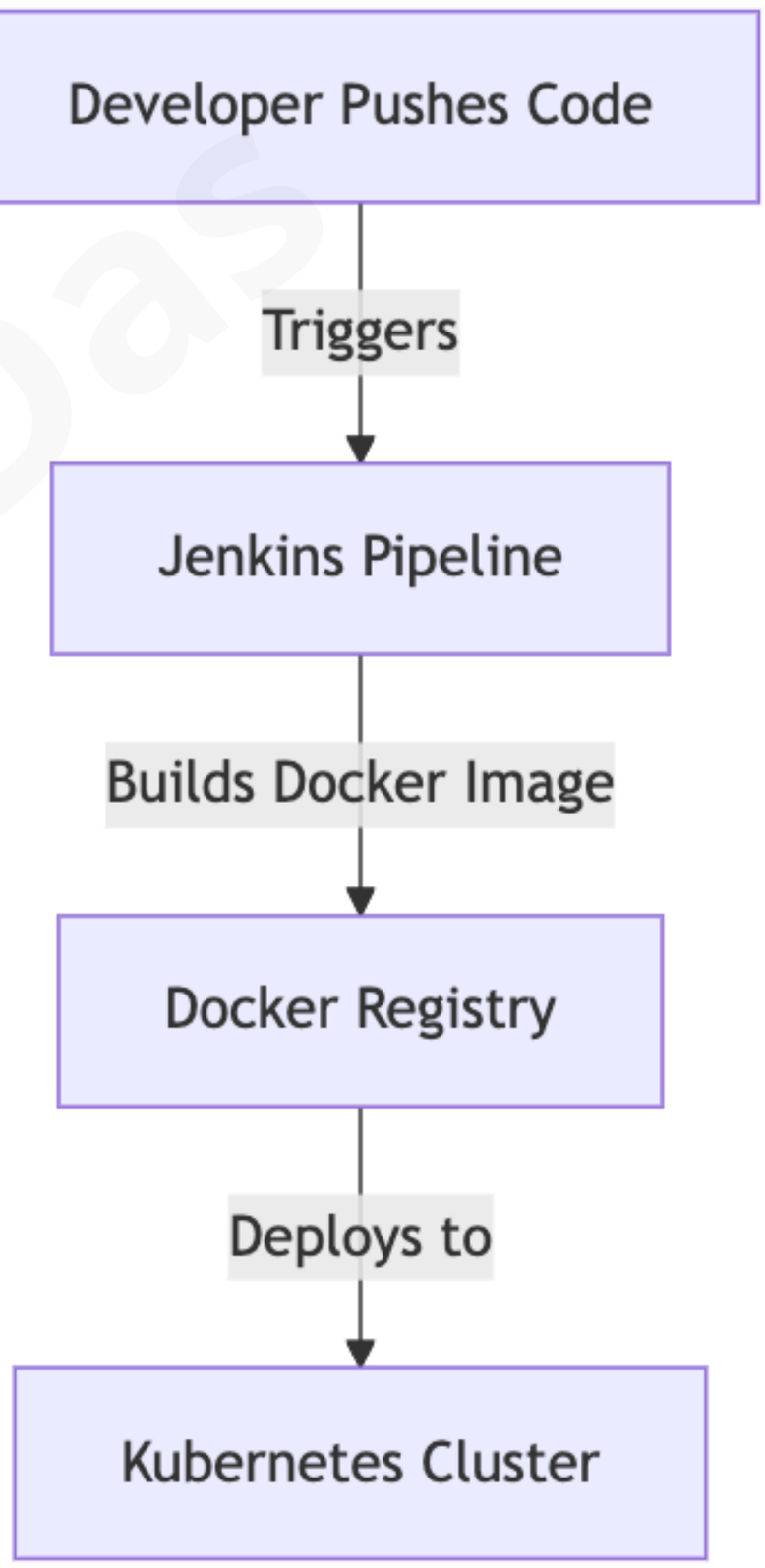
Jenkins automates building, testing, and deploying containerized applications.

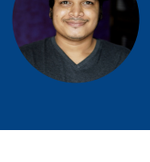
Install Jenkins on Kubernetes:

```
helm repo add jenkins https://charts.jenkins.io
helm install jenkins jenkins/jenkins
```

Example: Jenkinsfile for Kubernetes CI/CD

```
pipeline {
    agent any
    environment {
        IMAGE_NAME = "my-app"
        IMAGE_TAG = "v1.0"
        REGISTRY = "docker.io/myrepo"
    }
    stages {
        stage('Build') {
            steps {
                sh 'docker build -t $REGISTRY/$IMAGE_NAME:$IMAGE_TAG .'
            }
        }
        stage('Push to Registry') {
            steps {
                withDockerRegistry([credentialsId: 'docker-credentials', url: 'https://index.docker.io/v1/']) {
                    sh 'docker push $REGISTRY/$IMAGE_NAME:$IMAGE_TAG'
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f k8s/deployment.yaml'
            }
        }
    }
}
```





@Sandip Das

CI/CD Pipelines with Kubernetes

Helm Charts: Managing Kubernetes Releases

Helm packages Kubernetes applications into reusable charts for easy deployment.

Create a Helm Chart

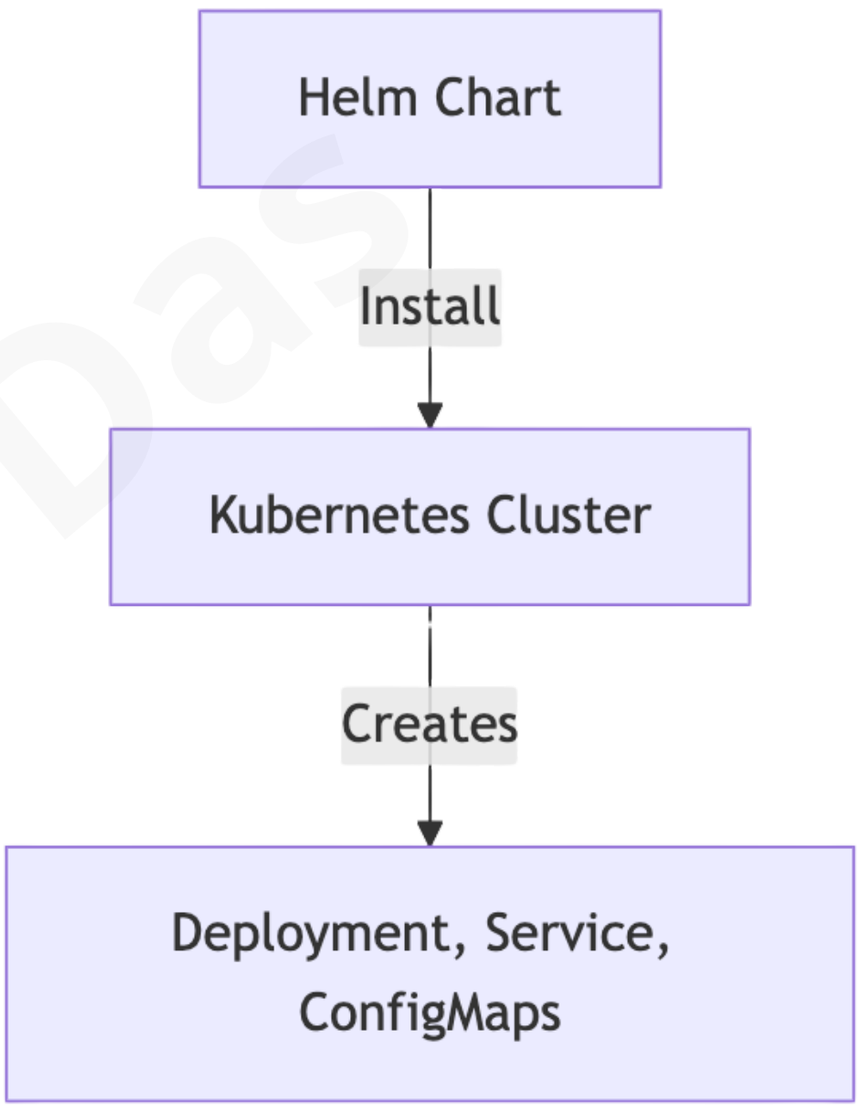
```
helm create my-app
```

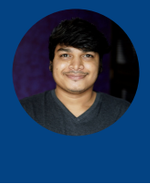
Deploy an Application with Helm

```
helm install my-app ./my-app
```

Example: Helm values.yaml

```
replicaCount: 2
image:
  repository: nginx
  tag: latest
service:
  type: LoadBalancer
  port: 80
```



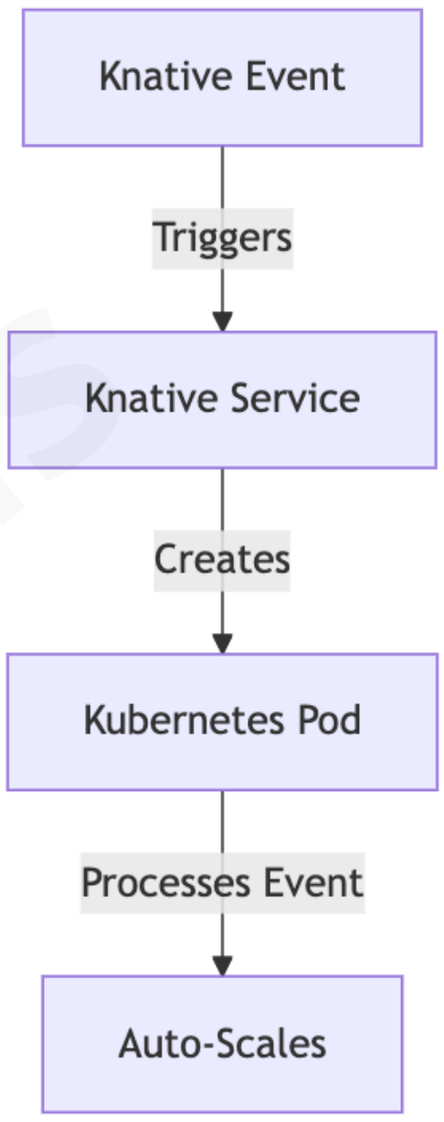


@Sandip Das

What Next You May Explore?

Serverless with Knative: Event-Driven Scaling

Knative brings serverless capabilities to Kubernetes, allowing applications to scale down to zero when idle.



Key Features:

✅ Event-driven scaling → Pods scale up/down based on requests

✅ Works with Kubernetes → Deploy serverless workloads without extra infrastructure

✅ Cold start optimization → Efficient resource usage

Installing Knative

```
kubectl apply -f
https://github.com/knative/serving/releases/latest/download/serving-crds.yaml
kubectl apply -f
https://github.com/knative/serving/releases/latest/download/serving-core.yaml
```

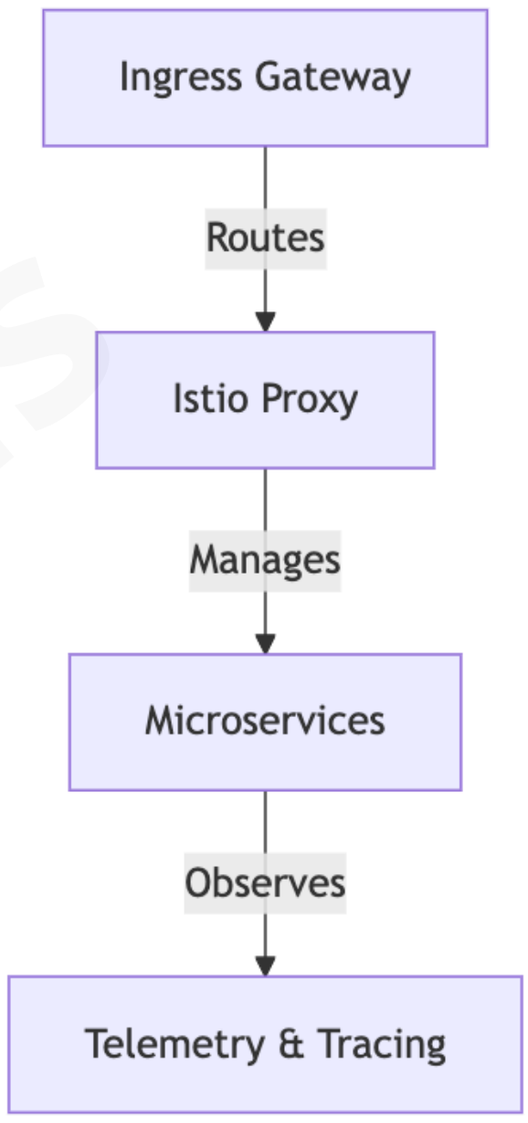
Example: Deploy a Serverless App

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello-knative
spec:
  template:
    spec:
      containers:
      - image: gcr.io/knative-samples/helloworld-go
```


What Next You May Explore?

Service Mesh with Istio: Advanced Traffic Management

Istio enhances Kubernetes networking with traffic routing, security, and observability.



Key Features:

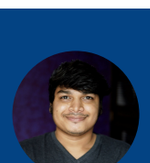
✅ Traffic management → A/B testing, canary releases
✅ Security → Mutual TLS (mTLS) encryption
✅ Observability → Built-in tracing and monitoring

Install Istio

```
curl -L https://istio.io/downloadIstio | sh -
cd istio-*
export PATH=$PWD/bin:$PATH
istioctl install --set profile=demo -y
```

Example: Canary Deployment

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: my-app
spec:
  hosts:
  - my-app.example.com
  http:
  - route:
    - destination:
        host: my-app
        subset: v1
      weight: 80
    - destination:
        host: my-app
        subset: v2
      weight: 20
```

What Next You May Expore?

Edge Computing with K3s & MicroK8s

Edge computing requires lightweight Kubernetes distributions optimized for low-resource environments.

K3s (Lightweight Kubernetes)

✅ Designed for IoT & Edge
✅ Single binary, less than 100MB
✅ Faster startup than full Kubernetes

Install K3s on a Raspberry Pi

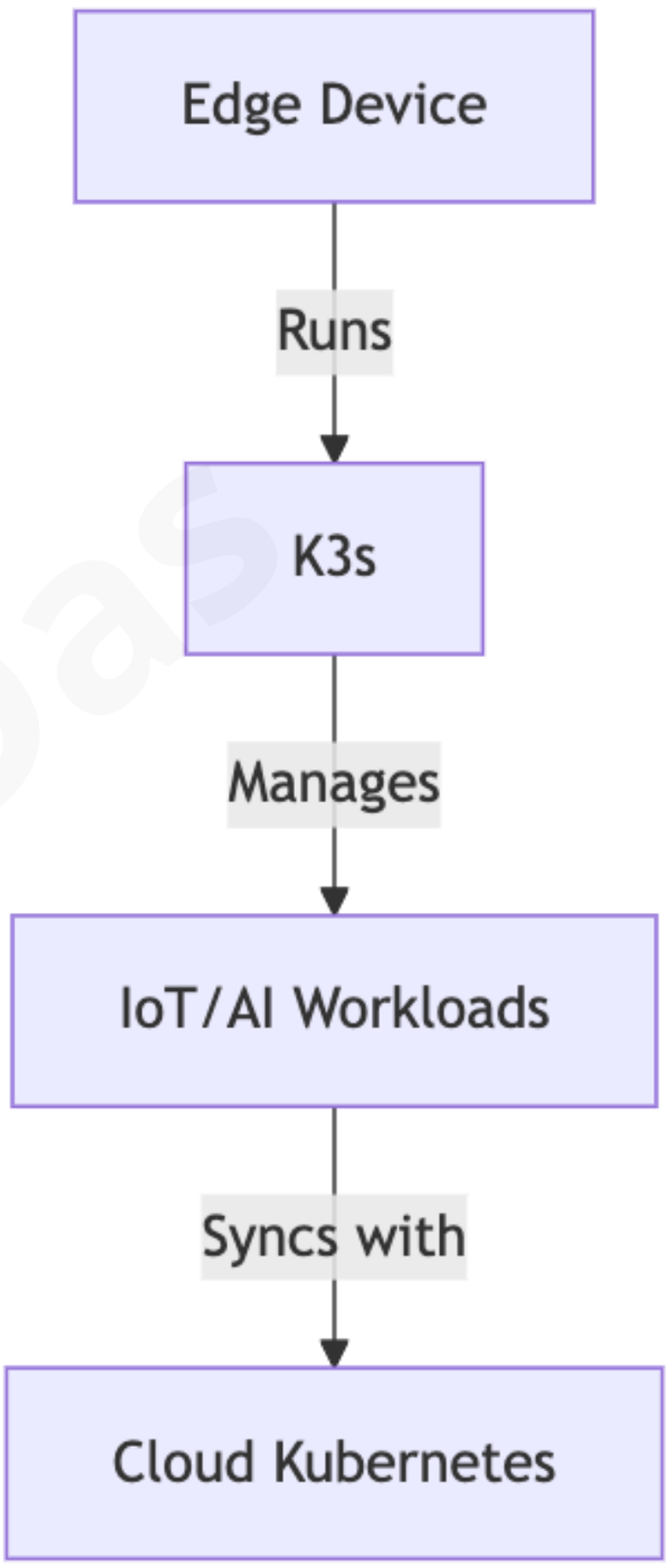
```
curl -sfL https://get.k3s.io | sh -
```

MicroK8s (Canonical's Lightweight K8s)

✅ Optimized for local development & small clusters
✅ Snap-based install for easy upgrades
✅ Includes built-in storage & networking

Install MicroK8s

```
sudo snap install microk8s --classic
```



GET SET GO

Keep Learning, Keep Building! 🚀

Congratulations on completing Kubernetes Handbook - From Beginner to Expert!

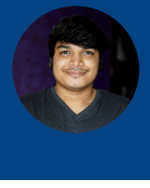
This is just the beginning of your journey in the world of Kubernetes and cloud-native technologies.

The tech landscape is ever-evolving, and the best way to stay ahead is to keep learning, experimenting, and sharing knowledge with others. Whether you are deploying your first containerized application or managing large-scale clusters, remember: every expert was once a beginner.

💡 **What's next?**

- Apply what you've learned by working on real-world projects.
- Contribute to open-source communities and collaborate with fellow engineers.
- Stay updated with the latest in Kubernetes and DevOps through blogs, conferences, and forums.
- Teach and mentor others – knowledge grows when shared!

If this book helped you, let me know! Connect with me on LinkedIn or follow my work at LearnXOps.
Happy learning, and may your clusters always be healthy! 🚀🔥
- Sandip Das



@Sandip Das