Core Concepts

#### Monolithic vs Microservices

Monolithic: Entire application packaged into a single unit (difficult to scale, update).

Microservices: Application broken into smaller independent services (scalable, fault-tolerant).

Kubernetes is built to manage microservices at scale.

#### Kubernetes Architecture

Control Plane: API Server, Scheduler, Controller Manager, ETCD (stores cluster state).

Worker Nodes: Run Pods, kubelet, kube-proxy, container runtime (Docker/Containerd).

Pods: Smallest deployable unit in Kubernetes.

# Setup on Local/AWS EC2

Local: Using Minikube, Kind, or Docker Desktop.

Cloud: Deploying clusters on AWS EC2 (via Kops, EKS, kubeadm).

# Kubectl

CLI tool for interacting with Kubernetes clusters. Example:

```
kubectl get pods
kubectl describe pod <pod-name>
kubectl apply -f deployment.yaml
```

#### Pods

Smallest unit in Kubernetes (can hold one or more containers).

Each pod gets its own IP inside the cluster.

#### Namespaces

Logical isolation within the cluster (e.g., dev, test, prod).

#### Labels, Selectors, Annotations

Labels: Key-value pairs for grouping (e.g., app=frontend).

Selectors: Used to filter resources by labels.

Annotations: Metadata (not used for selection).

#### Workloads

These define how applications run in Kubernetes:

# Deployments – Manage stateless applications, handle scaling & rolling updates.

# StatefulSets – Manage stateful apps (databases, Kafka, etc.), provide stable

identity & storage.

# DaemonSets – Ensure a copy of a pod runs on all (or some) nodes (e.g., logging agents).

# ReplicaSets – Ensure a specified number of pod replicas are running.

# Jobs – Run tasks that complete and then exit (batch jobs).

# CronJobs – Scheduled jobs (like Linux cron, e.g., run backup every night).

#### Networking

# Cluster Networking

Each Pod gets a unique IP.

Kubernetes ensures connectivity across nodes using CNI plugins (Calico, Flannel, Weave).

# Services

ClusterIP: Internal access only.

NodePort: Exposes service on each node’s IP:port.

LoadBalancer: External traffic routing (uses cloud provider’s LB).

Headless: No cluster IP, DNS directly maps to pod IPs.

# Ingress

Manages external access to services (HTTP/HTTPS).

Acts as a reverse proxy and load balancer with routing rules.

# Network Policies

Define which Pods can communicate with each other (like firewalls inside the cluster).

#### Storage

# Persistent Volumes (PV)

Cluster-wide storage abstraction (can be backed by AWS EBS, NFS, GCP disk, etc.).

# Persistent Volume Claims (PVC)

Pod request for storage (claims PV).

# StorageClasses

Define how dynamic storage should be provisioned (e.g., SSD vs HDD).

# ConfigMaps

Store configuration data (non-sensitive). Example: environment variables.

# Secrets

Store sensitive information (passwords, API keys, TLS certs).

Base64 encoded, not encrypted by default.

# 🧑‍💻 Kubernetes Interview Q\&A (Advanced Topics)

## **1. Scaling and Scheduling**

### ❓ What is the difference between HPA and VPA?

Answer:

* **HPA (Horizontal Pod Autoscaler)** scales the number of pod replicas based on CPU, memory, or custom metrics.
* **VPA (Vertical Pod Autoscaler)** adjusts the **resources (CPU/memory)** of existing pods.
  👉 HPA = scale **out/in**, VPA = scale **up/down**.

Example:

```bash
kubectl autoscale deployment nginx --cpu-percent=50 --min=2 --max=10
```

### ❓ How do Node Affinity and Taints/Tolerations differ?

Answer:

* **Node Affinity**: Soft/hard rules to **attract pods** to specific nodes (e.g., "run this on SSD nodes").
* **Taints/Tolerations**: Mechanism to **repel pods** unless they tolerate the taint.
  👉 Affinity = "go there", Taint = "keep away unless tolerated".

### ❓ What are Resource Quotas and Limits?

Answer:

* **Resource Requests**: Minimum CPU/memory a pod needs.
* **Limits**: Max resources a pod can consume.
* **ResourceQuota**: Enforce per-namespace limits.

Example: Prevents one team from hogging cluster resources.

### ❓ What are Probes in Kubernetes?

Answer:

* **Liveness Probe**: Checks if the container is alive; restarts if unhealthy.
* **Readiness Probe**: Checks if container is ready to serve traffic.
* **Startup Probe**: Used for slow-starting apps.

## **2. Cluster Administration**

### ❓ Explain RBAC in Kubernetes.

Answer:
RBAC (Role-Based Access Control) restricts what users or service accounts can do.

* **Role**: Namespace-scoped permissions.
* **ClusterRole**: Cluster-wide permissions.
* **RoleBinding/ClusterRoleBinding**: Bind roles to users or service accounts.

👉 Example: Only allow a dev to `get` and `list` pods in `dev` namespace.

### ❓ What is a CRD (Custom Resource Definition)?

Answer:

* CRDs allow extending Kubernetes with custom objects.
* Example: Create a custom resource `Database` to manage MySQL clusters like any native Kubernetes object.
  👉 This is the foundation for building **Operators**.

### ❓ How do you upgrade a Kubernetes cluster?

Answer:

* In **managed services (EKS/GKE/AKS)**: Use provider’s upgrade command/console.
* In **self-managed cluster**:

  1. Upgrade `kubeadm`
  2. Upgrade control-plane nodes (`kubeadm upgrade`)
  3. Upgrade worker nodes (`kubectl drain`, update kubelet, uncordon).

## **3. Monitoring and Logging**

### ❓ How do you monitor resource usage in Kubernetes?

Answer:

* Install **Metrics Server** → enables `kubectl top nodes/pods`.
* For detailed monitoring, use **Prometheus + Grafana**.

### ❓ How do you handle logging in Kubernetes?

Answer:

* Default: `kubectl logs <pod>`.

* For centralized logging: use **EFK (Elasticsearch + Fluentd + Kibana)** or **Loki + Promtail + Grafana**.

## **4. Advanced Features**

### ❓ What are Operators in Kubernetes?

Answer:

* Operators extend Kubernetes to manage complex apps (e.g., databases).
* They use CRDs + controllers to automate lifecycle tasks like backup, failover, scaling.

### ❓ What is Helm and why is it used?

Answer:

* Helm is the **package manager for Kubernetes**.
* It bundles multiple YAML manifests into a reusable **chart**.
  👉 Instead of applying 10 different YAMLs, you deploy 1 Helm chart.

### ❓ What is a Service Mesh?

Answer:

* A Service Mesh (Istio, Linkerd) manages **service-to-service communication**.
* Features: observability, traffic routing, retries, security (mTLS).
  👉 Useful for microservices.

## **5. Security**

### ❓ What are Pod Security Standards (PSS)?

Answer:

* Defines security levels for pods:

  * **Privileged** → full access (least secure).
  * **Baseline** → minimal restrictions.
  * **Restricted** → strong security (no root user, limited privileges).

### ❓ How do you secure container images in Kubernetes?

Answer:

* Use **image scanning tools** (Trivy, Anchore).
* Ensure images are signed (Cosign, Notary).
* Use private registries with authentication.

### ❓ How do you secure secrets in Kubernetes?

Answer:

* Store sensitive data in **Secrets** (base64 encoded).
* Enable **encryption at rest** using KMS providers (AWS KMS, GCP KMS).
* Mount secrets as env vars or files.

## **6. Cloud-Native Kubernetes**

### ❓ What’s the difference between EKS, AKS, and GKE?

Answer:

* All are **managed Kubernetes services**.
* **EKS (AWS)**, **AKS (Azure)**, **GKE (Google Cloud)** → manage the control plane for you.
* You only manage worker nodes & workloads.

### ❓ How does Cluster Autoscaler work?

Answer:

* Scales worker nodes **up/down** depending on pending pods.
* Works with cloud provider APIs (EC2 Auto Scaling Groups, GCP Instance Groups).

### ❓ What are Spot/Preemptible Nodes?

Answer:

* **Spot (AWS)** / **Preemptible (GCP)** are cheap, short-lived nodes.
* Great for **batch jobs** or **non-critical workloads**.
* If the cloud provider needs capacity, your nodes may be terminated.

Great question, Suraj 👍 — Kubernetes **debugging & troubleshooting** is one of the **most asked interview topics**, and also the most important in real-world clusters. I’ll cover it in a structured way with **common issues, commands, YAML fixes, and interview-ready answers**.

# 🛠 Kubernetes Debugging & Troubleshooting

## **1. Pod Issues**

### ❓ My Pod is stuck in `Pending`. How do you debug?

Answer:

* Check if there are enough resources (CPU/Memory) on nodes.
* Verify **PVC binding** if using persistent storage.
* Look at events.

Commands:

```bash
kubectl get pods
kubectl describe pod <pod-name>
kubectl get events --sort-by=.metadata.creationTimestamp
```

Common Fix:

* If node lacks resources → scale cluster.
* If PVC not bound → check PV and StorageClass.

### ❓ My Pod is in `CrashLoopBackOff`. How do you debug?

Answer:

* Container keeps failing on startup.
* Check logs of the container.

Commands:

```bash
kubectl logs <pod-name>
kubectl logs <pod-name> -c <container-name>  # multi-container pod
kubectl describe pod <pod-name>
```

Fixes:

* Wrong image? → Fix `image` field.
* Misconfigured environment variables? → Check ConfigMaps/Secrets.
* App needs time? → Add **readiness/liveness probes** carefully.

### ❓ How do you get inside a running Pod for debugging?

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

👉 Useful to check app logs/configs inside the container.

## **2. Deployment & Replica Issues**

### ❓ My Deployment is not scaling. How do you debug?

* Check replica count:

  ```bash
  kubectl get deploy
  kubectl describe deploy <name>
  ```
* Check if HPA is working:

  ```bash
  kubectl get hpa
  ```

Common Fixes:

* Metrics server not installed → HPA won’t work.
* Node resource shortage.

### ❓ How do you roll back a failed Deployment?

```bash
kubectl rollout undo deployment <deployment-name>
```

## **3. Service & Networking Issues**

### ❓ My Service is not reachable. How do you debug?

1. Check Service:

   ```bash
   kubectl get svc
   kubectl describe svc <svc-name>
   ```
2. Check Endpoints (should map to pods):

   ```bash
   kubectl get endpoints <svc-name>
   ```
3. Exec into a pod and curl the service name:

   ```bash
   kubectl exec -it <pod> -- curl http://<svc-name>:<port>
   ```

Common Fixes:

* Service selector labels don’t match Pod labels.
* Pod not running on correct port.

### ❓ My Ingress is not working. How do you debug?

1. Check Ingress rules:

   ```bash
   kubectl describe ingress <name>
   ```
2. Verify Ingress Controller is deployed (`nginx-ingress`, `traefik`, etc.).
3. DNS must resolve to Ingress external IP.

## **4. Node Issues**

### ❓ A Node is in `NotReady` state. What do you check?

* Node status:

  ```bash
  kubectl get nodes
  kubectl describe node <node-name>
  ```

~~~
* Kubelet logs on the node.
* Check `docker` or `containerd` runtime.
* Ensure network (CNI plugin) is running.

**Fix:** Restart kubelet, check disk/memory, check CNI pods.

---

## **5. Storage Issues**

### ❓ My Pod is stuck waiting for PVC. How do you debug?

```bash
kubectl get pvc
kubectl describe pvc <pvc-name>
kubectl get pv
~~~

Fix:

* Ensure a matching PV exists or dynamic provisioning via StorageClass works.

## **6. RBAC Issues**

### ❓ I get “forbidden” errors while using kubectl. Why?

Answer:

* RBAC rules don’t allow your user/service account to perform that action.

Debug:

```bash
kubectl auth can-i get pods --as=developer
```

Fix: Add appropriate `Role`/`RoleBinding`.

## **7. Debugging Tools & Commands**

* **Check pod logs**

  ```bash
  kubectl logs <pod>
  ```

* **Check events in namespace**

  ```bash
  kubectl get events -n <namespace>
  ```

* **Debug pods without a shell**

  ```bash
  kubectl run tmp --rm -it --image=busybox -- sh
  ```

  👉 Create a temporary debug pod.

* **Describe resources**

  ```bash
  kubectl describe pod <pod>
  kubectl describe node <node>
  ```

* **Network testing pod**

  ```bash
  kubectl run net-test --image=busybox:1.28 -it --rm -- nslookup <service-name>
  ```

## **8. Interview-Worthy Troubleshooting Scenarios**

1. **Pod keeps restarting** → Misconfigured app/env variables.
2. **Service has no endpoints** → Labels mismatch.
3. **Ingress not routing** → Missing Ingress Controller.
4. **PVC not bound** → No suitable PV/StorageClass.
5. **Forbidden errors** → RBAC issue.
6. **Node NotReady** → Kubelet/CNI failure.

✅ With these debugging flows, you’ll show interviewers that you can **systematically troubleshoot** Kubernetes like a real SRE/DevOps engineer.

Would you like me to now create a **flowchart-style checklist (step-by-step debugging guide)** so you can quickly recall during interviews or on the job?