Kubernetes Interview Questions and Answers

## 1. Your pod is stuck in CrashLoopBackOff — what’s your first step in debugging it?

Start by inspecting the pod logs using `kubectl logs <pod-name> --previous` to see the error before the crash. Then check the pod's events with `kubectl describe pod <pod-name>` to identify issues like misconfigured environment variables, missing files, or failed health checks. Also verify the container image and entrypoint.

## 2. A pod is terminating for more than 5 minutes. What could be the possible reasons?

Possible reasons include a long-running process that ignores SIGTERM, a misconfigured terminationGracePeriodSeconds, or a stuck volume unmount. Check the pod's lifecycle hooks and logs to identify delays.

## 3. You applied a new ConfigMap, but your pod isn't reflecting the changes. Why?

ConfigMaps are mounted at pod startup. To reflect changes, you must restart the pod or use a sidecar container to reload configuration dynamically. Also ensure the pod is correctly referencing the updated ConfigMap.

## 4. Your pod keeps restarting every few seconds. What are common causes?

Common causes include application crashes, failed health checks, or resource limits being exceeded. Use `kubectl describe pod` and `kubectl logs` to identify the root cause.

## 5. Your deployment shows 3 pods running, but only 1 is ready. What could be going wrong?

Check readiness probes in the pod spec. If they are misconfigured or the application is slow to start, pods may not become ready. Also inspect container logs and events for errors.

## 6. You need to test a new container image in a pod without modifying your deployment. What’s the best approach?

Use `kubectl run` to create a temporary pod with the new image: `kubectl run test-pod --image=<new-image> --restart=Never`. This allows isolated testing.

## 7. A pod is OOMKilled (Out of Memory). What two areas should you immediately inspect?

Inspect the container's memory limits in the pod spec and the application's memory usage. Use metrics from monitoring tools and adjust resource requests/limits accordingly.

## 8. Your pod can’t resolve external DNS names (like google.com). Where do you look?

Check the pod’s DNS configuration in `/etc/resolv.conf`, the cluster’s CoreDNS logs, and network policies that may block DNS traffic.

## 9. You want to prevent a pod from accessing the internet, but allow internal cluster access. How would you configure this?

Use Kubernetes NetworkPolicies to deny egress to external IPs while allowing traffic to cluster CIDRs. Example: deny all egress except to `10.0.0.0/8`.

## 10. A pod is not reaching a service running in another namespace. What checks will you do?

Verify the service name and namespace in the DNS format `<service>.<namespace>.svc.cluster.local`. Check NetworkPolicies, service selectors, and pod labels.

## 11. How would you ensure a pod can only be scheduled on nodes with a specific GPU?

Use node selectors or affinity rules with labels like `nvidia.com/gpu.present=true`. Also use resource requests for `nvidia.com/gpu`.

## 12. You deleted a pod manually, but it immediately came back. Why?

The pod is managed by a controller like Deployment or ReplicaSet which ensures desired state. To stop it, delete the controller resource.

## 13. How would you inject environment variables securely into a pod?

Use Kubernetes Secrets and reference them in the pod spec via `envFrom` or `env`. Ensure RBAC policies restrict access to secrets.

## 14. You need a pod to start only after another pod becomes ready. How do you handle that?

Use init containers to wait for readiness, or implement readiness checks via shared volumes or APIs. Alternatively, use external orchestration.

## 15. You notice your pod is using an outdated image even after updating the deployment. What’s missing?

Ensure the image tag is updated and not using `latest`. Use `kubectl rollout restart deployment <name>` to force a restart.

## 16. How do you troubleshoot a pod that mounts a PVC but cannot write to it?

Check PVC and PV status, access modes, and permissions. Use `kubectl describe pvc` and inspect logs for permission errors.

## 17. You want to troubleshoot a running pod's logs but only for a specific container. How?

Use `kubectl logs <pod-name> -c <container-name>` to view logs for the specific container.

## 18. Your pod takes too long to become Ready. Which liveness/readiness configuration mistakes might cause this?

Misconfigured probes (wrong path, port, or delay) can delay readiness. Ensure the application is ready before the probe timeout.

## 19. You rolled out a deployment, but the pods are not updating. What could be blocking the rollout?

Check for paused rollout, failed health checks, or resource quota limits. Use `kubectl rollout status` and `kubectl describe deployment`.

## 20. How do you schedule a pod on a specific node without using taints?

Use nodeName in the pod spec or node affinity rules to target specific nodes based on labels.

**Node affinity** and **anti-affinity** are Kubernetes scheduling rules that control **where pods are placed** in a cluster based on labels assigned to nodes. They help optimize resource usage, improve fault tolerance, and enforce workload isolation.

**🔹 Node Affinity**

**Node affinity** ensures that a pod is scheduled **on nodes that match specific labels**.

**✅ Example Use Case:**

You have GPU-enabled nodes labeled gpu=true, and you want your ML workloads to run only on those nodes.

**📄 Example YAML:**

This ensures the pod is scheduled **only on nodes with gpu=true**.

**🔹 Node Anti-Affinity**

**Node anti-affinity** ensures that a pod is **not scheduled on nodes that match certain labels**, or **not co-located with other pods** with specific labels.

**✅ Example Use Case:**

You want to **spread replicas of a service across different nodes** to avoid a single point of failure.

**📄 Example YAML:**

This ensures that **pods with label app=my-app are not scheduled on the same node**.

**🔸 When to Use**

| **Scenario** | **Use** |
| --- | --- |
| Run workloads on specialized hardware (e.g., GPU) | **Node Affinity** |
| Isolate workloads for security or performance | **Node Affinity** |
| Spread replicas across nodes for high availability | **Pod Anti-Affinity** |
| Avoid placing similar workloads together | **Pod Anti-Affinity** |

10 Terraform Scenario Questions Every DevOps Engineer Should Know  
  
🚀 **What happens if your state file is accidentally deleted?**  
⏭️Answer: Terraform loses track of all managed infrastructure. On the next apply, it will attempt to  
recreate everything from scratch, potentially causing conflicts with existing resources.  
  
🚀 **What happens if multiple team members run terraform apply simultaneously?**  
⏭️Answer: State file locking fails, risking corrupted state and inconsistent infrastructure. One process  
succeeds while others error out, potentially leading to drift if not managed properly.  
  
🚀 **What happens if a resource fails halfway through a terraform apply?**⏭️Answer: Terraform leaves successfully created resources running but marks the state as tainted. Subsequent apply operations will attempt to recreate failed resources, but you're left in partial state.  
  
🚀 **What happens when AWS API rate limits are hit during a large terraform apply?**⏭️Answer: Operations fail with throttling errors. Terraform retries a few times then fails the apply. Resources created before the limit was hit remain, creating partial deployments.  
  
🚀 **What happens if terraform plan shows no changes but infrastructure was modifie**d outside Terraform? ⏭️Answer: Terraform won't detect the drift until you run terraform refresh or terraform plan -refresh-only. This can lead to unexpected behavior when making future changes.  
  
🚀 **What happens if you delete a resource definition from your configuration?**⏭️Answer: On next apply, Terraform will destroy that resource in your infrastructure unless you use  
terraform state rm to remove it from state first or use lifecycle { prevent\_destroy = true }.  
  
🚀 **What happens if a provider API changes between Terraform versions?**  
⏭️Answer: You may encounter compatibility issues and failed plans/applies. Resources might need to be rebuilt or configurations updated to match new API requirements.  
  
🚀 **What happens if you have circular dependencies in your Terraform modules?**  
⏭️Answer: Terraform will fail to initialize or plan with dependency cycle errors. You'll need to refactor your module structure to break the circular references.  
  
🚀 **What happens if you exceed AWS service quotas during deployment?**⏭️Answer: Resources will fail to create with quota exceeded errors. Terraform marks them as failed, and you'll need to request quota increases before retrying the apply  
  
🚀 **What happens if you lose access to the remote backend storing your state?**  
⏭️Answer: All Terraform operations fail until access is restored. Teams can't collaborate, and changes can't be applied safely. This effectively blocks all infrastructure changes.