KK Funda – Kubernetes Real-Time Interview Q&A (Q1–Q40)

# Q1: CrashLoopBackOff Issue in Our Microservices Project

* 🎙️ Interviewer asks:

In your project, did you face: crashloopbackoff issue in our microservices project? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

In our eCommerce project on AWS EC2 with Kubernetes, the payment-service kept restarting, affecting order flow.

⚠️ What Was the Issue:

Pod status: CrashLoopBackOff. Cause: app tried to connect to DB before it was ready.

📚 Kubernetes Concept Involved:

Pod lifecycle, readiness probe, CrashLoopBackOff

🛠️ Troubleshooting Steps:

* - kubectl get pods -n ecommerce
* - kubectl logs <pod-name> -n ecommerce
* - kubectl describe pod <pod-name> -n ecommerce
* - Updated readinessProbe initialDelaySeconds
* - kubectl apply -f updated-deployment.yaml

✅ Final Fix and Outcome:

I explained in the project that the issue was due to app starting faster than the DB. Added a readiness probe delay and redeployed. Resolved the crash issue.

----------------------------------------------------------------------------------------------------

# Q2: App Not Accessible via Domain Name in Our Staging Environment

* 🎙️ Interviewer asks:

In your project, did you face: app not accessible via domain name in our staging environment? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

In our Java 3-tier app on staging, app.staging.kkfunda.in was not resolving despite all pods and services running.

⚠️ What Was the Issue:

DNS was still pointing to old LoadBalancer IP.

📚 Kubernetes Concept Involved:

Ingress Controller, Ingress Resource, DNS Mapping

🛠️ Troubleshooting Steps:

* - kubectl get ingress -n staging
* - kubectl get svc ingress-nginx-controller -n ingress-nginx
* - nslookup app.staging.kkfunda.in
* - Updated A-record in GoDaddy
* - Verified using dig and browser

✅ Final Fix and Outcome:

In the project, I found the LoadBalancer IP changed after redeploy. I updated DNS to reflect new IP and verified DNS propagation. Site became reachable.

----------------------------------------------------------------------------------------------------

# Q3: Readiness Probe Blocking Traffic in Production

* 🎙️ Interviewer asks:

In your project, did you face: readiness probe blocking traffic in production? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

In our inventory-service deployment, users complained of 503 errors during updates despite pods running.

⚠️ What Was the Issue:

The app wasn’t ready, but traffic was being sent. Readiness probe was either misconfigured or absent.

📚 Kubernetes Concept Involved:

Readiness Probe, RollingUpdate Strategy

🛠️ Troubleshooting Steps:

* - kubectl describe pod inventory-service-\*
* - Checked readinessProbe config in deployment YAML
* - Added proper HTTP check with delay and timeout
* - kubectl rollout restart deployment inventory-service

✅ Final Fix and Outcome:

In my project, we hit a bug where traffic was routed to pods that were not yet ready. I configured a proper readiness probe and adjusted the rollout strategy to avoid downtime.

----------------------------------------------------------------------------------------------------

# Q4: Volume Not Mounted: Pod Crash with Read-only File System

* 🎙️ Interviewer asks:

In your project, did you face: volume not mounted: pod crash with read-only file system? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Logging app failed because the log directory was not writable.

⚠️ What Was the Issue:

PVC was not mounted; logs tried to write to a read-only layer.

📚 Kubernetes Concept Involved:

PersistentVolume, PVC, Volume Mounts

🛠️ Troubleshooting Steps:

* - kubectl get pvc -n logging
* - kubectl describe pod logging-agent-\*
* - Checked volumeMounts path and claimName
* - Fixed YAML to mount /logs correctly from PVC
* - kubectl apply -f logging-deployment.yaml

✅ Final Fix and Outcome:

In the logging setup of our cluster, the pod crashed due to lack of writable volume. I traced it back to missing volumeMounts and corrected the deployment YAML to use PVC properly.

----------------------------------------------------------------------------------------------------

# Q5: Pod Scheduling Fails Due to Taints

* 🎙️ Interviewer asks:

In your project, did you face: pod scheduling fails due to taints? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Kafka pods were not coming up on new nodes in a multi-node cluster setup.

⚠️ What Was the Issue:

Nodes had taints and pods had no tolerations.

📚 Kubernetes Concept Involved:

Taints and Tolerations

🛠️ Troubleshooting Steps:

* - kubectl describe node node-3 | grep -i taints
* - kubectl describe pod kafka-\* | grep Tolerations
* - Added toleration in StatefulSet YAML
* - kubectl apply -f kafka-statefulset.yaml

✅ Final Fix and Outcome:

While scaling Kafka, we added new nodes that were tainted for infra workloads. The Kafka pods failed to schedule. I fixed it by adding matching tolerations to the StatefulSet.

----------------------------------------------------------------------------------------------------

# Q6: Intermittent Container Exit Due to OOM

* 🎙️ Interviewer asks:

In your project, did you face: intermittent container exit due to oom? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Image processing app exited unexpectedly after high-res uploads.

⚠️ What Was the Issue:

App consumed more memory than allowed, resulting in OOMKilled.

📚 Kubernetes Concept Involved:

Resource Limits, Container Memory Management

🛠️ Troubleshooting Steps:

* - kubectl describe pod image-processor-\* | grep -i oom
* - Checked resources in deployment YAML
* - Increased memory limit to 1Gi
* - kubectl apply -f updated-deployment.yaml

✅ Final Fix and Outcome:

We saw container crashes in the image-processing pod. I diagnosed an OOMKilled error and increased memory limits in the deployment. App stabilized post that.

----------------------------------------------------------------------------------------------------

# Q7: Frontend Fails to Connect to Backend (Service Misconfiguration)

* 🎙️ Interviewer asks:

In your project, did you face: frontend fails to connect to backend (service misconfiguration)? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

React app couldn't call backend APIs, resulting in network errors in browser.

⚠️ What Was the Issue:

Backend service name or port was incorrect in frontend config.

📚 Kubernetes Concept Involved:

ClusterIP Services, DNS Resolution

🛠️ Troubleshooting Steps:

* - kubectl get svc -n app
* - Checked backend service name and port
* - Updated frontend configmap with correct name
* - kubectl apply -f frontend-config.yaml
* - kubectl rollout restart deployment frontend

✅ Final Fix and Outcome:

We had a mismatch between backend service name and what frontend was using. I updated the configmap, reloaded the frontend, and restored end-to-end connectivity.

----------------------------------------------------------------------------------------------------

# Q8: Deployment Fails With ImagePullBackOff

* 🎙️ Interviewer asks:

In your project, did you face: deployment fails with imagepullbackoff? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

A new service failed to deploy during CI pipeline.

⚠️ What Was the Issue:

Incorrect Docker image name or no credentials for private repo.

📚 Kubernetes Concept Involved:

ImagePullSecrets, Docker Hub/Auth

🛠️ Troubleshooting Steps:

* - kubectl describe pod <pod-name>
* - Checked image name and tag
* - Created secret: kubectl create secret docker-registry
* - Added imagePullSecrets to deployment
* - kubectl apply -f deployment.yaml

✅ Final Fix and Outcome:

Our CI pushed to a private Docker registry. The deployment failed due to missing imagePullSecret. I added it and redeployed, resolving the ImagePullBackOff error.

----------------------------------------------------------------------------------------------------

# Q9: Can’t Connect From Pod to External API

* 🎙️ Interviewer asks:

In your project, did you face: can’t connect from pod to external api? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

A microservice in the cluster failed to reach a third-party API.

⚠️ What Was the Issue:

No internet access from pod due to missing NAT gateway.

📚 Kubernetes Concept Involved:

AWS VPC, Outbound Internet Access, NAT Gateway

🛠️ Troubleshooting Steps:

* - kubectl exec -it <pod> -- curl https://api.external.com
* - Checked VPC route tables for private subnets
* - Deployed NAT gateway and updated route table
* - Tested outbound connectivity again

✅ Final Fix and Outcome:

In our AWS-hosted K8s cluster, the pod was in a private subnet with no NAT gateway. After attaching a NAT gateway and updating routes, the pod regained internet access.

----------------------------------------------------------------------------------------------------

# Q10: High CPU Usage: HPA Not Scaling

* 🎙️ Interviewer asks:

In your project, did you face: high cpu usage: hpa not scaling? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Under load test, app pods hit 100% CPU but didn’t scale.

⚠️ What Was the Issue:

HPA wasn’t triggering due to missing metrics server.

📚 Kubernetes Concept Involved:

Horizontal Pod Autoscaler, Metrics Server

🛠️ Troubleshooting Steps:

* - kubectl top pod
* - kubectl get hpa
* - Installed metrics server
* - kubectl autoscale deployment webapp --cpu-percent=70 --min=2 --max=10

✅ Final Fix and Outcome:

HPA wasn’t working in our test cluster because metrics server was missing. I installed it, recreated the HPA, and verified that pods auto-scaled under CPU pressure.

----------------------------------------------------------------------------------------------------

# Q11: Pods Stuck in Pending State Due to PVC Issues

* 🎙️ Interviewer asks:

In your project, did you face: pods stuck in pending state due to pvc issues? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

In our file-upload service, new pods were not starting. They stayed in 'Pending' state for over 15 minutes.

⚠️ What Was the Issue:

The PersistentVolumeClaim (PVC) was not bound to a PersistentVolume (PV).

📚 Kubernetes Concept Involved:

PVC, PV Binding, StorageClass

🛠️ Troubleshooting Steps:

* - kubectl get pods -n upload-ns
* - kubectl describe pod <pod-name> -n upload-ns
* - kubectl get pvc -n upload-ns
* - kubectl describe pvc <pvc-name> -n upload-ns
* - Ensured matching StorageClass and volume availability

✅ Final Fix and Outcome:

In our Kubernetes cluster, a pod was stuck because the PVC couldn't find a matching PV. I described the PVC to identify the issue, added a matching StorageClass, and the pod started after the volume was bound.

----------------------------------------------------------------------------------------------------

# Q12: Service to Service DNS Resolution Failing in Another Namespace

* 🎙️ Interviewer asks:

In your project, did you face: service to service dns resolution failing in another namespace? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Our app in the `dev` namespace couldn't access the DB service in the `db` namespace.

⚠️ What Was the Issue:

Service name was used without full DNS path.

📚 Kubernetes Concept Involved:

Kubernetes DNS, Cross-Namespace Communication

🛠️ Troubleshooting Steps:

* - Checked logs in application pod
* - Verified service name in DB config
* - Used full DNS: db-service.db.svc.cluster.local
* - kubectl exec into pod to test DNS resolution

✅ Final Fix and Outcome:

This was a classic namespace miscommunication. I corrected the DB host to its full DNS name and tested with curl inside the pod. DNS resolved and connectivity worked.

----------------------------------------------------------------------------------------------------

# Q13: Deployment Update Rolled Out Without Triggering New Pods

* 🎙️ Interviewer asks:

In your project, did you face: deployment update rolled out without triggering new pods? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

We updated environment variables in deployment, but pods didn’t restart.

⚠️ What Was the Issue:

Kubernetes doesn’t auto-restart pods on config change unless a rollout is triggered.

📚 Kubernetes Concept Involved:

kubectl rollout, Deployment Strategy

🛠️ Troubleshooting Steps:

* - kubectl set env deployment webapp UPDATED\_AT=$(date +%s)
* - kubectl rollout restart deployment webapp
* - kubectl rollout status deployment webapp

✅ Final Fix and Outcome:

In our project, config changes didn’t reflect because pods were not restarted. I used the `rollout restart` to force apply and verify the updated environment variables.

----------------------------------------------------------------------------------------------------

# Q14: Failed to Mount Secret into Pod

* 🎙️ Interviewer asks:

In your project, did you face: failed to mount secret into pod? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Payments app couldn't read DB credentials stored in a secret.

⚠️ What Was the Issue:

Secret was not mounted or environment variable was misspelled.

📚 Kubernetes Concept Involved:

Kubernetes Secrets, Pod Environment

🛠️ Troubleshooting Steps:

* - kubectl get secret db-secret -n payments
* - kubectl describe pod <pod-name>
* - Checked envFrom and secretKeyRef in deployment YAML
* - Fixed the secret key name and redeployed

✅ Final Fix and Outcome:

In our real-time app, the secret key used in envFrom was wrong. After correcting it and reapplying the deployment, the pod could access DB credentials successfully.

----------------------------------------------------------------------------------------------------

# Q15: ResourceQuota Preventing Pod Creation

* 🎙️ Interviewer asks:

In your project, did you face: resourcequota preventing pod creation? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

QA team couldn’t create pods in their namespace.

⚠️ What Was the Issue:

Namespace exceeded its assigned CPU limit via ResourceQuota.

📚 Kubernetes Concept Involved:

ResourceQuota, Namespace Limits

🛠️ Troubleshooting Steps:

* - kubectl get resourcequota -n qa
* - kubectl describe resourcequota compute-quota -n qa
* - kubectl describe pod <pod-name>
* - Adjusted limits or deleted old pods to free up quota

✅ Final Fix and Outcome:

During load testing, the QA namespace hit CPU limits. I inspected the ResourceQuota and freed up unused pods to allow new ones to be scheduled.

----------------------------------------------------------------------------------------------------

# Q16: App Breaks When ConfigMap is Updated

* 🎙️ Interviewer asks:

In your project, did you face: app breaks when configmap is updated? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

App started throwing errors after config update. It didn’t reflect until pod restart.

⚠️ What Was the Issue:

Mounted ConfigMap volumes do not trigger auto-reload in app containers.

📚 Kubernetes Concept Involved:

ConfigMap, Volume Mount, Application Reload

🛠️ Troubleshooting Steps:

* - kubectl edit configmap app-config
* - kubectl rollout restart deployment app
* - Checked logs post rollout

✅ Final Fix and Outcome:

I learned that apps don’t auto-reload mounted config changes. I restarted the deployment using rollout to apply the updated config live.

----------------------------------------------------------------------------------------------------

# Q17: CrashLoop When Application Container Lacks Execution Permission

* 🎙️ Interviewer asks:

In your project, did you face: crashloop when application container lacks execution permission? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

A Python service crashed with 'Permission denied' on startup.

⚠️ What Was the Issue:

Entrypoint script had no execution rights.

📚 Kubernetes Concept Involved:

Dockerfile, ENTRYPOINT, File Permissions

🛠️ Troubleshooting Steps:

* - kubectl logs <pod>
* - Dockerfile updated with `RUN chmod +x entrypoint.sh`
* - Rebuilt image and pushed to registry
* - kubectl rollout restart deployment

✅ Final Fix and Outcome:

In our Docker image, the script was copied but not made executable. I fixed the Dockerfile with chmod, rebuilt, and deployed it. That resolved the CrashLoop.

----------------------------------------------------------------------------------------------------

# Q18: Helm Install Fails Due to Missing Value in values.yaml

* 🎙️ Interviewer asks:

In your project, did you face: helm install fails due to missing value in values.yaml? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Helm install for the backend-service failed with template error.

⚠️ What Was the Issue:

`.Values.ingress.enabled` was not defined in values.yaml.

📚 Kubernetes Concept Involved:

Helm Templating, Default Values

🛠️ Troubleshooting Steps:

* - helm install --dry-run --debug ./backend-chart
* - Added missing ingress block in values.yaml
* - Set default values in templates using `| default`

✅ Final Fix and Outcome:

This issue taught me the importance of template validation. I used dry-run debug, added safe defaults in Helm templates, and installed successfully after fixing values.yaml.

----------------------------------------------------------------------------------------------------

# Q19: Pods Scheduling Unevenly Across Nodes

* 🎙️ Interviewer asks:

In your project, did you face: pods scheduling unevenly across nodes? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

All pods were running on a single node while others were idle.

⚠️ What Was the Issue:

No spread constraints or affinity rules defined.

📚 Kubernetes Concept Involved:

Topology Spread Constraints

🛠️ Troubleshooting Steps:

* - kubectl get pods -o wide
* - Checked node labels and topology constraints
* - Added topologySpreadConstraints to deployment

✅ Final Fix and Outcome:

This impacted our performance. I applied spread constraints to ensure pods were balanced across nodes and not congested on one. It improved fault-tolerance and resource use.

----------------------------------------------------------------------------------------------------

# Q20: App Logs Not Available After Pod Crashes

* 🎙️ Interviewer asks:

In your project, did you face: app logs not available after pod crashes? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

A Java pod crashed and logs were missing after it restarted.

⚠️ What Was the Issue:

Logs were written to container and not persisted.

📚 Kubernetes Concept Involved:

Volume Mounts for Logs, Persistent Logging

🛠️ Troubleshooting Steps:

* - Checked `kubectl logs <pod>`
* - Verified logging directory was not backed by volume
* - Updated pod spec to mount `/var/logs` to PVC
* - Rechecked logs post restart

✅ Final Fix and Outcome:

We lost crash data because logs weren’t persisted. I added a PVC to `/var/logs` and ensured log rotation. This helped us debug future crashes properly.

----------------------------------------------------------------------------------------------------

# Q21: NodePort Service Not Accessible Externally

* 🎙️ Interviewer asks:

In your project, did you face: nodeport service not accessible externally? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

We deployed a Node.js app and exposed it via a NodePort service, but it wasn’t accessible from the browser.

⚠️ What Was the Issue:

Security group or firewall didn't allow traffic on the NodePort.

📚 Kubernetes Concept Involved:

NodePort, Service Exposure, Networking

🛠️ Troubleshooting Steps:

* - kubectl get svc -n web
* - Checked the nodePort (e.g., 32001)
* - Verified node external IP via `kubectl get nodes -o wide`
* - Checked firewall/security group on cloud (AWS/GCP)
* - Allowed access to port 32001 in the security group

✅ Final Fix and Outcome:

In our AWS setup, the EC2 instances had restricted ports. I added the NodePort to the security group and verified access via browser. The app became reachable.

----------------------------------------------------------------------------------------------------

# Q22: Static Website in NGINX Shows 404

* 🎙️ Interviewer asks:

In your project, did you face: static website in nginx shows 404? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Deployed a React app using NGINX container but it only returned 404 errors on page load.

⚠️ What Was the Issue:

Files were not mounted correctly to NGINX root directory.

📚 Kubernetes Concept Involved:

Volume Mount, Static Content in NGINX

🛠️ Troubleshooting Steps:

* - kubectl logs <nginx-pod>
* - Checked volumeMounts path in pod spec
* - Mapped content to `/usr/share/nginx/html`
* - Rebuilt container with corrected COPY command in Dockerfile

✅ Final Fix and Outcome:

We mistakenly mounted content to the wrong path. Once corrected to match NGINX root, the React app started working correctly in the browser.

----------------------------------------------------------------------------------------------------

# Q23: Readiness Probe Misconfigured – Causing Downtime

* 🎙️ Interviewer asks:

In your project, did you face: readiness probe misconfigured – causing downtime? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Pods were marked ‘Ready’ before app was fully initialized, resulting in failed requests.

⚠️ What Was the Issue:

Missing or incorrect readiness probe.

📚 Kubernetes Concept Involved:

Readiness Probe, Deployment Strategy

🛠️ Troubleshooting Steps:

* - kubectl describe pod <pod-name>
* - Readiness probe was missing
* - Added HTTP check on `/health` with delay
* - kubectl rollout restart deployment <name>

✅ Final Fix and Outcome:

Traffic was routed too early. Adding a readiness probe with a proper delay fixed the issue and avoided further downtime during rolling updates.

----------------------------------------------------------------------------------------------------

# Q24: Different Configs for Different Environments

* 🎙️ Interviewer asks:

In your project, did you face: different configs for different environments? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Needed different DB URLs for dev, QA, and prod environments using the same base code.

⚠️ What Was the Issue:

No config separation strategy in place.

📚 Kubernetes Concept Involved:

ConfigMap per Namespace or Kustomize

🛠️ Troubleshooting Steps:

* - Created separate ConfigMaps in each namespace
* - Overlaid base YAML using Kustomize for each environment
* - Mounted correct ConfigMap to deployment

✅ Final Fix and Outcome:

To isolate environments, I used Kustomize overlays and namespace-specific ConfigMaps. This helped us avoid manual changes and kept pipelines clean.

----------------------------------------------------------------------------------------------------

# Q25: CronJob Failing Without Any Output

* 🎙️ Interviewer asks:

In your project, did you face: cronjob failing without any output? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Daily backup CronJob didn’t produce logs or results, but showed as completed.

⚠️ What Was the Issue:

Job container completed too quickly or no command was defined.

📚 Kubernetes Concept Involved:

Kubernetes CronJob, Logging

🛠️ Troubleshooting Steps:

* - kubectl get cronjob
* - kubectl get job --watch
* - kubectl logs job/<name>
* - Verified `command` and `args` in YAML

✅ Final Fix and Outcome:

The container exited without doing anything due to missing command. I added `command: ["sh", "-c", "backup.sh"]` and validated logs after rerun.

----------------------------------------------------------------------------------------------------

# Q26: Developer Requires Limited Access to Logs Only

* 🎙️ Interviewer asks:

In your project, did you face: developer requires limited access to logs only? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Dev team needed access to pod logs without allowing delete or update rights.

⚠️ What Was the Issue:

RBAC was too broad or non-existent.

📚 Kubernetes Concept Involved:

RBAC, Role, RoleBinding

🛠️ Troubleshooting Steps:

* - kubectl create role pod-log-reader --verb=get,list --resource=pods/log
* - kubectl create rolebinding dev-readonly --role=pod-log-reader --user=devuser

✅ Final Fix and Outcome:

I created a namespace-scoped RBAC role for `pods/log` and bound it to dev users. This followed least privilege principle and passed our audit.

----------------------------------------------------------------------------------------------------

# Q27: Helm Chart Install Fails – nil pointer error

* 🎙️ Interviewer asks:

In your project, did you face: helm chart install fails – nil pointer error? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Helm install failed due to missing value in `.Values.ingress.enabled`

⚠️ What Was the Issue:

Missing or undefined key in values.yaml

📚 Kubernetes Concept Involved:

Helm Templating, Default Values

🛠️ Troubleshooting Steps:

* - helm install --dry-run --debug ./mychart
* - Defined `ingress.enabled: false` in values.yaml
* - Used default function in template: `| default false`

✅ Final Fix and Outcome:

This error was due to templating logic without default values. Adding safe defaults and validating via dry-run fixed the deployment.

----------------------------------------------------------------------------------------------------

# Q28: Pod on a Node is Unreachable by Other Services

* 🎙️ Interviewer asks:

In your project, did you face: pod on a node is unreachable by other services? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

One node had services that were not reachable within the cluster.

⚠️ What Was the Issue:

CNI plugin was misconfigured or kube-proxy failed.

📚 Kubernetes Concept Involved:

Kubernetes Networking, CNI

🛠️ Troubleshooting Steps:

* - kubectl get pods -o wide
* - kubectl logs kube-proxy -n kube-system
* - Reinstalled CNI plugin (e.g., Calico)
* - Tested with `ping` and `curl` between pods

✅ Final Fix and Outcome:

I traced the issue to kube-proxy and Calico config. After reinstalling the CNI and restarting the node, service connectivity was restored.

----------------------------------------------------------------------------------------------------

# Q29: Slow Startup Causes Readiness Probe to Fail

* 🎙️ Interviewer asks:

In your project, did you face: slow startup causes readiness probe to fail? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Java apps with large cache took 2 mins to load; probe failed in 30s.

⚠️ What Was the Issue:

Initial delay was too short for readiness probe.

📚 Kubernetes Concept Involved:

Readiness Probe, Application Warm-Up

🛠️ Troubleshooting Steps:

* - kubectl describe pod <pod>
* - increased `initialDelaySeconds` from 30 to 120
* - monitored logs and readiness transition

✅ Final Fix and Outcome:

I tuned the probe timing to allow startup warm-up. Post update, pods stayed stable and weren’t marked as unready unnecessarily.

----------------------------------------------------------------------------------------------------

# Q30: Backup YAML for Namespace Not Available

* 🎙️ Interviewer asks:

In your project, did you face: backup yaml for namespace not available? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Lost deployment and had no git history for some resources.

⚠️ What Was the Issue:

No backup automation was in place.

📚 Kubernetes Concept Involved:

kubectl export, GitOps Practice

🛠️ Troubleshooting Steps:

* - kubectl get all -n mynamespace -o yaml > backup.yaml
* - Scheduled nightly export with cronjob
* - Added GitOps using ArgoCD later

✅ Final Fix and Outcome:

We lacked YAML backups. I implemented nightly `kubectl get` exports and integrated GitOps to auto-sync K8s state from source.

----------------------------------------------------------------------------------------------------

# Q31: Pod Logs Lost After Crash – Logging Not Persisted

* 🎙️ Interviewer asks:

In your project, did you face: pod logs lost after crash – logging not persisted? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

We faced issues debugging a pod crash because the logs were lost immediately after the container restarted.

⚠️ What Was the Issue:

Logs were written inside the container without persistent storage or central logging.

📚 Kubernetes Concept Involved:

Log Persistence, Volume Mount, Sidecar Logging

🛠️ Troubleshooting Steps:

* - kubectl logs <pod-name> --previous
* - Checked logging configuration and volume mounts
* - Configured pod to mount /var/log to PVC
* - Set up Fluentd to forward logs to Elasticsearch

✅ Final Fix and Outcome:

I added volume mounts to persist logs across pod restarts and configured Fluentd for log forwarding. This ensured logs are retained for crash analysis.

----------------------------------------------------------------------------------------------------

# Q32: Service Cannot Reach External API – No Outbound Internet

* 🎙️ Interviewer asks:

In your project, did you face: service cannot reach external api – no outbound internet? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Our backend service failed to call a third-party API; DNS resolution worked but the connection was refused.

⚠️ What Was the Issue:

Private subnet without NAT Gateway, hence no internet access.

📚 Kubernetes Concept Involved:

AWS NAT Gateway, EKS Networking

🛠️ Troubleshooting Steps:

* - kubectl exec <pod> -- curl https://api.external.com
* - Checked VPC route tables
* - Attached NAT Gateway to private subnet route table
* - Tested with curl again

✅ Final Fix and Outcome:

In EKS, our private subnets needed a NAT gateway. I updated the route table and verified successful access to external APIs from within the pod.

----------------------------------------------------------------------------------------------------

# Q33: App Exposed via LoadBalancer Fails Health Checks

* 🎙️ Interviewer asks:

In your project, did you face: app exposed via loadbalancer fails health checks? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Service type LoadBalancer was created, but the external IP stayed in pending.

⚠️ What Was the Issue:

Cloud controller failed to provision due to missing health checks or unsupported region.

📚 Kubernetes Concept Involved:

LoadBalancer Service, Readiness Probe, Cloud Integration

🛠️ Troubleshooting Steps:

* - kubectl get svc
* - Checked cloud console for load balancer
* - Added proper readiness and liveness probes
* - Verified cloud controller logs

✅ Final Fix and Outcome:

Our readiness probe was missing, which caused health check failure in AWS ALB. Adding probes resolved the issue and LoadBalancer got provisioned.

----------------------------------------------------------------------------------------------------

# Q34: Secret Injection Fails – App Shows Null for Password

* 🎙️ Interviewer asks:

In your project, did you face: secret injection fails – app shows null for password? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

The Spring Boot app showed null when trying to read DB\_PASSWORD from env.

⚠️ What Was the Issue:

Wrong secret key or syntax in deployment YAML.

📚 Kubernetes Concept Involved:

Kubernetes Secrets, envFrom and secretKeyRef

🛠️ Troubleshooting Steps:

* - kubectl get secret -n app
* - kubectl describe pod <pod>
* - Fixed typo in secretKeyRef
* - kubectl rollout restart deployment

✅ Final Fix and Outcome:

The secret key had a typo (`DB\_PASSWD` vs `DB\_PASSWORD`). I corrected the key reference in the deployment and restarted the app, resolving the issue.

----------------------------------------------------------------------------------------------------

# Q35: Application Restarted During Load Testing

* 🎙️ Interviewer asks:

In your project, did you face: application restarted during load testing? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

App crashed during stress test, affecting user sessions.

⚠️ What Was the Issue:

Resource limits were too low for actual workload.

📚 Kubernetes Concept Involved:

Resource Requests & Limits, HPA

🛠️ Troubleshooting Steps:

* - kubectl describe pod <pod>
* - Observed `OOMKilled` reason
* - Increased memory and CPU limits
* - Implemented HPA for autoscaling

✅ Final Fix and Outcome:

We underestimated load impact. I increased limits and added HPA. Post that, pods scaled properly and survived load tests.

----------------------------------------------------------------------------------------------------

# Q36: ConfigMap Changes Not Reflected Until Manual Restart

* 🎙️ Interviewer asks:

In your project, did you face: configmap changes not reflected until manual restart? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

We updated a config file in ConfigMap but the application didn’t pick it up.

⚠️ What Was the Issue:

ConfigMap was mounted as volume and app didn’t auto-reload.

📚 Kubernetes Concept Involved:

ConfigMap, Volume Mount, Rollout Strategy

🛠️ Troubleshooting Steps:

* - kubectl edit configmap <name>
* - kubectl rollout restart deployment <app>
* - Confirmed new config loaded via logs

✅ Final Fix and Outcome:

Since the app didn’t watch the file system, I restarted the pods post update. Also added an init container to validate config presence on startup.

----------------------------------------------------------------------------------------------------

# Q37: Helm Upgrade Overwrites Existing Values

* 🎙️ Interviewer asks:

In your project, did you face: helm upgrade overwrites existing values? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Helm upgrade removed some user-defined settings during a chart update.

⚠️ What Was the Issue:

Helm was not using values.yaml or --reuse-values flag.

📚 Kubernetes Concept Involved:

Helm Upgrade, values.yaml

🛠️ Troubleshooting Steps:

* - Compared previous vs current values.yaml
* - Used `helm upgrade --reuse-values`
* - Stored critical values in separate override file

✅ Final Fix and Outcome:

This was a mistake in CI/CD. I enforced use of `--reuse-values` in automation and stored override values explicitly for every release.

----------------------------------------------------------------------------------------------------

# Q38: Pods Scheduled Only on Specific Nodes

* 🎙️ Interviewer asks:

In your project, did you face: pods scheduled only on specific nodes? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Wanted logging agents only on infra nodes labeled for monitoring.

⚠️ What Was the Issue:

DaemonSet lacked node affinity rules.

📚 Kubernetes Concept Involved:

Node Affinity, DaemonSet

🛠️ Troubleshooting Steps:

* - kubectl label nodes node1 role=infra
* - Added nodeAffinity to DaemonSet
* - kubectl apply -f logging-agent.yaml

✅ Final Fix and Outcome:

To isolate log agents, I used node labels and affinity rules in DaemonSet spec. Pods now deploy only on infra nodes.

----------------------------------------------------------------------------------------------------

# Q39: No Resource Enforcement in Namespace

* 🎙️ Interviewer asks:

In your project, did you face: no resource enforcement in namespace? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Developers deployed high-CPU pods and choked the cluster.

⚠️ What Was the Issue:

No LimitRange or ResourceQuota set for namespace.

📚 Kubernetes Concept Involved:

LimitRange, ResourceQuota

🛠️ Troubleshooting Steps:

* - kubectl get limitrange -n dev
* - kubectl get resourcequota -n dev
* - Created LimitRange with default limits
* - Created ResourceQuota for CPU and memory

✅ Final Fix and Outcome:

I implemented LimitRange and ResourceQuota policies in each dev namespace to enforce fair usage and prevent resource starvation.

----------------------------------------------------------------------------------------------------

# Q40: Pods in Stuck Terminating State

* 🎙️ Interviewer asks:

In your project, did you face: pods in stuck terminating state? What caused the issue, and how did you resolve it?

💬 Your Answer (Project Experience):

Some pods stayed in 'Terminating' for a long time after scale-down.

⚠️ What Was the Issue:

Finalizers not cleaning up or volume unmount delays.

📚 Kubernetes Concept Involved:

Pod Lifecycle, Finalizers

🛠️ Troubleshooting Steps:

* - kubectl get pod <name> -o yaml | grep finalizers
* - kubectl patch pod <name> -p '{"metadata":{"finalizers":[]}}'
* - Checked volume mount points

✅ Final Fix and Outcome:

Finalizer cleanup logic failed in a custom controller. I patched the pod manually and also fixed controller logic for future cleanup handling.

----------------------------------------------------------------------------------------------------