



200 Kubernetes Scenario Based Q&A

By DevOps Shack



Q1: Pod evicted — reason NodeHasDiskPressure. Fix?**Root Cause:**

Node ran out of disk space; kubelet triggered eviction.

Fix:

- SSH into node → check df -h
- Run: docker system prune -a or clean /var/log, /tmp
- Delete unused pods
- Node auto-recovers

Impact:

Pods crash, app downtime

Best Practice:

Set alerts at 70% disk usage, separate volume for /var/lib/containerd

Q2: Pod Pending — “insufficient memory”. What to do?**Root Cause:**

Pod's memory request > any node's free memory

Fix:

- Reduce request
- Scale nodes
- Enable Cluster Autoscaler

Impact:

Deployment blocked

Best Practice:

Use historical usage to right-size resources

Q3: Pod ImagePullBackOff. Reason and fix?

Root Cause:

Image not found, wrong tag, or no registry auth

Fix:

- Check image/tag
- If private, add imagePullSecret
- Restart pod

Impact:

Pod won't start

Best Practice:

Always test docker pull before deploying

Q4: ConfigMap updated but pod uses old config — why?**Root Cause:**

Pods load ConfigMaps at startup only

Fix:

- Run: `kubectl rollout restart deployment <name>`

Impact:

App uses stale config

Best Practice:

Use checksum annotations for auto-rollout

Q5: Pod stuck in Terminating. Solution?**Root Cause:**

SIGTERM handler issue or node unreachable

Fix:

```
kubectl delete pod <name> --grace-period=0 --force
```

Impact:

Deployment hangs

Best Practice:

Handle SIGTERM in app, use terminationGracePeriodSeconds

Q6: Ingress returns 404 — fix?**Root Cause:**

Path mismatch or missing service endpoints

Fix:

- Check Ingress paths
- Verify backend svc + endpoints
- Check rewrite annotations

Impact:

App inaccessible via domain

Best Practice:

Validate with curl, monitor ingress logs

Q7: Pod stuck in ContainerCreating. PVC attached. Why?**Root Cause:**

PVC can't bind or mount volume

Fix:

- Check PVC: kubectl describe pvc
- Check node logs for mount issues

Impact:

App doesn't start

Best Practice:

Use dynamic storage class, test PVCs in staging

Shall I continue with **Q8–Q20** in this refined format?

(Short front-facing questions + deep, crisp answers)

You said:

OK CONTINUE

ChatGPT said:

Excellent. Continuing with the next batch in your approved format:

Q8: Node shows NotReady. What happened and fix?

Root Cause:

Kubelet crashed, node unreachable, or disk/network failure.

Fix:

- SSH → restart kubelet:

```
sudo systemctl restart kubelet
```

- If unresponsive → replace node via autoscaler

Impact:

Pods rescheduled; risk of downtime if not replicated

Best Practice:

Enable node auto-repair or monitoring for heartbeats

Q9: DaemonSet not running on all nodes. Why?

Root Cause:

Node taints or missing tolerations/labels

Fix:

- Add tolerations:

```
- key: "node-role.kubernetes.io/master"
```

```
operator: "Exists"
```

```
effect: "NoSchedule"
```

Impact:

Monitoring/log agents may miss nodes

Best Practice:

Always test DaemonSet on tainted nodes in dev

Q10: Pod stuck in CrashLoopBackOff. What to check?

Root Cause:

App exits with error → Kubernetes restarts it repeatedly

Fix:

- Run kubectl logs <pod>
- Fix env vars, configs, or startup scripts

Impact:

Service never starts

Best Practice:

Use readiness probes + log startup exceptions

Q11: Readiness probe failing. Traffic not routed. Fix?**Root Cause:**

Probe path/port is wrong or app isn't ready yet

Fix:

- Match readiness probe with actual app health endpoint
- Increase initialDelaySeconds if app is slow to boot

Impact:

Pod marked unready → removed from service

Best Practice:

Use stable /health endpoint; test locally first

Q12: Service not routing traffic to pods. Why?**Root Cause:**

Pods are not in "Ready" state → endpoints list is empty

Fix:

- Check readiness probes
- Run: kubectl get endpoints <svc>

Impact:

App unreachable despite healthy pods

Best Practice:

Always confirm endpoints during deployment

Q13: Horizontal Pod Autoscaler not scaling. Why?

Root Cause:

No CPU/memory requests set, or metrics-server missing

Fix:

- Set resources.requests.cpu in pod spec
- Install metrics-server

kubectl top pods

Impact:

Pods never scale → performance issues

Best Practice:

Monitor with HPA + Prometheus for insight

Q14: Logs not showing with kubectl logs. Cause?

Root Cause:

Container may be crash-looping too fast or logs rotated

Fix:

`kubectl logs <pod> --previous`

Impact:

No visibility → blind debugging

Best Practice:

Export logs to ELK/Fluentd stack or Loki

Q15: Service exposed, but DNS name not resolving. Why?

Root Cause:

CoreDNS not working or config issue

Fix:

- Check CoreDNS pods:

`kubectl get pods -n kube-system | grep coredns`

- Restart if needed, check kubectl logs



Impact:

Internal services can't talk to each other

Best Practice:

Use headless services for stateful sets; monitor CoreDNS health

Q16: ClusterIP service accessible from inside, but not outside. Why?**Root Cause:**

ClusterIP is internal-only — no external access

Fix:

- Use NodePort or LoadBalancer type
- Or expose via Ingress

Impact:

External tools can't reach the service

Best Practice:

Use Ingress + cert-manager for proper external routing

Q17: Can't exec into pod — permission denied. Why?**Root Cause:**

RBAC restricts kubectl exec

Fix:

- Check role bindings:
`kubectl auth can-i exec --as=<user>`
- Grant appropriate permissions

Impact:

Debugging blocked for restricted users

Best Practice:

Create limited debug roles, not cluster-admin

Q18: Application restart causes long downtime. Why?**Root Cause:**

No readinessProbe → traffic hits pod before it's ready



Fix:

- Add readiness probe
- Configure rolling update strategy:

strategy:`type: RollingUpdate``rollingUpdate:``maxUnavailable: 0`**Impact:**

App crashes under load during deploy

Best Practice:

Use readinessProbe + zero-downtime rollout config

Q19: Deployment rollout stuck. Why?**Root Cause:**

New pods fail to become ready → deployment paused

Fix:

- Check kubectl rollout status deployment <name>
- Fix underlying pod issue

Impact:

CI/CD blocked, app update fails

Best Practice:

Set maxUnavailable=0 for safer rollouts

Q20: PVC not attaching to pod. What's wrong?**Root Cause:**

No suitable PV, or accessMode mismatch

Fix:

- Check PVC and PV binding:

`kubectl describe pvc <name>`

- Ensure StorageClass and accessModes match

Impact:

App fails to start with volume error

Best Practice:

Test StorageClass with dummy pod before production use

Q21: Pod IP keeps changing — how to access it reliably?**Root Cause:**

Pod IPs are ephemeral; recreated pods get new IPs.

Fix:

- Use a **Service** (ClusterIP) to provide stable access to pods.

Impact:

Direct pod IP access breaks on restart.

Best Practice:

Never hardcode pod IPs; always use Services or DNS.

Q22: Deployment rollback required — how to do it?**Root Cause:**

New deployment version is faulty.

Fix:

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

Impact:

Outage or broken feature live in production.

Best Practice:

Use health checks + canary strategy for safer updates.

Q23: HPA doesn't scale up on high memory. Why?**Root Cause:**

HPA scales on CPU by default, not memory.

Fix:

-
- Install custom-metrics-apiserver or Prometheus Adapter
 - Define custom HPA on memory

Impact:

App runs out of memory but doesn't scale.

Best Practice:

Use Prometheus-based autoscaling for more control.

Q24: Service routing traffic to pod on another node — is that okay?**Root Cause:**

Yes. Kube-proxy and CNI route traffic across nodes.

Fix:

No action needed unless locality is a concern.

Impact:

Slight latency; cross-node traffic may hit performance.

Best Practice:

Use topologyKeys or sessionAffinity if needed.

Q25: Need blue-green deployment — how to do it in K8s?**Root Cause:**

Canary/rolling not suitable for your use case.

Fix:

- Deploy two versions with different labels
- Switch Service selector from blue → green

Impact:

User sees partial rollout if not managed carefully.

Best Practice:

Automate with Argo Rollouts or Flagger.

Q26: Node's /var/lib/kubelet is full. Fix?

Root Cause:

Pod logs, container layers, and orphaned volumes pile up.

Fix:

- Prune unused images
- Clean orphaned volumes
- Reboot if required

Impact:

New pods can't start; node enters DiskPressure.

Best Practice:

Use log rotation, separate volume for /var/lib.

Q27: Pods can't reach external internet. Why?

Root Cause:

Node has no internet access or firewall blocks egress.

Fix:

- Check NAT rules
- Validate node route tables
- Add NAT Gateway if in private subnet (AWS)

Impact:

Image pull failures, app errors

Best Practice:

Use egress policies + NAT for outbound traffic

Q28: Service type LoadBalancer not working. Why?

Root Cause:

Cloud controller not provisioned or misconfigured

Fix:

-
- On cloud, ensure LB controller is installed (e.g., AWS ALB, GCP LB)
 - For bare metal: use MetallLB or ingress

Impact:

External traffic can't reach cluster

Best Practice:

Prefer Ingress + LoadBalancer for production

Q29: Cluster has idle resources, but pods are pending. Why?

Root Cause:

Resource **requests** don't match node capacity (e.g., CPU fragmentation)

Fix:

- Adjust pod requests
- Enable binpacking-aware scheduler or reconfigure limits

Impact:

False negative of “resource exhaustion”

Best Practice:

Use VerticalPodAutoscaler or right-size apps with Prometheus

Q30: You want pod to always run on a specific node. How?

Root Cause:

Need node affinity or toleration

Fix:

nodeSelector:

disktype: ssd

Or use affinity: or nodeName: <name>

Impact:

Wrong node may cause performance hit

Best Practice:

Use affinity + labels for better control

Q31: Pods randomly get killed. Why?**Root Cause:**

OOMKilled or CPU throttling due to no limits

Fix:

- Set resource limits:

resources:

limits:

memory: "512Mi"

cpu: "500m"

Impact:

Crashes, degraded service

Best Practice:

Use monitoring to tune limits

Q32: How to drain a node safely?**Root Cause:**

Maintenance or upgrade required

Fix:

kubectl drain <node> --ignore-daemonsets --delete-emptydir-data

Impact:

Pods rescheduled elsewhere

Best Practice:

Cordon first, monitor workloads

Q33: Need to stop scheduling on a node. How?**Fix:**

kubectl cordon <node>

Impact:

New pods won't be placed there

Best Practice:

Use for controlled migration or draining

Q34: Pod accesses secret as env variable — safe?**Root Cause:**

Env variables can be exposed via logs or env command

Fix:

Use volumeMount-based secrets instead

Impact:

Risk of leaking credentials

Best Practice:

Mount secrets as read-only files

Q35: Secret changed, but app didn't reload. Why?**Root Cause:**

Secrets aren't dynamically reloaded in pods

Fix:

Trigger rolling restart:

```
kubectl rollout restart deployment <name>
```

Impact:

App still using old values

Best Practice:

Use external-secrets or hot reload logic

Q36: You need to run a cron job in K8s. How?**Fix:**

```
kind: CronJob
```

```
schedule: "*/5 * * * *"
```

Impact:

Missed or failed jobs if misconfigured

Best Practice:

Set successfulJobsHistoryLimit, monitor status

Q37: How to limit pod CPU to avoid node starvation?**Fix:**

resources:

limits:

cpu: "500m"

Impact:

App throttled but won't crash

Best Practice:

Always define both requests and limits

Q38: How to detect which pod is using most resources?**Fix:**

kubectl top pod --all-namespaces

Impact:

Overuse can crash node

Best Practice:

Use Prometheus + Grafana for historical trends

Q39: Pod restarts frequently — how to debug?**Fix:**

kubectl logs <pod> --previous

kubectl describe pod <pod>

Check readiness, probes, crash errors

Impact:

Service flaps, SREs paged

Best Practice:

Alert on high restart count

Q40: How to force rollout of unchanged deployment?**Fix:**

```
kubectl rollout restart deployment <name>
```

Impact:

Used to refresh config or secrets

Best Practice:

Use GitOps annotations or checksum triggers

Q41: Pod cannot access Service — what to check?**Root Cause:**

Pod DNS config, service selector mismatch, or endpoints missing.

Fix:

- Check `kubectl get svc <name>`, validate selector.
- Check endpoints:

```
kubectl get endpoints <svc-name>
```

Impact:

Service unreachable inside cluster.

Best Practice:

Always validate pod labels and service selectors match.

Q42: Pods stuck in ContainerCreating due to image pull. Fix?**Root Cause:**

Image pull delay or network issue.

Fix:

- Test image pull manually on node.
- Check kubelet logs:

```
journalctl -u kubelet
```

Impact:

Pod boot blocked.

Best Practice:

Use local registry mirrors if possible.

Q43: Pod takes too long to schedule. Why?**Root Cause:**

Resource constraints, affinity rules too strict.

Fix:

- Relax affinity.
- Check scheduler events:

```
kubectl describe pod <pod>
```

Impact:

App startup delays.

Best Practice:

Avoid over-constraining scheduler.

Q44: Node running many pods becomes slow. Why?**Root Cause:**

CPU starvation, memory exhaustion, or disk I/O bottleneck.

Fix:

- Monitor with kubectl top node.
- Reschedule heavy pods elsewhere.

Impact:

Cluster instability.

Best Practice:

Spread critical workloads with anti-affinity.

Q45: Pod can't access Kubernetes API. Why?**Root Cause:**

Missing RBAC permissions or service account misconfigured.

Fix:

- Check ServiceAccount permissions.
- Apply correct RBAC roles.

Impact:

App features relying on API fail.

Best Practice:

Use least privilege RBAC.

Q46: Deployment stuck in progress — how to check?**Fix:**

Check rollout status:

`kubectl rollout status deployment <name>`

- Check `kubectl describe deployment <name>` events.

Impact:

Partial deploy causes broken services.

Best Practice:

Enable rollout alarms.

Q47: CronJob fails randomly. Debugging steps?**Root Cause:**

Missed schedule due to controller issues or resource quota.

Fix:

-
- Check CronJob history:

```
kubectl get cronjobs
```

```
kubectl describe cronjob <name>
```

- Check pod failures.

Impact:

Tasks missed (e.g., backup failure).

Best Practice:

Enable job retries, monitor completions.

Q48: Some pods can access Internet, others can't. Why?**Root Cause:**

Network policies restricting egress selectively.

Fix:

- Check NetworkPolicy:

```
kubectl get networkpolicy -A
```

- Adjust egress rules.

Impact:

Broken external API calls.

Best Practice:

Define minimal necessary egress in policies.

Q49: How to manually evict a pod?**Fix:**

```
kubectl delete pod <pod-name> --grace-period=0 --force
```

Impact:

Immediate pod removal.

Best Practice:

Use sparingly; prefer graceful rollout.

Q50: Pod evicted — message "Evicted: The node was low on resource". Reason?

Root Cause:

Memory or disk pressure triggered kubelet eviction.

Fix:

- Monitor resource usage.
- Scale up nodes or optimize app.

Impact:

Pod loss, service disruption.

Best Practice:

Set requests and limits carefully.

Q51: Need canary deployments. Best way?

Fix:

- Use two deployments with weighted traffic in service.
- Automate via Argo Rollouts or Flagger.

Impact:

Progressive rollout reduces blast radius.

Best Practice:

Use metrics-based auto-promotion.

Q52: Pod restart count increasing slowly. Should you worry?

Root Cause:

Minor crashes, memory leaks, or liveness probe issues.

Fix:

-
- Investigate pod logs and readiness.

Impact:

Gradual degradation → possible outage later.

Best Practice:

Set restart alert thresholds.

Q53: How to list all resources in a namespace?**Fix:**

```
kubectl get all -n <namespace>
```

Impact:

Quick inventory visibility.

Best Practice:

Use label selectors to narrow searches.

Q54: What happens if two pods have same hostPort?**Root Cause:**

Host port conflict; second pod fails to start.

Fix:

- Assign unique hostPort per pod.
- Avoid hostPort unless necessary.

Impact:

Pod scheduling errors.

Best Practice:

Use Service abstraction instead.

Q55: Application inside pod can't resolve DNS. Debug?**Fix:**

- Check CoreDNS logs.

-
- Check /etc/resolv.conf inside pod.
 - Restart CoreDNS if needed.

Impact:

Apps fail to find services.

Best Practice:

Monitor CoreDNS health constantly.

Q56: How to label a running node?

Fix:

```
kubectl label nodes <node-name> disktype=ssd
```

Impact:

Labels enable scheduling policies.

Best Practice:

Use consistent labeling across cluster.

Q57: How to remove a label from node or pod?

Fix:

```
kubectl label nodes <node-name> disktype-
```

Impact:

Scheduling behavior changes.

Best Practice:

Document all label changes in GitOps.

Q58: Pods scheduled but stuck initializing. Why?

Root Cause:

InitContainer failure or volume issues.

Fix:

- Check init container logs.



-
- Inspect volume mounts.

Impact:

Startup bottlenecks.

Best Practice:

Test init containers separately.

Q59: How to quickly check rollout history?

Fix:

`kubectl rollout history deployment <name>`

Impact:

Track previous versions.

Best Practice:

Tag important rollouts with annotations.

Q60: How to cleanly delete a namespace stuck in Terminating?

Fix:

- Remove finalizers:

```
kubectl get namespace <ns> -o json | jq '.spec.finalizers=[]' | kubectl replace --raw "/api/v1/namespaces/<ns>/finalize" -f -
```

Impact:

Namespace deletion blocks cluster cleanup.

Best Practice:

Avoid resources stuck with finalizers during delete.

Q61: Pod crashloop after secret update. Why?

Root Cause:

New secret missing required keys or wrong format.

Fix:

- Validate secret fields.



-
- Rollout restart deployment.

Impact:

App fails to boot.

Best Practice:

Validate secrets pre-deployment.

Q62: Rolling update too slow. How to speed up?**Fix:**

Tune strategy:

strategy:

`rollingUpdate:`

`maxUnavailable: 50%`

`maxSurge: 50%`

Impact:

Faster rollout but needs resource margin.

Best Practice:

Balance speed and stability.

Q63: Kubernetes dashboard not opening. Why?**Root Cause:**

RBAC missing or dashboard pod crash.

Fix:

- Check dashboard pod logs.
- Apply correct ClusterRoleBinding.

Impact:

No UI access to cluster.

Best Practice:

Expose dashboard securely with auth.

Q64: Node upgrade needed without downtime. How?

Fix:

- Cordon node.
- Drain node.
- Upgrade.
- Uncordon after.

Impact:

Zero downtime if pods rescheduled properly.

Best Practice:

Use node pools for rolling upgrades.

Q65: How to quickly find which node a pod runs on?**Fix:**

```
kubectl get pod <pod> -o wide
```

Impact:

Faster troubleshooting.

Best Practice:

Monitor pods and nodes mapping.

Q66: Kube-proxy crash — network breaks. Recovery?**Root Cause:**

Node cannot route services internally.

Fix:

- Restart kube-proxy:

```
kubectl rollout restart daemonset kube-proxy -n kube-system
```

Impact:

Service communication fails.

Best Practice:

Alert if kube-proxy dies.

Q67: How to check which scheduler placed a pod?

Fix:

Look under pod spec:

```
kubectl get pod <pod> -o yaml | grep schedulerName
```

Impact:

Custom schedulers need validation.

Best Practice:

Use default scheduler unless necessary.

Q68: What happens if no resource limits set?**Root Cause:**

Pod may consume unlimited CPU/memory.

Fix:

- Set resource requests and limits.

Impact:

Noisy neighbor effect — node starvation.

Best Practice:

Mandate limits via admission controllers.

Q69: ETCD out of space — what happens?**Root Cause:**

Keyspace fills up; API server fails.

Fix:

- Compact ETCD:

```
etcdctl compact <revision>
```

- Defrag:

```
etcdctl defrag
```

Impact:

Cluster becomes unresponsive.

Best Practice:

Monitor ETCD disk usage carefully.

Q70: How to expose service externally without LoadBalancer?

Fix:

Use Ingress + NodePort service internally.

Impact:

No cloud load balancer needed.

Best Practice:

Ingress is better for multi-service management.

Q71: How to bind pod to specific availability zone?

Fix:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: topology.kubernetes.io/zone

operator: In

values:

- us-east-1a

Impact:

Better latency and HA.

Best Practice:

Always spread across zones.

Q72: Pod stuck in Init:CrashLoopBackOff. Why?

Root Cause:

InitContainer failing repeatedly.

Fix:

- Check init container logs.
- Fix its configuration/scripts.

Impact:

Main app container never starts.

Best Practice:

Use minimal init containers.

Q73: High API server latency observed. What to check?**Root Cause:**

ETCD overload or authentication bottlenecks.

Fix:

- Check etcd latency.
- Scale API server horizontally if HA.

Impact:

kubectl, controllers slow down.

Best Practice:

Distribute load with multiple masters.

Q74: How to restart all pods in a namespace fast?**Fix:**

Delete pods:

```
kubectl delete pods --all -n <namespace>
```

Impact:

Downtime unless replicas available.

Best Practice:

Use rollout restart per deployment instead.

Q75: CronJob overlapping — two runs at once. Why?**Root Cause:**

concurrencyPolicy not set.

Fix:

`concurrencyPolicy: Forbid`

Impact:

Race conditions, duplicate tasks.

Best Practice:

Use Forbid or Replace concurrency wisely.

Q76: Secret accidentally exposed — immediate action?**Fix:**

- Rotate secret credentials immediately.
- Revoke old keys/tokens.

Impact:

Security breach risk.

Best Practice:

Use sealed-secrets or vault-based secret management.

Q77: Pod needs privileged access — how to configure?**Fix:**

`securityContext:`

`privileged: true`

Impact:

Higher security risk.

Best Practice:

Limit privileged pods strictly.

Q78: Pod can't mount volume — permission denied. Why?

Root Cause:

Volume permissions mismatch.

Fix:

- Adjust fsGroup in pod spec:

securityContext:

 fsGroup: 2000

Impact:

Pod crashes at boot.

Best Practice:

Use correct UID/GID handling.

Q79: Controller Manager logs show "throttling" — impact?**Root Cause:**

API server limiting excessive requests.

Fix:

- Optimize controller settings.
- Tune client-go QPS/burst settings.

Impact:

Controller slow responses.

Best Practice:

Throttle back non-critical controllers.

Q80: Kubernetes event logs too noisy. How to reduce?**Fix:**

- Increase event garbage collection settings.
- Monitor only critical event types.

Impact:

Wasted storage + harder troubleshooting.

Best Practice:

Alert on only Warning/Errors.

Q81: Pod in "Error" state — difference from "CrashLoopBackOff"?**Root Cause:**

- "Error" = pod exits once and doesn't retry.
- "CrashLoopBackOff" = continuous crash and restart.

Fix:

Investigate container exit code and logs.

Impact:

Service availability affected.

Best Practice:

Use restartPolicy wisely (Always, OnFailure, Never).

Q82: How to view all failed pods?**Fix:**

```
kubectl get pods --field-selector=status.phase=Failed
```

Impact:

Identify issues early.

Best Practice:

Alert on high failed pod counts.

Q83: Pod deleted but keeps reappearing. Why?**Root Cause:**

Controlled by a Deployment/ReplicaSet.

Fix:

- Delete higher controller:

```
kubectl delete deployment <name>
```

Impact:

Manual deletion ineffective.

Best Practice:

Understand controller hierarchy.

Q84: Pod says "OOMKilled" — what does it mean?**Root Cause:**

Out of Memory — killed by Linux kernel.

Fix:

- Increase memory limits.
- Optimize app memory usage.

Impact:

Service crashes randomly.

Best Practice:

Set appropriate requests/limits and monitor.

Q85: Service pointing to wrong pods. Reason?**Root Cause:**

Label selector mismatch.

Fix:

Update Deployment pod labels or Service selectors.

Impact:

Traffic routed to wrong apps.

Best Practice:

Use unique app labels per service.

Q86: How to manually trigger HPA scale out?**Fix:**

- Increase CPU artificially or
- Edit HPA spec:



```
kubectl edit hpa <name>
```

Increase target utilization temporarily.

Impact:

Force scale to test behavior.

Best Practice:

Simulate load carefully.

Q87: Why are kube-dns/CoreDNS pods critical?

Root Cause:

Cluster DNS relies on them.

Fix:

Always ensure minimum replicas (2+).

Impact:

Service discovery failure if DNS down.

Best Practice:

Scale CoreDNS appropriately.

Q88: Node goes into "Unknown" state. Why?

Root Cause:

Heartbeat timeout — node unreachable.

Fix:

- SSH if possible.
- If lost, remove from cluster and replace.

Impact:

Pods need rescheduling; risk of downtime.

Best Practice:

Set node health checks + auto-repair.

Q89: Why is readiness probe more important than liveness?

Root Cause:

- Readiness = routing traffic only to ready pods.



-
- Liveness = killing/restarting bad pods.

Fix:

Use both properly, prioritize readiness for stability.

Impact:

Wrong probe = user-facing errors.

Best Practice:

Separate health endpoints for readiness/liveness.

Q90: How to migrate a running pod to another node?

Fix:

- Cordon current node: `kubectl cordon <node>`
- Delete pod → it reschedules on another node.

Impact:

Minimal downtime if managed carefully.

Best Practice:

Use PodDisruptionBudgets if app critical.

Q91: Deployment automatically triggers rollout even without changes. Why?

Root Cause:

Mutable fields like configmap checksum changes or sidecar updates.

Fix:

Investigate spec changes triggering new ReplicaSets.

Impact:

Unexpected pod churn.

Best Practice:

Use immutable tags; track changes via GitOps.

Q92: What if etcd backup is missing during disaster?

Root Cause:

No cluster state restore possible.

Fix:



-
- Manually recreate cluster (very hard).
 - Recommend: **always backup etcd!**

Impact:

Full cluster loss.

Best Practice:

Scheduled etcd snapshots to secure storage.

Q93: New node joins but no pods scheduled — why?

Root Cause:

- Node tainted.
- Unschedulable (cordoned).

Fix:

- Check taints:

`kubectl describe node <node>`

- Remove taints if needed.

Impact:

Wasted cluster capacity.

Best Practice:

Auto-un-cordon nodes after joining.

Q94: Why avoid setting resource limits too low?

Root Cause:

- Causes CPU throttling.
- Out-of-memory killing.

Fix:

Right-size based on real metrics (Prometheus).

Impact:

Performance degradation.

Best Practice:

Adjust with load tests + autoscaling.



Q95: Pod failing with "CrashLoopBackOff" only on one node. Why?**Root Cause:**

Node-specific issue (disk, network, corrupt runtime).

Fix:

- Drain the bad node.
- Investigate system logs.

Impact:

Single-node issues affecting pods.

Best Practice:

Run node diagnostics periodically.

Q96: How to delete stuck finalizers without downtime?**Fix:**

Patch resource:

```
kubectl patch <resource> -p '{"metadata":{"finalizers":[]}}' --type=merge
```

Impact:

Force cleanups.

Best Practice:

Fix controller bugs causing stuck finalizers.

Q97: New pods get scheduled on full nodes. Why?**Root Cause:**

Kubernetes scheduler works on resource **requests**, not actual usage.

Fix:

Set accurate requests/limits.

Impact:

Hidden resource crunch.

Best Practice:

Enable resource monitoring to detect drift.

Q98: How to recover a deleted namespace accidentally?**Root Cause:**

Namespace deletion irreversible if completed.

Fix:

- Preventative: Backup manifests.
- Recreate resources manually.

Impact:

Major outage if critical NS deleted.

Best Practice:

Enable RBAC restrictions on delete operations.

Q99: Service says "no endpoints available" but pods are running. Why?**Root Cause:**

Pods not passing readiness probes.

Fix:

- Validate health checks.
- Check pod readiness status.

Impact:

Service unreachable despite pod presence.

Best Practice:

Separate startup vs readiness probes.

Q100: kubectl commands hang intermittently. Cause?**Root Cause:**

API server overload or client timeout.

Fix:

- Scale API server replicas.
- Use local caching (kubectl client-go improvements).

Impact:

Slow cluster operations.

Best Practice:

Monitor API server metrics (qps, latency).

Q101: Pod using hostPath volume. Risk?**Root Cause:**

Full node filesystem access if misused.

Fix:

Use PersistentVolumeClaims instead of hostPath.

Impact:

Security risk, node corruption.

Best Practice:

Restrict hostPath usage via PodSecurityPolicies.

Q102: Service not load balancing properly. Why?**Root Cause:**

Session affinity misconfigured or uneven readiness states.

Fix:

- Check service.spec.sessionAffinity.
- Validate pod readiness.

Impact:

Load skew → performance impact.

Best Practice:

Use health checks and even pod distribution.

Q103: How to guarantee pod restart on node failure?**Fix:**

Use Deployment or StatefulSet, not naked pods.

Impact:

Guaranteed pod rescheduling.

Best Practice:

Never use standalone pods in production.

Q104: Node ready, but pods stuck at ContainerCreating. Why?**Root Cause:**

Image pull errors, CSI driver mount issues.

Fix:

Check kubelet logs + container runtime logs.

Impact:

Delayed service startups.

Best Practice:

Automate runtime health alerts.

Q105: Pod pending due to "No nodes match pod affinity rules". What now?**Root Cause:**

Affinity too strict; no node satisfies constraints.

Fix:

Relax affinity conditions.

Impact:

Stuck deployment.

Best Practice:

Use preferred affinity rules instead of required when possible.

Q106: Multiple pods writing to same volume. Problem?**Root Cause:**

PVC with ReadWriteOnce mounted by multiple pods.

Fix:

- Use ReadWriteMany storage class if needed.

-
- Avoid multiple writers otherwise.

Impact:

Data corruption possible.

Best Practice:

Separate volumes or RWX-capable storage.

Q107: Node joins but fails network setup. Cause?

Root Cause:

CNI plugin misconfigured or not installed.

Fix:

- Check CNI plugin pods.
- Reinstall CNI components.

Impact:

Pods cannot get network.

Best Practice:

Verify CNI status during node join.

Q108: Cluster scaling slow despite HPA working. Why?

Root Cause:

Cluster autoscaler too conservative or misconfigured.

Fix:

Tune autoscaler parameters: scale-up-delay, max-nodes.

Impact:

Apps suffer under load.

Best Practice:

Simulate load to pre-test scaling behaviors.

Q109: Deployment rollback also fails. What next?

Root Cause:

Both old and new replicas are broken.



Fix:

Manually redeploy last known good configuration.

Impact:

Outage persists.

Best Practice:

Tag stable releases in Git + container registry.

Q110: How to force delete a stuck namespace?**Fix:**

Remove finalizers manually (force finalize).

`kubectl proxy &`

```
curl -k -H "Content-Type: application/json" -X PUT --data-binary @finalize.json
http://127.0.0.1:8001/api/v1/namespaces/<ns>/finalize
```

Impact:

Namespace finally deleted.

Best Practice:

Fix root cause of stuck resources.

Q111: Pods crash after node reboot. Why?**Root Cause:**

Lost volumes or node-specific storage not reattached.

Fix:

Check PersistentVolume attachment and CSI status.

Impact:

Downtime until rescheduling.

Best Practice:

Use dynamic volume provisioning with health checks.

Q112: CronJob runs multiple times wrongly. How?**Root Cause:**

Controller restart during scheduled time.

Fix:

Set startingDeadlineSeconds to restrict late runs.

Impact:

Duplicate operations (e.g., double backup).

Best Practice:

Set strict concurrency policies.

Q113: High etcd memory usage — fix?**Root Cause:**

Large number of short-lived keys (events).

Fix:

- Tune etcd compaction frequency.
- Defragment regularly.

Impact:

Slower API responses.

Best Practice:

Compact every few hours under load.

Q114: How to detect Pod eviction reasons easily?**Fix:**

Describe pod:

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

Look under "Events" for eviction messages.

Impact:

Quick root cause identification.

Best Practice:

Alert on eviction thresholds.

Q115: New node ready but pods not scheduled. Why?

Root Cause:

Taints like node.kubernetes.io/not-ready not cleared.

Fix:

- Wait for full kubelet readiness.
- Remove taints if needed.

Impact:

Cluster underutilized.

Best Practice:

Validate node after join.

Q116: ReplicaSet vs Deployment — difference?**Root Cause:**

ReplicaSet maintains pod replicas. Deployment manages ReplicaSets + rolling updates.

Fix:

Prefer Deployment unless advanced replica control needed.

Impact:

Manual ReplicaSet management is error-prone.

Best Practice:

Always use Deployment abstraction.

Q117: NetworkPolicy blocks internal app access. Solution?**Root Cause:**

No allow rules configured; default deny.

Fix:

Define ingress/egress allow rules explicitly.

Impact:

Broken service communication.

Best Practice:

Use progressive allow-listing of traffic.

Q118: Pod mounts ConfigMap but fails to find files. Why?

Root Cause:

Mount path mismatch or wrong key usage.

Fix:

Check volumeMounts and volumes section carefully.

Impact:

App boot failure.

Best Practice:

Standardize config layouts.

Q119: Pod scaling too aggressively — HPA misbehaving. Why?**Root Cause:**

Wrong metric thresholds or misinterpreted CPU spikes.

Fix:

Tune targetAverageUtilization wisely.

Impact:

Resource wastage, node saturation.

Best Practice:

Smooth metrics input with Prometheus relabelling.

Q120: Service type ExternalName — use case?**Fix:**

Map service DNS inside cluster to external hostname (e.g., SaaS API).

spec:

type: ExternalName

externalName: db.external.com

Impact:

No need for proxies.

Best Practice:

Use for simple external access without ingress.

Q121: How to get pod logs of a completed Job?

Fix:

Use --previous flag:

```
kubectl logs <pod-name> --previous
```

Impact:

Retrieve logs even if pod exited.

Best Practice:

Export logs before pod GC deletes them.

Q122: Pod stuck in "Evicted" — can you restart it?**Root Cause:**

Evicted pods are **terminated**, not restartable.

Fix:

Create a new pod or redeploy.

Impact:

Manual intervention needed.

Best Practice:

Fix eviction root cause early (disk, memory).

Q123: How to limit Job retries if it keeps failing?**Fix:**

Set backoffLimit:

```
backoffLimit: 3
```

Impact:

Prevent endless job retries.

Best Practice:

Always define retry logic clearly.

Q124: Pod goes Pending — PVC stuck in "Pending". Why?**Root Cause:**

No suitable PersistentVolume available.

Fix:

Check PVC events and storage class provisioning.



Impact:

App deployment blocks.

Best Practice:

Pre-provision storage or enable dynamic provisioning.

Q125: Service running, but port 80 blocked externally. Cause?**Root Cause:**

NetworkSecurityGroups/Firewall missing rules.

Fix:

Allow ingress TCP 80 in cloud VPC/NLB.

Impact:

App inaccessible externally.

Best Practice:

Define firewall rules at deployment time.

Q126: Liveness probe failure leads to pod restart. Normal?**Root Cause:**

Yes — Kubernetes restarts unhealthy containers automatically.

Fix:

Investigate liveness probe logic and app behavior.

Impact:

Avoid silent cascading failures.

Best Practice:

Use readiness probe stricter than liveness.

Q127: How to list all services in a namespace?**Fix:**

```
kubectl get svc -n <namespace>
```

Impact:

Quick environment mapping.

Best Practice:

Label services for easy grouping.

Q128: What happens if a NodePort service port is already taken?**Root Cause:**

NodePort range conflict.

Fix:

Choose an unused NodePort manually (30000–32767 default).

Impact:

Service creation fails.

Best Practice:

Avoid manual NodePort assignments unless necessary.

Q129: App using NodePort service is accessible from wrong IPs. Why?**Root Cause:**

No firewall/NACL restricting incoming traffic.

Fix:

Restrict security groups to trusted IP ranges.

Impact:

Security breach possible.

Best Practice:

Always enforce external firewall.

Q130: How to check pod events quickly?**Fix:**

`kubectl describe pod <pod-name>`

Check "Events" section.

Impact:

Immediate visibility into issues.

Best Practice:

Automate event monitoring.

Q131: CrashLoopBackOff happens due to config error. How to fix without redeploying?

Fix:

If config in ConfigMap/Secret → update + rollout restart.

Otherwise, rebuild corrected image.

Impact:

Redeploy time critical.

Best Practice:

Decouple configs from images always.

Q132: App metrics missing in Prometheus. Cause?

Root Cause:

ServiceMonitor not created or wrong selector.

Fix:

Create ServiceMonitor with correct labels.

Impact:

Observability loss.

Best Practice:

Use consistent app metrics naming.

Q133: Node suddenly stops running pods. Why?

Root Cause:

PIDs exhausted, disk full, networking issues.



Fix:

Check kubelet logs, OS dmesg/syslog.

Impact:

Partial app outages.

Best Practice:

Monitor OS-level resources proactively.

Q134: How to cordon multiple nodes at once?**Fix:**

```
kubectl cordon node1 node2 node3
```

Impact:

Block scheduling in batch.

Best Practice:

Automate with scripts during upgrades.

Q135: Pod volume mount errors after upgrade. Cause?**Root Cause:**

Storage plugin upgrade issues or API version changes.

Fix:

Check CSI driver compatibility.

Impact:

Data loss risks.

Best Practice:

Backup volumes before major upgrades.

Q136: Pod needs to run on GPU node only. How to enforce?**Fix:**

Use node selectors:

nodeSelector:

accelerator: nvidia-gpu

Impact:

Dedicated GPU scheduling.

Best Practice:

Label GPU nodes properly.

Q137: Cluster upgraded, but some nodes still old version. Why?

Root Cause:

Rolling upgrade incomplete.

Fix:

Upgrade nodes manually or restart node pool.

Impact:

Inconsistent API behavior.

Best Practice:

Automate upgrade plans cluster-wide.

Q138: CoreDNS pods crash repeatedly. How to debug?

Fix:

- Check CoreDNS logs.
- Validate config maps.
- Inspect node networking.

Impact:

Service discovery failures.

Best Practice:

Backup known good CoreDNS configs.

Q139: How to find which pod is consuming most disk?

Fix:

Exec into pod and use:

du -sh /*

Or monitor at node/container runtime level.

Impact:

DiskPressure events if unnoticed.

Best Practice:

Alert on abnormal volume usage growth.

Q140: Pod logs show "connection refused" to another service. Likely reason?

Root Cause:

- Target pod not ready.
- Service misconfigured.
- NetworkPolicy blocking.

Fix:

- Check endpoints.
- Check readiness probes.
- Check NetworkPolicy rules.

Impact:

Microservice communication breaks.

Best Practice:

Automate service health checks before traffic.

Q141: Pod lifecycle stuck in "Init". Reason?

Root Cause:

InitContainers failing or long-running.

Fix:

- Check kubectl describe pod <pod> → init container logs.
- Fix init tasks.

Impact:

Main container delayed.

Best Practice:

Keep InitContainers lightweight and fast.



Q142: Pod crash on startup after config update. Likely cause?**Root Cause:**

Invalid environment variable or bad config file.

Fix:

Check container logs immediately after crash.

Impact:

New rollout stuck.

Best Practice:

Validate configs via staging pipelines first.

Q143: Why avoid latest tag in container images?**Root Cause:**

latest is mutable → causes unexpected pulls.

Fix:

Use immutable version tags (e.g., v1.2.3).

Impact:

Non-reproducible deployments.

Best Practice:

Strict version pinning.

Q144: How to debug a Job that never runs?**Fix:**

Check:

`kubectl get jobs`

`kubectl describe job <name>`

- Look for controller errors, quota issues.

Impact:

Missed scheduled tasks.

Best Practice:

Set alerts on Job completions.

Q145: NetworkPolicy blocks pods after rollout. What changed?**Root Cause:**

New labels not matching old NetworkPolicies.

Fix:

Update NetworkPolicy selectors to match new pods.

Impact:

Application communication breakage.

Best Practice:

Version-control policies with apps.

Q146: App randomly unavailable — liveness probe flapping. Cause?**Root Cause:**

Liveness probe too strict or flaky app behavior.

Fix:

Relax thresholds: increase failureThreshold, periodSeconds.

Impact:

Pods restarted unnecessarily.

Best Practice:

Separate healthcheck endpoints from app logic.

Q147: StatefulSet pod stuck terminating. Why?**Root Cause:**

PersistentVolume stuck unmounting or Finalizer blocked.

Fix:

Force delete pod if safe; investigate volume.

Impact:

Stateful apps blocked (e.g., DB nodes).

Best Practice:

Gracefully handle pod shutdown in apps.

Q148: Kubelet logs show "pleg was last seen X seconds ago". What does it mean?

Root Cause:

Pod lifecycle event generator stalled → container runtime issue.

Fix:

Restart container runtime and kubelet.

Impact:

Node scheduling and health degraded.

Best Practice:

Alert on kubelet health metrics.

Q149: Pod stuck in CrashLoopBackOff due to DB connection. Cause?**Root Cause:**

App starts before DB becomes available.

Fix:

Add readiness probe with DB connectivity check.

Impact:

Unnecessary restarts.

Best Practice:

Graceful retries inside app on boot.

Q150: Can't delete a pod immediately. Why?**Root Cause:**

Graceful shutdown period (`terminationGracePeriodSeconds`).

Fix:

Force delete:

```
kubectl delete pod <pod> --grace-period=0 --force
```

Impact:

Potential data corruption if not safe.

Best Practice:

Tune shutdown periods carefully.

Q151: Service of type ClusterIP needed externally. Quick solution?

Fix:

Expose temporarily using port forwarding:

```
kubectl port-forward svc/<service-name> <local-port>:<service-port>
```

Impact:

Dev testing made easy.

Best Practice:

Use only temporarily for testing.

Q152: What causes "failed to assign an IP address" error in pod?**Root Cause:**

CNI plugin failure.

Fix:

Check CNI logs, node networking.

Impact:

Pod never gets scheduled.

Best Practice:

Monitor CNI health and node IP pools.

Q153: Pods not evenly distributed across zones. Why?**Root Cause:**

No topology-aware scheduling.

Fix:

Enable topologySpreadConstraints or multi-zone awareness.

Impact:

Risk of zone failure impact.

Best Practice:

Always architect for zone resilience.

Q154: How to restrict pod scheduling to specific nodes only?**Fix:**

Use node affinity or nodeSelector.

```
nodeSelector:
```

disktype: ssd

Impact:

Better workload placement.

Best Practice:

Use tolerations + taints combo for special nodes.

Q155: Application restart causes cache loss. Solution?**Root Cause:**

Pods ephemeral — no persistent store.

Fix:

- Use PersistentVolumes.
- Or move to external cache (e.g., Redis, Memcached).

Impact:

User sessions/data lost.

Best Practice:

Avoid app caching inside pod memory only.

Q156: Deployment rollout completed, but service still unreachable. Cause?**Root Cause:**

Readiness probe failures prevent Service binding.

Fix:

Fix readiness probes or app startup.

Impact:

Zero visible downtime, hidden functional downtime.

Best Practice:

Monitor readiness as critical metric.

Q157: Node labels disappear after reboot. Why?**Root Cause:**

Manually applied labels without persistence.



Fix:

Use startup scripts or cloud-specific node labels.

Impact:

Pod scheduling errors.

Best Practice:

Automate node labeling at bootstrap.

Q158: K8s API server shows "too many open files" error. What to check?**Root Cause:**

Ulimit settings too low.

Fix:

Increase nofile limits on API server hosts.

Impact:

API server crash/unstable.

Best Practice:

Configure system limits appropriately in HA clusters.

Q159: How to block a namespace from accidental deletion?**Fix:**

Use ResourceQuota + LimitRange policies carefully;
use GitOps to control creation/deletion via PR approvals.

Impact:

Protect critical namespaces.

Best Practice:

Lock down critical namespaces via RBAC.

Q160: How to ensure pod termination doesn't affect users?**Fix:**

- Use readiness probes to unlist before shutdown.
- Graceful shutdown handlers inside app code.

Impact:

Zero-downtime deployments.



Best Practice:

Shift connections off before termination.

Q161: Pod can't pull image after node reboot. Why?**Root Cause:**

Node credentials lost, or docker daemon/containerd not restarted.

Fix:

- Restart container runtime.
- Revalidate imagePullSecrets.

Impact:

Deployment failures post-reboot.

Best Practice:

Healthcheck container runtime after node restarts.

Q162: Pod stuck in Terminating during scaling down. Cause?**Root Cause:**

Pod finalizers hanging or shutdown tasks not completing.

Fix:

Force delete if safe:

```
kubectl delete pod <pod> --grace-period=0 --force
```

Impact:

Scaling operations delayed.

Best Practice:

Fix app shutdown behavior inside container.

Q163: CoreDNS shows SERVFAIL errors. What's wrong?

Root Cause:

Misconfigured upstream DNS servers or Corefile syntax issues.

Fix:

- Correct upstream settings.
- Restart CoreDNS.

Impact:

Service discovery broken inside cluster.

Best Practice:

Always validate CoreDNS configs after cluster updates.

Q164: Pod affinity rules slow down deployments. Why?**Root Cause:**

Scheduler tries to satisfy hard affinity constraints → delays.

Fix:

Use preferred affinity over required wherever possible.

Impact:

Slower rollout, resource wastage.

Best Practice:

Balance scheduling strictness vs rollout speed.

Q165: Liveness probe triggers unnecessary restarts. Fix?**Root Cause:**

Too aggressive thresholds.

Fix:

- Increase timeoutSeconds, failureThreshold.
- Review app healthcheck responsiveness.

Impact:

False alarms, resource churn.

Best Practice:

Test probes thoroughly under load.

Q166: PersistentVolume shows "Released" state. Now what?

Root Cause:

PVC deleted, PV stuck in released.

Fix:

Manually reclaim or rebind volume.

Impact:

Wasted storage.

Best Practice:

Use dynamic provisioners with reclaimPolicy: Delete.

Q167: K8s Ingress works internally but not externally. Why?

Root Cause:

External DNS/LoadBalancer misconfigured.

Fix:

- Validate public IP mappings.
- Check ingress controller external exposure.

Impact:

Users can't reach application.

Best Practice:

Test Ingress paths internally and externally both.

Q168: HPA scales pods up but doesn't scale down. Cause?

Root Cause:

Metrics stickiness or cooldown periods not tuning.

Fix:

Adjust stabilizationWindowSeconds.

Impact:

Overprovisioned resources.

Best Practice:

Use right HPA cooldown policies.

Q169: App logs missing even after pod runs fine. Why?**Root Cause:**

Logging agent misconfigured or /var/log missing mounts.

Fix:

- Reconfigure Fluentd, Loki, etc.
- Check pod file paths.

Impact:

Lost observability.

Best Practice:

Centralize logging early.

Q170: Deployment rolling update triggers full outage. Reason?**Root Cause:**

Misconfigured maxUnavailable during rollout.

Fix:

Set:

strategy:

rollingUpdate:

maxUnavailable: 0

Impact:

Zero-availability window.

Best Practice:

Use safe rollout configurations always.

Q171: StatefulSet scaling down deletes wrong pod first. Why?**Root Cause:**

StatefulSets delete highest ordinal pod first by design.

Fix:

Understand StatefulSet behavior → design apps for it.

Impact:

Possible data loss if mishandled.

Best Practice:

Manual deletion if order matters.

Q172: Pod not using updated ConfigMap even after restart. Why?**Root Cause:**

Pod caching old config internally or mounting issues.

Fix:

Check if volumes or env refs were properly refreshed.

Impact:

App running old configs silently.

Best Practice:

Validate config reload behavior in app code.

Q173: API server CPU usage spikes during heavy kubectl use. Why?**Root Cause:**

kubectl commands directly hit API server heavily.

Fix:

Rate-limit heavy kubectl scripts.

Use kubectl get with field selectors.

Impact:

Controller performance degradation.

Best Practice:

Use efficient queries + cache where possible.

Q174: DaemonSet not deploying new pods on new nodes. Cause?**Root Cause:**

Node label/taint mismatch or missing tolerations.

Fix:

Add tolerations and correct node selectors.

Impact:

Monitoring/logging gaps.

Best Practice:

DaemonSets should tolerate default taints.

Q175: Kubernetes Dashboard exposed publicly without auth. Danger?**Root Cause:**

Anonymous access allowed by service configuration.

Fix:

Use secured Ingress + RBAC + OIDC auth.

Impact:

Severe security vulnerability.

Best Practice:

Restrict access; use VPN for dashboard access.

Q176: CronJobs pile up old pods. Why?**Root Cause:**

History limits not set.

Fix:

Set:

`successfulJobsHistoryLimit: 3`

`failedJobsHistoryLimit: 1`

Impact:

Resource clutter.

Best Practice:

Prune old Jobs via spec limits.

Q177: Node clock skew causes pod issues. Why?**Root Cause:**

Time difference between nodes disrupts certificates, auth, etc.

Fix:

Enable NTP (Network Time Protocol) on all nodes.

Impact:

TLS errors, service failures.

Best Practice:

Time sync critical in Kubernetes clusters.

Q178: Pod anti-affinity ignored by scheduler. Why?**Root Cause:**

Anti-affinity rules are preferred, not required.

Fix:

Use requiredDuringSchedulingIgnoredDuringExecution for strict behavior.

Impact:

Pods may still colocate undesirably.

Best Practice:

Document anti-affinity strategies clearly.

Q179: Pod crashes only after 1 hour of running. Why?**Root Cause:**

Memory leaks, resource starvation over time.

Fix:

- Investigate resource usage trends.
- Tune garbage collection inside app.

Impact:

Hidden stability issues.

Best Practice:

Long-running soak tests before production deployment.

Q180: Deployment shows completed but app still inaccessible. Why?**Root Cause:**

Pods are ready, but Service or Ingress misconfigured.

Fix:

- Verify service endpoints.
- Validate ingress routing rules.



Impact:

False sense of rollout success.

Best Practice:

Automate smoke tests post-deployment.

Q181: How to troubleshoot a stuck pod deletion?**Fix:**

Force delete pod:

```
kubectl delete pod <pod> --force --grace-period=0
```

Check for finalizers causing block.

Impact:

Deployment pipelines may hang.

Best Practice:

Monitor for stuck Terminating states.

Q182: Ingress returns 502 Bad Gateway. Likely reason?**Root Cause:**

Backend service is down or not ready.

Fix:

- Validate pod readiness.
- Check Ingress controller logs.

Impact:

Application partially offline.

Best Practice:

Use backend health checks with Ingress.

Q183: CrashLoopBackOff with "connection refused" — what to check?**Root Cause:**

Service endpoint not available when pod starts.

Fix:

Add retry logic in app startup.

Use readiness probes carefully.

Impact:

Frequent unnecessary restarts.

Best Practice:

Make services retry-aware.

Q184: Rolling update too aggressive, overwhelming nodes. How to fix?**Fix:**

Lower maxSurge and maxUnavailable settings.

Example:

```
maxSurge: 1
```

```
maxUnavailable: 0
```

Impact:

Node CPU/memory exhaustion.

Best Practice:

Progressive rollouts tuned per workload type.

Q185: ETCD alarms firing on disk space usage. Urgent fix?**Fix:**

- Compact and defrag ETCD:

```
etcdctl compact <revision>
```

```
etcdctl defrag
```

- Expand ETCD storage.

Impact:

API server may fail if ignored.

Best Practice:

Alert early at 70% ETCD disk usage.

Q186: Pod continuously evicted on same node. Root cause?

Root Cause:

Node resource exhaustion not fixed.

Fix:

- Free disk/memory.
- Cordon problematic node.

Impact:

Unstable workloads.

Best Practice:

Enforce node health checks.

Q187: Service name typo in deployment manifest. What happens?**Root Cause:**

Pod tries to connect wrong DNS.

Fix:

Correct the service name.

Rollout restart.

Impact:

Immediate connection failures.

Best Practice:

Automate manifest linting (e.g., kubeval).

Q188: Pod taking 10+ minutes to terminate. Acceptable?**Root Cause:**

Long-running shutdown hooks, slow DB draining.

Fix:

Shorten shutdown tasks or increase shutdown aggressiveness.

Impact:

Slow scaling down → capacity issues.

Best Practice:

Optimize shutdown process within pods.

Q189: Node conditions show "MemoryPressure=True". Action plan?

Fix:

- Investigate memory leaks on pods.
- Scale out nodes.
- Evict non-critical pods.

Impact:

Kubelet prioritizes critical workloads only.

Best Practice:

Monitor node memory usage closely.

Q190: Kube-proxy iptables mode vs IPVS mode — which better?**Root Cause:**

IPVS is faster, more scalable at high pod counts.

Fix:

Use IPVS if node has more than 1000 pods.

Impact:

Better service performance under load.

Best Practice:

Plan cluster sizing with proxy mode choice.

Q191: How to quickly check K8s cluster version?**Fix:**

```
kubectl version --short
```

Impact:

Know capabilities and compatibility immediately.

Best Practice:

Track cluster versions across environments.

Q192: CronJob missed execution window after controller crash. How to avoid?

Fix:

Set startingDeadlineSeconds appropriately.

Example:

```
startingDeadlineSeconds: 300
```

Impact:

Guaranteed critical job execution even after downtime.

Best Practice:

Never leave CronJobs without tight execution windows.

Q193: Kubelet constantly restarts containers. Why?**Root Cause:**

Container runtime health check failing, bad probes.

Fix:

Fix app, or relax liveness/readiness probes.

Impact:

Resource churn, instability.

Best Practice:

Benchmark probe settings under load.

Q194: Helm upgrade rollback fails. Why?**Root Cause:**

State drift or manual cluster changes.

Fix:

Recreate resources cleanly or force Helm release delete.

Impact:

CI/CD failures.

Best Practice:

Enforce GitOps — Helm templates fully reflect cluster state.

Q195: Kubernetes Service slow to detect new pods. Reason?**Root Cause:**

Endpoints controller delays.



Fix:

Tune controller flags if critical.

Generally, wait a few seconds (normal behavior).

Impact:

Short connection drops during scale events.

Best Practice:

Graceful retries inside client apps.

Q196: Pod volume mount has wrong permissions. How to fix?**Fix:**

Set securityContext:

`fsGroup: 2000`

Impact:

Pod crashes or app permission denied errors.

Best Practice:

Match app UID/GID to volume.

Q197: Service unreachable because wrong namespace used. How to catch early?**Fix:**

Explicitly define namespace in all manifests.

Impact:

Hidden communication failures.

Best Practice:

Use namespace validation policies.

Q198: Kubernetes cluster certificates expired. Disaster recovery?**Fix:**

Renew certificates using kubeadm certs renew.

Restart critical components.

Impact:

API server fully unusable until recovery.

Best Practice:

Rotate certificates proactively.

Q199: Pod runs but Service unreachable from another namespace. Why?**Root Cause:**

Need full DNS name: <service>.<namespace>.svc.cluster.local

Fix:

Use full DNS in cross-namespace calls.

Impact:

App-level connection errors.

Best Practice:

Standardize full internal service addresses.

Q200: Pod accesses wrong ConfigMap after rollout. Cause?**Root Cause:**

Outdated volume mounts or wrong ConfigMap selected.

Fix:

Validate volume references carefully.

Impact:

Unexpected app behavior post-deploy.

Best Practice:

Rollout ConfigMap updates with checksum annotations.
