You’re Not Ready for Kubernetes On-Call — Until You Can Answer These 5 Scenarios

LinkedIn: Amit Singh

Medium: 🌐 Amit Singh – Medium

If you’ve ever been on-call for a Kubernetes environment, you know the moment will come: 2 AM, pager buzzing, and you’re staring at a broken cluster.

These five questions *will* come up eventually — and you’d better be ready to handle them with confidence. Let’s break them down with a practical, no-nonsense approach so you can stay calm under fire.

1. What if kubectl Freezes?

Imagine running kubectl get pods, and it just… sits there. Forever. That’s one of the most gut-wrenching feelings on call.

Here’s how to respond:

✅ **Check network connectivity**

- ping <apiserver_ip>
- telnet <apiserver_ip> 6443
- verify firewall rules and security groups

✅ **Check API server health**

- kubectl get --raw /healthz
- or curl https://localhost:6443/healthz --insecure if on the master node

✅ **Validate kubeconfig**

- kubectl config get-contexts
- kubectl config view
- make sure you’re pointing at the correct cluster

✅ **Enable verbose output**

- kubectl get pods -v=8

✅ **Investigate the control plane**

- systemctl status kube-apiserver
- journalctl -u kube-apiserver

✅ **Escalate if needed**

- Confirm etcd is healthy
- Alert your team if the control plane is failing

When kubectl freezes, you're really testing whether your control plane is alive — **always start there**.

2. Can You Fix a Stuck Rollout with kubectl rollout undo?

It happens to everyone: you deploy a new version, and pods get stuck in a crashloop. The fastest way to recover is rolling back to a known working revision.

✅ **Check the rollout status**

- kubectl rollout status deployment/<name>
- kubectl describe deployment <name>

✅ **Review rollout history**

- kubectl rollout history deployment/<name>

✅ **Undo the rollout**

- kubectl rollout undo deployment/<name>

✅ **Validate**

- kubectl rollout status deployment/<name>

✅ **If still stuck**

- Check container images
- Verify ConfigMaps and Secrets
- Validate readiness and liveness probes
- Confirm RBAC and NetworkPolicies aren't blocking traffic

✅ **Manual fix**

- kubectl edit deployment <name>
- correct fields manually and apply changes

Rollbacks are your safety net — but they only work if your rollout history is healthy. **Test your rollback before you need it!**

3. What if Your Service Works in Staging, but Not Production?

This one is classic — it passes all staging tests, you ship to prod, and it blows up.

✅ **Verify the image**

- Are you using the same SHA or accidentally running :latest?
- Confirm with kubectl describe pod <pod>

✅ **Compare configuration**

- Environment variables
- ConfigMaps
- Secrets
- Feature flags

✅ **Cluster-level differences**

- NetworkPolicies
- PodSecurityPolicies or Kyverno
- RBAC roles

✅ **Test from within the pod**

- kubectl logs <pod>
- kubectl exec -it <pod> -- /bin/sh
- test connectivity to dependent services

✅ **Validate DNS and service discovery**

- nslookup <service-name>
- dig <service-name>

✅ **Resource constraints**

- Check for OOMKills or CPU throttling
- Production often has stricter quotas than staging

✅ **Consider traffic mirroring**

- Safely mirror production traffic to staging to debug realistic scenarios

Remember: *prod is never identical to staging*. That's why on-call engineers get paid to solve these differences fast.

4. What's the Fastest Way to Trace Pod-to-Pod Network Latency?

Networking is one of the trickiest parts of Kubernetes. If one service can't talk to another, it's crucial to identify if the issue is latency, packet loss, or network policy.

✅ **Quick checks**

- Deploy a debug pod (like nicolaka/netshoot)
- kubectl exec -it <debug-pod> -- ping <target-pod-ip>
- curl -w "@curl-format.txt" to measure timings

✅ **Deep dive**

- iperf between pods
- mtr or traceroute for hops
- tcpdump on the worker node
- Check the CNI plugin logs (Calico, Cilium, Flannel, etc.)

✅ **Advanced**

- Enable NetworkPolicy flow logs
- Service mesh metrics (e.g., Istio, Linkerd tap)
- Cilium Hubble for flow tracing

✅ **If latency is node-to-node**

- Check cloud VPC routes
- Validate firewall rules
- Confirm correct subnet routing

In production, a network hiccup can bring down entire services — so learn these tools **before** your first 3 AM incident.

5. What if a Node Goes NotReady, but the Kubelet Logs Are Clean?

This is one of the most misleading situations:

- node status is NotReady
- but journalctl -u kubelet shows nothing alarming

✅ **Inspect node conditions**

- kubectl describe node <node>
  - look for DiskPressure
  - MemoryPressure
  - NetworkUnavailable

✅ **Check Kubernetes events**

- kubectl get events --field-selector involvedObject.kind=Node

✅ **Check container runtime**

- journalctl -u containerd
- docker info if Docker
- check disk usage in /var/lib/kubelet

✅ **Inspect CNI plugin**

- Logs for CNI pods
- bridge interfaces on the node

✅ **Check kube-proxy**

- kubectl logs -n kube-system <kube-proxy-pod>
- validate iptables rules

✅ **Node-level resource issues**

- top for load
- df -h for disk
- dmesg for kernel events

✅ **Cloud-specific**

- hypervisor maintenance
- autoscaler scale-downs
- cloud provider instance status

✅ **Remediate**

- cordon the node: kubectl cordon <node>
- drain:
- bash
- CopyEdit
- kubectl drain <node> --ignore-daemonsets --delete-emptydir-data
- replace or investigate offline

NotReady doesn’t always mean kubelet is broken — think bigger: runtime, network, or even the cloud node itself.

Conclusion

Kubernetes is an incredible platform, but on-call responsibilities can be stressful — especially when you don’t have a clear checklist to fall back on.

If you master these five scenarios, you’ll be far more confident, your team will trust you, and you’ll sleep a lot better when that next 2 AM alert hits.

Thanks Everyone!

Connect with me: Amit Singh