420+ DevOps Interview Questions

Learn With Shubham Praharaj

DevOps Interview Master Guide (Structured Questions Bank + What Interviewers Expect)

This document is a **complete DevOps interview question bank** designed to help you prepare for roles like:

✅ DevOps Engineer
✅ Cloud DevOps Engineer
✅ Platform Engineer
✅ Site Reliability Engineer (SRE)
✅ Kubernetes / Terraform / CI-CD Engineer

It is structured by **skills + difficulty level + production scenarios**, so you can revise quickly and answer confidently in interviews.

✅ How to Use This Guide (Fast Preparation Plan)

Step 1 — First Pass (Quick Scan)

Try answering each question in **60–90 seconds**.

Step 2 — Second Pass (Weak Areas Only)

Mark the ones where you struggle and revise again.

Step 3 — Scenario Practice

Focus on scenario-based questions (these are asked in senior roles).

✅ Best interview structure to answer **any question**:

Concept → Why → How it works → Example → Best Practices → Troubleshooting

---

📌 Skills Covered

✅ AWS DevOps
✅ SRE
✅ Kubernetes
✅ Jenkins
✅ Terraform
✅ Ansible
✅ Shell & Python Automation
✅ Monitoring, Logging & Incident Response
✅ Security & DevSecOps

📚 Table of Contents

- Intermediate
- Experienced

Section 4 — Jenkins Interview Questions (65)

- Beginner
- Intermediate / Advanced
- Expert

Section 5 — Terraform Interview Questions (70+)

- Beginner
- Intermediate
- Expert

Section 6 — Ansible Interview Questions (40)

- Basics
- Inventory
- Playbooks & Troubleshooting
- Variables & Modules
- Advanced

Section 7 — Shell & Python Interview Questions (50)

- Beginner to Expert
- Production Scenarios

✅ Section 1 — AWS DevOps Interview Questions (50)

✅ Basic Conceptual Level (1–10)

1. Explain the core principles of DevOps and its benefits in AWS

Answer (What interviewer expects):
DevOps improves delivery speed and stability by promoting:

- Automation (CI/CD + IaC)
- Continuous feedback
- Collaboration between Dev + Ops
- Monitoring-driven improvement

In AWS, DevOps becomes faster because AWS provides managed services like:
CloudWatch, CodePipeline, Auto Scaling, ECS/EKS, Lambda.

---

2. IaC vs IaaS

Answer:

- **IaaS:** AWS provides infrastructure resources like EC2, EBS, VPC.
- **IaC:** You manage infra using code (Terraform/CloudFormation), enabling reproducibility.

---

3. 3 main service categories in AWS

Answer:

- **Compute** (EC2, Lambda, ECS/EKS)
- **Storage** (S3, EBS, EFS)
- **Networking** (VPC, Route53, CloudFront)

---

4. Types of EC2 instances and choosing the right one

Answer:
EC2 instance families include:

- General purpose (balanced)
- Compute optimized (CPU heavy)
- Memory optimized (in-memory workloads)
- Storage optimized (high IOPS)
- GPU instances (ML/graphics)

Choice depends on workload: CPU usage, memory needs, traffic pattern, and budget.

---

5. Security Groups vs NACLs

Answer:

- **Security Group:** stateful firewall at instance level
- **NACL:** stateless firewall at subnet level

Security groups allow rules only, NACL supports allow + deny.

---

6. Benefits of VPC

Answer:
VPC provides:

- network isolation
- control of subnets and routing
- security using SG/NACL
- private connectivity (VPC endpoints)
- hybrid integration (VPN/Direct Connect)

---

7. S3 storage classes

Answer:
Common S3 classes:

- Standard
- Standard-IA
- One Zone-IA
- Glacier Instant Retrieval
- Glacier Flexible Retrieval
- Deep Archive

Used based on access frequency and retention.

---

8. CloudWatch usage

Answer:
CloudWatch provides:

- Metrics

- Logs
- Alarms
- Dashboards
- Events (EventBridge)

Used for monitoring CPU, latency, errors, and sending alerts.

---

9. AWS Lambda use-case

Answer:
Lambda runs code without managing servers.
Best for:

- event-based systems
- automation
- APIs
- scheduled tasks
- lightweight microservices

---

10. Autoscaling in AWS

Answer:
Auto Scaling increases or decreases compute capacity based on metrics like:
CPU, memory, request count, queue length.

Implemented using:

- Auto Scaling Groups + Launch templates
- Target tracking policies

✅ Advanced Conceptual Level (11–20)

11. CodePipeline vs CodeDeploy

Answer:

- **CodePipeline:** end-to-end CI/CD orchestration
- **CodeDeploy:** deployment engine (EC2, Lambda, ECS)

CodePipeline manages stages, CodeDeploy handles deployment strategies.

---

12. Serverless architecture: benefits + challenges

Answer:
✅ Benefits:

- no servers to manage
- scales automatically
- cost-efficient for event workloads

⚠️ Challenges:

- cold start latency
- debugging complexity
- vendor lock-in
- tracing distributed flows is harder

---

13. ECS vs EKS

Answer:

- ECS = simpler AWS-managed container orchestration
- EKS = Kubernetes-based (more flexibility, industry standard)

Choose ECS for simplicity, EKS for K8s ecosystem + portability.

---

14. Role of Terraform/CloudFormation in DevOps

Answer:
They implement Infrastructure as Code enabling:

- repeatable deployments
- reduced manual errors
- version-controlled infra changes
- faster environment creation

---

15. IaC testing approach

Answer:
IaC testing validates:

- syntax and formatting (linting)
- security checks (policy as code)
- unit/integration validation (test environments)

Tools: terraform validate, tflint, checkov, terratest.

---

16. Disaster recovery strategies in AWS

Answer:

- Backup & restore
- Pilot light
- Warm standby
- Active-active multi-region

Chosen based on RTO and RPO requirements.

---

17. IAM + VPC security best practices

Answer:

- least privilege roles
- avoid access keys (use IAM roles)
- enable MFA
- use private subnets + NAT
- use VPC endpoints
- logging with CloudTrail + Config

---

18. AWS cost optimization strategies

Answer:

- right-size EC2
- Spot and Savings Plans
- S3 lifecycle policies
- turn off unused resources

- set budgets + alerts
- use load testing to avoid overprovisioning

---

19. Observability for serverless

Answer:
Use:

- CloudWatch Logs Insights
- X-Ray tracing
- OpenSearch
- structured logging + correlation IDs

---

20. Blue/Green deployment in AWS

Answer:
Blue/Green deployments reduce downtime by running:

- Blue = current stable environment
- Green = new version

Traffic switches after validation using ALB or CodeDeploy.

✅ Intermediate Level (21–30)

21. Explain a DevOps project you worked on

Answer format:

- problem statement
- tools used
- pipeline flow
- deployment strategy
- monitoring + incident results
- what you improved

---

22. Infrastructure changes with minimal downtime

Answer:
Use:

- rolling updates
- blue/green
- canary releases
- feature flags
- database migrations carefully (backward compatible schema)

---

23. Ansible/Chef automation experience

Answer:
Use configuration management for:

- OS patching
- installing packages
- configuring services

- environment consistency
- repeatability across dev/stage/prod

---

24. Troubleshooting AWS deployments

Answer approach:
Check in this order:

- CloudWatch alarms/logs
- deployment pipeline logs
- IAM permissions
- security group/network routing
- resource limits/throttling

---

25. Monitoring AWS applications

Answer:
Use metrics:

- latency, error rate, throughput
- CPU/memory
- database connections
- queue depth

Tools: CloudWatch, X-Ray, OpenTelemetry, dashboards.

---

26. Experience with IaC scripts

Answer:
Mention:

- modules
- remote backend/state locking
- code reviews
- environment separation (dev/stage/prod)
- drift detection

---

27. Kubernetes usage in AWS

Answer:
EKS used for microservices:

- deployments
- ingress
- autoscaling
- secrets/configmaps
- logging/monitoring integration

---

28. CI/CD pipelines in AWS

Answer:
Typical flow:
Git → build → test → package → deploy → verify → rollback plan

Tools: CodePipeline, Jenkins, GitHub Actions, ArgoCD.

---

29. Collaboration with Dev & Security teams

Answer:

- shared pipeline ownership
- DevSecOps scanning in CI
- shared runbooks
- incident review meetings
- clear approval workflows

---

30. Incident response experience

Answer:

- detect → triage → mitigate → fix → postmortem
- rollback strategy
- root cause analysis
- preventive automation

✅ Expert Level (31–40)

31. Advanced AWS services usage

Examples:

- Step Functions for workflows
- Lambda Layers for shared dependencies
- custom CloudFormation resources

---

32. Encryption for sensitive data

Answer:
Use:

- KMS
- Secrets Manager
- TLS everywhere
- encryption at rest for EBS/RDS/S3
- key rotation policies

---

33. Serverless security best practices

Answer:

- least privilege IAM role
- restrict Lambda network with VPC
- validate inputs
- API Gateway rate limiting
- WAF integration
- logging + anomaly detection

---

34. HA and scalable web architecture

Answer:
Multi-AZ design:

- ALB

- Auto Scaling
- RDS Multi-AZ or Aurora
- CloudFront + caching
- disaster recovery plan

---

35. AWS performance optimization

Answer:

- caching (CloudFront, ElastiCache)
- reduce DB calls
- asynchronous design (SQS/Kafka)
- tune instance types and scaling triggers

---

36. Automated compliance & audits

Answer:
Use:

- AWS Config rules
- Security Hub
- GuardDuty
- IaC policy checks (OPA, Checkov)

---

37. Staying updated

Answer:

- AWS What's New
- architecture blogs and documentation
- hands-on labs
- internal knowledge sharing

---

38. Hard technical problem you solved

Best structure:

- impact
- root cause
- fix
- prevention
- lessons learned

---

39. Cost management strategy

Answer:

- cost explorer analysis
- tagging enforcement
- scheduled shutdown
- rightsizing
- savings plans
- reserved capacity for steady workloads

---

40. Building DevOps culture

Answer:

- encourage automation
- reduce blame culture
- shared responsibility
- documentation/runbooks
- standardization across teams

✅ Expert Production Scenarios (41–50)

41. Flash sale traffic surge causing outages

Answer:

- validate ALB target health
- check scaling triggers
- enable caching + CloudFront
- scale DB read replicas
- use queue for async traffic
- rollback quickly if release caused failure

---

42. Production database corrupted

Answer:

- stop writes
- restore from snapshot
- use point-in-time recovery (PITR)
- validate integrity
- introduce backup policy + deletion protections

---

43. Slow build times in CI/CD

Answer:

- parallelize builds
- enable caching dependencies
- use dedicated agents
- optimize tests (unit vs integration separation)
- remove unnecessary pipeline steps

---

44. Security vulnerability in public API

Answer:

- immediate mitigation (WAF rule, rate-limit, block attack vector)
- patch + deploy fix
- rotate credentials
- audit logs
- postmortem + prevention

---

45. Migrate legacy app to AWS

Answer:

- assessment (dependencies, DB, traffic)
- choose migration strategy (rehost/refactor)
- build staging infra using IaC
- testing + cutover plan
- rollback plan

---

46. High latency issue

Answer:

- check CloudWatch metrics
- ALB response time
- DB slow queries
- external API latency
- caching improvements + scaling

---

47. Unauthorized S3 access attempt

Answer:

- check CloudTrail logs
- block access via policy

- enforce MFA delete
- enable versioning
- rotate credentials and review IAM policies

---

48. Automate microservices deployment

Answer:
CI/CD design:

- build Docker images → push registry
- deploy to Kubernetes/ECS
- canary + rollback
- integrate tests + security scan
- add monitoring gates

---

49. AWS cost is too high

Answer:

- identify top cost services
- remove unused resources
- rightsizing + Spot
- S3 lifecycle
- database optimization
- enforce budgets + alerts

---

50. Adopting DevOps culture

Answer:

- start with CI + automation
- build shared pipelines
- define ownership
- improve collaboration between Dev/Ops
- introduce monitoring + fast feedback

SRE (SITE RELIABILITY ENGINEERING) — INTERVIEW QUESTIONS + ANSWERS (70+ Qs)

Beginner → Intermediate → Production Scenarios | Google Docs Ready Format

---

How to Use This SRE Section

✅ **Round 1:** Answer in 1–2 lines (quick recall)
✅ **Round 2:** Answer in 30–60 seconds (interview mode)
✅ **Round 3:** Practice scenario questions like incident calls

---

✅ PART 1 — SRE ABSOLUTE BEGINNER QUESTIONS (20)

1. What is SRE?

Answer: Site Reliability Engineering (SRE) is a discipline that applies software engineering principles to IT operations to improve reliability, scalability, and efficiency.

---

2. What is the primary goal of an SRE?

Answer: To ensure systems are **reliable, scalable, and efficient** while balancing the need for **feature velocity vs stability**.

---

3. What is the difference between DevOps and SRE?

Answer:

- **DevOps:** culture/practices to improve delivery and collaboration
- **SRE:** implementation model that uses engineering + reliability metrics to run production systems

---

4. What are the key responsibilities of an SRE?

Answer:

- Monitoring and alerting
- Incident response and root cause analysis
- Reliability engineering and automation
- Capacity planning and performance tuning
- On-call and operational excellence

---

5. What is reliability in SRE terms?

Answer: Reliability means a system behaves as expected under normal and failure conditions, meeting availability and performance targets.

---

6. What is Availability?

Answer: Availability is the percentage of time a service is operational and accessible to users (example: 99.9%).

---

7. What is Latency?

Answer: Latency is the time taken to serve a request (example: response time in milliseconds).

---

8. What is Throughput?

Answer: Throughput is the number of requests processed per unit time (RPS—requests per second).

---

9. What is Monitoring and why is it needed?

Answer: Monitoring measures service health through metrics/logs/traces to detect issues early and ensure reliability.

---

10. What is Alerting?

Answer: Alerting notifies engineers when systems breach defined thresholds (CPU high, error spikes, latency increase).

---

11. What is Incident Management?

Answer: The structured process of detecting, responding to, resolving, and learning from production incidents.

---

12. What is On-call in SRE?

Answer: Engineers rotate responsibility to respond to production issues when alerts trigger during or outside work hours.

---

13. What are runbooks?

Answer: Runbooks are documented step-by-step procedures to handle common incidents (restart service, rollback, clear queue).

---

14. Why is automation important in SRE?

Answer: Automation reduces human error, speeds up recovery, and improves operational efficiency.

---

15. What does “infrastructure as code” mean in SRE context?

Answer: Managing infra using version-controlled code to improve consistency, repeatability, and reliability.

---

16. What is “toil” in SRE?

Answer: Toil is repetitive manual work that produces no long-term value and should ideally be automated.

---

17. What does it mean to be “production focused”?

Answer: It means prioritizing reliability and customer impact over non-critical improvements.

---

18. Why are strong problem-solving skills required for SRE?

Answer: SREs must quickly diagnose system failures, reduce downtime, and restore services under pressure.

---

19. What is the role of collaboration for SRE?

Answer: SREs work with dev/security/product teams to improve system design, release safety, and operational readiness.

---

20. What is the difference between proactive and reactive work in SRE?

Answer:

- **Reactive:** responding to outages/incidents
- **Proactive:** preventing incidents (capacity planning, chaos testing, automation)

✅ PART 2 — SRE INTERMEDIATE QUESTIONS (20)

21. What are SLIs, SLOs, and SLAs?

Answer:

- **SLI (Indicator):** a metric (latency, error rate, availability)
- **SLO (Objective):** target value for SLIs (99.95% uptime)
- **SLA (Agreement):** contractual commitment with penalties

---

22. Give examples of SLIs.

Answer:

- Request success rate (2xx)
- p95/p99 latency
- Availability (uptime)
- Queue length
- CPU/memory saturation

---

23. What is an error budget?

Answer: The allowed amount of failure (downtime/errors) within an SLO period. If exhausted, new feature releases slow down until reliability improves.

---

24. How do error budgets help decision making?

Answer: They create a measurable balance between shipping features and maintaining stability.

---

25. What is MTTR?

Answer: Mean Time To Recovery—average time taken to restore service after a failure.

---

26. What is MTBF?

Answer: Mean Time Between Failures—average time between one failure and the next.

---

27. What is an RCA (Root Cause Analysis)?

Answer: A structured analysis of why an incident occurred and how to prevent it from recurring.

---

28. What should a good postmortem include?

Answer:

- What happened (timeline)
- Impact
- Root cause
- Contributing factors
- Fixes and preventive actions
- Lessons learned

---

29. What is “blameless postmortem”?

Answer: A postmortem culture focused on improving systems rather than blaming individuals.

---

30. What is capacity planning?

Answer: Forecasting needed resources (CPU/memory/network/DB capacity) based on traffic growth and workload patterns.

---

31. What is autoscaling and when is it helpful?

Answer: Automatically scaling resources up/down based on metrics like CPU/RPS. Helpful for variable traffic workloads.

---

32. What is the difference between vertical and horizontal scaling?

Answer:

- **Vertical:** bigger machine (scale-up)
- **Horizontal:** more machines (scale-out)

---

33. Explain redundancy and failover.

Answer:

- **Redundancy:** extra components to avoid single points of failure
- **Failover:** switching traffic to healthy standby resources during failures

---

34. What is High Availability (HA)?

Answer: Designing systems with redundancy so they remain operational even when parts fail.

---

35. What is disaster recovery (DR)?

Answer: A strategy to restore systems and data after major failures like region outage, data corruption, ransomware, etc.

---

36. What is RTO and RPO?

Answer:

- **RTO (Recovery Time Objective):** max acceptable downtime
- **RPO (Recovery Point Objective):** max acceptable data loss window

---

37. What are common DR strategies?

Answer:

- Backup & Restore
- Pilot Light
- Warm Standby
- Active-Active / Multi-region

---

38. What is observability?

Answer: Ability to understand internal system behavior using **metrics, logs, and traces** (plus events).

---

39. Monitoring vs Observability?

Answer:

- **Monitoring:** detects known problems
- **Observability:** helps debug unknown problems using deep signals

---

40. What is distributed tracing?

Answer: Tracking a request as it travels through multiple microservices (example tools: Jaeger, AWS X-Ray).

✅ PART 3 — SRE SITUATION-BASED TECHNICAL QUESTIONS (31)

41. A production outage occurs. Walk through your response.

Answer:

1. Acknowledge incident + alert stakeholders
2. Assess severity and impact
3. Stop the bleeding (rollback, scale, failover)
4. Identify root cause (logs/metrics/traces)
5. Apply permanent fix
6. Postmortem + prevention actions

---

42. How would you design a CI/CD pipeline for microservices?

Answer:

- Per-service pipelines + versioned artifacts
- Automated tests (unit → integration → smoke)
- Security scans (SAST, dependency, container)
- Deploy to staging, then prod
- Canary/blue-green rollout
- Observability + automated rollback

---

43. How would you handle a cloud security breach?

Answer:

1. Contain (block access, isolate systems)
2. Identify entry vector (IAM, keys, public endpoints)
3. Rotate secrets + revoke credentials
4. Collect logs for forensics
5. Patch vulnerability
6. Improve guardrails (least privilege, WAF, SIEM alerts)

---

44. Application response time increases significantly. Steps?

Answer:

- Check latency metrics (p95/p99)
- Identify layer: DNS/CDN → LB → app → DB
- Look for CPU/memory saturation
- Check DB query latency / locks
- Analyze traces/logs
- Scale or optimize bottleneck

---

45. How do you ensure database high availability in cloud?

Answer:

- Multi-AZ replication
- Automated backups
- Read replicas
- Failover testing
- Connection pooling
- Monitor slow queries + locks

---

46. How do you implement blue-green deployment in Kubernetes?

Answer:

- Deploy new version alongside old
- Route traffic via service selector or ingress weights
- Validate readiness + metrics
- Switch traffic fully
- Rollback fast by routing back

---

47. How do you automate infra provisioning with Terraform?

Answer:

- Use modular Terraform design
- Remote state + locking
- CI pipeline runs validate/plan/apply
- Policy checks (security scanning)
- Separate workspaces/environments

---

48. You must improve scalability of an application. What do you do?

Answer:

- Identify bottleneck with profiling and metrics
- Add caching (Redis/CDN)
- Scale horizontally

- Optimize DB (indexes, connection pooling)
- Async workloads using queues
- Load testing + capacity planning

---

49. How would you integrate security scanning into CI/CD?

Answer:

- SAST scans on pull requests
- Dependency scanning
- Container image scanning
- IaC scanning (Terraform/Helm)
- Secrets scanning
- Block merges on critical issues

---

50. How do you manage secrets in production?

Answer:
Use Vault / AWS Secrets Manager / Parameter Store + rotate keys regularly and avoid putting secrets in logs or Git.

---

51. Deploying a feature requiring changes in multiple services—how do you coordinate?

Answer:

- Backward-compatible APIs
- Feature flags
- Versioned contract testing
- Gradual rollout (canary)
- Rollback plan ready
- Communication + deployment checklist

---

52. Describe your disaster recovery plan approach.

Answer:

- Define RTO/RPO
- Identify critical systems
- Backup strategy + restore drills
- Multi-region if required
- Automate failover steps
- Run DR tests quarterly

---

53. How do you monitor containerized workloads?

Answer:

- Metrics: CPU/memory, restarts, latency
- Logs: centralized logging
- Traces: distributed tracing

- Alerts: error rate, saturation
- Dashboard: service health overview

---

54. Migrating a legacy application to cloud—main challenges?

Answer:

- Dependencies and hidden configs
- Data migration risks
- Security model shift
- Networking complexity
- Performance tuning
- Downtime avoidance strategy

---

55. How do you optimize cloud costs without impacting reliability?

Answer:

- Right-sizing
- Autoscaling
- Reserved/spot strategy
- Storage lifecycle policies
- Remove unused resources

- Use managed services wisely
- Monitor cost anomalies

---

56. High CPU alerts in production—how do you handle?

Answer:

- Confirm impact (latency/error increase?)
- Identify pod/instance consuming CPU
- Check traffic patterns
- Scale horizontally
- Profile app for loops/memory leak
- Add caching or optimize queries

---

57. How do you implement automated testing in CI/CD?

Answer:
Unit → Integration → Security → Performance → Smoke tests before production promotion.

---

58. Network outage affecting cloud connectivity—your response?

Answer:

- Confirm scope (local vs region-wide)
- Failover to alternate region/VPN

- Check routing/DNS/peering
- Engage provider status + internal network team
- Restore + validate + postmortem

---

59. Explain autoscaling in real systems.

Answer: Scale based on CPU/RPS/queue depth, but ensure DB/cache layers can handle scaling too.

---

60. How do you ensure compliance in cloud infra?

Answer:

- Policy-as-code
- Audit logs
- Encryption
- IAM least privilege
- Automated compliance scans
- Regular security reviews

---

61. Blue-green vs Canary—when to use which?

Answer:

- **Blue-green:** best for fast switch/rollback
- **Canary:** best for gradual safe rollout with real users

---

62. How do you ensure traceability in CI/CD pipelines?

Answer:

- Git commit tagging
- Build IDs linked to deployments
- Artifact versioning
- Audit trails (who deployed what, when)

---

63. Network architecture change without disrupting production—approach?

Answer:

- Deploy changes in parallel
- Use staged rollout
- Validate with test traffic
- Use feature flags and fallback routes

---

64. What is capacity planning and how do you do it?

Answer:
Use traffic trends, load testing, resource utilization history, and forecast growth to prevent future bottlenecks.

---

65. Intermittent connectivity between services—how do you debug?

Answer:

- Check DNS resolution
- Network policies / SG rules
- Packet loss metrics
- Service discovery issues
- Timeouts and retries
- Trace request paths

---

66. Critical security patch needed everywhere—your execution plan?

Answer:

- Prioritize by severity
- Patch staging first
- Automate rollout using CI/CD
- Use rolling updates
- Validate monitoring
- Document and confirm completion

---

67. How do you test a DR plan?

Answer:
By running scheduled DR drills: restore backups, failover traffic, validate application, verify RTO/RPO.

---

68. Cloud costs too high—what's your step-by-step approach?

Answer:

- Identify top cost services
- Find idle resources
- Right-size compute
- Optimize storage
- Use commitments (reserved instances)
- Improve architecture (caching/CDN)

---

69. Multi-region redundancy strategy?

Answer:

- Active-passive or active-active
- Global load balancing (Route53/Traffic manager)
- Multi-region DB replication
- Automated failover

---

70. Dependency/version management in CI/CD pipelines—how?

Answer:

- Lock versions
- Use artifact repositories
- Use semantic versioning
- Automate dependency checks and upgrades
- Maintain rollback compatibility

---

71. Integrating third-party APIs reliably—how?

Answer:

- Retries with exponential backoff
- Circuit breaker pattern
- Timeouts
- Monitoring of external latency
- Fallback responses
- Rate limiting

✅ SRE QUICK REVISION CHEAT SHEET (Interview Gold)

Core Metrics

- Availability
- Latency (p95/p99)
- Error Rate
- Saturation (CPU/memory/disk/threads)
- Traffic (RPS)

Key Concepts

- SLI / SLO / SLA
- Error Budget
- Toil
- MTTR / MTBF
- Postmortem
- Observability (Metrics + Logs + Traces)

Golden Signals (Google SRE Standard)

✅ Latency • Traffic • Errors • Saturation

✅ KUBERNETES INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(Beginner → Intermediate → Advanced → Production Scenarios)

Kubernetes Interview Guide (28 Q&A)

Section A — Beginner Level (1–10)

1. What is Kubernetes and why is it used?

Answer: Kubernetes is an open-source container orchestration platform used to deploy, scale, and manage containerized applications automatically across a cluster of machines.

---

2. What is a Kubernetes Cluster?

Answer: A Kubernetes cluster is a group of machines (nodes) working together to run containerized applications. It consists of a **control plane** and **worker nodes**.

---

3. What is a Node in Kubernetes?

Answer: A node is a machine (VM or physical server) that runs application workloads. Nodes host **Pods**, and are managed by the control plane.

---

4. What is a Pod in Kubernetes?

Answer: A Pod is the smallest deployable unit in Kubernetes. It contains one or more containers sharing the same **network namespace** and **storage volumes**.

---

5. What is a Deployment in Kubernetes?

Answer: A Deployment manages a set of identical Pods and ensures the desired number of replicas are running. It supports **rolling updates** and **rollbacks**.

---

6. What is a ReplicaSet?

Answer: A ReplicaSet ensures a specific number of Pod replicas are running. Deployments use ReplicaSets internally.

---

7. What is a Service in Kubernetes?

Answer: A Service provides a stable network endpoint (DNS + IP) to expose Pods and load balance traffic across them.

---

8. What are the different types of Kubernetes Services?

Answer:

- **ClusterIP:** internal access only (default)
- **NodePort:** exposes service on node's port
- **LoadBalancer:** uses cloud provider LB
- **ExternalName:** maps service to external DNS name

---

9. What is a Namespace in Kubernetes?

Answer: A Namespace is a logical separation within a cluster used to organize resources and manage access boundaries (dev, stage, prod).

---

10. What are labels and selectors in Kubernetes?

Answer:

- **Labels:** key-value metadata attached to resources
- **Selectors:** used by Services/Deployments to match Pods by labels

---

Section B — Intermediate Level (11–20)

11. What is a ConfigMap?

Answer: A ConfigMap stores non-sensitive configuration data such as environment variables or config files, used by Pods at runtime.

---

12. What is a Secret in Kubernetes?

Answer: A Secret stores sensitive data like passwords, tokens, and keys. It can be mounted into Pods or exposed as environment variables.

---

13. What is the difference between ConfigMap and Secret?

Answer:

- ConfigMap = non-sensitive data
- Secret = sensitive data (base64-encoded and managed more securely)

---

14. What is Ingress in Kubernetes?

Answer: Ingress is an API resource used to manage external HTTP/HTTPS access to services, typically with routing rules and TLS termination.

---

15. What is an Ingress Controller?

Answer: An Ingress Controller is the actual component (like NGINX Ingress Controller) that implements the Ingress rules and routes traffic.

---

16. What is the difference between a StatefulSet and a Deployment?

Answer:

- **Deployment:** for stateless workloads

- **StatefulSet:** for stateful apps (stable pod identity, stable storage)

---

17. What is a Persistent Volume (PV) and Persistent Volume Claim (PVC)?

Answer:

- **PV:** storage resource in the cluster
- **PVC:** a request for storage by a Pod
  PVC binds to PV based on size and access mode.

---

18. What are liveness and readiness probes?

Answer:

- **Liveness probe:** checks if container is alive (restart if fails)
- **Readiness probe:** checks if container is ready to serve traffic

---

19. What is a DaemonSet in Kubernetes?

Answer: A DaemonSet ensures one Pod runs on each node (or selected nodes), commonly used for logging/monitoring agents.

---

20. What is a Job and a CronJob?

Answer:

- **Job:** runs a task until completion
- **CronJob:** runs Jobs on a schedule (like Linux cron)

Section C — Advanced Level (21–24)

21. How does Kubernetes perform rolling updates?

Answer: Kubernetes gradually replaces old Pods with new Pods while maintaining availability. It uses `maxSurge` and `maxUnavailable` settings.

---

22. How do you rollback a failed deployment?

Answer:
You can rollback using:
`kubectl rollout undo deployment <deployment-name>`
It restores the previous ReplicaSet revision.

---

23. What is Horizontal Pod Autoscaler (HPA)?

Answer: HPA automatically scales the number of Pod replicas based on CPU/memory usage or custom metrics.

---

24. What is Cluster Autoscaler?

Answer: Cluster Autoscaler automatically scales worker nodes up/down when Pods can't be scheduled due to insufficient resources.

Section D — Production Scenario Questions (25–28)

25. A Pod is stuck in CrashLoopBackOff. How do you troubleshoot it?

Answer:
Steps:

1. Check logs: `kubectl logs <pod>`

2. Describe pod: `kubectl describe pod <pod>`
3. Verify env variables, configmaps, secrets
4. Validate image, command, and args
5. Check probes (readiness/liveness causing restarts)

---

26. A Pod is Pending for a long time. What could be the reasons?

Answer:
Common causes:

- Insufficient CPU/memory in nodes
- Node selectors/taints not matching
- PVC not bound
- Cluster Autoscaler not adding nodes
  Check using: `kubectl describe pod <pod>`

---

27. Service is not reachable from outside. What checks do you perform?

Answer:

- Service type correct? (NodePort / LoadBalancer / Ingress)
- Ingress controller running?
- DNS pointing correctly?
- Security Groups / firewall open?
- Pod labels match Service selectors?

---

28. High latency in production Kubernetes app. What's your approach?

Answer:

- Check resource saturation (CPU/memory throttling)
- Check HPA scaling behavior
- Analyze metrics + traces
- Check DB latency and connection pool
- Inspect networking (DNS, ingress, service mesh)
- Add caching or optimize queries

✅ Kubernetes Quick Revision (One-Liners)

- **Pod:** smallest unit
- **Deployment:** manages stateless pods
- **StatefulSet:** stable identity + storage
- **Service:** stable endpoint + load balancing
- **Ingress:** HTTP routing from outside
- **HPA:** scales pods
- **Cluster Autoscaler:** scales nodes
- **ConfigMap vs Secret:** config vs sensitive data
- **PV/PVC:** storage supply vs storage request

✅ JENKINS INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(Beginner → Intermediate → Advanced → Production Scenarios)

Jenkins Interview Guide (28 Q&A)

Section A — Beginner Level (1–10)

1. What is Jenkins and why is it used?

Answer: Jenkins is an open-source automation server used to implement **CI/CD pipelines**. It helps automate building, testing, and deploying software reliably.

---

2. What is the difference between CI and CD?

Answer:

- **CI (Continuous Integration):** frequent code merges + automated builds/tests
- **CD (Continuous Delivery/Deployment):** automated delivery and release to environments (staging/production)

---

3. What is a Jenkins Job?

Answer: A Jenkins job is a task configured in Jenkins such as building code, running tests, or deploying an application.

---

4. What is Jenkins Pipeline?

Answer: A Jenkins Pipeline is a set of automated steps written as code to define how software is built, tested, and deployed.

---

5. What are the two types of Jenkins Pipelines?

Answer:

- **Declarative Pipeline:** structured and easier to read
- **Scripted Pipeline:** flexible and uses Groovy scripting heavily

---

6. What is a Jenkinsfile?

Answer: A Jenkinsfile is a text file containing pipeline code, stored in the repository. It enables **Pipeline-as-Code** and version control.

---

7. What is a Jenkins Agent?

Answer: A Jenkins agent is a machine that runs builds and pipeline stages. Agents help distribute workloads and scale CI/CD execution.

---

8. What is a Jenkins Controller (Master)?

Answer: The Jenkins controller manages jobs, schedules builds, handles UI/API, stores configs, and coordinates with agents.

---

9. What is a build artifact in Jenkins?

Answer: A build artifact is an output file generated during pipeline execution, such as `.jar`, `.war`, Docker image tags, reports, etc.

---

10. What is the workspace in Jenkins?

Answer: Workspace is a directory on the Jenkins node where project files are checked out and build operations happen.

Section B — Intermediate Level (11–20)

11. How does Jenkins integrate with Git?

Answer: Jenkins connects to Git repositories via plugins and can automatically pull code using webhook triggers or polling.

---

12. What are Jenkins build triggers?

Answer: Build triggers define when jobs run, such as:

- SCM polling
- Webhooks (GitHub/GitLab)
- Scheduled (cron)
- Upstream job completion
- Manual triggers

---

13. What is the difference between Freestyle job and Pipeline?

Answer:

- **Freestyle:** UI-based, easier but limited and not scalable
- **Pipeline:** code-driven, reusable, version-controlled, best for real CI/CD

---

14. What are Jenkins plugins and why are they important?

Answer: Plugins extend Jenkins functionality (Git, Docker, Kubernetes, Slack alerts, Maven, etc.). Jenkins is plugin-driven.

---

15. How do you securely store credentials in Jenkins?

Answer: Use **Jenkins Credentials Manager** to store secrets such as tokens, usernames/passwords, SSH keys, and API keys.

---

16. How do you use credentials inside a Jenkins pipeline?

Answer: Use `withCredentials()` to access secrets safely without exposing them in logs.

Example:

- Inject username/password
- Inject secret text token
- Inject SSH private key

---

17. What are Jenkins Shared Libraries?

Answer: Shared Libraries allow reusing common pipeline code across multiple repositories (for standard stages like build/test/deploy).

---

18. How do you handle environment variables in Jenkins?

Answer: You can set environment variables:

- globally in Jenkins
- per-job configuration
- in Jenkinsfile using `environment {}` block

---

19. What is a parallel stage in Jenkins Pipeline?

Answer: Parallel stages run multiple tasks at the same time to speed up pipelines, such as running tests for multiple modules in parallel.

---

20. How do you archive artifacts in Jenkins?

Answer: Jenkins stores artifacts using `archiveArtifacts` so they can be downloaded later from build history.

Example use cases:

- Build outputs
- Reports
- Logs
- Deployment packages

Section C — Advanced Level (21–24)

21. How do you implement rollback in Jenkins pipelines?

Answer:
Rollback strategy depends on the deployment type:

- redeploy previous build artifact
- rollback Kubernetes deployment
- rollback Helm release
- use Blue/Green or Canary strategy

Rollback must be automated and tested like normal deployment.

---

22. How do you build Docker images using Jenkins?

Answer: Jenkins can build Docker images using:

- Docker CLI inside pipeline
- Docker plugin
- Kaniko (for Kubernetes environments)
- BuildKit

Typical flow: build → tag → scan → push to registry.

---

23. How do you integrate Jenkins with Kubernetes?

Answer: Jenkins can run dynamic agents as Kubernetes Pods using Jenkins Kubernetes Plugin. It improves scalability and reduces cost.

---

24. How do you secure Jenkins in production?

Answer: Key Jenkins security practices:

- enable authentication (SSO/LDAP/OAuth)
- role-based access control (RBAC)
- limit admin users
- restrict script approvals
- rotate credentials
- keep Jenkins/plugins updated
- disable anonymous access
- secure secrets using Vault (optional)

Section D — Production Scenario Questions (25–28)

25. A Jenkins job fails randomly (flaky builds). How do you fix it?

Answer:
Steps:

1. Identify pattern: agent-specific or global issue
2. Check logs for resource/timeouts
3. Verify dependency versions
4. Add retry logic only for flaky stages
5. Make build idempotent
6. Stabilize test environment (mock dependencies if needed)

---

26. Your Jenkins pipeline is very slow. What do you optimize?

Answer:

- use parallel stages
- reduce checkout depth (shallow clone)
- cache dependencies (Maven, npm, pip)
- use faster agents (auto-scaling)
- reduce Docker build time via layer caching
- split pipeline into stages with clear boundaries

---

27. Jenkins controller went down. How do you recover quickly?

Answer:
Best-practice recovery steps:

- restore from backup (JENKINS_HOME)
- store Jenkins home on persistent storage
- use Infrastructure-as-Code to recreate Jenkins
- keep plugins version-locked
- use external secrets storage
- implement disaster recovery plan

---

28. Secrets got exposed in Jenkins logs. What do you do immediately?

Answer:

1. Rotate the exposed credentials immediately
2. Remove logs or restrict access to them
3. Enable masking of secrets
4. Update pipeline to use `withCredentials()`
5. Add security audit and access review
6. Implement secret scanning in CI/CD

✅ Jenkins Quick Revision (One-Liners)

- **Jenkinsfile:** pipeline as code

- **Controller:** manages scheduling and UI
- **Agent:** runs builds
- **Credentials Manager:** stores secrets safely
- **Shared Libraries:** reuse pipeline code
- **Artifacts:** stored build outputs
- **Triggers:** webhook, schedule, SCM polling
- **Security:** RBAC + updates + limited plugins

✅ TERRAFORM INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(Beginner → Intermediate → Advanced → Production Scenarios)

Terraform Interview Guide (28 Q&A)

Section A — Beginner Level (1–10)

1. What is Terraform?

Answer: Terraform is an **Infrastructure as Code (IaC)** tool used to provision and manage infrastructure across cloud providers (AWS, Azure, GCP) using declarative configuration files.

---

2. What is Infrastructure as Code (IaC)?

Answer: IaC is managing infrastructure using code instead of manual processes. It allows infrastructure to be:
✅ repeatable • ✅ version-controlled • ✅ automated • ✅ auditable

---

3. Is Terraform declarative or imperative?

Answer: Terraform is **declarative**, meaning you describe the desired final state and Terraform figures out how to reach it.

---

4. What is a Terraform Provider?

Answer: A provider is a plugin that lets Terraform interact with APIs of platforms like AWS, Kubernetes, GitHub, etc.

Example providers:

- `aws`
- `azurerm`
- `google`
- `kubernetes`

---

5. What is a Terraform Resource?

Answer: A resource is a component that Terraform creates and manages, such as:

- EC2 instance
- S3 bucket
- VPC
- IAM role
- Kubernetes deployment

---

6. What are Terraform Variables used for?

Answer: Variables make Terraform code reusable and flexible across environments like dev/stage/prod.

Example use cases:

- instance type
- region
- environment name
- CIDR blocks

---

7. What are Terraform Outputs?

Answer: Outputs allow Terraform to print useful values after execution, such as:

- public IP address
- DNS name
- VPC ID
- database endpoint

---

8. What is the purpose of `terraform init`?

Answer: It initializes Terraform by:

- downloading providers
- setting up backend
- preparing modules
- creating `.terraform/`

---

9. What is the purpose of `terraform plan`?

Answer: It generates an execution plan that shows what Terraform will create, update, or destroy **before applying changes**.

---

10. What is the purpose of `terraform apply`?

Answer: It executes the plan and applies the changes to create or update infrastructure.

Section B — Intermediate Level (11–20)

11. What is a Terraform State file?

Answer: Terraform state (`terraform.tfstate`) stores the mapping between resources in your cloud and your configuration code.

Terraform uses state to track:
✅ what exists • ✅ what changed • ✅ what to update next

---

12. Where should Terraform state be stored in real projects?

Answer: In production, use **remote state** such as:

- S3 + DynamoDB lock (AWS)
- Terraform Cloud
- Azure Storage
- GCS bucket

Remote state ensures collaboration and safety.

---

13. What is state locking and why is it important?

Answer: State locking prevents multiple users/pipelines from modifying infrastructure at the same time, avoiding corruption.

Example: DynamoDB table used for locking in AWS.

---

14. What is a Terraform Module?

Answer: A module is a reusable set of Terraform files grouped together, such as:

- VPC module
- EKS module
- S3 module
- ECS module

Modules improve maintainability and reuse.

---

15. What is the difference between `count` and `for_each`?

Answer:

- `count`: best for identical resources using numeric indexes
- `for_each`: best for unique resources using keys (maps/sets)

✅ `for_each` is generally safer for long-term changes.

---

16. What are Terraform Data Sources?

Answer: Data sources allow Terraform to **read existing infrastructure** without creating it.

Example: Fetch existing VPC ID, subnet IDs, AMI ID, etc.

---

17. What is the difference between `terraform refresh` and plan?

Answer:

- **refresh:** syncs state with real infra
- **plan:** compares config vs state and shows changes

(Note: In newer workflows, refresh behavior is handled as part of plan/apply.)

---

18. What is drift in Terraform?

Answer: Drift happens when real infrastructure changes outside Terraform (manual changes), causing mismatch between Terraform state and reality.

---

19. How do you detect and fix drift?

Answer:
✅ Detect drift: `terraform plan`
✅ Fix drift: `terraform apply` (reconcile)
✅ Prevent drift: enforce IaC-only changes using IAM policies and governance

---

20. What is the Terraform Backend?

Answer: Backend defines where state is stored and how state operations are performed.

Common backend example:

- S3 backend for state
- DynamoDB for locking

Section C — Advanced Level (21–24)

21. How do you handle secrets in Terraform securely?

Answer: Best practices:

- avoid putting secrets in `.tfvars` in Git
- use secret managers (AWS Secrets Manager / Vault)
- mark values as `sensitive = true`
- restrict state file access
- encrypt remote state storage

---

22. What are Terraform Workspaces and when do you use them?

Answer: Workspaces allow using the same codebase for multiple environments by keeping separate state files.

Examples:

- `dev`
- `stage`
- `prod`

✅ Good for simple setups
⚠️ For large organizations, separate state and repos are often better.

---

23. What is the lifecycle block in Terraform?

Answer: The lifecycle block controls resource behavior.

Common lifecycle settings:

- `prevent_destroy = true` (protect critical resources)

- `ignore_changes` (ignore specific changes like tags)

---

24. How do you manage infrastructure dependencies in Terraform?

Answer: Terraform automatically builds dependencies through references.

Example:

- subnet depends on VPC
- EC2 depends on subnet/security group

You can also force dependency using:

- `depends_on`

Section D — Production Scenario Questions (25–28)

25. Terraform apply failed halfway. What happens and how do you recover?

Answer: Terraform may leave partial infrastructure created.

Recovery steps:

1. Check error logs
2. Fix the issue (permissions/quota/provider bug)
3. Run `terraform plan`
4. Run `terraform apply` again
5. If resource is broken, delete carefully or use `terraform taint` / recreate

---

26. Two engineers ran Terraform at the same time. What can break?

Answer: It may cause:

- corrupted state file
- conflicting resource updates
- unexpected deletions/overwrites

Fix: implement remote state + locking (example: DynamoDB lock).

---

27. Your Terraform state file got exposed. What do you do?

Answer: Immediate actions:

1. Treat it as a security incident
2. Rotate exposed credentials/secrets immediately
3. Restrict backend access (IAM)
4. Encrypt state storage (S3 SSE-KMS)
5. Move sensitive secrets to secret manager
6. Enable logging/auditing

---

28. You want a safe and controlled Terraform deployment in CI/CD. How?

Answer: Recommended CI/CD approach:
✅ `terraform fmt` (format check)
✅ `terraform validate`
✅ `terraform plan` and store the plan artifact
✅ manual approval for production
✅ `terraform apply` with approval
✅ use remote state + locking
✅ restrict privileges using IAM roles

✅ Terraform Quick Revision (One-Liners)

- **Terraform:** IaC tool for provisioning infra
- **Provider:** connects Terraform to cloud API
- **State:** mapping of code ↔ infra
- **Remote state:** collaboration + security
- **Locking:** prevents state corruption
- **Modules:** reusable infra building blocks
- **Drift:** manual infra changes outside Terraform
- **Workspaces:** multiple environments, separate state

✅ ANSIBLE INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(Beginner → Intermediate → Advanced → Production Scenarios)

Ansible Interview Guide (28 Q&A)

Section A — Beginner Level (1–10)

1. What is Ansible?

Answer: Ansible is an **agentless configuration management and automation tool** used for:
✅ configuration management • ✅ application deployment • ✅ orchestration • ✅ provisioning

It uses **SSH (Linux)** and **WinRM (Windows)** to connect to machines.

---

2. Why is Ansible called agentless?

Answer: Because Ansible does **not require any agent/software** to be installed on the target servers.
Only the control node needs Ansible installed.

---

3. What is an Ansible Control Node?

Answer: The **Control Node** is the machine where Ansible is installed and from which we execute automation tasks.

---

4. What is an Ansible Managed Node?

Answer: Managed nodes are the target machines/servers that Ansible configures using playbooks/modules.

---

5. What is an Inventory in Ansible?

Answer: Inventory is a list of servers (hosts) Ansible will manage.

It can be:

- **Static** (INI/YAML file)
- **Dynamic** (AWS / Azure / Kubernetes inventory sources)

---

6. What is a Playbook in Ansible?

Answer: A playbook is a YAML file containing a set of tasks that defines automation steps.

A playbook runs:
Play → Tasks → Modules → Results

---

7. What is an Ansible Module?

Answer: Modules are reusable units of work used in tasks, such as:

- `yum`, `apt` (packages)
- `copy`, `template` (files)
- `service`, `systemd` (services)
- `user` (users)
- `shell`, `command` (commands)

---

8. What is the difference between `command` and `shell` module?

Answer:

- `command`: does **NOT** support shell features like pipes, redirects, variables
- `shell`: supports shell features like `|`, `&&`, redirection `>`

✅ Best practice: use `command` when possible (safer).

---

9. What are Ansible Facts?

Answer: Facts are system details automatically collected from managed nodes, like:

- OS type
- IP address
- CPU/memory
- hostname

Facts are stored inside `ansible_facts`.

---

10. What is idempotency in Ansible?

Answer: Idempotency means running the same playbook multiple times results in the **same state**, without unnecessary changes.

✅ Example: installing nginx repeatedly won't reinstall it again and again.

Section B — Intermediate Level (11–20)

11. What are Roles in Ansible?

Answer: Roles are a way to organize Ansible code into structured reusable components such as:

- tasks
- handlers
- templates
- vars
- defaults
- files

Roles make automation clean and reusable in projects.

---

12. What are Handlers in Ansible?

Answer: Handlers are tasks triggered only when a change happens.

Example: restart nginx only if config changes.

✅ Handlers run at the end of a play unless flushed explicitly.

---

13. What is Ansible Vault?

Answer: Ansible Vault is used to **encrypt secrets** like:

- passwords
- API keys
- certificates
- sensitive variables

✅ Prevents secret leakage in Git repositories.

---

14. What is the difference between Variables, Defaults, and Vars?

Answer:

- **defaults/** → lowest priority, safe fallback
- **vars/** → higher priority, overrides defaults
- Variables can also come from: inventory, CLI, facts, etc.

✅ This determines which variable wins.

---

15. What are Tags in Ansible?

Answer: Tags allow running specific tasks only.

Example use cases:

- run only deployment tasks
- skip database tasks
- run only “install” section

---

16. What is a Jinja2 template in Ansible?

Answer: Jinja2 templates allow inserting variables dynamically in config files.

Example: generate an Nginx config file based on environment values.

---

17. How do you run playbooks on only one host in a group?

Answer: You can limit execution using:

- `--limit hostname`
- or limit to a subgroup pattern

Example: run only on one server during testing.

---

18. What is the difference between `vars_files` and inventory variables?

Answer:

- `vars_files`: variables stored in separate YAML files
- inventory variables: variables directly attached to hosts/groups

Use inventory variables for environment-specific configuration.

---

19. What is `become` in Ansible?

Answer: `become` is used for privilege escalation (like sudo).

Example: install packages as root.

✅ Very common in Linux automation.

---

20. How do you check syntax and validate a playbook before running?

Answer:

- syntax check: `--syntax-check`
- dry-run simulation: `--check`
- show changes: `--diff`

✅ Prevents breaking production accidentally.

Section C — Advanced Level (21–24)

21. What is a Dynamic Inventory and why is it useful?

Answer: Dynamic inventory automatically fetches host details from sources like:

- AWS EC2 instances
- Kubernetes clusters
- Azure VMs
- CMDB tools

✅ Useful when servers scale up/down frequently.

---

22. What is the difference between `include_tasks` and `import_tasks`?

Answer:

- `import_tasks`: static, loaded at playbook parsing time
- `include_tasks`: dynamic, loaded during runtime

✅ Use include when conditions decide tasks at runtime.

---

23. How do you handle failures in Ansible playbooks?

Answer: Common strategies:

- `ignore_errors: yes` (only if safe)
- `failed_when:` for custom failure logic
- `retries:` and `until:` for retry loops
- `block / rescue / always` for error recovery flows

---

24. How do you run tasks in parallel in Ansible?

Answer: Ansible runs tasks in parallel by default (based on forks).

You can tune performance using:

- `forks` settings
- `serial` for controlled batch rollout

✅ Useful for safe deployments (like 10 servers at a time).

Section D — Production Scenario Questions (25–28)

25. Scenario: A playbook works in staging but fails in production. What do you check?

Answer: Check common mismatches:
✅ different OS version / packages
✅ different permissions
✅ firewall / DNS issues
✅ missing environment variables

✅ secrets differences (Vault)
✅ inventory group variables incorrect

Fix by adding validation tasks, environment checks, and strict inventories.

---

26. Scenario: You ran a playbook and it broke a service. How do you rollback safely?

Answer: Rollback strategies:

- maintain backup configs before applying changes
- use versioned deployments (blue/green)
- use Ansible to restore previous config + restart service
- deploy in batches using `serial`

✅ Always test rollback automation before production.

---

27. Scenario: You need to deploy updates with zero downtime using Ansible. How?

Answer: Options:
✅ rolling deployment (batch servers)
✅ load balancer drain + update + reattach
✅ health checks after each batch
✅ `serial` and handlers to control restart timing
✅ canary release to small percentage first

---

28. Scenario: How do you secure Ansible in real organizations?

Answer: Best practices:

- use Ansible Vault for secrets
- restrict SSH keys and sudo access

- use least privilege accounts
- store playbooks in Git with review process
- use centralized logging/audit trails
- separate inventories per environment (dev/stage/prod)

✅ Ansible Quick Revision (One-Liners)

- **Ansible:** automation & configuration management tool
- **Agentless:** no software needed on managed nodes
- **Inventory:** list of servers
- **Playbook:** YAML automation workflow
- **Module:** unit of work
- **Idempotent:** safe repeated runs
- **Handlers:** run only when notified
- **Vault:** encrypt secrets
- **Roles:** reusable structure
- **Tags:** run selected tasks only

✅ KUBERNETES INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(Beginner → Intermediate → Advanced → Production Scenarios)

Kubernetes Interview Guide (28 Q&A)

Section A — Beginner Level (1–10)

1. What is Kubernetes and why is it used?

Answer: Kubernetes (K8s) is an **open-source container orchestration platform** used to:
✅ deploy containerized applications
✅ scale applications automatically
✅ manage self-healing (restart, reschedule)
✅ perform rolling updates and rollbacks
✅ handle service discovery & load balancing

---

2. What problem does Kubernetes solve?

Answer: Kubernetes solves problems such as:

- managing large numbers of containers
- maintaining application availability
- scaling services reliably
- automating deployments and infrastructure coordination

Without Kubernetes, managing containers across multiple servers becomes manual and error-prone.

---

3. What is a Kubernetes Cluster?

Answer: A Kubernetes cluster is a group of machines that run containerized workloads.
It includes:
✅ **Control Plane (Master Components)**
✅ **Worker Nodes (where apps run)**

---

4. What is a Node in Kubernetes?

Answer: A node is a machine (VM or physical server) that runs application workloads.
Each node contains:

- **kubelet** (node agent)
- **container runtime** (Docker/containerd)
- **kube-proxy** (network rules)

---

5. What is a Pod?

Answer: A pod is the **smallest deployable unit in Kubernetes**.
It can contain:

- **one container** (most common)
- or **multiple containers** that share:
    - network (same IP)
    - storage volumes

✅ Pods are ephemeral (can be destroyed and recreated).

---

6. What is the difference between a Pod and a Container?

Answer:

- A **container** is a runtime package of an application
- A **pod** is a Kubernetes wrapper for containers

✅ A pod can run **one or more containers**, while a container is a single unit.

---

7. What is a Deployment in Kubernetes?

Answer: A Deployment is used to manage application pods declaratively. It provides:
✅ desired number of replicas
✅ rolling updates
✅ rollbacks
✅ self-healing replacement pods

---

8. What is a ReplicaSet?

Answer: A ReplicaSet ensures the required number of pod replicas are running at all times.

✅ Deployment manages ReplicaSets automatically.

---

9. What is a Kubernetes Service?

Answer: A Service provides a stable way to access pods, because pod IPs change frequently.

Types of Services:

- **ClusterIP** (internal only)
- **NodePort** (expose on worker node port)
- **LoadBalancer** (cloud load balancer)
- **ExternalName** (DNS mapping)

---

10. What is a Namespace in Kubernetes?

Answer: A namespace is a logical separation inside a Kubernetes cluster used to:
✅ isolate teams/projects
✅ manage resources separately
✅ apply RBAC and policies differently

Common namespaces: `default`, `kube-system`, `kube-public`.

Section B — Intermediate Level (11–20)

11. What are Labels and Selectors in Kubernetes?

Answer:

- **Labels:** key-value metadata attached to objects (`app=web`)
- **Selectors:** used to match objects based on labels

✅ Services use selectors to connect to pods.

---

12. What is a ConfigMap?

Answer: A ConfigMap stores **non-sensitive configuration data** like:

- environment variables
- application configs
- URLs, flags, settings

✅ Keeps configuration separate from container image.

---

13. What is a Secret in Kubernetes?

Answer: Secrets store **sensitive data**, such as:

- DB passwords
- API keys
- TLS certificates

✅ Best practice: do not hardcode secrets in YAML or Git.

---

14. What are Liveness and Readiness Probes?

Answer:

- **Liveness Probe:** checks if the container is alive → restarts if failed
- **Readiness Probe:** checks if the container is ready to serve traffic → removes from service if failed

✅ They improve reliability and zero-downtime deployments.

---

15. What is a StatefulSet and when do you use it?

Answer: StatefulSet is used for **stateful applications** requiring:
✅ stable pod identity (fixed name)
✅ stable storage
✅ ordered startup/shutdown

Examples: MySQL, MongoDB, Kafka, Elasticsearch.

---

16. What is the difference between Deployment and StatefulSet?

Answer:
✅ **Deployment:** stateless workloads, random pod names, easy scaling
✅ **StatefulSet:** stable identity + stable storage, ordered rolling operations

---

17. What is a Persistent Volume (PV) and Persistent Volume Claim (PVC)?

Answer:

- **PV:** actual storage resource in cluster
- **PVC:** request for storage by a pod

✅ Pod uses PVC → PVC binds to PV.

---

18. What is Ingress in Kubernetes?

Answer: Ingress provides **HTTP/HTTPS routing** to services, allowing:
✅ domain-based routing
✅ path-based routing
✅ TLS termination (HTTPS)

Ingress needs an **Ingress Controller** (Nginx, ALB, Traefik).

---

19. What is a DaemonSet?

Answer: DaemonSet ensures **one pod runs on every node** (or selected nodes).

Common use cases:

- logging agents (Fluentd)
- monitoring agents (Node Exporter)
- security agents

---

20. What is a Job and CronJob?

Answer:

- **Job:** runs a task until completion (one-time batch job)
- **CronJob:** runs jobs on a schedule (like Linux cron)

Examples: backups, cleanup scripts, scheduled reports.

Section C — Advanced Level (21–24)

21. What is the Kubernetes Control Plane?

Answer: The control plane manages the cluster and scheduling. Components include:

- **API Server** (entry point)
- **Scheduler** (places pods on nodes)
- **Controller Manager** (maintains desired state)
- **etcd** (cluster database)

✅ If control plane fails, workloads may run but cluster becomes unmanaged.

---

22. What is etcd and why is it important?

Answer: etcd is a distributed key-value store used to store:
✅ cluster configuration
✅ secrets (unless external)
✅ state of nodes, pods, services

✅ etcd must be backed up regularly in production.

---

23. What is HPA (Horizontal Pod Autoscaler)?

Answer: HPA automatically scales pods based on metrics such as:

- CPU usage
- memory usage
- custom metrics (via Prometheus adapter)

✅ Useful for handling load spikes dynamically.

---

24. What is a NetworkPolicy?

Answer: NetworkPolicy controls network traffic rules between pods, namespaces, and services.

✅ Provides micro-segmentation and zero-trust networking inside clusters.

Example: allow only backend pods to talk to database pods.

Section D — Production Scenario Questions (25–28)

25. Scenario: Pods are stuck in `CrashLoopBackOff`. What do you do?

Answer: Steps to troubleshoot:
✅ check logs (`kubectl logs`)
✅ describe the pod (`kubectl describe pod`)
✅ verify env vars, secrets, ConfigMaps
✅ check image name/tag and pull permissions
✅ check probes (liveness/readiness failures)
✅ check resource limits (OOMKilled)

Most common causes: missing configs, wrong command, app crash, memory limit.

---

26. Scenario: A deployment is slow and keeps restarting pods. How do you fix it?

Answer:

- verify readiness probes are correct
- increase initialDelaySeconds if app needs time
- increase CPU/memory resources
- check external dependencies (DB, API)
- enable autoscaling if needed
- check node capacity and scheduling delays

✅ Readiness probe misconfiguration is one of the most common reasons.

---

27. Scenario: You need zero downtime deployment for a production app. What's your approach?

Answer:
✅ use rolling updates with proper probes
✅ set maxUnavailable=0 and maxSurge=1 (if needed)
✅ ensure graceful shutdown using:

- preStop hooks
- terminationGracePeriodSeconds
  ✅ use Ingress/Service stable routing
  ✅ perform canary release if high risk

---

28. Scenario: How do you secure a Kubernetes cluster in production?

Answer: Strong production security practices:
✅ RBAC (least privilege)
✅ namespace isolation
✅ network policies
✅ secrets encryption + external secret managers
✅ disable anonymous API access
✅ restrict cluster-admin usage
✅ scan images (Trivy, etc.)
✅ use Pod Security Standards (restricted/baseline)
✅ audit logs enabled

✅ Kubernetes Quick Revision (One-Liners)

- **Pod:** smallest unit, runs container(s)
- **Deployment:** manages stateless pods + rollouts
- **StatefulSet:** stable identity + storage
- **Service:** stable access to pods
- **Ingress:** routing via HTTP/HTTPS

- **PV/PVC:** storage management
- **ConfigMap:** non-sensitive config
- **Secret:** sensitive config
- **HPA:** autoscale pods
- **NetworkPolicy:** restrict pod traffic

✅ PYTHON + SHELL INTERVIEW QUESTIONS + ANSWERS (28 Questions)

(DevOps Focused | Beginner → Intermediate → Advanced → Production Scenarios)

Python + Shell Interview Guide (28 Q&A)

✅ Section A — Beginner Level (1–10)

1. What is the main difference between Shell scripting and Python in DevOps?

Answer:

- **Shell scripting** is best for **quick OS-level automation** (files, services, commands, pipelines).
- **Python** is best for **complex automation**, API calls, data parsing, and scalable scripts.

✅ Rule of thumb:
Shell = glue automation | **Python = structured automation**

---

2. When should you choose Shell over Python?

Answer: Use Shell when the task involves:
✅ system administration commands (grep, awk, sed)
✅ managing services (systemctl, ps, kill)
✅ automating Linux tasks quickly
✅ chaining commands via pipes

---

3. When should you choose Python over Shell?

Answer: Use Python when you need:
✅ API integration (AWS, GitHub, monitoring APIs)
✅ JSON/YAML parsing
✅ handling complex logic and error handling
✅ better maintainability and reusable modules

---

4. What is the purpose of `#!/bin/bash` in a shell script?

Answer: It's called a **shebang**. It tells the system which interpreter should execute the script.

Example:
`#!/bin/bash` → run using Bash shell

---

5. How do you make a shell script executable?

Answer:
Use:
```
chmod +x script.sh
```
Then run:
```
./script.sh
```

---

6. What are environment variables and why are they important?

Answer: Environment variables store system values used by processes, such as:

- `PATH`

- `HOME`
- `USER`
- `AWS_REGION`

✅ Used in DevOps for configuration without hardcoding.

---

7. How do you print a variable in Shell and Python?

Answer:
✅ Shell:

```
echo $VAR
```

✅ Python:

```
print(var)
```

---

8. How do you handle user input in Shell and Python?

Answer:
✅ Shell:

```
read name
echo "Hello $name"
```

✅ Python:

```
name = input("Enter name: ")
print("Hello", name)
```

---

9. How do you check exit status in Shell?

Answer:
In Shell, `$?` gives exit code of last command:

- `0` → success
- `non-zero` → failure

Example:

```
ls /tmp
echo $?
```

---

10. What are the most common Shell commands used in DevOps daily work?

Answer: Some essentials are:

- `ls`, `cd`, `pwd`
- `cp`, `mv`, `rm`, `mkdir`
- `cat`, `tail`, `head`, `grep`
- `ps`, `top`, `kill`
- `systemctl`, `journalctl`
- `curl`, `wget`, `ssh`, `scp`

✅ Section B — Intermediate Level (11–20)

11. Explain pipes (`|`) in Shell scripting with an example.

Answer: Pipes send output of one command as input to another.

Example:

```
cat app.log | grep ERROR | wc -l
```

✅ counts total ERROR lines in a log file.

---

12. What is the difference between `>` and `>>` in Shell?

Answer:

- `>` overwrites file
- `>>` appends to file

Example:

```
echo "Hello" > file.txt
echo "World" >> file.txt
```

---

13. What are `grep`, `awk`, and `sed` used for?

Answer:
✅ `grep` → search text
✅ `awk` → text processing + column extraction
✅ `sed` → stream editor (replace/edit text)

Example:

```
awk '{print $1}' file.txt
```

prints first column.

---

14. What is a function in Shell and Python?

Answer:
✅ Shell function example:

```
hello() {
  echo "Hello DevOps"
}
hello
```

✅ Python function example:

```
def hello():
    print("Hello DevOps")
hello()
```

---

15. How do you run a command in background and keep it running after logout?

Answer:
✅ run in background:

```
command &
```

✅ keep running after logout:

```
nohup command &
```

---

16. How do you find which process is using a specific port?

Answer:
Use:

```
sudo lsof -i :8080
```

or

```
sudo netstat -tulnp | grep 8080
```

---

17. How do you parse JSON in Python for DevOps tasks?

Answer: Use Python's `json` module:

```
import json

data = json.loads('{"env":"prod","region":"us-east-1"}')
print(data["region"])
```

✅ Very useful in cloud automation.

---

18. What is the difference between `subprocess` and `os.system()` in Python?

Answer:

- `os.system()` → simple but limited output handling
- `subprocess` → better control, captures output, safer

Example:

```
import subprocess
result = subprocess.run(["ls", "-l"], capture_output=True, text=True)
print(result.stdout)
```

---

19. How do you handle exceptions in Python?

Answer: Using `try-except`:

```
try:
    x = 10 / 0
except Exception as e:
    print("Error:", e)
```

✅ Important in production scripts to avoid crashes.

---

20. How do you schedule scripts in Linux?

Answer: Use **cron jobs**.

Example:

```
crontab -e
```

Run script every day at 2 AM:

```
0 2 * * * /path/backup.sh
```

✅ Section C — Advanced Level (21–24)

21. What is idempotency and why does it matter in DevOps automation?

Answer: Idempotency means running a script multiple times produces the same final result.

✅ Example:
A script that creates a directory should not fail if the directory already exists.

Shell best practice:

```
mkdir -p /opt/app
```

---

22. How do you write a safe Bash script for production usage?

Answer: Recommended safety settings:

```
set -euo pipefail
```

Meaning:

- `-e` → stop on error
- `-u` → fail on undefined variables
- `pipefail` → fail if any command in pipeline fails

✅ Prevents silent failures.

---

23. Explain logging best practices for automation scripts.

Answer: Best practices:
✅ timestamp logs
✅ log to file and console
✅ include error codes
✅ log structured messages (INFO/WARN/ERROR)

Shell example:

```
echo "$(date) [INFO] Starting deployment..."
```

Python example using logging module:

```
import logging
logging.basicConfig(level=logging.INFO)
logging.info("Starting deployment")
```

---

24. How do you securely manage secrets in scripts?

Answer: Avoid hardcoding secrets inside scripts. Instead:
✅ use environment variables
✅ use secret managers (AWS Secrets Manager, Vault)
✅ restrict permissions
✅ rotate credentials

Bad example:

```
PASSWORD="admin123"
```

Good example:

```
PASSWORD="$DB_PASSWORD"
```

✅ Section D — Production Scenario Questions (25–28)

25. Scenario: A production server is running out of disk space. What do you do?

Answer:
✅ check disk usage:

```
df -h
```

✅ find heavy directories:

```
du -sh /* | sort -h
```

✅ check logs:

```
journalctl --disk-usage
```

✅ cleanup safely:

- remove old logs
- rotate logs
- remove unused docker images:

```
docker system prune
```

✅ prevent recurrence: monitoring + alerts.

---

26. Scenario: Your script works manually but fails in Jenkins/CI pipeline. Why?

Answer: Common causes:
✅ missing environment variables in CI
✅ permission issues (non-root agents)
✅ different PATH in CI
✅ missing files (workspace not correct)
✅ script requires interactive input

Fix tips:

- always define full paths
- print env values inside pipeline for debugging
- use non-interactive flags (like `-y`)

---

27. Scenario: A deployment script accidentally brings down production. How do you prevent this in the future?

Answer:
✅ implement safeguards:

- confirm environment (prod/staging)
- add approvals for prod deploy
- use feature flags
- use blue/green or canary deployments
- use rollback support
- add automated tests before deployment

✅ Most failures come from missing validation checks.

---

28. Scenario: You need to automate log parsing and alerting. How would you do it?

Answer: Approach:
✅ Shell quick parsing:

```
grep "ERROR" app.log | tail -20
```

✅ Python automation for better logic:

- parse logs
- count frequency
- send alert via email/Slack webhook
- run as cronjob

Python idea (high level):

- read file
- filter errors
- if threshold crossed → notify

✅ Python is ideal for scalable alert automation.

✅ Python + Shell Quick Revision Sheet

Shell Must-Know

- `grep`, `awk`, `sed`
- `ps`, `top`, `kill`

- `df -h`, `du -sh`
- pipes `|`, redirection `> >>`
- cron jobs
- `set -euo pipefail`

Python Must-Know

- `try/except`
- `subprocess`
- `json` parsing
- logging module
- scripts for API automation