

DevOps Interview Questions/Answers:

✓ Structure I'll follow:

- CI/CD Tools (Jenkins, GitHub Actions, GitLab CI/CD)
- Configuration Management (Ansible, Chef, Puppet)
- Containerization (Docker)
- Container Orchestration (Kubernetes, ECS, EKS)
- Infrastructure as Code (Terraform, CloudFormation)
- Monitoring and Logging (Prometheus, Grafana, ELK Stack)
- Cloud (AWS Services – EC2, S3, VPC, IAM, Lambda, RDS, ECS, EKS, etc.)
- Security and Networking in DevOps
- General DevOps Methodologies & Culture
- Scenario-Based Questions

✓ Entire guide now has 160+ questions with deep explanations.

Section 1. CI/CD Tools (Jenkins, GitHub Actions, GitLab CI/CD)

Q1. What is CI/CD and why is it important in DevOps?

Answer:

- **CI (Continuous Integration)** is the practice of frequently integrating code into a shared repository and running automated tests to detect issues early.
- **CD (Continuous Delivery/Deployment)** ensures that the application is automatically deployed to staging or production environments after passing tests.
- **Importance:**
 - Reduces integration problems.
 - Speeds up release cycles.
 - Improves quality via automated testing.

Extra Tip: Mention tools like **Jenkins, GitHub Actions, GitLab CI/CD, AWS CodePipeline** and say which you've used in projects.

Q2. How does Jenkins work internally?

Answer:

- Jenkins is a **Java-based automation server** that runs on a servlet container like **Apache Tomcat**.
- It uses a **master-agent architecture**:

- **Master (Controller):** Manages configuration, schedules jobs, monitors agents.
- **Agents (Nodes):** Execute build jobs.
- Jobs can be configured via **UI** or **Jenkinsfile (Pipeline as Code)**.
- Supports **plugins** for integration with tools like Git, Docker, Kubernetes, AWS.

Extra Tip: If asked about scaling Jenkins, talk about **Kubernetes-based Jenkins Agents** for elasticity.

Q3. Difference between Declarative and Scripted Jenkins Pipelines?

Answer:

- **Declarative Pipeline:**
 - Uses a structured, predefined syntax.
 - Easier to read, more opinionated.

```
groovy

pipeline {
    agent any
    stages {
        stage('Build') { steps { echo 'Building...' } }
    }
}
```

Scripted Pipeline:

- Written in Groovy, highly flexible.
- Good for complex logic.

```
groovy

node {
    stage('Build') {
        echo 'Building...'
    }
}
```

Extra Tip: Mention you prefer Declarative for maintainability and Scripted for complex logic.

Q4. How do you create a Jenkins pipeline for a microservices app?

Answer:

- Create **Jenkinsfile** in repo:

```
groovy                                                                    Copy Edit

pipeline {
  agent any
  stages {
    stage('Build') { steps { sh 'mvn clean package' } }
    stage('Docker Build') { steps { sh 'docker build -t myapp .' } }
    stage('Push to Registry') { steps { sh 'docker push myrepo/myapp' } }
    stage('Deploy') { steps { sh 'kubectl apply -f k8s/deployment.yaml' } }
  }
}
```

- Configure webhooks for CI triggers. ↓

Q5. What is the difference between Continuous Delivery and Continuous Deployment?

Answer:

- **Continuous Delivery:** Deployment to production requires **manual approval**.
- **Continuous Deployment:** **Fully automated** to production after tests pass.

Q6. How do you integrate Jenkins with GitHub?

Answer:

- Install **GitHub plugin** in Jenkins.
- Generate **Personal Access Token** in GitHub.
- Add webhook in GitHub pointing to `http://jenkins-server/github-webhook/`.
- Configure pipeline with SCM as GitHub repo.

Q7. How do you secure sensitive credentials in Jenkins?

Answer:

- Use **Jenkins Credentials Store**.
- Inject via withCredentials in pipeline:

groovy

Copy Edit

```
withCredentials([string(credentialsId: 'AWS_KEY', variable: 'AWS_KEY')]) {  
    sh "echo $AWS_KEY"  
}
```

Q8. Explain how you would set up a multi-branch pipeline in Jenkins.

Answer:

- Install **Multibranch Pipeline Plugin**.
- Configure Jenkins job to scan branches for Jenkinsfile.
- Each branch gets its own pipeline run automatically.

Q9. Difference between Jenkins and GitHub Actions?

Answer:

- **Jenkins:** Self-hosted, plugin-rich, more customizable.
- **GitHub Actions:** Cloud-based, built into GitHub, YAML workflows, auto-scaling runners.

Q10. How do you scale Jenkins for high availability?

Answer:

- Use **Jenkins master + multiple agents**.
- Run agents as **Kubernetes pods** using Jenkins Kubernetes Plugin.
- Use **Jenkins Operations Center** for distributed masters.

Q11. What is a Jenkins Shared Library?

Answer:

- A reusable pipeline code repository.
- Store common steps like build, test, deploy.
- Include in Jenkinsfile:

```
groovy
```

```
@Library('my-shared-lib') _
```

Q12. Explain Blue/Green Deployment in a Jenkins pipeline.

Answer:

- Deploy new version to **Green environment** while Blue is live.
- Test Green, then switch traffic.
- Rollback by pointing back to Blue if issues occur.

Q13. What are GitHub Actions runners?

Answer:

- **Hosted runners:** Provided by GitHub (Linux, Windows, macOS).
- **Self-hosted runners:** Installed on your own infrastructure for customization.

Q14. What is a GitLab CI/CD pipeline and how does it differ from Jenkins?

Answer:

- GitLab CI/CD uses .gitlab-ci.yml in the repo.
- Native to GitLab, integrated with repo & issues.
- Jenkins is external and integrates with multiple SCMs.

Q15. How do you troubleshoot a failing Jenkins pipeline?

Answer:

- Check **console output logs**.
- Validate **syntax in Jenkinsfile** with lint.
- Enable **Pipeline Replay** for debugging.
- Check **agent connectivity** and plugin versions.

✅ That's the **CI/CD section done with 15 detailed Q&A.**

✅ **Section 2: Docker (Containerization)**

Q1. What is Docker and why do we use it?

Answer:

- Docker is a **containerization platform** that packages an application and its dependencies into a single **lightweight, portable container**.
- **Benefits:**
 - Consistent environment across dev, test, prod.
 - Faster deployments than VMs.
 - Efficient resource utilization.

Q2. Difference between Docker image and Docker container?

Answer:

- **Image:** Blueprint of the application with all dependencies. Immutable.
- **Container:** Running instance of an image.

Q3. Explain Docker architecture.

Answer:

- **Client:** CLI or API that sends commands.
- **Docker Daemon:** Runs on the host, manages containers/images.
- **Registry:** Stores images (Docker Hub, AWS ECR).
- Communication via **REST API**.

Q4. What is a Dockerfile?

Answer:

- A text file containing instructions to build a Docker image.
- Example:

```
dockerfile
```

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y nginx
COPY index.html /var/www/html
CMD ["nginx", "-g", "daemon off;"]
```

Q5. Explain Docker storage drivers.

Answer:

- **Overlay2 (Linux):** Default, efficient copy-on-write.

- **aufs, devicemapper:** Older alternatives.
- Allows container layers to share base image layers.

Q6. How does Docker networking work?

Answer:

- **Bridge (default):** Containers communicate via internal network.
- **Host:** Shares host network stack.
- **Overlay:** For multi-host networking (Swarm/Kubernetes).
- **None:** No networking.

Q7. What is the difference between ENTRYPOINT and CMD in Dockerfile?

Answer:

- **ENTRYPOINT:** Always executed as the main process.
- **CMD:** Provides default arguments to ENTRYPOINT.
- Example:

```
dockerfile

ENTRYPOINT ["python"]
CMD ["app.py"]
```

Q8. How do you reduce Docker image size?

Answer:

- Use smaller base images (e.g., **alpine**).
- Combine RUN commands with && to reduce layers.
- Remove unnecessary packages and temp files.
- Use **multi-stage builds**.

Q9. Explain Docker Compose.

Answer:

- Tool to define and manage multi-container apps using docker-compose.yml.
- Example:

```
yaml
```

```
version: '3'
services:
  web:
    image: nginx
  db:
    image: mysql
```

Q10. How do you secure Docker containers?

Answer:

- Use **non-root users**.
- Keep images updated.
- Use **Docker Bench for Security**.
- Scan images with **Trivy** or **Clair**.

Q11. What is Docker Swarm and how does it compare to Kubernetes?

Answer:

- Docker Swarm: Native clustering tool for Docker.
- Easier setup than Kubernetes, but less feature-rich.
- Kubernetes is the industry standard now.

Q12. How do you persist data in Docker?

Answer:

- Use **Volumes** (docker volume create) or **Bind mounts**.
- Example:

```
bash
```

```
docker run -v /host/path:/container/path myapp
```

Q13. How do you troubleshoot a Docker container that keeps crashing?

Answer:

- Check logs: docker logs <container_id>.
- Inspect container: docker inspect.
- Check resource limits and entrypoint errors.

Q14. What is the difference between Docker and a Virtual Machine?

Answer:

- **Docker:** Uses host OS kernel, lightweight.
- **VM:** Requires full OS per instance, heavy.
- Containers start in seconds; VMs take minutes.

Q15. What is a multi-stage build in Docker?

Answer:

- Allows using multiple FROM statements to reduce final image size.
- Example:

```
dockerfile

FROM golang:1.18 AS builder
WORKDIR /app
RUN go build -o app

FROM alpine:latest
COPY --from=builder /app/app .
CMD ["/app"]
```

✓ Docker section complete.

SECTION 3: Kubernetes (Container Orchestration)

Q1. What is Kubernetes and why do we use it?

Answer:

Kubernetes (K8s) is an **open-source container orchestration platform** for automating deployment, scaling, and management of containerized applications.

- Features: **Self-healing, Auto-scaling, Service discovery, Rolling updates.**

Q2. Explain Kubernetes architecture.

Answer:

- **Master Node (Control Plane):**
 - API Server (handles requests), Scheduler (assigns pods), Controller Manager, etcd (key-value store).
- **Worker Nodes:**
 - Kubelet (manages pods), Kube-proxy (networking), Container Runtime (Docker/Containerd).

Q3. What is a Pod in Kubernetes?

Answer:

- Smallest deployable unit in K8s.
- Can contain **one or more containers** sharing the same network namespace and storage volumes.

Q4. Difference between ReplicaSet and Deployment?

Answer:

- **ReplicaSet:** Ensures a certain number of pod replicas run at any time.
- **Deployment:** Manages ReplicaSets and enables **rolling updates & rollbacks**.

Q5. Explain Kubernetes Services.

Answer:

- **ClusterIP (default):** Internal communication.
- **NodePort:** Exposes service on node's IP at a static port.
- **LoadBalancer:** Uses cloud provider's LB for external access.
- **ExternalName:** Maps to an external DNS name.

Q6. What are ConfigMaps and Secrets?

Answer:

- **ConfigMap:** Stores non-sensitive config data (e.g., environment variables).
- **Secret:** Stores sensitive data (e.g., passwords, API keys). Encoded in Base64.

Q7. Explain Kubernetes Ingress.

Answer:

- Ingress is an API object that **manages external access** (HTTP/HTTPS) to services in a cluster.

- Uses **Ingress Controller** (e.g., NGINX, Traefik).

Q8. How does Kubernetes handle service discovery?

Answer:

- Each service gets a **DNS name** via CoreDNS.
- Pods can communicate using service name instead of IP.

Q9. Explain Rolling Update vs Recreate Strategy.

Answer:

- **RollingUpdate (default)**: Updates pods incrementally without downtime.
- **Recreate**: Deletes all old pods, then creates new ones (downtime possible).

Q10. How does Kubernetes handle scaling?

Answer:

- **Horizontal Pod Autoscaler (HPA)**: Scales pods based on CPU/memory metrics.
- **Vertical Pod Autoscaler (VPA)**: Adjusts pod resources.
- **Cluster Autoscaler**: Adds/removes worker nodes.

Q11. What is etcd in Kubernetes?

Answer:

- A distributed **key-value store** that stores all cluster state data.
- Highly available and fault-tolerant.

Q12. Explain taints and tolerations.

Answer:

- **Taint**: Applied to nodes to restrict which pods can run on them.
- **Toleration**: Allows pods to run on tainted nodes.

Q13. How do you monitor Kubernetes clusters?

Answer:

- Use **Prometheus + Grafana, Kube-State-Metrics, ELK stack, Lens dashboard**.

Q14. How does Kubernetes manage networking?

Answer:

- Flat network model: Every pod gets a unique IP.
- Uses **CNI plugins** like Flannel, Calico, Weave for networking.

Q15. What is a StatefulSet in Kubernetes?

Answer:

- Manages stateful applications.
- Provides **stable network identities** and **persistent storage** for each pod.

Q16. Explain Kubernetes Namespaces.

Answer:

- Logical partitions for organizing resources.
- Useful for multi-team clusters and resource isolation.

Q17. What is the difference between Docker Swarm and Kubernetes?

Answer:

- Swarm: Simple, less powerful, native to Docker.
- Kubernetes: Feature-rich, industry standard for orchestration.

Q18. How do you troubleshoot a failing pod?

Answer:

- Check logs: `kubectl logs pod-name`.
- Describe pod: `kubectl describe pod pod-name`.
- Events: `kubectl get events`.

Q19. What is the difference between NodePort and LoadBalancer?

Answer:

- **NodePort:** Exposes service on all nodes at a fixed port (manual LB).
- **LoadBalancer:** Uses cloud LB for automatic external access.

Q20. How do you secure Kubernetes clusters?

Answer:

- Enable **RBAC**.
- Use **Network Policies**.

- Rotate secrets regularly.
- Limit container privileges.

✓ **Kubernetes section complete.**

SECTION 4: Infrastructure as Code (Terraform & CloudFormation)

Q1. What is Infrastructure as Code (IaC)?

Answer:

- IaC is the practice of managing infrastructure using **code**, not manual processes.
- Benefits: **Consistency, Version Control, Automation.**

Q2. What is Terraform and why is it popular?

Answer:

- Open-source IaC tool by HashiCorp.
- Uses **HCL (HashiCorp Configuration Language)**.
- Multi-cloud support (AWS, Azure, GCP).

Q3. Explain Terraform architecture.

Answer:

- **Core:** Reads config and manages lifecycle.
- **Providers:** Plugins for cloud platforms.
- **State File:** Stores infrastructure state for tracking changes.

Q4. What is the difference between Terraform and CloudFormation?

Answer:

- **Terraform:** Multi-cloud, uses HCL.
- **CloudFormation:** AWS-only, uses JSON/YAML.

Q5. What is Terraform state and why is it important?

Answer:

- State file (terraform.tfstate) tracks resources created by Terraform.
- Enables incremental updates.
- Should be stored in **remote backends** (S3, etc.).

Q6. What are Terraform backends?

Answer:

- Mechanism for storing state remotely (S3, GCS, Consul).
- Allows **collaboration** and **locking**.

Q7. Explain terraform init, plan, apply, destroy.

Answer:

- init – Initialize working directory.
- plan – Show execution plan.
- apply – Apply changes.
- destroy – Delete resources.

Q8. How do you manage secrets in Terraform?

Answer:

- Use **Vault**, **AWS Secrets Manager**, or environment variables.
- Never hardcode secrets in .tf files.

Q9. Explain Terraform modules.

Answer:

- Reusable configurations stored in separate directories or repos.
- Called using module block in main .tf file.

Q10. How do you handle drift in Terraform?

Answer:

- Use terraform plan to detect differences between real infrastructure and state file.
- Run terraform apply to fix.

Q11. What is a Terraform provider?

Answer:

- Plugin that interacts with APIs (AWS, Azure, GCP).
- Declared in configuration:

```
hcl

provider "aws" {
  region = "us-east-1"
}
```

Q12. How do you upgrade Terraform version safely?

Answer:

- Use terraform version to check.
- Run terraform 0.x upgrade command if major changes.

Q13. What is the difference between count and for_each in Terraform?

Answer:

- **count:** Simple replication based on an integer.
- **for_each:** Creates resources based on a map or set.

Q14. Explain depends_on in Terraform.

Answer:

- Forces explicit resource dependency.
- Useful when implicit dependencies are not enough.

Q15. How do you ensure team collaboration in Terraform?

Answer:

- Use **remote backend with locking** (e.g., S3 + DynamoDB).
- Use **Terraform Cloud/Enterprise** for governance.

✓ **Terraform section done.**

SECTION 5: Configuration Management (Ansible, Chef, Puppet)

Q1. What is configuration management in DevOps?

Answer:

Configuration management is the process of **automating the setup and**

maintenance of servers and applications to ensure consistency across environments.

Q2. What is Ansible and why is it popular?

Answer:

- **Agentless**, uses **SSH** for communication.
- Written in Python, uses **YAML playbooks**.
- Easy to learn, no extra software on nodes.

Q3. Explain Ansible architecture.

Answer:

- **Control Node:** Runs playbooks.
- **Managed Nodes:** Machines being configured.
- **Inventory:** List of hosts.
- **Modules:** Pre-built scripts for tasks.

Q4. What is an Ansible playbook?

Answer:

- A YAML file that defines **tasks to execute on target hosts**.

```
yaml
- hosts: webservers
  tasks:
    - name: Install NGINX
      apt:
        name: nginx
        state: present
```

Q5. How does Ansible differ from Chef and Puppet?

Answer:

- **Ansible:** Agentless, YAML, simple.
- **Chef:** Uses Ruby DSL, requires agent.
- **Puppet:** Declarative, uses manifests, agent-based.

Q6. How do you manage secrets in Ansible?

Answer:

- Use **Ansible Vault** to encrypt sensitive data:

Q7. What are Ansible roles?

Answer:

- A way to organize playbooks into reusable components (tasks, vars, handlers).

Q8. How do you test Ansible playbooks before running in production?

Answer:

- Use `--check` flag for dry-run.
- Use **Molecule** for testing.

Q9. How does Ansible ensure idempotency?

Answer:

- Tasks are written to produce the same result even if executed multiple times.

Q10. What is the difference between Ansible ad-hoc commands and playbooks?

Answer:

- **Ad-hoc:** One-time tasks.
- **Playbooks:** Structured, reusable automation.

Q11. How do you handle dynamic inventories in Ansible?

Answer:

- Use scripts or **cloud inventory plugins** (AWS, GCP).

Q12. What are Ansible facts?

Answer:

- System properties gathered by Ansible before running tasks.

Q13. How do you scale Ansible for large environments?

Answer:

- Use **Ansible Tower/AWX** for GUI and RBAC.

Q14. What is the difference between handlers and tasks in Ansible?

Answer:

- **Handlers:** Run only when notified (e.g., restart service).
- **Tasks:** Run in order always.

Q15. How do you debug Ansible playbooks?

Answer:

- Use `-vvvv` for verbose mode.
- Use debug module to print variables.

✓ **Ansible section complete.**

SECTION 6: Monitoring & Logging (Prometheus, Grafana, ELK)

Q1. Why is monitoring important in DevOps?

Answer:

- Ensures system reliability, detects issues early, supports **SLO/SLI/SLA compliance**.

Q2. What is Prometheus and why is it used?

Answer:

- Open-source **metrics-based monitoring** system.
- Pulls metrics from targets via HTTP.
- Stores data in **time-series database**.

Q3. How does Prometheus architecture work?

Answer:

- **Server:** Scrapes and stores metrics.
- **Exporters:** Collect metrics (e.g., node_exporter).
- **Alertmanager:** Sends alerts.

Q4. What is Grafana?

Answer:

- Visualization tool for Prometheus, InfluxDB, Elasticsearch.
- Creates dashboards and alerts.

Q5. Explain ELK Stack.

Answer:

- **Elasticsearch:** Stores logs.
- **Logstash:** Collects & processes logs.
- **Kibana:** Visualization UI.

Q6. Difference between push and pull metrics in Prometheus?

Answer:

- **Pull:** Prometheus scrapes targets.
- **Push:** Targets send metrics to Pushgateway (rare case).

Q7. How do you monitor Kubernetes clusters?

Answer:

- **Prometheus + Grafana** with **kube-state-metrics** and **cAdvisor**.

Q8. What is a Prometheus Alert Rule?

Answer:

- Defines conditions for alerts.

yaml

```
- alert: HighCPU
  expr: node_cpu_seconds_total > 90
  for: 5m
```

Q9. How do you secure Prometheus and Grafana?

Answer:

- Enable **TLS** and **authentication**.
- Use RBAC in Grafana.

Q10. What are blackbox and whitebox monitoring?

Answer:

- **Blackbox:** Tests from outside (ping, HTTP).
- **Whitebox:** Internal metrics.

Q11. How do you monitor logs in microservices?

Answer:

- Use **centralized logging** with ELK or Loki.

Q12. How do you set up alerts in Grafana?

Answer:

- Create thresholds on panels, integrate with Slack, email, or PagerDuty.

✅ **Monitoring section complete.**

SECTION 7: AWS Cloud Services

Q1. What are the core AWS services for DevOps?

Answer:

- **Compute:** EC2, Lambda, ECS, EKS.
- **Storage:** S3, EBS, EFS.
- **Networking:** VPC, Route 53, ALB/NLB.
- **CI/CD:** CodeBuild, CodeDeploy, CodePipeline.
- **Monitoring:** CloudWatch, X-Ray.

Q2. Explain EC2 instance types.

Answer:

- **General Purpose (t2, t3):** Balanced.
- **Compute Optimized (c5):** High CPU.
- **Memory Optimized (r5):** High RAM.
- **GPU (p2, g4):** ML, graphics.

Q3. What is an Auto Scaling Group (ASG)?

Answer:

- Automatically adjusts EC2 instances based on demand.
- Works with **Load Balancer** for HA.

Q4. Explain IAM roles vs IAM policies.

Answer:

- **Roles:** Assign temporary credentials to resources.
- **Policies:** JSON documents defining permissions.

Q5. Difference between S3 and EBS?

Answer:

- **S3:** Object storage, accessible via API, infinite scaling.
- **EBS:** Block storage for EC2 instances.

Q6. What is an AWS VPC?

Answer:

- Virtual Private Cloud for isolating resources.
- Contains **subnets (public/private)**, **route tables**, **NAT Gateway**.

Q7. Explain AWS security best practices.

Answer:

- Enable **MFA**, use **least privilege**, rotate access keys.
- Encrypt data with **KMS**.

Q8. How does AWS Lambda work?

Answer:

- Serverless compute.
- Runs code in response to events (S3 upload, API Gateway, etc.).

Q9. Difference between ECS and EKS?

Answer:

- **ECS:** AWS-managed container orchestration.
- **EKS:** Kubernetes on AWS.

Q10. What is CloudFormation?

Answer:

- AWS IaC service for deploying resources using YAML/JSON templates.

Q11. How do you secure S3 buckets?

Answer:

- Disable **public access**, enable **bucket policies**, encrypt objects (SSE-S3, SSE-KMS).

Q12. How do you implement CI/CD in AWS?

Answer:

- Use **CodePipeline** (or Jenkins on EC2) + CodeBuild + CodeDeploy.

Q13. How does AWS Load Balancer work?

Answer:

- **ALB**: HTTP/HTTPS, Layer 7 routing.
- **NLB**: TCP, Layer 4, high performance.

Q14. How do you implement high availability in AWS?

Answer:

- Deploy across **multiple AZs**.
- Use **Load Balancer + Auto Scaling**.

Q15. Explain AWS Route 53.

Answer:

- DNS service.
- Supports **Routing Policies**: Simple, Weighted, Latency, Failover.

Q16. How do you monitor AWS resources?

Answer:

- **CloudWatch Metrics, Logs, Alarms**.

Q17. How do you handle secrets in AWS?

Answer:

- Use **Secrets Manager** or **SSM Parameter Store**.

Q18. What is the difference between RDS and DynamoDB?

Answer:

- **RDS:** Relational database (MySQL, PostgreSQL).
- **DynamoDB:** NoSQL, key-value.

Q19. How do you implement disaster recovery in AWS?

Answer:

- **Backup to S3, Cross-region replication, Multi-AZ RDS.**

Q20. How does AWS support DevOps culture?

Answer:

- Automation tools: **CodePipeline, CloudFormation, Lambda.**
- Scalability with ASG, ECS/EKS.

✓ **AWS section complete.**

SECTION 8: Security & Networking in DevOps

Q1. How do you secure a CI/CD pipeline?

- Use **role-based access control**, secrets vaults, scan for vulnerabilities.

Q2. What is the principle of least privilege?

- Give only the **minimum permissions** needed to perform tasks.

Q3. How do you secure containers?

- Use **image scanning**, run as **non-root**, enable **AppArmor/SELinux**.

Q4. What is a WAF?

- Web Application Firewall that blocks attacks like **SQL injection, XSS**.

Q5. How do you implement zero trust in DevOps?

- No implicit trust; verify every user, device, and request.

SECTION 9: General DevOps Methodologies

Q1. What is DevOps and why is it important?

Answer:

DevOps is a **cultural and technical movement** that integrates **development and operations teams** to improve **collaboration, automation, and delivery speed**.

- **Benefits:**
 - Faster time to market.
 - Reduced failure rate.
 - Faster recovery from failures.
- **Key practices:** CI/CD, Infrastructure as Code, Monitoring, Automation.

Q2. What are the main principles of DevOps?

Answer:

- **Automation:** Everything from builds to deployments.
- **Collaboration:** Break silos between Dev & Ops.
- **Continuous Integration and Delivery (CI/CD):** Frequent releases.
- **Monitoring & Feedback:** Observability for improvement.
- **Security (DevSecOps):** Shift-left approach for security.

Q3. Explain the DevOps lifecycle.

Answer:

- **Plan → Code → Build → Test → Release → Deploy → Operate → Monitor**
- Continuous feedback loops feed improvements back to the planning phase.

Q4. What is the difference between Agile and DevOps?

Answer:

- **Agile:** Focuses on **software development methodology** for iterative development.
- **DevOps:** Focuses on **end-to-end delivery**, including operations and infrastructure.

Q5. What is the difference between Continuous Integration, Continuous Delivery, and Continuous Deployment?

Answer:

- **CI:** Frequent merging and automated testing.
- **CD (Delivery):** Code is always ready for production (manual approval).
- **CD (Deployment):** Fully automated to production after tests pass.

Q6. What metrics do you track in a DevOps pipeline?

Answer:

- **Deployment frequency.**
- **Lead time for changes.**
- **Change failure rate.**
- **Mean time to recovery (MTTR).**
- **Pipeline success rate.**

Q7. What is DevSecOps?

Answer:

- **DevOps + Security.**
- Security is integrated into the pipeline (static analysis, secrets scanning, dependency checks) instead of being an afterthought.

Q8. What is shift-left testing in DevOps?

Answer:

- Testing early in the development cycle to catch defects sooner.
- Implemented using **unit tests, automated tests in CI.**

Q9. What is the difference between “pets vs cattle” analogy in infrastructure?

Answer:

- **Pets:** Individually maintained servers (traditional approach).
- **Cattle:** Disposable, identical servers (cloud-native approach). If one fails, replace it.

Q10. How do you handle cultural change when adopting DevOps?

Answer:

- **Promote collaboration** across teams.
- **Introduce automation gradually.**
- **Train teams in CI/CD, IaC, Cloud.**
- **Use metrics to show success.**

✓ **General DevOps Methodologies section complete.**

✓ **SECTION 10: Scenario-Based DevOps Questions (20 Scenarios)**

- **How would you deploy a microservices app on AWS with zero downtime?**
(Answer: Blue/Green or Canary deployment using ALB + ECS/EKS).
- **How would you troubleshoot slow response in a Kubernetes app?**
(Check pod logs, resource limits, HPA, network policies).

Scenario 1:

Your production application is running on Kubernetes. Suddenly, some pods keep restarting. How do you troubleshoot?

Answer:

1. Check logs: `kubectl logs <pod>`
2. Describe pod: `kubectl describe pod <pod>` (look for OOMKilled, CrashLoopBackOff)
3. Check resource limits (requests/limits) and adjust if needed.
4. Verify health checks (livenessProbe, readinessProbe).
5. Check for image pull errors or config errors.

Scenario 2:

You need zero downtime deployment for a microservice on AWS. How would you achieve this?

Answer:

- Use **Blue/Green deployment**:
 - Deploy new version (green) alongside old (blue).
 - Switch ALB target group to green after testing.
- Or use **Canary deployment** with weighted traffic in ALB or Kubernetes.

Scenario 3:

A Jenkins pipeline fails at the build stage randomly. How do you debug?

Answer:

- Check **console logs** for specific errors.
- Validate Jenkinsfile syntax.
- Check if **node agents have enough resources**.
- Run `mvn clean package` locally (if Maven build).

- If random: suspect **network issues or cache problems**.

Scenario 4:

Your AWS EC2 instances are getting high CPU usage during traffic spikes. What is your solution?

Answer:

- Enable **Auto Scaling Group** with CPU-based scaling policy.
- Attach **Application Load Balancer** to distribute traffic.
- Optimize app code and enable caching (Redis/CloudFront).

Scenario 5:

How would you migrate an on-prem app to AWS with minimal downtime?

Answer:

- Use **AWS Database Migration Service (DMS)** for DB migration.
- Sync files using **AWS DataSync**.
- Deploy app to **ECS/EKS** or EC2 in parallel.
- Use **Route 53 failover routing** for final cutover.

Scenario 6:

Your CI pipeline takes 30 minutes to run tests. How can you optimize it?

Answer:

- Use **parallel test execution**.
- Cache dependencies (Maven, npm).
- Use **Docker images with pre-installed dependencies**.
- Shift left by running **unit tests before integration tests**.

Scenario 7:

You find that a secret key is accidentally committed to Git. What do you do?

Answer:

- Immediately **revoke the key**.
- Rotate credentials in IAM.
- Remove key from Git history using git filter-branch or BFG Repo Cleaner.
- Implement **pre-commit hooks** and use **secret scanning tools** (e.g., TruffleHog).

Scenario 8:

One microservice is slow; others are fine. How do you identify the bottleneck?

Answer:

- Enable **distributed tracing** (AWS X-Ray, Jaeger).
- Check **service-level metrics** (CPU, memory, response time).
- Analyze database queries or external API latency.

Scenario 9:

You need to deploy an app across multiple AWS regions for disaster recovery.

How?

Answer:

- Deploy in **multiple regions with Route 53 latency-based routing**.
- Sync data using **S3 Cross-Region Replication** or **RDS Multi-Region**.
- Use **Global DynamoDB tables** for NoSQL.

Scenario 10:

You see a Kubernetes pod stuck in Pending state. What do you check?

Answer:

- Check node capacity: `kubectl describe node`.
- Check **taints and tolerations**.
- Check **storage class** for persistent volumes.
- Check **network policies**.

Scenario 11:

Your Docker container keeps growing in size after each build. Why?

Answer:

- Layers are not being cleaned up.
- Use `docker system prune`.
- Optimize Dockerfile (combine RUN commands, use multi-stage builds).

Scenario 12:

Your deployment failed because of a Terraform state file corruption. What now?

Answer:

- Use **remote backend** (S3 + DynamoDB lock) for state.
- Restore from backup (Terraform Cloud or versioned S3 bucket).
- If needed, run `terraform refresh` to rebuild state from infrastructure.

Scenario 13:

You need to roll back a failed deployment in Kubernetes. How?

Answer:

- Use `kubectl rollout undo deployment <name>`.
- If rollback is disabled, apply the previous manifest.

Scenario 14:

You need to monitor logs for all microservices in Kubernetes. How do you centralize logging?

Answer:

- Use **ELK Stack** or **Fluentd + Elasticsearch + Kibana**.
- Alternative: **Loki + Grafana** for cost-effective logging.

Scenario 15:

Your team needs secrets management across multiple environments. How do you handle it?

Answer:

- Use **HashiCorp Vault**, **AWS Secrets Manager**, or **Kubernetes Secrets** (with encryption).

Scenario 16:

How do you implement Canary deployment in Kubernetes?

Answer:

- Use **two deployments** with different labels.
- Service splits traffic using **Istio** or **NGINX Ingress annotations**.

Scenario 17:

Your Jenkins master crashed. How do you recover?

Answer:

- Restore from **backup of Jenkins home directory**.
- Use **Jenkins Configuration as Code (JCasC)** for reproducibility.
- For HA: run Jenkins on **Kubernetes with persistent storage**.

Scenario 18:

You find multiple failed login attempts in AWS CloudTrail. What do you do?

Answer:

- Enable **MFA for all users**.
- Rotate credentials.
- Use **AWS GuardDuty** to detect suspicious activity.

Scenario 19:

Your build artifacts need to be shared across multiple pipelines. How do you store them?

Answer:

- Use **artifact repository** like Nexus, Artifactory, or **S3 bucket** with lifecycle policies.

Scenario 20:

You have 100+ servers to configure. How do you automate this?

Answer:

- Use **Ansible playbooks** or **Terraform + Ansible combo**.
- Store configs in **Git for version control**.

 **General + Scenarios section complete.**

Happy learning
By Raj Nandgaonkar