20 Real-Time Cloud & DevOps Scenarios with Answers

Terraform State File Conflict

Problem: Multiple engineers ran `terraform apply`, causing state corruption.
Solution: Configured S3 backend with DynamoDB state locking + versioning.
Answer: We solved Terraform state conflicts by moving state to S3 with DynamoDB locking. This ensured no parallel applies and added versioning for recovery.

Drift in Infrastructure

Problem: EC2 instances were modified manually, causing drift from Terraform.
Solution: Ran `terraform plan` regularly + set up drift detection in CI/CD.
Answer: I enforced infrastructure drift detection in our pipeline, so if someone made manual changes, it was flagged before the next deployment.

Kubernetes Pod CrashLoopBackOff

Problem: Pods kept restarting due to wrong environment variables.
Solution: Checked logs → corrected secret reference → validated config in CI/CD.
Answer: Pods crashed in Kubernetes due to a misconfigured secret. I fixed the manifest and added validation checks in CI/CD to avoid future runtime errors.

Kubernetes Node Resource Exhaustion

Problem: Pods were evicted due to insufficient CPU/memory.
Solution: Implemented resource requests/limits + Cluster Autoscaler.
Answer: We had pod evictions due to resource exhaustion. I fixed it by defining CPU/memory requests/limits and enabling cluster autoscaling for peak load.

AWS Cost Spike

Problem: AWS bill increased due to idle instances.
Solution: Used Cost Explorer → scheduled Lambda to stop idle EC2 → budgets.
Answer: When AWS costs spiked, I identified idle EC2s and automated start/stop schedules via Lambda. This saved 30% in costs.

Load Balancer High Latency

Problem: Users reported latency during traffic spikes.
Solution: Enabled Auto Scaling + ALB Target Groups with health checks.
Answer: We faced latency due to high load on EC2. I enabled auto-scaling with proper ALB health checks. This ensured elasticity and 99.9% availability.

IAM Over-Permission

Problem: Developers had admin access in AWS.
Solution: Applied least privilege IAM policies + role-based access.
Answer: I noticed developers had admin access. I implemented least privilege IAM roles with strict

boundaries, improving security posture significantly.

Jenkins Pipeline Failures

Problem: Builds failed due to inconsistent agents.
Solution: Used Docker-based Jenkins agents + dependency caching.
Answer: We stabilized Jenkins by running builds inside Docker agents and caching dependencies. This increased pipeline reliability.

Slow Build Times

Problem: Maven/NPM builds were slow.
Solution: Enabled caching + parallel builds in Jenkins.
Answer: I optimized our Jenkins builds by adding caching and parallel execution. Build time dropped from 25 minutes to 8 minutes.

Git Merge Conflicts in Multi-Team

Problem: Frequent conflicts when merging branches.
Solution: Adopted Git Flow + pull request reviews + smaller feature branches.
Answer: To reduce merge conflicts, I enforced Git Flow branching and smaller PRs. This improved collaboration and reduced conflicts.

Docker Image Too Large

Problem: CI/CD pipeline slowed due to 3GB Docker images.
Solution: Used multi-stage builds + Alpine base image.
Answer: I reduced our Docker image from 3GB to 400MB by using multi-stage builds and Alpine base images. This sped up deployments drastically.

Docker Container Networking Issue

Problem: Containers couldn’t communicate with each other.
Solution: Created a custom bridge network in Docker Compose.
Answer: Our containers couldn’t talk to each other. I created a custom bridge network in Compose, which fixed inter-service communication.

Monitoring Gaps

Problem: No alerts when EC2 instances crashed.
Solution: Integrated CloudWatch + SNS alerts.
Answer: We faced downtime because EC2 crashed silently. I integrated CloudWatch alarms with SNS alerts, ensuring the team was notified instantly.

Log Management Issue

Problem: Logs scattered across servers, difficult to debug.
Solution: Implemented ELK Stack (Elasticsearch, Logstash, Kibana).
Answer: Troubleshooting was tough due to scattered logs. I set up ELK for centralized logging, which improved visibility and reduced MTTR.

Zero-Downtime Deployment

Problem: Users faced downtime during updates.
Solution: Used Kubernetes Rolling Updates + Blue/Green deployment.
Answer: To ensure zero downtime, I implemented Kubernetes rolling updates and, for critical apps, blue/green deployments. This gave seamless upgrades.

CloudFront Caching Issue

Problem: Users still saw old website content after updates.
Solution: Invalidated CloudFront cache using CLI.
Answer: I solved CloudFront caching issues by running invalidations post-deployment. I also added cache-busting headers in CI/CD.

RDS High CPU Usage

Problem: Database CPU usage spiked under load.
Solution: Optimized queries + enabled Read Replicas + Auto Scaling.
Answer: When RDS CPU usage spiked, I optimized queries and added read replicas. This balanced load and improved DB performance.

Secret Management

Problem: Secrets were stored in plain text in Jenkins pipelines.
Solution: Moved to AWS Secrets Manager / Vault with dynamic credentials.
Answer: I secured credentials by moving from plain-text Jenkins variables to AWS Secrets Manager. This improved compliance and security.

Event-Driven Automation

Problem: Manual steps for scaling during traffic bursts.
Solution: Used CloudWatch + EventBridge + Lambda for automation.
Answer: To automate scaling, I used CloudWatch + EventBridge to trigger Lambda for adding/removing instances dynamically. This reduced manual effort.

Disaster Recovery Test

Problem: No DR plan tested, risk of outage.
Solution: Implemented multi-region backups + automated failover.
Answer: I designed a DR plan by replicating S3/RDS across regions and testing failover. This gave us confidence in meeting RTO/RPO requirements.