# DevOps Q&A Transcript

## 1. What is your role in your project as a DevOps engineer? Explain in detail.

A DevOps engineer bridges the gap between development and operations. Key responsibilities include:  
- CI/CD Pipeline Management: Automating build, test, and deployment processes.  
- Infrastructure as Code (IaC): Using tools like Terraform or CloudFormation to provision infrastructure.  
- Monitoring & Logging: Setting up tools like Prometheus, Grafana, ELK stack.  
- Cloud Management: Managing AWS, Azure, or GCP resources.  
- Containerization & Orchestration: Using Docker and Kubernetes.  
- Security & Compliance: Implementing DevSecOps practices.  
- Collaboration: Working closely with developers, QA, and IT teams.

## 2. What is VPC Peering and how does it work?

VPC Peering connects two Virtual Private Clouds (VPCs) to route traffic between them using private IPs.  
How it works:  
- Create a peering connection between VPC A and VPC B.  
- Accept the request from the other VPC.  
- Update route tables in both VPCs to allow traffic.  
- Ensure security groups and NACLs allow communication.

## 3. Write Terraform code to create 5 EC2 instances with different names and sizes.

provider "aws" {  
 region = "us-east-1"  
}  
  
resource "aws\_instance" "example" {  
 count = 5  
 ami = "ami-0c55b159cbfafe1f0" # Replace with valid AMI  
 instance\_type = element(["t2.micro", "t2.small", "t2.medium", "t3.micro", "t3.small"], count.index)  
 tags = {  
 Name = "Instance-${count.index + 1}"  
 }  
}

## 4. How to check the last 5 users who logged into a Linux system?

Use the command:  
last -n 5

## 5. What is a multi-stage Dockerfile? Can we use the second stage as the first?

A multi-stage Dockerfile uses multiple FROM statements to:  
- Build in one stage (e.g., compile code).  
- Copy only necessary artifacts to a smaller final image.  
Yes, you can use the second stage as the first by referencing it with --target during build.

## 6. How does an application communicate with the outside world?

- Through network interfaces (e.g., public IP, NAT).  
- Load balancers, Ingress controllers, or API gateways.  
- DNS resolution and firewall rules must allow traffic.

## 7. A pod is in CrashLoopBackOff. How do you troubleshoot it?

1. kubectl describe pod <pod-name> – Check events.  
2. kubectl logs <pod-name> – View logs.  
3. Check readiness/liveness probes.  
4. Validate config maps, secrets, and environment variables.  
5. Check resource limits and image issues.

## 8. What are the types of services in Kubernetes?

- ClusterIP (default, internal access)  
- NodePort (exposes service on node IP)  
- LoadBalancer (external access via cloud LB)  
- ExternalName (maps service to DNS name)

## 9. What is a headless service in Kubernetes?

A headless service (clusterIP: None) allows direct access to individual pods without load balancing.  
Useful for:  
- StatefulSets  
- DNS-based service discovery

## 10. Write a Dockerfile for a ReactJS application.

# Stage 1: Build  
FROM node:18 AS build  
WORKDIR /app  
COPY package\*.json ./  
RUN npm install  
COPY . .  
RUN npm run build  
  
# Stage 2: Serve  
FROM nginx:alpine  
COPY --from=build /app/build /usr/share/nginx/html  
EXPOSE 80  
CMD ["nginx", "-g", "daemon off;"]

## 11. In a Kubernetes cluster, one ReplicaSet is not working. How do you debug it?

1. kubectl describe rs <replicaset-name>  
2. Check pod status: kubectl get pods  
3. View logs: kubectl logs <pod-name>  
4. Check deployment or template issues.

## 12. How to check CPU usage of a system?

Use commands like:  
top  
mpstat  
vmstat

## 13. What AWS services are used in your project?

- EC2 – Compute  
- S3 – Storage  
- RDS – Databases  
- VPC – Networking  
- CloudWatch – Monitoring  
- IAM – Access control  
- ECS/EKS – Container orchestration  
- Lambda – Serverless

## 14. What are the CI/CD steps while developing?

1. Code Commit  
2. Build (e.g., compile, lint)  
3. Test (unit/integration)  
4. Package (e.g., Docker image)  
5. Deploy (to staging/prod)  
6. Monitor (logs, metrics)

## 15. Write a CI/CD pipeline to build and push a Node.js app image to Docker Hub.

# .github/workflows/docker.yml  
name: Build and Push  
  
on:  
 push:  
 branches: [ main ]  
  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Docker  
 uses: docker/setup-buildx-action@v3  
  
 - name: Login to Docker Hub  
 uses: docker/login-action@v3  
 with:  
 username: ${{ secrets.DOCKER\_USERNAME }}  
 password: ${{ secrets.DOCKER\_PASSWORD }}  
  
 - name: Build and push  
 uses: docker/build-push-action@v5  
 with:  
 context: .  
 push: true  
 tags: yourdockerhubuser/node-app:latest