## 1.Walk me through your current project architecture and your role in it.

**In my current project, we manage a microservices-based application deployed on AWS EKS (Kubernetes).**

**We use Terraform for infrastructure provisioning, Jenkins for CI/CD, and GitHub for version control.**

**Each microservice runs as a Docker container, deployed through Helm charts.**

**My role involves managing CI/CD pipelines, automating deployments, monitoring using CloudWatch and Prometheus, and resolving infra-level issues.**

**I also handle IAM roles, S3 backups, and cost optimization on AWS.**

## 2.Which DevOps tools have you worked with in the last 2 years?

- **Version Control:** Git & GitHub
- **CI/CD:** Jenkins, GitHub Actions
- **Containerization:** Docker, Kubernetes, Helm
- **Infrastructure as Code:** Terraform
- **Monitoring:** Prometheus, Grafana, AWS CloudWatch
- **Configuration Management:** Ansible
- **Cloud Provider:** AWS (EC2, S3, IAM, VPC, EKS, RDS, CloudFormation)

## 3.What AWS services have you used in production?

**I've used several AWS services in production:**

- **EC2 for compute resources**
- **S3 for storage and backups**
- **RDS (MySQL) for managed databases**
- **EKS (Elastic Kubernetes Service) for container orchestration**
- **IAM for access control**
- **VPC, Subnets, NAT Gateway for networking**
- **CloudWatch for monitoring and alerts**
- **CloudFormation/Terraform for provisioning**

- Load Balancers (ALB/NLB) for traffic routing

**4.Describe your experience with CI/CD pipelines.**

I've designed and maintained Jenkins pipelines integrated with GitHub. Whenever code is pushed, Jenkins automatically triggers:

1. Build stage: Compiles code and runs unit tests.
2. Docker stage: Builds and pushes Docker images to ECR.
3. Deploy stage: Deploys the image to EKS using Helm.
4. Post-deploy checks: Verifies health and sends notifications to Slack.
   This automated flow reduced manual work and improved deployment speed.

**5.How do you expose a Kubernetes application to external traffic?**

You can expose a Kubernetes app using:

- Service of type LoadBalancer — automatically provisions an external IP in cloud environments (like AWS).
- Or use Ingress Controller (like NGINX Ingress) with a domain name for routing traffic to multiple services.
  Example:

```
kubectl expose deployment myapp --type=LoadBalancer --port=80 --target-port=8080
```

**6.What is the difference between Deployment and StatefulSet in Kubernetes?**

| Feature | Deployment | StatefulSet |
|---|---|---|
| Purpose | For stateless applications (e.g., web apps, APIs) | For stateful apps (e.g., databases) |
| Pod Name | Random pod names | Fixed, predictable pod names |
| Storage | No persistent volume by default | Uses PersistentVolumeClaim (PVC) |

| Feature | Deployment | StatefulSet |
|---|---|---|
| Scaling | Easy, stateless scaling | Maintains stable identities per pod |

Example: Deployment for frontend, StatefulSet for MongoDB or Redis.

## 7.What is a ConfigMap, and how is it different from a Secret?

- ConfigMap: Stores non-sensitive configuration data (like environment variables, URLs, ports).
- Secret: Stores sensitive data (like passwords, API keys).
- Secrets are base64-encoded and can be integrated with AWS Secrets Manager for better security.

Example:

```
kind: ConfigMap

data:

  APP_ENV: prod

kind: Secret

data:

  DB_PASSWORD: bXlwYXNzMTIz
```

## 8.What is the purpose of a NAT Gateway?

A NAT Gateway allows private subnet instances to access the internet for updates or downloads —
but prevents inbound traffic from the internet to those instances.
Example: Private EC2s pulling Docker images or OS updates from internet repositories.

## 9.How do you check network connectivity between two servers?

- Use ping to check basic connectivity:

```
ping <destination_IP>
```

- Use telnet or nc (netcat) to check port connectivity:

```
telnet <destination_IP> <port>
```

```
nc -zv <destination_IP> <port>
```

 For AWS, also check security groups, NACLs, and VPC route tables.

**10.How do you check running processes in Linux?**

- To list all running processes:

```
ps -ef
```

- For live process monitoring:

```
top
```

- For detailed CPU/memory usage:

```
htop
```

**11.What command would you use to find files larger than 100MB?**

bash

```
find / -type f -size +100M
```

You can add -exec ls -lh {} \; to list details:

bash

Copy code

```
find / -type f -size +100M -exec ls -lh {} \;
```

## Round Two Interview Questions

**1. You have an application in Account A that needs to access an S3 bucket in Account B. How would you configure this?**

**Answer (short): Use cross-account IAM + a bucket policy. Create an IAM role in Account A (or use an IAM user) and allow it to assume a role in Account B, or grant the Account A principal access via the bucket policy in Account B. Prefer role-based access (AssumeRole) for security.**

**Example (recommended flow)**

1. **In Account B (bucket owner) — add a bucket policy allowing a role from Account A to s3:GetObject/s3:PutObject:**

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",

"Principal":{"AWS":"arn:aws:iam::111111111111:role/AccessFromAccountA"
},
      "Action":["s3:GetObject","s3:PutObject"],
      "Resource":"arn:aws:s3:::my-bucket/*"
    }
  ]
}
```

2. **In Account A — create an IAM role AccessFromAccountA with a trust policy that allows the service or user to assume it (if using assume-role pattern), and attach a policy allowing sts:AssumeRole if needed, or a direct role with S3 permissions.**

3. **Use temporary credentials (AssumeRole) from Account A to call S3, or configure SDK/CLI with that role.**

---

**2. Your EC2 instance in a private subnet needs to download packages without a NAT Gateway. What alternatives exist?**

**Answer (short): Use S3 VPC Gateway Endpoint (for packages on S3), VPC Interface endpoints (AWS services), NAT instance, proxy server in public subnet, or use AWS Systems Manager (SSM) Session Manager / SSM Run Command to run commands via SSM (SSM requires SSM endpoints or internet access if no endpoints).**

**Options**

- **S3 Gateway Endpoint — best for yum/apt repos if packages mirrored to S3.**
- **VPC Interface Endpoint (AWS PrivateLink) — e.g., for ECR, SSM, SNS, etc.**
- **NAT instance — cheaper but less managed than NAT Gateway.**
- **Proxy in public subnet — squid proxy + security groups.**
- **SSM Agent with IAM role — use SSM to execute package install without outbound internet (requires interface endpoints for SSM if completely private).**

---

**3. How would you set up geolocation-based routing using AWS services?**

**Answer (short): Use Amazon Route 53 with Geolocation routing (or Geoproximity with Traffic Flow) to route users to region-specific endpoints (ALB, CloudFront distributions, regional API Gateways). Optionally combine with CloudFront + origin failover + regional origins for caching/latency.**

**Steps**

1. **Create records in Route 53 with Routing policy = Geolocation mapping continents/countries to specific endpoints (ALB DNS names or CloudFront).**
2. **Optionally add health checks to avoid routing to unhealthy endpoints.**
3. **For more advanced weighting/latency control, use Traffic Flow or Latency routing.**

---

**4. Write a Dockerfile for a Node.js application with multi-stage builds.**

**Answer (example):**

```
# Stage 1: build

FROM node:18-alpine AS builder

WORKDIR /app

COPY package*.json ./

RUN npm ci --production=false

COPY . .

RUN npm run build


# Stage 2: runtime

FROM node:18-alpine

WORKDIR /app

ENV NODE_ENV=production

COPY --from=builder /app/package*.json ./

COPY --from=builder /app/dist ./dist

RUN npm ci --production=true

EXPOSE 3000

CMD ["node", "dist/server.js"]
```

**Explain: first stage compiles/transpiles assets; final stage only contains production dependencies and built artifacts — smaller image.**

---

**5. What's the difference between COPY and ADD commands in Dockerfile?**

**Answer (short):**

- **COPY: simple, recommended for copying files/directories from build context into image.**
- **ADD: does everything COPY does plus: can fetch remote URLs and automatically extract local tar archives. Because ADD has extra behavior, prefer COPY unless you need those features.**

- `COPY` is the simple and recommended option. It just copies files or directories from the build context into the image. It's predictable and doesn't do anything extra, so it's the best choice in most cases.
- `ADD` can do everything `COPY` does, but it also has some extra features. For example, it can **download files from a remote URL** during the build, and it can **automatically extract local tar archives** into the image.
- Because `ADD` has extra behavior that can sometimes cause unexpected results or larger images, the best practice is to use `COPY` unless you specifically need the extra features of `ADD`.

---

**6. How do you debug a container that has exited?**

**Answer (steps/commands):**

1. **docker ps -a — find the exited container.**
2. **docker logs <container> — view stdout/stderr.**
3. **docker inspect <container> — check exit code and config.**
4. **Run container interactively to reproduce:**
5. **docker run --rm -it --entrypoint /bin/sh <image>**
6. **If Kubernetes: kubectl describe pod <pod> and kubectl logs <pod> --previous (for crashed container) and kubectl exec -it <pod> -- /bin/sh if pod is running.**

7. Check health checks, environment variables, file permissions, and startup command.

---

**7. How would you handle secrets in a Docker container for a PHP application connecting to MySQL?**

Answer (best practices): Do not bake secrets into images. Use a secrets manager or runtime injection.

Options

- Kubernetes: Use Kubernetes Secret (mounted as file or env var) or integrate with External Secrets Operator (pulls from AWS Secrets Manager).
- ECS/Fargate: Use AWS Secrets Manager or SSM Parameter Store with task definition secrets.
- Docker Compose (dev): use .env file (not committed) or Docker secrets (Swarm).
- Env vars vs files: Prefer mounting secrets as files for better rotation. Encrypt at rest.
  Example (K8s):

```
apiVersion: v1

kind: Secret

metadata: {name: db-secret}

type: Opaque

data:

  username: <base64>

  password: <base64>
```

Mount or reference in container env.

---

**8. How would you implement blue-green deployment in Kubernetes?**

Answer (patterns): Two main ways:

1. Service selector switch (simple):
   - Deploy green (v1) as deployment-v1 with labels app: myapp, version: v1. Service selector points to version: v1.
   - Deploy blue (v2) as version: v2. After smoke tests, update the Service selector to version: v2 (atomic switch).
2. Ingress/Traffic-split (advanced): Use Istio/Linkerd/Traefik or Argo Rollouts to shift traffic gradually (canary → 100% green), weighted routing, rollback easy.

Mention health checks, readiness probes, and DB migration strategy (zero-downtime or backward compatible migrations).

---

**9. How do you implement network policies to restrict pod-to-pod communication in Kubernetes?**

Answer (short): Use NetworkPolicy resources (namespaced) to allow/deny traffic based on pod selectors, namespace selectors, and ports. Apply a default deny policy and add allow rules for required traffic.

Example (allow only app frontend to talk to backend on 8080):

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
  namespace: prod
spec:
  podSelector:
    matchLabels:
      app: backend
```

```
 policyTypes:
 - Ingress
 ingress:
 - from:
   - podSelector:
       matchLabels:
         app: frontend
   ports:
   - protocol: TCP
     port: 8080
```

Note: requires CNI that supports NetworkPolicy (Calico, Cilium, etc.).

---

**10–11. A critical production Kubernetes cluster is experiencing multiple issues: Pods stuck in ImagePullBackOff / Pods being evicted / Users reporting 503 errors — What troubleshooting steps will you follow, and how to avoid this in the future?**

**Answer (structured troubleshooting + prevention):**

**Troubleshooting steps (quick triage)**

1. Gather context
   - kubectl get pods -A — overview
   - kubectl describe pod <problem-pod> — events and reasons
   - kubectl logs <pod> / kubectl logs <pod> -c <container> --previous
2. ImagePullBackOff
   - Check kubectl describe events: ErrImagePull or ImagePullBackOff.

- Verify image name/tag, registry credentials (imagePullSecrets), ECR auth (refresh token), network connectivity to registry.
- kubectl get secret for docker config; test docker pull <image> from node.

3. Pods evicted
   - kubectl describe node <node> and kubectl describe pod <pod> — eviction reason (e.g., nodefs, memory, diskpressure, pod-oom-kill).
   - Check node resource utilization: kubectl top node, kubectl top pod.
   - Free up disk, increase node size, set proper resource requests/limits, configure eviction thresholds.
4. 503 errors (users)
   - Check Ingress / Service: kubectl describe svc <svc>, kubectl get endpoints <svc> — ensure endpoints exist.
   - Check Ingress controller logs and ALB/NLB health checks.
   - Check application pod logs for errors and readiness/liveness probe failures.
   - Check Service type and port mappings; verify DNS.
5. Root causes correlation
   - ImagePull issues reduce pod replicas → endpoints missing → 503 from load balancer.
   - Node pressure evicts pods → insufficient replicas → 503.

Immediate mitigations

- Reapply correct imagePullSecrets or fix image tags.
- Scale up replica count / node group temporarily.
- Remove disk pressure (clean logs/images) or cordon/replace bad node.
- Roll back to known-good image.

**Long-term prevention**

- **Use readiness probes so traffic only goes to healthy pods.**
- **Use image pull policies and immutable tags (avoid latest).**
- **Implement auto-scaling (HPA/Cluster Autoscaler).**
- **Enforce resource requests & limits to avoid node OOM.**
- **Use private registry authentication automation (ECR credential helper).**
- **Monitoring & alerts: Prometheus/Grafana, node exporter, cluster autoscaler metrics.**
- **Use CI tests for container images and staging rollout before production.**

---

**12. How do you handle Terraform state file corruption?**

**Answer (steps):**

1. **Stop all Terraform runs to avoid concurrent damage.**
2. **Restore from backup: Use your backend snapshots (S3 versioning or CI/CD artifact) — terraform state pull to inspect current state.**
3. **If part of the state is corrupt: use terraform state rm to remove corrupt resources, then terraform import to reimport correct resources.**
4. **Locking and remote backend: Use S3 with DynamoDB lock to prevent corruption; turn on S3 versioning to roll back.**
5. **Manual fix: terraform state edit (not recommended) — use only as last resort and with extreme caution.**

---

**13. You need to import an existing AWS VPC into Terraform. What are the steps?**

**Answer (steps):**

1. **Create resource block in your .tf (example):**

```
resource "aws_vpc" "existing" {
  # fill attributes later
}
```

2. Run:

```
terraform init
terraform import aws_vpc.existing vpc-0abc12345
```

3. Run terraform plan — it will show differences. Populate the .tf with the actual attributes output by terraform show or use tools like terraformer to auto-generate TF code.
4. Import related resources (subnets, route tables, security groups) one by one and reconcile with config.

---

**14. How do you manage secrets in Terraform without hardcoding them?**

Answer (best practices):

- Don't store secrets in .tf files or checked-in terraform.tfstate unencrypted.
- Use remote backends (S3 with SSE and DynamoDB locking).
- Use environment variables for sensitive variables (TF_VAR_db_password).
- Use Terraform Cloud/Enterprise variables with sensitive flag.
- Integrate external secret stores: aws_secretsmanager_secret (represent secrets in SM) or use external data source to fetch secrets from Vault / AWS Secrets Manager at runtime.
- Use sensitive = true in variable blocks to prevent output.

---

**15. How would you implement cross-account resource provisioning using Terraform?**

Answer (pattern): Use multiple providers with assume_role and provider aliases.

**Example:**

```
provider "aws" {
  alias  = "accountA"
  region = "us-east-1"
  # credentials for account A
}


provider "aws" {
  alias = "accountB"
  region = "us-east-1"
  assume_role {
    role_arn = "arn:aws:iam::222222222222:role/TerraformRole"
  }
}
resource "aws_s3_bucket" "bucket_in_b" {
  provider = aws.accountB
  bucket   = "my-bucket-account-b"
}
```

**Grant the IAM role in each account the minimal permissions and use STS assume-role for security.**

---

**16. An S3 bucket was created via Terraform, but someone manually added a policy. How do you handle this drift?**

**Answer (approach):**

1. **Detect drift: terraform plan will show differences.**
2. **Decide:**

    - If manual change is authorized, update your Terraform code to include the new policy so future applies will keep it (adopt the change).
    - If change is unauthorized, revert it — update Terraform to desired state and terraform apply to remove manual change.
3. Prevention: enable IAM permissions guardrails, use AWS Config rules to detect non-compliant changes, enforce workflows (pull requests, code reviews), and restrict console access.

---

**17. Write a Python script to backup all files older than 30 days from a directory.**

**Answer (script):**

```
#!/usr/bin/env python3

import os

import shutil

import time

from pathlib import Path


SOURCE_DIR = "/path/to/source"

BACKUP_DIR = "/path/to/backup"

DAYS = 30

CUT_OFF = time.time() - (DAYS * 24 * 60 * 60)


os.makedirs(BACKUP_DIR, exist_ok=True)


for root, dirs, files in os.walk(SOURCE_DIR):

    for fname in files:
        full_path = os.path.join(root, fname)
        try:
            mtime = os.path.getmtime(full_path)
            if mtime < CUT_OFF:
                # preserve relative path
                rel_path = os.path.relpath(full_path, SOURCE_DIR)
                dest_path = os.path.join(BACKUP_DIR, rel_path)
                os.makedirs(os.path.dirname(dest_path), exist_ok=True)
                shutil.move(full_path, dest_path)
                print(f"Moved: {full_path} -> {dest_path}")
        except Exception as e:
            print(f"Error processing {full_path}: {e}")
```

(Explain: script moves older files to backup directory preserving structure. You can change shutil.move ➜ shutil.copy2 if you want copy instead.)

---

18. Your company's cloud costs are increasing rapidly. How would you approach cost optimization without impacting performance?

Answer (framework + quick wins):

1. Measure & Identify

- Enable cost allocation tags, use Cost Explorer, and identify top cost drivers (EC2, RDS, EBS, S3, Data Transfer).
- Set budget alerts and daily spending dashboards.

2. Quick wins

- Right-size instances: identify underutilized EC2 instances and downsize or use burstable instance families.

- Reserved Instances / Savings Plans: for predictable workloads use RIs or Savings Plans (1–3 year).
- Terminate unused resources: stray EBS volumes, unattached Elastic IPs, idle load balancers.
- S3 lifecycle rules: move infrequently accessed data to IA/Glacier.
- Use spot instances for non-critical or batch workloads.

3. Architectural changes (sustained)
   - Auto Scaling (scale down at night) and Cluster Autoscaler for k8s.
   - Use managed services where they reduce ops cost (RDS/Aurora vs self-managed DB).
   - Optimize data transfer with CloudFront and correct region placement.
4. Process & Governance
   - Implement tagging + chargeback to hold teams accountable.
   - CI/CD cost gates, review infrastructure changes in PRs for cost impact.
   - Use budgets & automated actions to prevent runaway spend.
5. Monitoring & Continuous Improvement
   - Use cost anomaly detection, continuous rightsizing recommendations, and schedule regular cost reviews.