Here are **real-time scenario-based interview questions and answers** based on the job description:

1. Software Development & TDD

Q1: You are developing a new microservice in Java/Python. How would you apply Test-Driven Development (TDD) in your workflow?

A1:

TDD involves writing tests before writing the actual code. Here's how I would apply it:

- **Write a failing test case** – Using JUnit (Java) or PyTest (Python), I write a unit test for a function that doesn't exist yet.
- **Implement minimal code** – Just enough to make the test pass.
- **Run the test** – It should initially fail.
- **Write the function logic** – Implement the actual business logic.
- **Refactor & optimize** – Clean up the code while ensuring tests still pass.

Example:

- Suppose I need a function to check if an email is valid.
- I first write a unit test checking different email formats.
- Implement the isValidEmail() function to pass the tests.
- This ensures correctness before moving to integration testing.

2. DevOps Tools (JFrog, SonarQube, Docker)

Q2: Your team complains about slow dependency resolution in

JFrog Artifactory. How would you troubleshoot and optimize it?

A2:

- **Check network and DNS latency** – Slow downloads may be due to network issues.
- **Analyze repository structure** – Use **local vs. remote repositories** effectively.
- **Enable caching** – Configure Artifactory to cache remote dependencies.
- **Use virtual repositories** – Combine multiple sources for faster resolution.
- **Monitor Artifactory**

performance – Use logs (artifactory.log) and metrics to find bottlenecks.

Example:

If a Maven build is slow, I'd check ~/.m2/settings.xml for misconfigured repository URLs and analyze Artifactory logs for 404s or long response times.

3. CI/CD & GitHub Actions

Q3: You notice frequent build failures in a GitHub Actions pipeline. How do you debug and fix them?

A3:

- **Check logs** – Use GitHub Actions logs (Actions > Failed Job > View logs).
- **Run jobs locally** – Debug using act (GitHub Actions runner for local testing).
- **Verify secrets & environment variables** – Check if missing or incorrect.
- **Isolate issues** – Run pipeline jobs separately to pinpoint failure.
- **Optimize caching** – Use actions/cache to reduce redundant downloads.

Example:
A pipeline using pytest for testing fails. Checking logs shows missing dependencies. Adding cache: 'pip' speeds up installation and fixes the issue.

4. Cloud (AWS, Azure, GCP) & Infrastructure as Code (Terraform)

Q4: Your team wants to migrate an on-premises application to AWS with minimal downtime. How would you approach this using Terraform?

A4:

- **Assess the current infrastructure** – Identify dependencies and networking.
- **Write Terraform scripts** – Define **EC2, RDS, VPC, and IAM roles**.
- **Use Terraform state management** – Store state in S3 with DynamoDB locking.
- **Test deployment in a staging environment** – Validate application behavior.
- **Perform blue-green deployment** – Minimize downtime by switching traffic.

Example:

If moving a PostgreSQL database to AWS RDS, I'd first use Terraform to create an **RDS instance with a read replica**. Then, I'd switch the application's database connection to the new instance during off-peak hours.

5. Troubleshooting & Debugging

Q5: Your application in a Kubernetes cluster is running but not responding. How would you debug it?

A5:

- **Check pod status:** kubectl get

pods -o wide

- **Inspect logs:** kubectl logs <pod-name>
- **Check events:** kubectl get events
- **Describe the pod:** kubectl describe pod <pod-name>
- **Check network issues:** kubectl exec -it <pod-name> -- curl <service-url>

Example:

If a pod is failing due to an OOM (Out of Memory) issue, I'd increase resource requests in the deployment YAML: resources: requests: memory:

"512Mi" cpu: "250m" limits: memory: "1Gi" cpu: "500m"
Then, restart the pod to apply changes.

6. Automation & Scripting

Q6: You need to automate log analysis from multiple servers. How would you achieve this using Bash/Python?

A6:

- **Bash Approach:**

```
for server in server1 server2 server3; do ssh $server 'grep "ERROR" /var/log/app.log' >> combined_errors.log done
```

- **Python Approach:**

```
import paramiko servers = ["server1", "server2"] for server in servers: ssh = paramiko.SSHClient() ssh.set_missing_host_key_policy (paramiko.AutoAddPolicy()) ssh.connect(server, username="user", password="pass") stdin, stdout, stderr = ssh.exec_command('grep "ERROR" /var/log/app.log') print(stdout.read().decode()) ssh.close()
```

Example:

If monitoring logs for failed

authentication attempts, I'd set up a **cron job** running every 5 minutes to notify the team.

7. Security Best Practices

Q7: How do you ensure secure application development in a DevOps environment?

A7:

- **Code scanning** – Use SonarQube, GitHub CodeQL.
- **Dependency scanning** – Use JFrog Xray, Snyk.
- **Secrets management** – Store secrets in AWS Secrets Manager, HashiCorp Vault.
- **Least privilege principle** –

Limit IAM roles and permissions.

- **CI/CD security checks** – Implement security gates before deployment.

Example:

If an API key is accidentally committed to a Git repository, I'd use:

```
git filter-branch --force --index-filter \ "git rm --cached --ignore-unmatch path/to/file" \ --prune-empty --tag-name-filter cat -- --all
```

Then, I'd rotate the compromised key immediately.

8. API & Integration

Q8: Your API is experiencing high latency under load. How would you diagnose and optimize it?

A8:

- **Check API Gateway logs** – Look for slow endpoints.
- **Enable caching** – Use AWS API Gateway caching or Redis.
- **Optimize database queries** – Add indexing, use pagination.
- **Use async processing** – Shift long-running tasks to queues (RabbitMQ, Kafka).
- **Enable autoscaling** – Increase instances dynamically.

Example:

If an API fetching user orders is slow due to large database queries, I'd paginate results using:
SELECT * FROM orders WHERE user_id = 1 LIMIT 10 OFFSET 0;
Then, return next_page_token for client-side handling.

Conclusion

These scenario-based questions test real-world problem-solving skills. Preparing with hands-on experience in **CI/CD, DevOps tools, cloud platforms, automation, security, and debugging** will help you succeed

in the interview.
Would you like me to generate a **mock interview session** or focus on specific topics?

Difference Between grep, awk, and sed in Linux (With Real-Time Examples)

1. grep (Global Regular Expression Print)

Use Case: Searching for error messages in log files.
Example: Find all occurrences of "ERROR" in a log file:

grep "ERROR" /var/log/syslog

Output:

Feb 25 10:05:12 server1 app: ERROR - Database connection failed Feb 25 10:06:45 server1 app: ERROR - Invalid user input

Explanation: grep filters lines containing "ERROR" from /var/log/syslog.

2. awk (Pattern Scanning & Processing)

Use Case: Extracting and formatting specific columns from structured data.

Example: Get the second column

(process names) from the ps command output:

```
ps aux | awk '{print $2, $11}'
```

Output:

```
1234 nginx 5678 postgres 9101 java
```

Explanation:

- $2 is the **process ID (PID)**
- $11 is the **process name**

Another Example: Calculate total memory usage from free -m:

```
free -m | awk '/Mem:/ {print "Used Memory: " $3 "MB out of " $2 "MB"}'
```

Output:

```
Used Memory: 2048MB out of
```

4096MB

3. sed (Stream Editor)

Use Case: Modifying text files by replacing content.

Example: Replace "foo" with "bar" in a file (test.txt):

sed 's/foo/bar/g' test.txt

Explanation:

- s/foo/bar/g → Replace "foo" with "bar" globally in each line.

Example: Delete all lines containing "DEBUG" in a log file:

sed '/DEBUG/d' /var/log/app.log

When to Use Each Command?

- **Use grep** when **you just need to find** lines matching a pattern.
- **Use awk** when **you need to extract, filter, or calculate** specific fields in structured data.
- **Use sed** when **you need to modify text, replace strings, or delete lines**.

Would you like more advanced examples, such as using these tools in **DevOps automation scripts**?