# CI/CD and DevOps Interview Questions and Answers

## 1. Explain the CI/CD workflow you follow and the kind of pipeline you use. How do you define and invoke pipelines in Jenkins?

CI/CD workflow involves continuous integration, testing, and deployment of code. In Jenkins, pipelines are defined using Jenkinsfile written in Groovy syntax. The pipeline can be declarative or scripted. It includes stages like build, test, and deploy. Pipelines are invoked automatically via webhooks or manually through Jenkins UI.

## 2. What are shared libraries in Jenkins, and how are they written and defined?

Shared libraries in Jenkins allow reuse of common pipeline code across multiple Jenkinsfiles. They are defined in a separate repository or folder structure with vars and src directories. The library is loaded using '@Library' annotation in Jenkinsfile.

## 3. What kind of applications do you deploy using Jenkins pipelines, and what deployment tools do you use?

Applications include web apps, microservices, and APIs. Deployment tools used include Kubernetes, Helm, Docker, AWS CLI, Terraform, and Ansible. Jenkins pipelines automate the deployment process using these tools.

## 4. If the Jenkins pipeline runs but the build doesn’t happen, what possible issues could be causing it?

Possible issues include misconfigured build steps, missing build tools, incorrect paths, permission issues, or failed pre-checks. Logs should be checked for errors and pipeline syntax validated.

## 5. What is the purpose of a webhook, and how is it used in a CI/CD pipeline?

A webhook is a callback URL triggered by events like code commits. In CI/CD, it notifies Jenkins to start a pipeline when changes are pushed to a repository.

## 6. How do you create and manage Kubernetes clusters (using tools like Terraform), and what are the master and worker nodes?

Kubernetes clusters are created using Terraform scripts that define infrastructure as code. Master nodes manage cluster state and scheduling, while worker nodes run application pods.

## 7. What are common Kubernetes errors you’ve faced (like CrashLoopBackOff, ImagePullError), and how did you resolve them?

CrashLoopBackOff occurs due to application crashes or misconfigurations. ImagePullError happens when the image is unavailable or credentials are missing. Resolutions include checking logs, fixing configurations, and ensuring image availability.

## 8. What is the command to access a pod and how can you define or create a Kubernetes class or object?

To access a pod: kubectl exec -it <pod-name> -- /bin/bash. Kubernetes objects like Deployment, Service, and ConfigMap are defined using YAML files and created using kubectl apply -f <file>.

## 9. Explain the folder structure of a basic Helm chart. What commands do you use to deploy with Helm?

Helm chart structure includes Chart.yaml, values.yaml, templates/, and charts/. To deploy: helm install <release-name> <chart-path> and helm upgrade <release-name> <chart-path>.

## 10. What are the stages in a Docker image build? Why do we use ENTRYPOINT and CMD instructions?

Stages include base image selection, copying files, installing dependencies, and setting execution commands. ENTRYPOINT sets the main command, CMD provides default arguments.

## 11. How do you manage and connect services like DBs, EC2, EKS, or ECS? Include the command to connect to ECS.

Services are managed using AWS CLI, Terraform, and IAM roles. To connect to ECS: aws ecs execute-command --cluster <cluster-name> --task <task-id> --container <container-name> --interactive --command '/bin/bash'.

## 12. Which container registry do you use for storing Docker images?

Common registries include Docker Hub, AWS ECR, Azure Container Registry, and GitHub Container Registry. AWS ECR is used with authentication and push commands.

**Advanced CI/CD and DevOps Interview Answers**

**1. What branching strategy do you follow, and how do you handle merges to avoid breaking the release branch? If a bug appears in production, what’s your approach to resolving it?**

We follow the Git Flow branching strategy, which includes feature branches, develop, release, and hotfix branches. All new features are developed in feature branches and merged into the develop branch after code review and testing. The release branch is created from develop and undergoes final testing before being merged into main and tagged for production. To avoid breaking the release branch, we enforce pull request reviews, automated tests, and CI checks. If a bug appears in production, we create a hotfix branch from main, fix the issue, test it, and merge it back into both main and develop to ensure consistency.

**2. Describe your typical deployment flow and CI/CD workflow. What stages do you define in your Jenkins pipeline, and how do you ensure full quality checks during deployment?**

Our CI/CD workflow includes stages such as Checkout, Build, Unit Test, Integration Test, Security Scan, Docker Build, Push to Registry, Deploy to Staging, Acceptance Test, and Deploy to Production. Jenkins pipelines are defined using declarative syntax in Jenkinsfiles. We ensure quality checks by integrating tools like SonarQube for code quality, Trivy for image scanning, and automated test suites. Each stage has conditional checks and notifications for failures.

**3. How do you use Jenkins shared libraries? Explain their typical structure and how they are integrated into your Jenkinsfiles.**

Jenkins shared libraries allow us to reuse common pipeline code across multiple projects. They are stored in a separate Git repository and follow a specific structure: vars/ for global functions, src/ for custom classes, and resources/ for static files. We integrate them into Jenkinsfiles using the @Library annotation and call shared functions directly. This promotes DRY principles and simplifies pipeline maintenance.

**4. Are you aware of security scanning tools? How do you scan Docker images—both during build and at the registry level? Are you using any extensions or tools for image scanning?**

Yes, we use tools like Trivy, Clair, and Aqua Security for scanning Docker images. During the build stage, Trivy is integrated into the Jenkins pipeline to scan the image before pushing to the registry. At the registry level, we use Harbor which supports image scanning and vulnerability reporting. These tools help ensure compliance and reduce security risks.

**5. How do you pass environment variables during Docker build commands? What services do you use for storing Docker images?**

Environment variables can be passed using the --build-arg flag in Docker build commands. For example: docker build --build-arg ENV\_VAR=value . We use AWS ECR and Docker Hub for storing Docker images. ECR provides secure, scalable storage and integrates well with AWS services.

**6. How do you establish a connection with databases in your deployments or infrastructure setup?**

Database connections are established using environment variables or secrets stored in tools like AWS Secrets Manager or HashiCorp Vault. During deployment, these secrets are injected into the application containers. We also use Kubernetes ConfigMaps and Secrets for managing DB credentials securely.

**7. How do you handle authentication for EKS clusters and store secrets securely in your environment?**

Authentication for EKS clusters is managed using IAM roles and aws-auth ConfigMap. We use kubectl and AWS CLI to configure access. Secrets are stored securely using AWS Secrets Manager or Kubernetes Secrets, and access is controlled via RBAC policies and IAM roles.

**8. How do you create AWS Lambda functions and manage the artifacts for deployment? What options do you use to push artifacts to Lambda?**

AWS Lambda functions are created using the AWS CLI, SAM (Serverless Application Model), or Terraform. Artifacts are packaged as ZIP files and pushed using the AWS CLI or SAM deploy. We also use CI/CD pipelines to automate packaging and deployment to Lambda.

**9. What is email signing and Helm chart signing? Which tools do you use to sign Helm charts?**

Email signing ensures the authenticity and integrity of email messages using digital signatures. Helm chart signing involves signing the chart package using GPG to verify its source and integrity. We use Helm's built-in signing feature with GPG keys and tools like cosign for enhanced security.

**1. Your pod keeps getting stuck in CrashLoopBackOff, but logs show no errors. How would you approach debugging and resolution?**  
Answer:- Check: kubectl describe pod for livenessProbe, readinessProbe, and OOMKilled.  
Debug: Resource limits too low, misconfigured probes, or failing commands in the entrypoint.  
Fix: Tune probes or increase CPU/memory requests/limits. Check container start command for logic errors.  
  
**2. You have a StatefulSet deployed with persistent volumes, and one of the pods is not recreating properly after deletion. What could be the reasons, and how do you fix it without data loss?**  
Answer: Cause: PVCs are tied to specific StatefulSet pod names (pod-0, pod-1, etc.).  
Fix: Ensure PVC still exists and matches the pod name. Do not delete PVCs manually. Restart StatefulSet if needed with kubectl rollout restart.  
  
  
**3. Your cluster autoscaler is not scaling up even though pods are in Pending state. What would you investigate?**  
Answer: Check: kubectl describe pod for unschedulable reason.  
Fix: Ensure:  
Node group has scaling enabled  
Requests are not too large for a node  
No taints preventing scheduling  
Pod has tolerations and nodeSelector/affinity match  
  
  
**4. A network policy is blocking traffic between services in different namespaces. How would you design and debug the policy to allow only specific communication paths?**  
Answer:   
Debug:  
Use kubectl get networkpolicy and test with netshoot or busybox pods.  
Use labels, not IPs.  
  
  
**5. One of your microservices has to connect to an external database via a VPN inside the cluster. How would you architect this in Kubernetes with HA and security in mind?**Answer:  
Options:  
Use a sidecar VPN container or a VPN DaemonSet  
Route DB traffic via a private network interface (e.g., VPC peering)  
Use NetworkPolicy, TLS, and secrets management for secure access  
HA: Run multiple VPN pods behind a ClusterIP or use a LoadBalancer.

**1. What is a Readiness Probe?**

**✅ Definition:**

A **readiness probe** determines whether a container is **ready to start accepting traffic**. It tells Kubernetes if the application inside the container is fully initialized and prepared to handle requests.

**🧭 Why It’s Used:**

* Prevents traffic from being sent to a pod that is still starting up or temporarily unable to serve.
* Ensures high availability by routing traffic only to healthy pods.
* Helps during rolling updates or restarts to avoid downtime.

**⚙️ How It Works:**

* Kubernetes checks the readiness probe at regular intervals.
* If the probe fails, the pod is **removed from the Service endpoints**.
* Once the probe passes, the pod is added back to the endpoints.

**🛠️ Common Probe Types:**

* httpGet: Makes an HTTP request to a specified path and port.
* tcpSocket: Checks if a TCP connection can be established.
* exec: Runs a command inside the container.

**📄 Example:**

**❤️ 2. What is a Liveness Probe?**

**✅ Definition:**

A **liveness probe** checks whether a container is **still running and healthy**. It helps detect situations where the application is stuck or has crashed.

**🧭 Why It’s Used:**

* Automatically restarts containers that are unresponsive or deadlocked.
* Improves fault tolerance and self-healing of applications.
* Prevents manual intervention for recovery.

**⚙️ How It Works:**

* Kubernetes periodically runs the liveness probe.
* If the probe fails, Kubernetes **restarts the container**.
* This does not affect the pod's inclusion in service endpoints.

**🛠️ Common Probe Types:**

Same as readiness probes:

* httpGet
* tcpSocket
* exec

**📄 Example:**

**🔁 Key Differences**

| **Feature** | **Readiness Probe** | **Liveness Probe** |
| --- | --- | --- |
| **Purpose** | Is the app ready to serve traffic? | Is the app alive and functioning? |
| **Action on Fail** | Remove pod from service endpoints | Restart the container |
| **Use Case** | Startup delays, temporary unavailability | Deadlocks, crashes, hangs |
| **Impact** | Affects traffic routing | Affects container lifecycle |

**🧪 Best Practices**

* Use **readiness probes** for apps that take time to initialize or depend on external services (like databases).
* Use **liveness probes** to detect and recover from runtime failures.
* Avoid overly aggressive probe settings to prevent false positives and unnecessary restarts.

Would you like help configuring these probes for a specific application or YAML file?