CI/CD - DevOps Interview Questions

Beginner Level (1-20 Questions)

1. What is CI/CD in DevOps?

Answer:
CI/CD stands for **Continuous Integration (CI) and Continuous Deployment (CD)**.

- **CI (Continuous Integration):** Developers frequently merge code into a shared repository, and automated tests are run to catch issues early.
- **CD (Continuous Deployment/Delivery):** Automates the deployment of software.
    - **Continuous Delivery:** Requires manual approval before deployment.
    - **Continuous Deployment:** Fully automated, no manual intervention.

2. What are the benefits of using CI/CD?

Answer:

- **Faster Releases:** Automates software delivery.
- **Early Bug Detection:** Runs tests automatically on new code.
- **Improved Collaboration:** Developers merge code frequently, reducing integration issues.
- **Consistent Deployments:** Eliminates manual errors with automated builds and releases.

3. What are some popular CI/CD tools?

Answer:

- **Jenkins** – Open-source, highly customizable.
- **GitHub Actions** – Integrated with GitHub.
- **GitLab CI/CD** – Built-in with GitLab.
- **CircleCI, Travis CI** – Cloud-based solutions.
- **Azure DevOps Pipelines, AWS CodePipeline** – Cloud-native CI/CD.

4. What is a CI pipeline?

Answer:
A **CI pipeline** is an automated workflow that builds, tests, and validates new code before merging it into production.

- Steps: **Code Commit → Build → Test → Artifact Storage → Deployment**
- Example tools: **Jenkinsfile, GitHub Actions YAML, GitLab CI/CD YAML**

5. What is a build artifact in CI/CD?

Answer:
A **build artifact** is a compiled and packaged version of code ready for deployment.

- Examples:
  - **JAR, WAR, or ZIP files** for Java projects
  - **Docker images** for containerized applications

6. How does source control work in CI/CD?

Answer:
Source control (e.g., **Git**) helps track changes in code.

- Developers push code to repositories (**GitHub, GitLab, Bitbucket**).
- CI/CD tools trigger **automated builds and tests** on new commits.

7. What is the purpose of unit tests in CI/CD?

Answer:
Unit tests validate **individual components of code** to catch early-stage bugs.

- Tools: **JUnit, pytest, Mocha, Jest**
- Example:

```
def add(x, y):
    return x + y

def test_add():
    assert add(2, 3) == 5
```

8. What is versioning in CI/CD?

Answer:
Versioning assigns unique numbers to each software release to track changes.

- **Semantic Versioning (SemVer): MAJOR.MINOR.PATCH** (e.g., **1.2.3**)
- **Git Tags:** CI/CD pipelines deploy specific versions using tags.

9. What is a rollback in CI/CD?

Answer:
A rollback reverts to a previous stable release when the new deployment fails.

- Example: Rolling back an application using Kubernetes:

```
kubectl rollout undo deployment my-app
```

10. What is a canary deployment?

Answer:
Canary deployment releases new changes to a **subset of users** before full deployment.

- Example: **Deploy to 10% of users → Monitor logs → Full release**

Intermediate Level (21-40 Questions)

21. What is the difference between GitHub Actions and GitLab CI/CD?

Answer:

Feature	GitHub Actions	GitLab CI/CD
Integration	Best with GitHub	Best with GitLab
Configuration	`.github/workflows/*.yml`	`.gitlab-ci.yml`
Runners	GitHub-hosted & self-hosted	GitLab Runners

Feature	GitHub Actions	GitLab CI/CD
Container Support	Uses Docker containers	Strong native container support

22. How do you trigger a Jenkins pipeline?

Answer:
Jenkins pipelines can be triggered using:

- **Webhooks:** Automatically triggered by a Git commit.
- **Cron Jobs:** Run at scheduled times.
- **Manually:** Click 'Build Now' in Jenkins UI.

23. What is a deployment strategy?

Answer:
Deployment strategies ensure smooth updates. Common types:

- **Rolling Deployment:** Replaces old instances gradually.
- **Blue-Green Deployment:** Deploys new version alongside the old one.
- **Canary Deployment:** Releases updates to a small group first.

24. How do you secure CI/CD pipelines?

Answer:

- **Use Secrets Management** (e.g., HashiCorp Vault, AWS Secrets Manager).
- **Restrict Access:** Use role-based access control (RBAC).
- **Scan for Vulnerabilities:** Use tools like **Snyk, SonarQube**.

25. How do you integrate CI/CD with Infrastructure as Code (IaC)?

Answer:
Integrating **CI/CD with Infrastructure as Code (IaC)** ensures that infrastructure changes are automated and version-controlled.

- **Best Practices:**
    - Store IaC scripts (**Terraform, Ansible, CloudFormation**) in **Git**.

    - Use **automated testing** (e.g., `terraform validate`, `ansible-lint`).
    - Apply changes using CI/CD pipelines (`terraform apply`).

- **Example GitHub Actions Pipeline for Terraform:**

```
jobs:
  terraform:
    steps:
      - run: terraform init
      - run: terraform validate
      - run: terraform apply -auto-approve
```

26. What is a pipeline as code?

Answer:
Pipeline as Code means defining **CI/CD workflows using configuration files**.

- Example tools: **Jenkinsfile, GitHub Actions YAML, GitLab CI/CD YAML**.

- **Example Jenkinsfile:**

```
pipeline {
  agent any
  stages {
    stage('Build') { steps { sh 'mvn package' } }
    stage('Test') { steps { sh 'mvn test' } }
  }
}
```

27. What is an ephemeral build environment in CI/CD?

Answer:
An **ephemeral build environment** is a temporary environment spun up **only during the build process** and discarded after execution.

- Used in **GitHub Actions Runners, Jenkins Agents, Kubernetes Jobs**.
- **Benefits:**
    - Ensures **clean state** for each build.
    - Reduces **resource costs**.

28. What is the purpose of a staging environment in CI/CD?

Answer:
A **staging environment** replicates production to test before deployment.

- **Why it matters:**
  - Helps catch bugs **before they reach production**.
  - Enables **performance testing, security testing**.
- **CI/CD flow:**
  - Dev → QA → **Staging** → Production

29. How does a monorepo impact CI/CD pipelines?

Answer:
A **monorepo** is a single repository for multiple projects/services.

- **Challenges:**
  - Running **CI/CD for only changed services** can be complex.
  - **Large build times** if not optimized.
- **Solution:**
  - Use **Bazel, NX, or GitHub Actions path filters** to build/test only **modified code**.

30. What are pipeline triggers, and how are they used?

Answer:
Triggers **automatically start CI/CD workflows** based on specific events.

- **Examples:**
  - **Git Push:** Run pipeline when new code is pushed.
  - **Pull Requests:** Trigger tests before merging.
  - **Schedule:** Run a job every night (`cron`).
- **Example GitLab CI/CD trigger:**

```
trigger:
  event: push
```

31. What is artifact versioning in CI/CD?

Answer:
Versioning assigns **unique identifiers** to builds for tracking.

- **Best Practices:**

    - Use **Semantic Versioning (1.2.3)** for clarity.
    - Tag artifacts using commit hashes (`v1.0.0-commitSHA`).

- **Example:**

```
docker tag my-app:latest my-app:1.2.3
```

32. How do you handle environment variables in CI/CD?

Answer:

- Use **.env files** or **CI/CD secrets storage**.
- **Example GitHub Actions Environment Variable:**

```
env:
  NODE_ENV: production
```

- **Best Practices:**
    - **Never hardcode secrets.**
    - Use tools like **Vault, AWS Secrets Manager**.

33. What is a multi-branch pipeline in CI/CD?

Answer:
A **multi-branch pipeline** runs different workflows for different Git branches.

- **Example (Jenkins):**
    - `main` → Deploy to production.
    - `develop` → Deploy to staging.
- **Jenkinsfile example:**

```
if (env.BRANCH_NAME == 'main') {
    deployToProd()
} else {
    deployToStaging()
}
```

34. How do you automate rollback in CI/CD?

Answer:
If a deployment fails, CI/CD should **automatically revert to a stable version**.

- **Strategies:**
  - **Git Revert:** Roll back code changes.
  - **Kubernetes Rollback:** `kubectl rollout undo deployment my-app`.
  - **Feature Flags:** Disable a new feature without redeployment.

35. What is test-driven development (TDD), and how does it integrate with CI/CD?

Answer:
TDD means **writing tests before writing code**.

- **CI/CD Best Practice:**
  - Run unit tests **before merging code**.
  - Block deployment if tests fail.
- **Example:**

```
def test_addition():
    assert add(2, 3) == 5
```

36. How do you handle dependencies in a CI/CD pipeline?

Answer:
Managing dependencies ensures **consistent builds**.

- **Solutions:**
  - Use **lock files** (`package-lock.json`, `Pipfile.lock`).
  - Cache dependencies (`npm ci`, `pip freeze`).
- **Example:**

```
- uses: actions/cache@v3
  with:
    path: ~/.npm
    key: node-${{ hashFiles('**/package-lock.json') }}
```

37. What is containerized CI/CD?

Answer:
Running CI/CD jobs inside **containers** ensures **consistency and isolation**.

- **Tools:** Docker, Kubernetes, GitHub Actions.
- **Example:**

```
jobs:
  build:
    runs-on: ubuntu-latest
    container: node:16
```

38. How do you optimize CI/CD pipelines for speed?

Answer:

- **Run Tests in Parallel**
- **Cache Dependencies**
- **Use Lightweight Docker Images**
- **Only Deploy Changed Services**

39. What is an approval stage in CI/CD pipelines?

Answer:
An **approval stage** requires **manual approval** before deploying to production.

- **Example:**
  - GitLab CI/CD: `when: manual`.
  - Jenkins: Use `input` step.

40. How do you handle secrets in CI/CD pipelines?

Answer:
Secrets should **never be stored in Git**.

- **Solutions:**
  - **Vault, AWS Secrets Manager**.
  - **GitHub Secrets (`secrets.MY_SECRET`)**.
  - **Environment variables**.
- **Example:**

```
env:
  DATABASE_PASSWORD: ${{ secrets.DB_PASSWORD }}
```

Advanced Level (41-60 Questions)

41. What are self-hosted runners in CI/CD?

Answer:
Self-hosted runners are custom machines for executing CI/CD jobs instead of cloud-hosted ones.

- Example: GitHub Actions supports **Linux, Windows, macOS** runners.

42. How does caching improve CI/CD performance?

Answer:
Caching stores **dependencies** and **artifacts** to speed up builds.

- Example: Caching npm dependencies in GitHub Actions:

```
steps:
  - uses: actions/cache@v3
    with:
      path: ~/.npm
      key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
```

43. What is parallel execution in CI/CD?

Answer:
Parallel execution runs multiple tasks **simultaneously** to speed up pipelines.

- Example: Running multiple tests at once in Jenkins.

44. What is dynamic vs. static analysis in CI/CD security?

Answer:

- **Static Analysis:** Scans code **before execution** (e.g., SonarQube).
- **Dynamic Analysis:** Scans code **during runtime** (e.g., OWASP ZAP).

45. What is a feature flag, and how does it work in CI/CD?

Answer:
A feature flag enables/disables features without deploying new code.

- Example: Toggle dark mode using a flag instead of redeploying.

46. How do you handle secrets in CI/CD pipelines?

Answer:

- Use **environment variables** securely.
- Store secrets in **AWS Secrets Manager, HashiCorp Vault**.
- Example:
- ```
  secrets:
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  ```

47. What is observability in CI/CD?

Answer:
Observability means **monitoring logs, metrics, and traces** to debug CI/CD failures.

48. What is immutable infrastructure?

Answer:
Immutable infrastructure means **servers are never updated** but replaced instead.

49. What are the key metrics for CI/CD performance?

Answer:

- **Lead Time:** Time from commit to deployment.
- **Mean Time to Recovery (MTTR):** Time to recover from failures.

50. How do you ensure zero-downtime deployments?

Answer:
  ```

- **Use rolling updates, blue-green, and canary deployments.**
- **Deploy with Kubernetes and load balancers.**

51. What is a release train in CI/CD?

Answer:
A **release train** is a deployment strategy where software releases are scheduled at **fixed intervals**, rather than waiting for all features to be ready.

- Common in **Agile** environments.
- Ensures **predictability** and **reduces deployment risks**.
- Example: **Google Chrome** releases every 4 weeks regardless of pending features.

52. How do you handle database migrations in a CI/CD pipeline?

Answer:
Database migrations ensure **schema changes** are applied safely in an automated pipeline.

- Use tools like **Liquibase, Flyway, Django Migrations**.
- Steps in CI/CD:
  i. **Check migrations** before deployment (`liquibase validate`).
  ii. **Apply migrations** during deployment (`flyway migrate`).
  iii. **Rollback if failure** (`flyway undo`).
- Example in a pipeline (Flyway):

```
steps:
  - name: Apply database migrations
    run: flyway migrate -url=jdbc:mysql://db -user=root -password=secret
```

53. What is trunk-based development, and how does it impact CI/CD?

Answer:
Trunk-based development means developers **commit directly to the main branch** (trunk) instead of using long-lived feature branches.

- **Pros:**

    - **Faster CI/CD cycles** with fewer merge conflicts.
    - Reduces integration complexity.
- **Cons:**
    - Requires **strict automated testing** to prevent breaking changes.
- Example workflow:
    - Commit to `main` → Automated Tests → Deploy to Staging → Deploy to Production.

54. How do you implement blue-green deployments in Kubernetes?

Answer:
A **blue-green deployment** runs **two versions** of an application simultaneously, allowing **instant rollback** if issues occur.

- Steps:
    i. Deploy **new version (green)** while **old version (blue) stays live**.
    ii. Switch traffic to green using a **load balancer or Ingress**.
    iii. Rollback if issues arise by redirecting traffic back to blue.
- Example Kubernetes YAML:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: blue-green
spec:
  rules:
    - http:
        paths:
          - path: "/"
            backend:
              service:
                name: green-service
                port:
                  number: 80
```

55. What is a service mesh, and how does it help CI/CD?

Answer:
A **service mesh** is a dedicated infrastructure layer for handling **service-to-service communication** in microservices.

- Examples: **Istio, Linkerd, Consul**.
- Benefits in CI/CD:
    - **Canary deployments**: Route traffic gradually.
    - **A/B Testing**: Split traffic between versions.
    - **Security**: Implements zero-trust policies (e.g., **mTLS**).

56. What is progressive delivery in CI/CD?

Answer:
Progressive delivery is an **evolution of CI/CD** that deploys features gradually, rather than all at once.

- **Includes:**
    - **Feature Flags:** Enable/disable features dynamically.
    - **Canary Releases:** Test with a small user group first.
    - **A/B Testing:** Deploy different versions for analytics.

57. How do you handle long-running tests in CI/CD pipelines?

Answer:
Long-running tests slow down deployments. Strategies to optimize:

- **Parallel Test Execution:** Run tests across multiple machines.
- **Test Selection:** Run only impacted tests using **test impact analysis**.
- **Mocking Dependencies:** Reduce external calls using **Mockito, WireMock**.
- **Shift-Left Testing:** Run tests **early in the pipeline** to detect failures faster.

58. What is Chaos Engineering, and how does it fit into CI/CD?

Answer:
Chaos Engineering involves **intentionally injecting failures** to test system resilience.

- **Example tools:**
    - **Gremlin, LitmusChaos** (Kubernetes-based).
    - **AWS Fault Injection Simulator (FIS)**.
- **In CI/CD Pipelines:**
    - Add a **chaos test stage** before production deployment.

- Example:

```
steps:
  - name: Run Chaos Test
    run: gremlin attack --target kubernetes --cpu 90%
```

59. How do you implement immutable deployments in CI/CD?

Answer:
Immutable deployments mean **never modifying running instances**—instead, deploying a new version entirely.

- Best for **containers, serverless, and cloud-native applications**.
- Tools:
  - **Docker images** (`image: my-app:v2`).
  - **Infrastructure as Code (Terraform, CloudFormation)** to replace instances.
- **Example:**
  - **Bad approach:** `ssh into a server & update the app.`
  - **Good approach:** `Deploy a new container & replace old one.`

60. What are the best practices for securing CI/CD pipelines?

Answer:
To secure CI/CD, follow **these best practices**:
✅ **Use Secret Management:** Store secrets in **Vault, AWS Secrets Manager, or Kubernetes Secrets**.
✅ **Enable Role-Based Access Control (RBAC):** Restrict who can trigger deployments.
✅ **Enforce Code Signing:** Sign artifacts to ensure they are not tampered with.
✅ **Run Security Scans:** Use **SAST, DAST, and dependency scanning** tools.
✅ **Monitor CI/CD Pipelines:** Detect suspicious activity using **SIEM tools** like Splunk or Datadog.