Implementation of devops

Git

Git serves as a distributed version control system to manage source code and track changes efficiently. It enables seamless collaboration among development teams, integrates with CI/CD pipelines for automated testing, and facilitates continuous deployment.

Jenkins

Jenkins is an open-source automation server used in DevOps to build, test, and deploy applications continuously.
It enables Continuous Integration (CI) and Continuous Delivery (CD) by automating workflows across the software development lifecycle

Relation between Git & Jenkins

Git is a distributed version control system that tracks and manages changes to code.
Jenkins is an automation server that uses Git (via plugins/webhooks) to trigger CI/CD workflows—automatically building, testing, and deploying code on every commit.

Integration between Git and Jenkins

1. Install Jenkins and Git

2. Install Required Plugins in Jenkins

3. Configure Git in Jenkins

4. Add GitHub Credentials

5. Create a Jenkins Job / Pipeline

6. Set Up Build Trigger

7. Configure GitHub Webhook

```
pipeline {
   agent any
   stages {
      stage('Checkout') {
         steps {
            git branch: 'main', url: 'git@github.com:user/repo.git'
         }
      }
      stage('Build') {
         steps {
            sh 'echo Building...'
         }
      }
   }
}
```

Docker

Docker packages applications and all their dependencies into lightweight, portable containers, ensuring consistent and

reproducible execution across development, testing, and production environments.
It optimizes CI/CD pipelines by accelerating deployment, improving isolation, and eliminating the "works on my machine" problem.

Implementation between Docker & Jenkins

Step-by-Step Integration of Docker with Jenkins

1. **Install Docker & Jenkins**
2. **Install Required Jenkins Plugins**
3. **Configure Docker in Jenkins**
4. **Add Docker Registry Credentials**
5. **Create a Jenkins Pipeline with Docker Commands**

```
pipeline {
 agent any
 environment {
  DOCKER_IMAGE = 'your-username/your-app'
  DOCKER_CREDENTIALS = 'docker-hub-credentials'
 }
 stages {
  stage('Checkout') {
   steps {
    git 'https://github.com/your/repo.git'
   }
```

```
    }
    stage('Build Image') {

    steps {
      script {
       docker.build(env.DOCKER_IMAGE)
      }
     }
    }
    stage('Push to Docker Hub') {
     steps {
      script {
       docker.withRegistry('https://registry.hub.docker.com',
env.DOCKER_CREDENTIALS) {
        docker.image(env.DOCKER_IMAGE).push('latest')
       }
      }
     }
    }
  }
}
```

6. **Run the Pipeline**

7. **Optional: Use `inside()` and Multi-Stage Scenarios**

```
def img = docker.build("my-app:${env.BUILD_ID}")

img.inside {

  sh 'make test'

}

img.push()
```

•