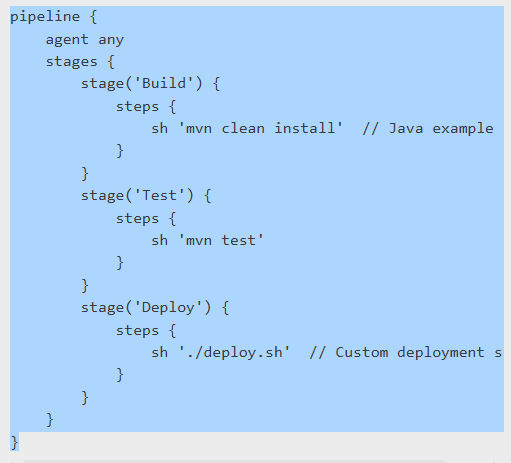
**Solution to Lab Exercise:**

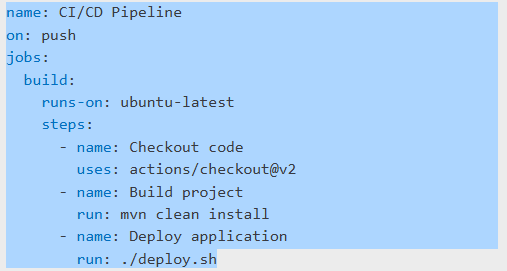
**Part 1: Setting Up a CI/CD Pipeline**

1. **Clone a Git repository**
   * Command: git clone <repo\_url>
   * This downloads the project repository to the local system.
2. **Create a sample application (Node.js/Java)**
   * Develop a simple application in Node.js or Java.
   * Example: A basic web server using Express.js (Node.js) or Spring Boot (Java).
3. **Configure a CI/CD pipeline in Jenkins, GitHub Actions, or GitLab CI/CD**
   * **Jenkins:**
     + Install required plugins (Git, Pipeline, Docker if needed).
     + Create a new pipeline job.
     + Use a Jenkinsfile to define build and deployment stages.
   * **GitHub Actions:**
     + Define .github/workflows/main.yml with build and deployment steps.
   * **GitLab CI/CD:**
     + Create .gitlab-ci.yml for pipeline automation.
4. **Define build, test, and deployment stages**

Example of Jenkinsfile:



Example of .github/workflows/main.yml:



**Part 2: Automating Deployment**

1. **Push code changes to the repository**
   * Command: git push origin main
   * This triggers the pipeline configured in Jenkins, GitHub Actions, or GitLab CI/CD.
2. **Observe the automated build and test execution**
   * Check the pipeline status in Jenkins/GitHub/GitLab.
   * Monitor logs for errors.
3. **Deploy the application automatically**
   * The deployment script (deploy.sh) is executed.
   * Application is deployed to the specified environment (e.g., server, cloud, container).
4. **Verify deployment**
   * Access the application URL.
   * Check logs (journalctl -u app.service for Linux, docker logs <container\_id> for Docker).

**Observations:**

* Pipeline execution status is recorded.
* Build and test results are validated.
* Deployment logs confirm successful deployment.

**Conclusion:**  
Automating deployment pipelines enhances software development by enabling rapid, reliable, and repeatable deployments. This experiment successfully demonstrated configuring and automating CI/CD workflows using Jenkins, GitHub Actions, and GitLab CI/CD.

**Viva Questions & Answers:**

1. **What is the purpose of CI/CD pipelines?**
   * They automate the process of building, testing, and deploying software.
2. **How does Jenkins automate software deployment?**
   * By using pipeline jobs with Jenkinsfile configurations.
3. **What is the role of GitHub Actions/GitLab CI/CD in deployment automation?**
   * They define workflows for CI/CD within repositories.
4. **How does Docker improve deployment automation?**
   * It packages applications into containers for easy deployment.
5. **How do you debug a failing CI/CD pipeline?**
   * Review pipeline logs and fix errors in scripts or configurations.

**References:**

* Jenkins Documentation (https://www.jenkins.io/doc/)
* GitHub Actions (https://docs.github.com/en/actions)
* GitLab CI/CD (<https://docs.gitlab.com/ee/ci/>)

**SOLUTION-02:**

**Part 1: Setting Up a CI/CD Pipeline**

1. **Clone a Git repository:**
2. git clone https://github.com/user/sample-repo.git

cd sample-repo

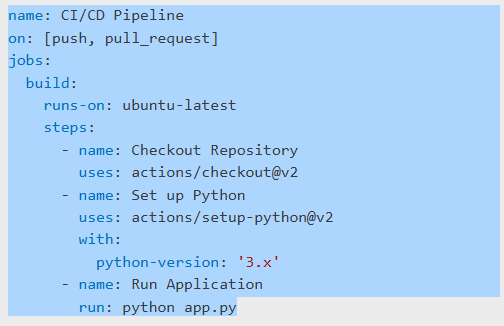
1. **Create a sample application (Python example):**
   * Create app.py:

print("Hello, CI/CD Pipeline!")

* + Add and commit the file:
    1. git add app.py
    2. git commit -m "Added sample application"
    3. git push origin main

**Configure a CI/CD pipeline using GitHub Actions:**

* + Create a .github/workflows/main.yml file:

Commit and push the file:

* + 1. git add .github/workflows/main.yml
    2. git commit -m "Added CI/CD pipeline"
    3. git push origin main

**Part 2: Automating Deployment**

1. **Push code changes to trigger the pipeline:**
   * Modify app.py:

print("CI/CD Pipeline Automation Successful!")

* + Commit and push:
    1. git add app.py
    2. git commit -m "Updated application output"
    3. git push origin main
  + GitHub Actions automatically runs the pipeline.

1. **Deploying the application using Docker (Optional):**
   * Create a Dockerfile:
     1. FROM python:3.8
     2. COPY app.py /app.py

CMD ["python", "/app.py"]

* + Build and run the container:
  + docker build -t my-app .

docker run my-app

1. **Verify Deployment:**
   * Check GitHub Actions logs for successful execution.
   * Run docker ps to confirm the container is running.

**Observations:**

* The pipeline automatically builds and executes the application.
* Any errors in execution can be found in GitHub Actions logs.
* The deployment process is triggered upon every code push.

**Conclusion:**  
Automating deployment pipelines ensures fast, reliable, and consistent software releases. This experiment successfully demonstrated configuring a CI/CD pipeline using GitHub Actions and deploying an application with Docker.