1 CI/CD Pipeline Design

1. Development
Objective: To commit and push code to the repository to trigger the CI pipeline.
Process:

- Code is developed using Java in IntelliJ IDEA.
- Commits are pushed to the main branch via the Git CLI, triggering the CI pipeline on GitHub Actions.
- Incremental pushes occur after each new feature implementation to ensure continuous validation.

2. Build
Objective: Ensure the application builds without errors.
Process:

- On every push, GitHub Actions runs the `mvn clean package` command.
- The system verifies dependencies and checks for syntax or compilation errors.
- Any compile-time issues, such as `NullPointerException` risks or unresolved dependencies, result in pipeline failure.

3. Test
Objective: Run all tests to validate the system's functionality and integration.
Process:

- Tests are triggered using the `mvn test` command.
- Includes unit tests (e.g., for input validation), integration tests (e.g., REST server communication), and system-level tests (e.g., endpoint responses).
- Failures at this stage halt the pipeline and prompt fixes before deployment.

4. Deploy
Objective: Package and deploy the application.
Process:

- Successfully built and tested applications are packaged as Docker images.
- Images are pushed to a Docker repository, ensuring consistency across deployments.

Automation Tools:

- **GitHub Actions:** Manages the pipeline workflow and executes commands across stages.
- **Docker:** Ensures consistent environments during deployment.

2 Test Automation

2.1 Unit Tests

For unit tests, I've implemented automated tests in Postman that run every hour. This schedule ensures continuous validation of the components' behavior after each change. By using Postman's built-in test scripting functionality, each unit test checks the expected output for various endpoints or services.

2.2 Future Plans for Automation

Looking ahead, the goal is to extend automation to include integration and system tests. While unit tests are fully automated at present, integration tests will be automated to check that various components interact correctly with one another. System-level testing will also be automated in the future, verifying that the application behaves as expected when processing specific inputs and generating the correct outputs. This can be done using Github actions, which I do in the next stage.

3 CI/CD Pipeline Demo for Your Project

3.1 Scenario 1 – Successful Pipeline Execution

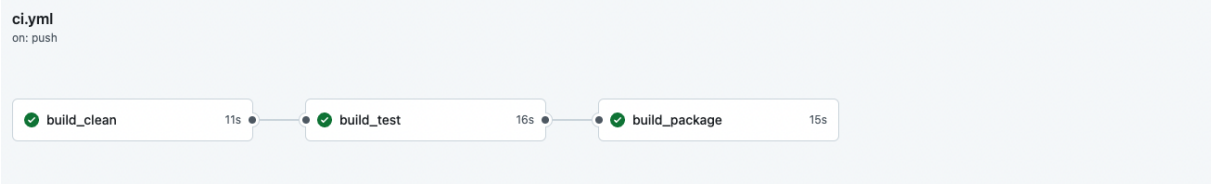

Figure 1: Successful CI/CD pipeline

Process Walkthrough:

- **Code Push to GitHub:**
  - The process begins with a developer pushing code changes to the GitHub repository.
  - This triggers the CI/CD pipeline, and the repository is pulled for the build process.
- **Build Stage:**
  - The project is built using Maven (`mvn clean install`). During this stage, the project is compiled, and dependencies are resolved.
  - As there are no compile-time errors in the code, the build completes successfully.
- **Test Execution:**
  - After the build, the automated tests are triggered to run. In this case, all the tests, including unit tests and integration tests, pass without any issues.
  - The tests ensure that the application behaves as expected and all code changes are functioning properly.
- **Packaging the Application:**
  - After a successful test execution, the project is packaged into a JAR file using the Maven command `mvn package`.

Outcome:

- Every stage of the CI/CD pipeline passes successfully.
- The project is in a ready-to-deploy state, with no errors during build or testing.

3.2 Scenario 2 – Unsuccessful Pipeline Execution (Test Failure)

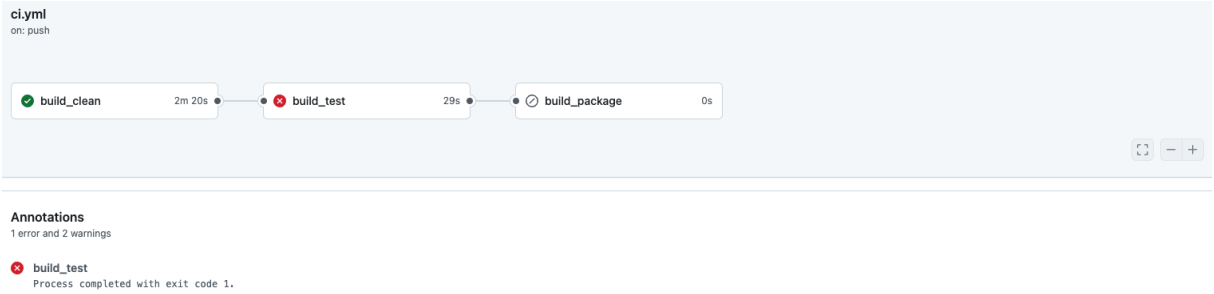

Figure 2: Failure at the test stage

Process Walkthrough:

- **Code Push to GitHub:**
  - Just like in Scenario 1, the process begins with a developer pushing code changes to GitHub.
  - This triggers the CI/CD pipeline.
- **Build Stage:**
  - The project is compiled using Maven, and there are no compile-time errors, so the build successfully completes.
- **Test Execution:**
  - However, when the automated tests are triggered, one or more tests fail.
  - This failure could be due to issues like failed assertions, incorrect logic, or bugs introduced by the new code changes.
- **Failure Impact:**
  - As the tests did not pass, the pipeline halts and does not proceed to the packaging stage.
  - The failure is reported back to the developer, indicating the specific tests that failed and providing relevant logs to help debug the issue.