Assessment 2: Brownfield

Development - Continuous Integration Report A

Cohort 1 Team 2

Team Members:
Trace Chinelle
Vidhi Chohan
Apollo Cowan
Siyuan Liu
Aryaman Marathe
Charlie Mason

Continuous Integration Methods and Approach Part A- Summary

Our project used some CI practices to ensure efficient development. We attempted this to automate workflows and minimise integration issues. GitHub Actions was selected as the CI platform for its seamless integration with our version control system and extensive library of reusable actions. Automated processes reduced manual work and allowed developers to focus on feature implementation and bug fixes .

1) **Automated Build and Testing**
   The CI pipeline is designed to automatically build the project and execute test cases whenever code is pushed to the repository or a pull request is made. The Gradle-based configuration ensures that the build process is streamlined, while the tests validate functionality and identify potential bugs early.
2) **Gradle-Based Dependency Management**
   The setup employs Gradle to manage dependencies. They are cached to reduce build time and ensure consistency in different environments. This allowed us to maintain a lightweight workflow.
3) **Version Control Integration**
   The pipeline integrates with GitHub, testing changes continuously. These are triggered via every commit to the main branch and pull request enabling us to identify bugs before merging changes.
4) **Feedback Mechanism**
   Developers receive immediate feedback that provides details of successful builds, test coverage, and any failures.

Continuous Integration Methods and Approach B- Infrastructure

CI Pipelines Used

1) **Automated Build and Testing Pipeline**
    - **Input:** Commits and pull requests targeting the main branch
    - **Processes:**
        - Utilises a GitHub Actions workflow configured with Gradle to automate the compilation of the project that uses dependency caching to optimise execution time
        - The build task ensures that all modules and their dependencies are compiled correctly and checked for errors
    - **Outputs:**
        - Test results, including failures logged in the GitHub Actions interface
        - Creates a dependency graph to monitor for outdated dependencies
    - **Trigger:** Automatically triggered by commits and pull requests targeting the main branch
    - **Appropriateness:** Ensures every change can and will integrate seamlessly, reducing likelihood of bugs in the main branch
2) **Artifact Management and Binary Distribution**
    - **Input:** Successful build outputs
    - **Processes:**
        - Packages the code into distributable JAR file
        - Uploads the JAR to a designated separate releases page
    - **Outputs:** A ready-to-download binary accessible to developers
    - **Trigger:** Runs upon command after a successful build
    - **Appropriateness:** Simplifies distribution and allows easy access to latest working versions
3) **Release Management**
    - **Input:** Tagged commits (e.g using dates, ‘mark’, ‘patch’)
    - **Processes:**
        - Runs a build and testing workflow to ensure the release is stable
        - Creates a new version in the repository with the binary attached
    - **Outputs:** A release version available on the repository
    - **Trigger:** Tags on releases
    - **Appropriateness:** Guarantees stable releases and provides a clear version history