CI Pipeline

Weiliang Zhao

January 28, 2023

1 Description of a CI pipeline

CI (Continuous Integration) pipeline is a set of automated processes that are designed to build, test, and deploy software code changes. It allows developers to integrate code changes frequently and catch errors early on in the development process. The pipeline typically includes the following stages:

1. Code Commit:
   The first step in a CI pipeline is to commit the code changes to a version control system (VCS), such as Git. This allows developers to collaborate on the code and provides a history of changes that can be used for debugging and rollbacks.

2. Build:
   Once code changes have been committed, the pipeline will automatically build the software. This can include tasks such as compiling the code, running linting and formatting checks and generating documentation.

3. Test:
   After the build step, the pipeline will automatically run a set of unit and integration tests to ensure that the code changes have not introduced any regressions.

4. Deployment:
   If the build and test steps are successful, the pipeline will deploy the software to a staging environment where it can be further tested by QA teams or beta users.

5. Monitoring:
   The final step in the pipeline is to monitor the deployed software for performance and stability, and to collect feedback and metrics that can be used to improve the software.

2 Description of Automate Aspects

1. **Unit Testing**
   Automating unit tests using a framework such as unittest or pytest can help ensure that individual components of the code are functioning correctly. These tests can be run on a regular basis to catch any issues early on in the development process.

2. **Integration testing**
   Automating for my software would involve using a testing framework or tool to test the integration of different units of code, such as modules or components, within the software. This step would be performed after conducting unit tests, in order to ensure that the various components of the software are working together as intended. Automating this process would help to improve the efficiency and reliability of the testing process, and would allow for more thorough and regular testing to be conducted.

3. **Performance Testing**
   Performance testing can be automated by using tools such as Apache JMeter or Gatling. These tools can simulate different loads and conditions to test the software's behaviour, including load testing, stress testing, and scalability testing. This automation can ensure that any changes

made to the software do not negatively impact its performance, and can be run regularly as part of the CI pipeline.

4. **Functional test**
   Functional tests can be automated using a tool like Selenium or Appium. This automation can be integrated into the pipeline by setting up a script that triggers the tests to run after the code is built. This script could also include the ability to pass in any necessary environment variables and test data. By automating the functional tests in this manner, it ensures that the software is thoroughly tested from the user's perspective and any issues can be identified and addressed early on in the development process. This can help to improve the overall reliability and quality of the software and give confidence to the stakeholders that the software is working as expected.

3 Issues it would identify

CI pipeline should be set up to automatically identify and report any issues that arise during the development process so that they can be addressed quickly and efficiently.

1. **Errors and Bugs**
   The CI pipeline should be able to detect any errors and bugs such as syntax errors, logical errors and compilation errors in the code during the compilation process and alert the development team or developer.

2. **Performance issues**
   The pipeline should also check for performance issues, such as long execution times or high resource usage. For example, if the simulation takes too long to run on a specific hardware configuration, the pipeline should flag this as an issue.

3. **Incorrect input handling**
   The CI pipeline should also check that the software is handling user input correctly. For example, if the software is supposed to accept a specific file format for input, the pipeline should check that the software is able to read and process that file format correctly.

4. **Compatability issues and Code Quality**: The pipeline should also check for compatibility issues, such as software not working on different operating systems or with different versions of dependencies. And check for code quality issues, such as code that does not conform to coding standards or has poor organization.

5. **Security vulnerabilities**
   The pipeline should also check for any known security vulnerabilities in the software or its dependencies.