SWE 264P Project Report 4

Continuous Integration

Continuous Integration (CI) is a pivotal practice in modern software development that focuses on team members' frequent integration of work. Typically, each team member integrates their work multiple times throughout the day. Automated builds and tests bolster this practice to verify each integration, which catches conflicts and errors swiftly and improves the software's overall quality.

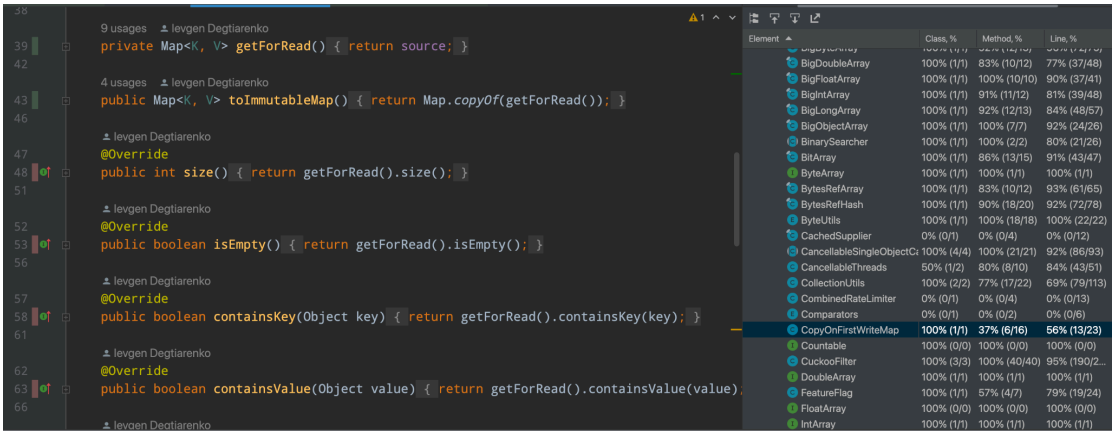
The objectives of CI are manifold, but key among them are the early detection of errors, which allows for more manageable and less costly fixes, and the improvement of software quality through continuous feedback. It reduces integration issues that typically arise when merging efforts close to release dates, thus facilitating smoother and quicker release cycles. CI promotes transparency in the development process, allowing teams to monitor progress and tackle challenges proactively. Automation plays a critical role in CI, eliminating the need for manual testing and building and increasing efficiency. Moreover, CI fosters collaboration, encouraging team members to collaborate and share the responsibility of maintaining code quality and readiness.

CI in ElasticSearch

The ElasticSearch project employs various CI pipelines contained within the .ci and .github directories. While Jenkins is used within the .ci folder to manage different pipelines, the specifics of these pipelines on the Jenkins server are not accessible, possibly due to permissions constraints.

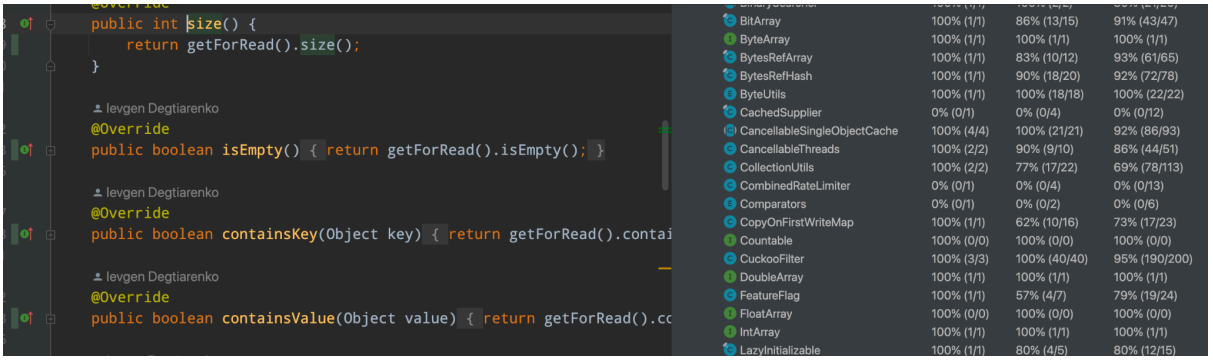
In contrast, within the .github directory, ElasticSearch utilizes GitHub Actions to manage distinct CI pipelines. Initially, there were two workflows, configured by **gradle-wrapper-validation.yml** and **docs-preview-links.yml**. The former is used to validate the integrity of the project's Gradle Wrapper, ensuring it hasn't been tampered with, while the latter automates the generation of preview links for documentation changes submitted via pull requests.

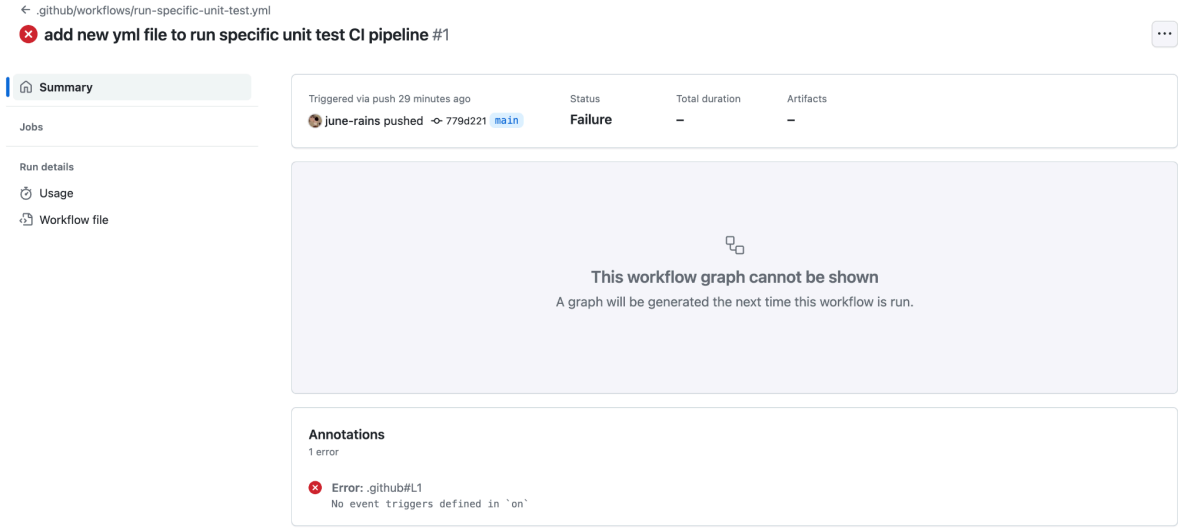
However, these workflows didn't automatically execute unit tests, which led to the creation of run-specific-unit-test.yml. This new configuration file is geared towards running tests for the CopyOnFirstWriteMapTests class. When the coverage of this test class was initially examined, it was at a certain level, which was expected to increase after adding more unit tests.



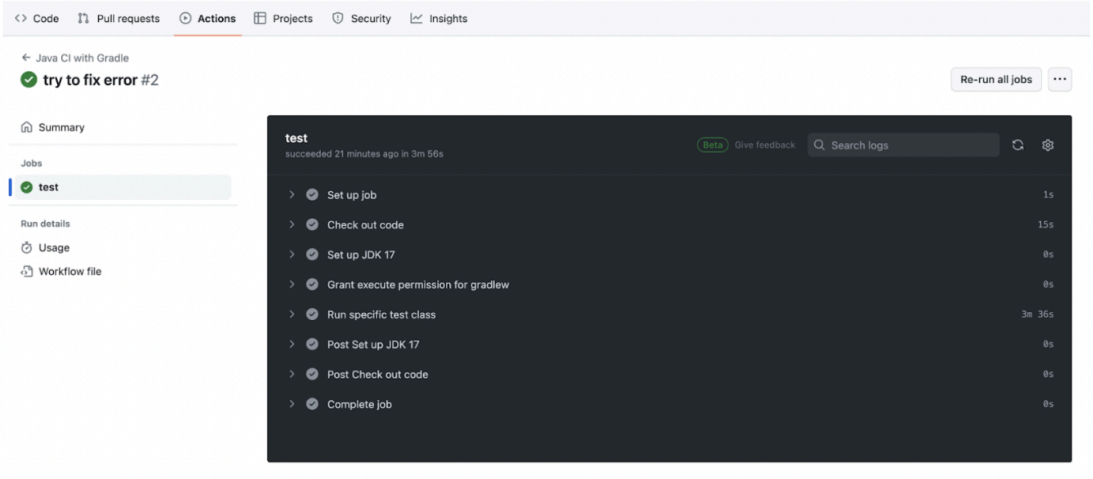
Implementation and Testing of the CI Pipeline

To implement and evaluate the new CI pipeline, new unit tests were added to CopyOnFirstWriteMapTests, located at server/src/test/java/org/elasticsearch/common/util/CopyOnFirstWriteMapTests.java. Following this, the changes were pushed to the main branch, triggering the pipeline. The initial push failed, indicating an error due to missing event triggers defined in the keyword of the YAML file.





After correcting the YAML file and pushing the changes again, the CI pipeline was successfully executed, as evidenced by a screenshot of the GitHub Actions interface. The workflow was configured to check out the code, set up the Java environment, make the Gradle wrapper executable, and run the specified test class upon any push or pull request to the branches.



This CI process exemplifies the seamless integration of code changes and testing, essential for maintaining a high standard of code quality and ensuring the reliability of software releases.