**Slide 1: Title Slide**

**Script:** “Hello, everyone. Welcome to our presentation on Continuous Integration (CI) for the Software Engineering and DevOps coursework. In this video demonstration, we will be walking you through the CI pipeline for our Python project, Dec2Hex.py. We’ll cover the steps of the pipeline, from code pushing to GitHub, all the way through Jenkins' automated build process, test execution, and static code analysis with SonarQube. Let’s get started.”

**Slide 2: Overview of the Demonstration**

**Script:** “Before we dive into the details, let me give you a quick overview of the steps involved in our CI pipeline. First, we’ll push our code changes to GitHub, which will automatically trigger Jenkins to start the build process. Jenkins will then pull the latest code, run our tests using pytest, and trigger static code analysis with SonarQube. Finally, we’ll review the results from Jenkins and SonarQube to confirm the build’s success. Each of these steps is an essential part of our CI workflow, helping us automate and ensure the quality of our code.”

**Slide 3: Step 1 – Pushing Changes to GitHub**

**Script:** “Let’s start with Step 1: pushing changes to GitHub. In this part of the pipeline, we make a small change to the code, for example, modifying a print statement in our Dec2Hex.py file. After committing this change, we push it to the GitHub repository. This action triggers our CI pipeline, as Jenkins is configured to monitor the repository for any code updates. You can see the file in our GitHub repository here. Once the push is complete, Jenkins will detect the change and start the process automatically.”

**Slide 4: Step 2 – Jenkins Detects the Push**

**Script:** “Now let’s move on to Step 2: Jenkins detecting the push. After we push our code to GitHub, Jenkins automatically picks up this change. In our setup, Jenkins is configured to check for changes in the repository every 5 minutes, so the push is detected almost immediately. You can see here in the Jenkins interface that a new build has been triggered. Jenkins starts by pulling the latest commit from GitHub and prepares to run the build.”

**Slide 5: Step 3 – Jenkins Build Execution**

**Script:** “In Step 3, Jenkins runs the build and executes the tests. After pulling the latest code, Jenkins starts the build process. In our case, since we’re working with Python, the build step essentially runs the Dec2Hex.py script with a test input, like the number 15. Here, you can see the Jenkins console output. As we can observe, Jenkins runs the tests we’ve written using pytest. In this case, all tests pass, confirming that the code works correctly. If any tests failed, Jenkins would mark the build as failed, but that didn’t happen in this run.”

**Slide 6: Step 4 – SonarQube Static Code Analysis**

**Script:** “Next, in Step 4, we run SonarQube static code analysis. After Jenkins finishes running the tests, it triggers SonarQube to analyze our code. SonarQube checks the code for issues like bugs, vulnerabilities, and code smells. It’s an essential tool in ensuring code quality. In our case, SonarQube runs smoothly, and you can see in this screenshot that the analysis is completed successfully. It checks all the rules defined for the project and shows a detailed report in the SonarQube dashboard. If any issues were found, Jenkins would have flagged the build as failed.”

**Slide 7: Step 5 – Final Jenkins Output**

**Script:** “In Step 5, we finally reach the end of the Jenkins pipeline, where the build completes successfully. After running the tests and performing the SonarQube analysis, Jenkins concludes the process. As you can see in the Jenkins interface, the build status is marked as **SUCCESS**. This means that everything has passed — the code runs correctly, the tests have all passed, and the static analysis with SonarQube shows no critical issues. Jenkins provides a detailed log for each step in the pipeline, so we know exactly what’s happening at each stage of the process.”

**Slide 8: Step 6 – Final SonarQube Report**

**Script:** “Finally, in Step 6, we review the results in SonarQube. As part of our CI pipeline, SonarQube scans the code for potential issues. Once the static analysis is complete, SonarQube provides a summary of the code’s health. In this case, the report is clean. We have no bugs, vulnerabilities, or code smells, and the **Quality Gate** status is green, which indicates that our code meets the quality standards we’ve set. This is a great sign — it confirms that the code we pushed and tested is not only functional but also clean and maintainable.”

**Slide 9: Conclusion**

**Script:** “Let’s wrap up with a brief conclusion. In this video, we’ve demonstrated a fully automated Continuous Integration pipeline that starts with a code push to GitHub and runs through Jenkins for testing and SonarQube for static analysis. This process ensures that every code change is tested, and any issues are flagged early, making the development cycle faster and more efficient. Continuous Integration is a vital part of modern DevOps, as it allows us to maintain high code quality and catch errors early in the process. Thank you for watching.”

**Slide 10: Q&A / Thank You**

**Script:** “Thank you for your time. We hope you found the demonstration helpful in understanding how we set up our CI pipeline. If you have any questions, feel free to ask. We’re happy to clarify any part of the process. Otherwise, thanks again for watching, and we appreciate your attention.”