Name: Harsh Dalvi
Roll No: 13

Subject: Advance DevOps , Sem: SEM V
Class / Batch: TE-IT / Batch B

Experiment No. 7

Aim: To describe Static Analysis SAST process and learn to integrate Jenkins SAST to SonarQube/GitLab. **(LO1, LO4)**

Theory:

SAST (Static Analysis Security Testing):

Static application security testing (SAST), or static analysis, is a testing methodology that analyses source code to find security vulnerabilities that make your organization's applications susceptible to attack. It scans an application before the code is compiled. It's also known as white box testing.

Why SAST is an important Security Activity?

Developers dramatically outnumber security staff. It can be challenging for an organization to find the resources to perform code reviews on even a fraction of its applications. A key strength of SAST tools is the ability to analyze 100% of the code base. Additionally, they are much faster than manual secure code reviews performed by humans. These tools can scan millions of lines of code in a matter of minutes. SAST tools automatically identify critical vulnerabilities—such as buffer overflows, SQL injection, cross-site scripting, and others—with high confidence. Thus, integrating static analysis into the SDLC can yield dramatic results in the overall quality of the code developed.

SonarQube:

- SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications.
- It is used to test the quality of the code and execute the automatic reviews with the help of identifying the bugs, code analysis and security exposures on various programming languages such as Java, C#, JavaScript, PHP, Ruby, Cobol, C/C++ and so on of the web applications.
- It supports 25+ major programming languages through built-in rulesets and can also be extended with various plugins.
- SonarQube tool is written on the JAVA programming language.
- It will generate the reports of the code coverage, complexity of code, repeated code, security weakness, and bugs.
- offers complete analysis with multiple tools like Ant, Maven, Gradle, Jenkins, and so on.

Benefits of SonarQube:

1. **Sustainability:** Reduces complexity, possible vulnerabilities, and code duplications, optimizing the life of applications.
2. **Increase productivity:** Reduces the scale, cost of maintenance, and risk of the application; as such, it removes the need to spend more time changing the code.
3. **Quality code:** Code quality control is an inseparable part of the process of software development.
4. **Detect Errors:** Detects errors in the code and alerts developers to fix them automatically before submitting them for output.
5. **Increase consistency:** Determines where the code criteria are breached and enhances the quality.
6. **Business scaling:** No restriction on the number of projects to be evaluated.
7. **Enhance developer skills:** Regular feedback on quality problems helps developers to improve their coding skills.

Why to use SonarQube?

- Developers working with hard deadlines to deliver the required functionality to the customer. It is so important for developers that many times they compromise with the code quality, potential bugs, code duplications, and bad distribution of complexity.
- Additionally, they tend to leave unused variables, methods, etc. In this scenario, the code would work in the desired way.
- To avoid these issues in code, developers should always follow the good coding practice, but sometimes it is not possible to follow the rules and maintain the good quality as there may be many reasons.
- In order to achieve continuous code integration and deployment, developers need a tool that not only works once to check and tell them the problems in the code but also to track and control the code to check continuous code quality. To satisfy all these requirements, here comes SonarQube in the picture.

Features of SonarQube:

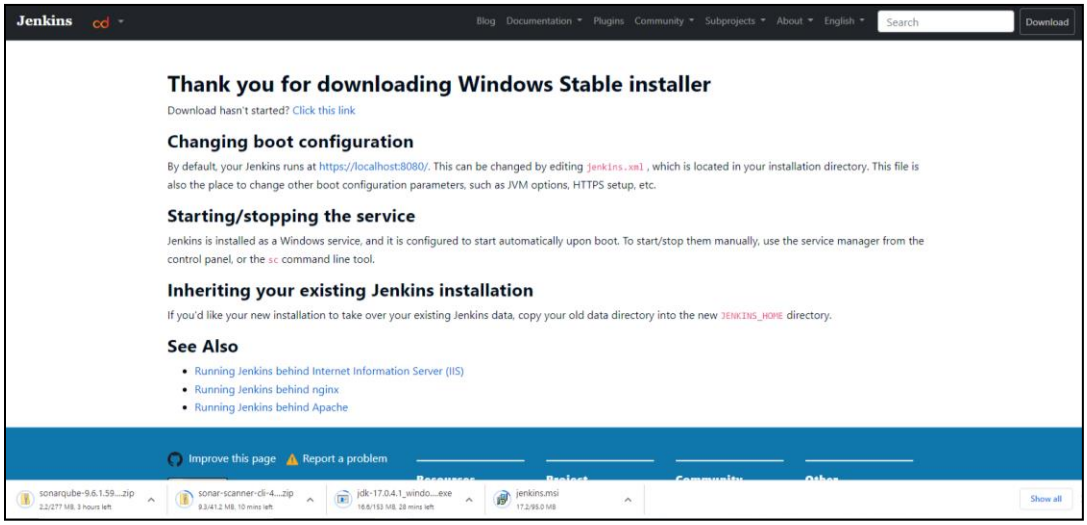
- It will integrate with multiple development environments like Visual Studio, Eclipse, and IntelliJ IDEA over the SonarLint plug-ins.
- It also supports some external tools such as GitHub, LDAP, and Active Directory.
- It can record the metric history and deliver the evolution graphs.
- It will help us to identify the complex issues.
- It will provide application security.

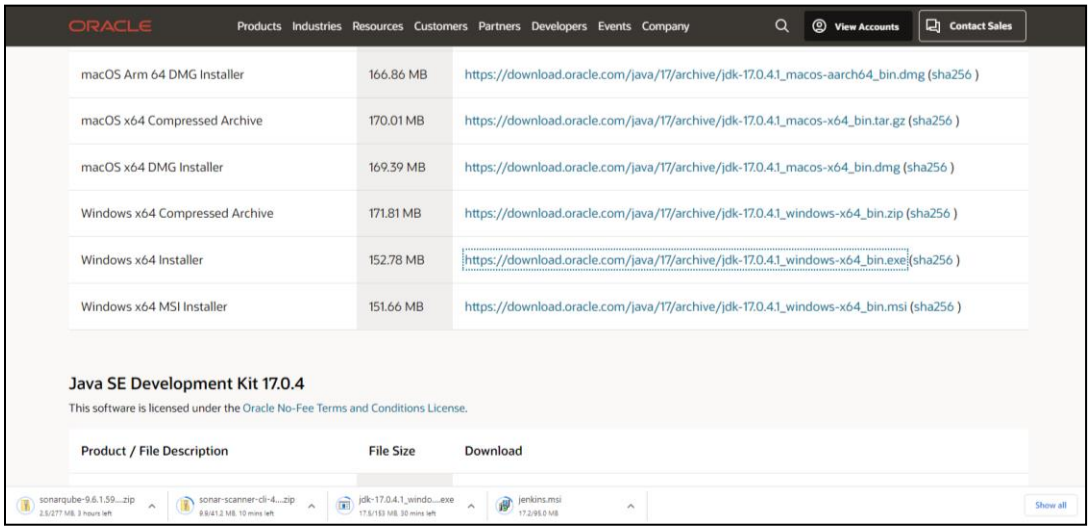
Name: Harsh Dalvi
Roll No: 13

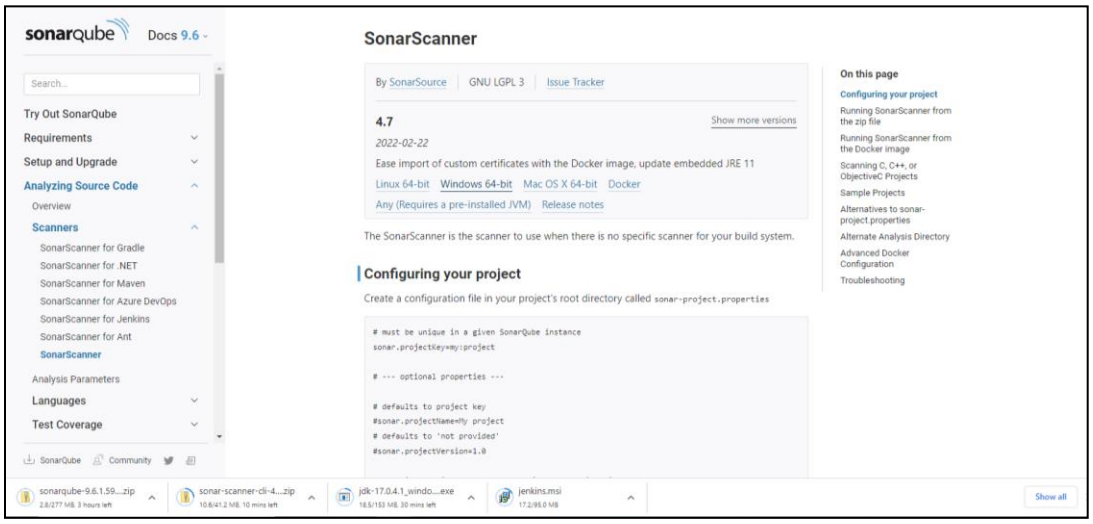
Subject: Advance DevOps , Sem: SEM V
Class / Batch: TE-IT / Batch B

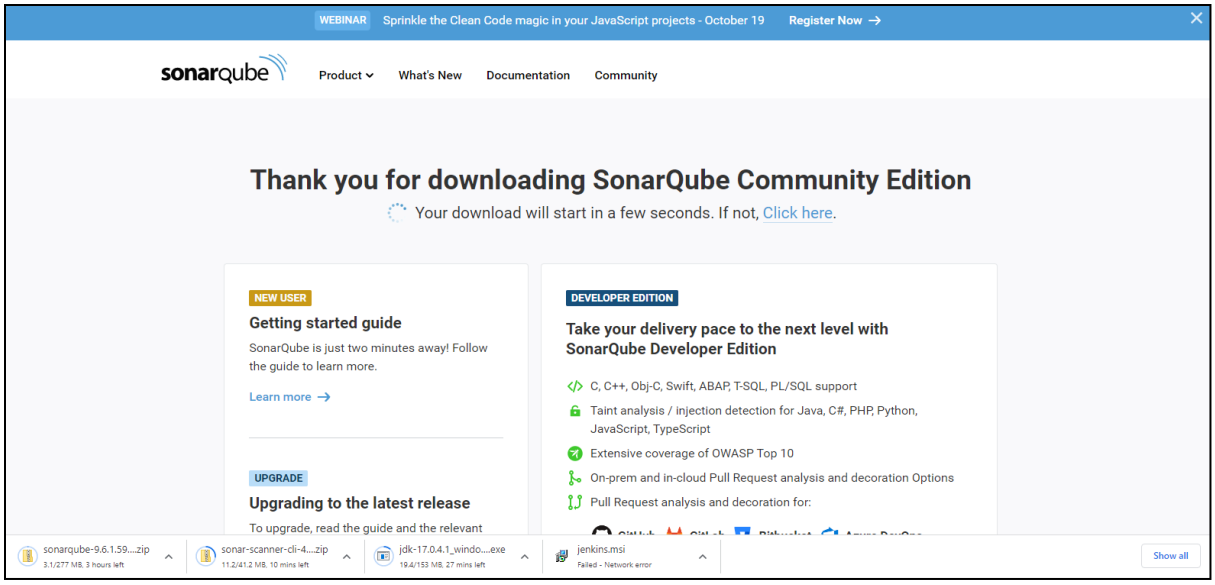
Steps to perform the Experiment:

Step 1: Download Jenkins, JDK 17, SonarScanner and SonarQube from their respective websites.

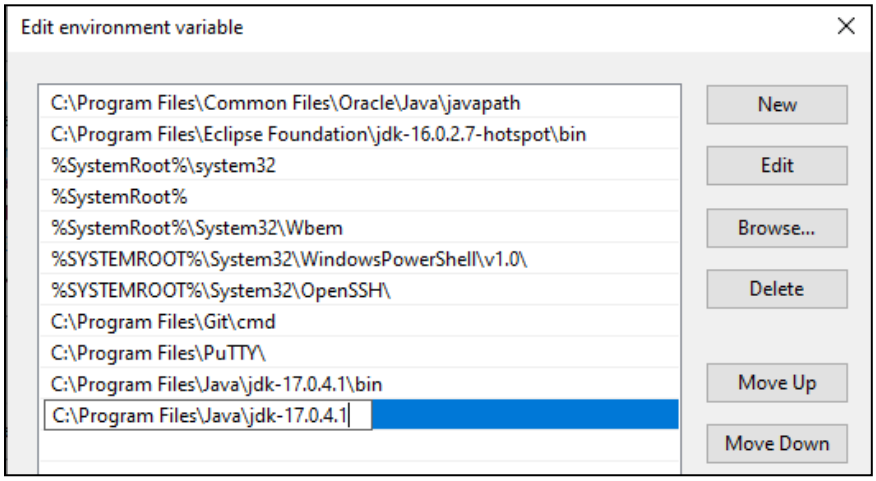


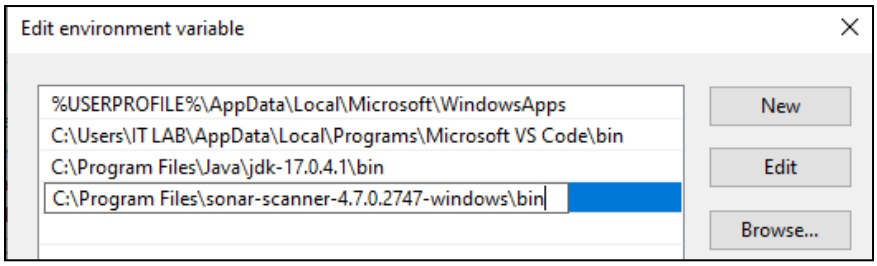


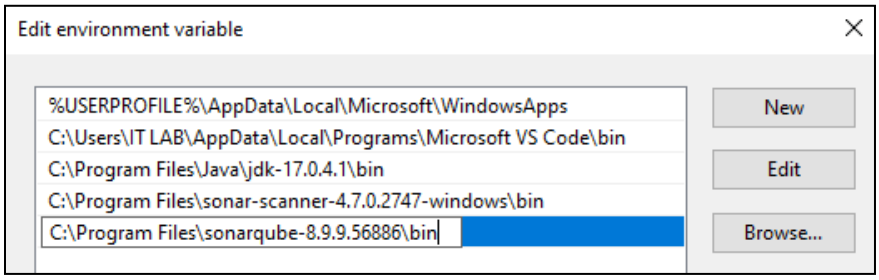




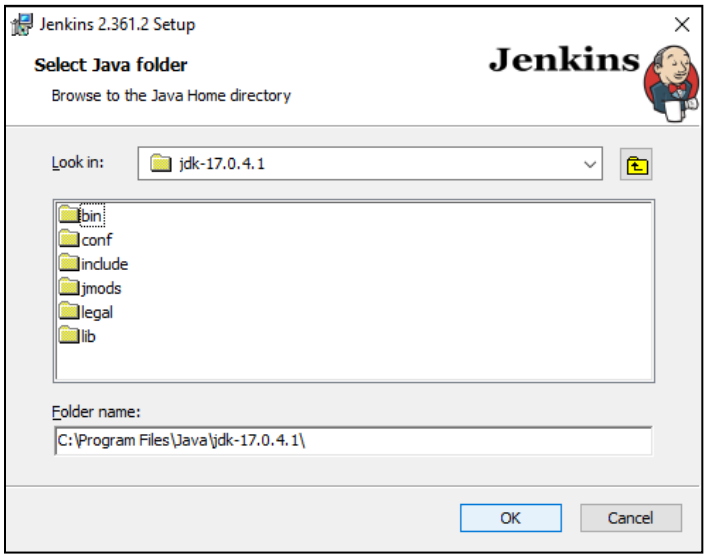
Step 2: Now give the path to JDK, SonarScanner and SonarQube in your Environment Variables.



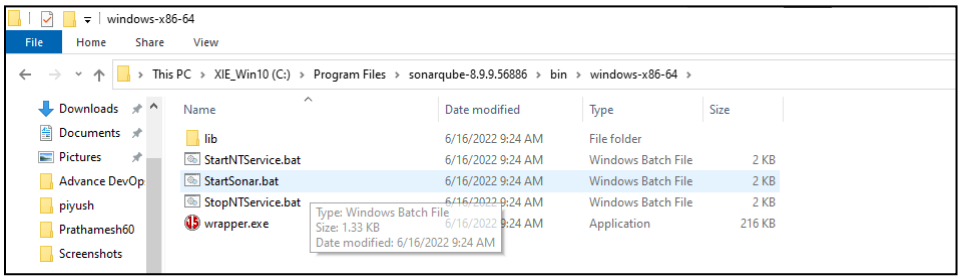


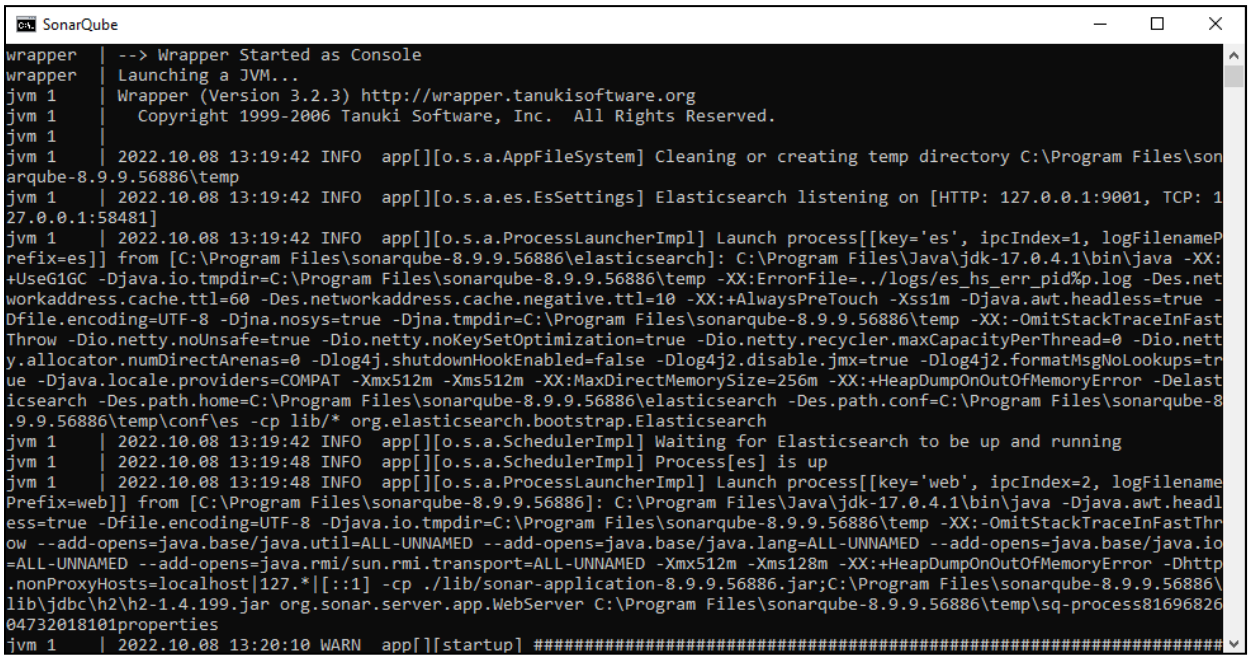


Step 3: Install Jenkins and select the JDK folder. During the installation download all the recommended plugins.

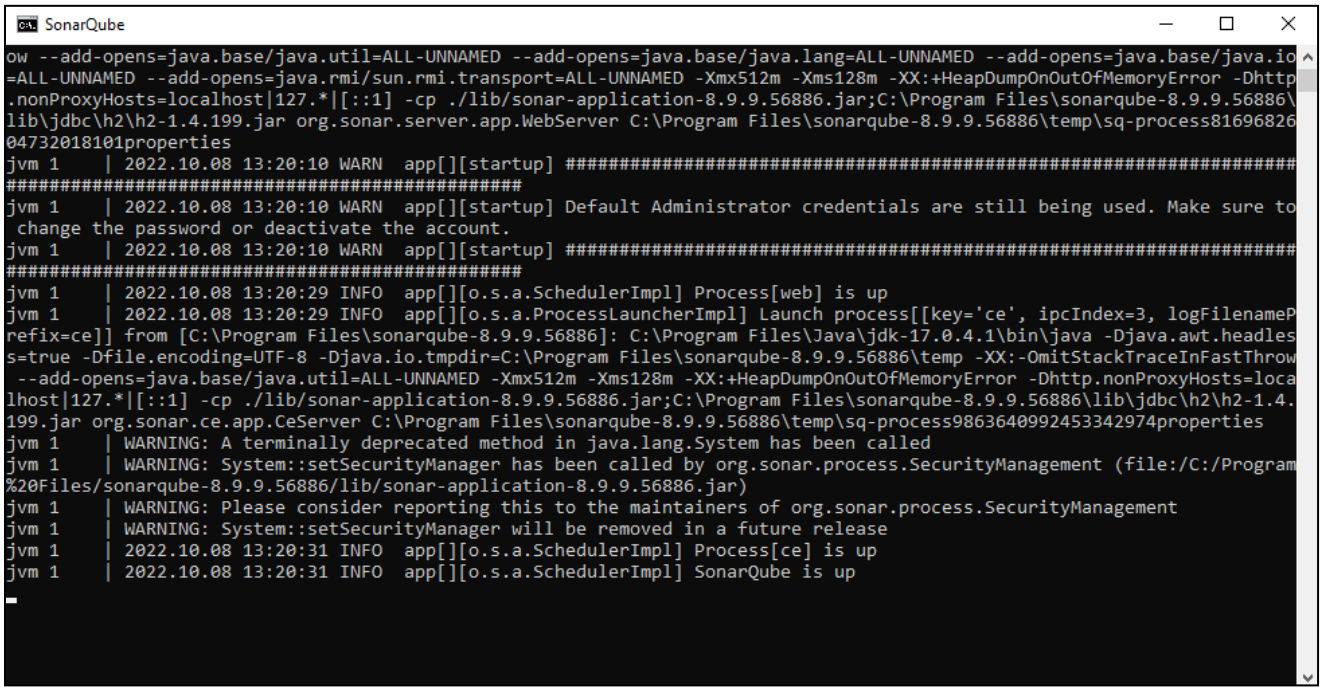


Step 4: After the installation of Jenkins, go to the SonarQube folder and open the "StartSonar.bat" file by following the path given below.

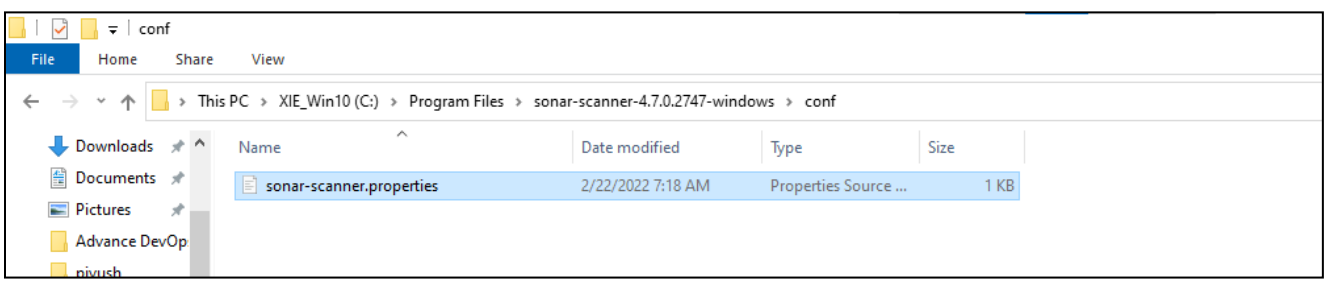




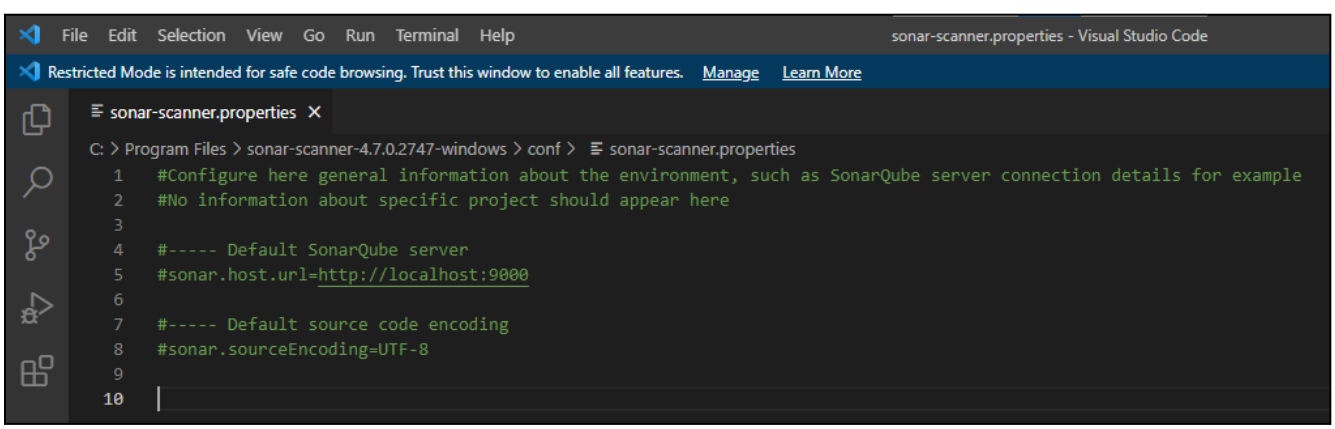
Step 5: The required output is that the terminal should show “Process is up” and “SonarQube is up” to get the desired output as shown below.



Step 6: Now, go to the SonarScanner folder and open the “sonar-scanner.properties” file by following the path given below.



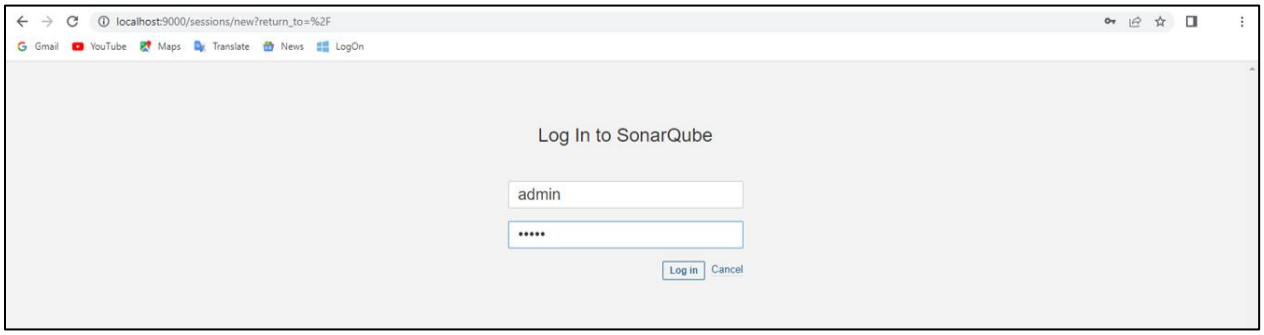
Step 7: Copy the URL and paste it on the Browsers Search Bar



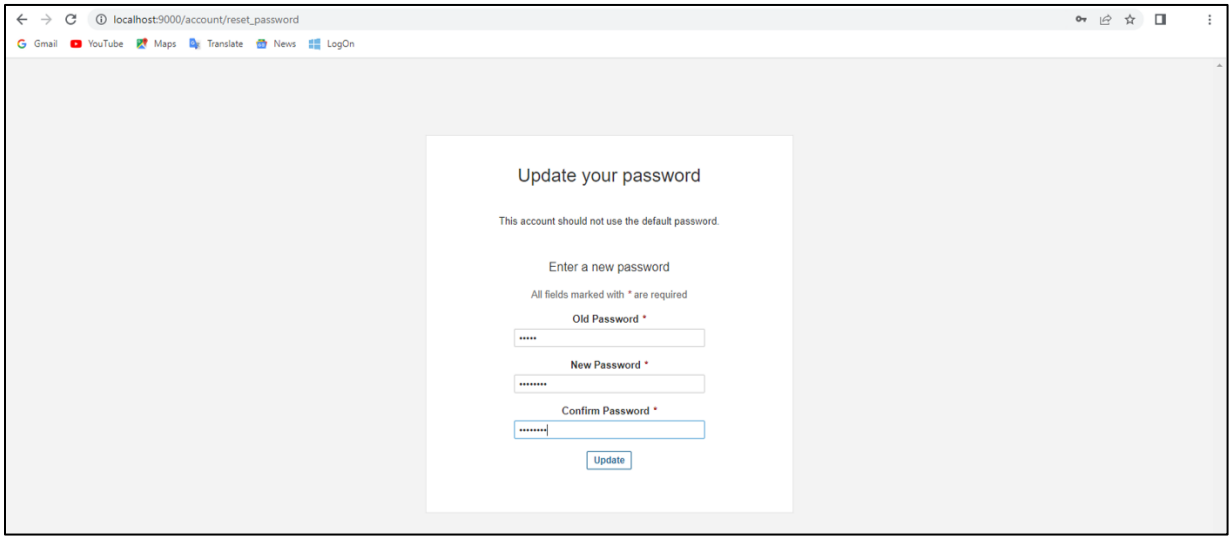
Name: Harsh Dalvi
Roll No: 13

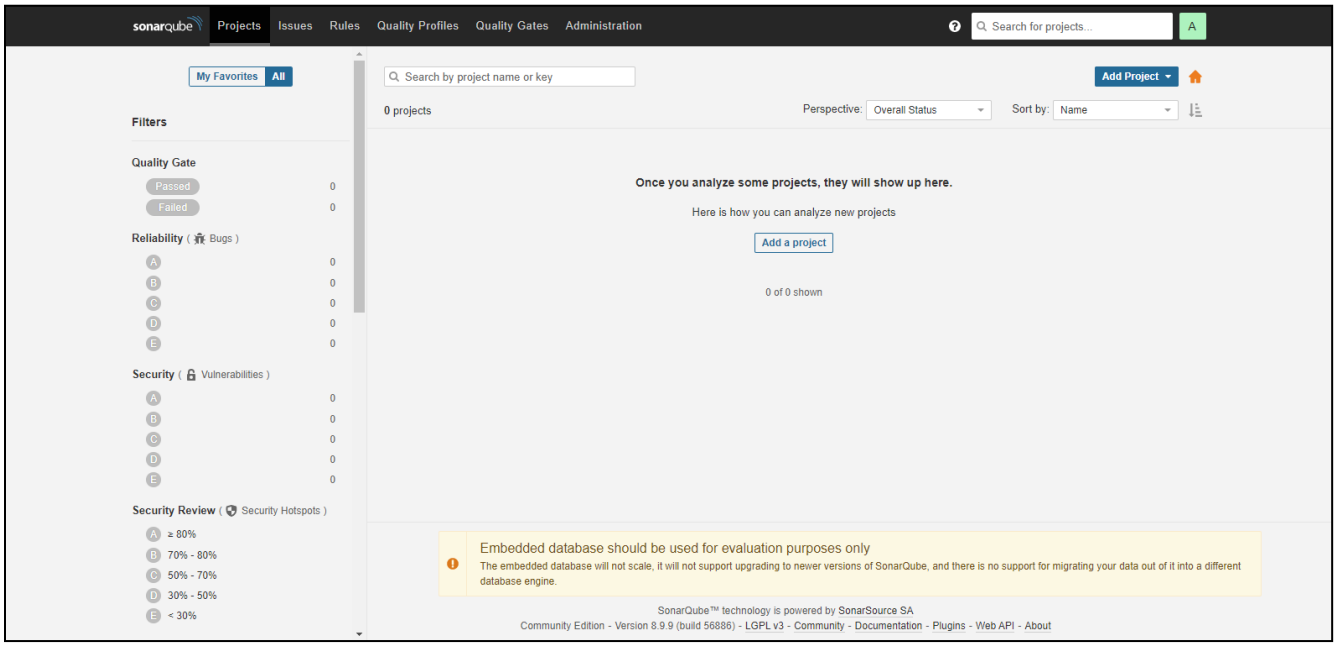
Subject: Advance DevOps , Sem: SEM V
Class / Batch: TE-IT / Batch B

Step 8: Sign in as “admin”

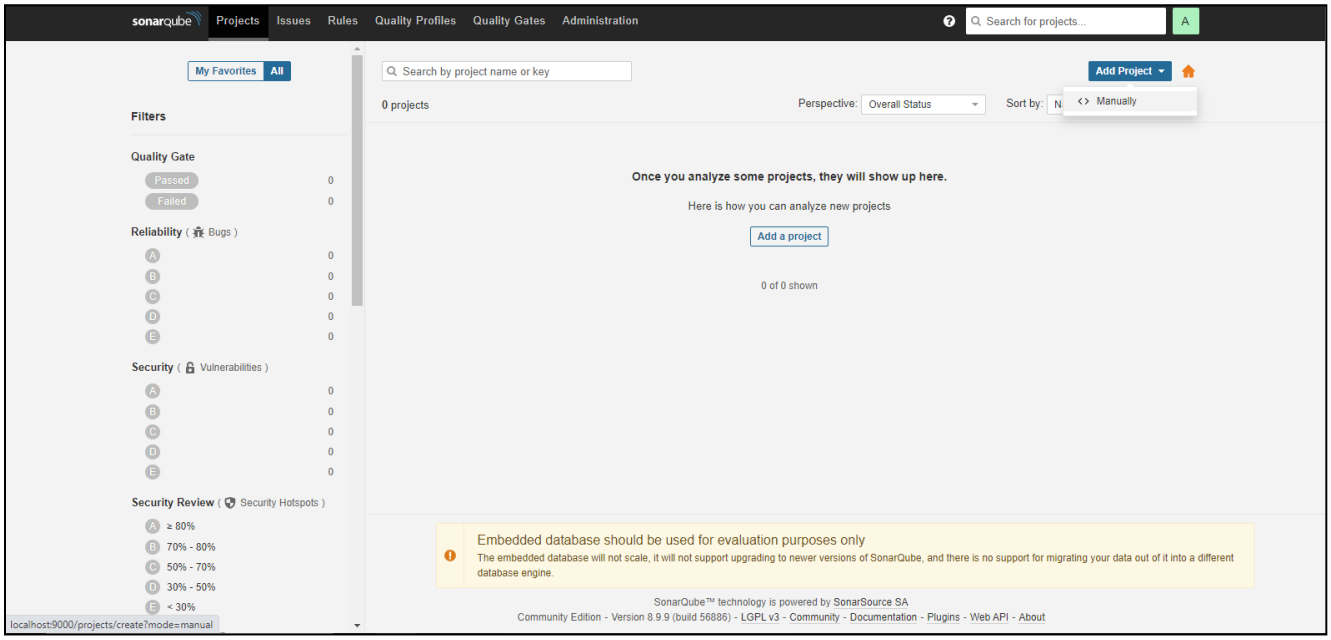


Step 9: Change the Password

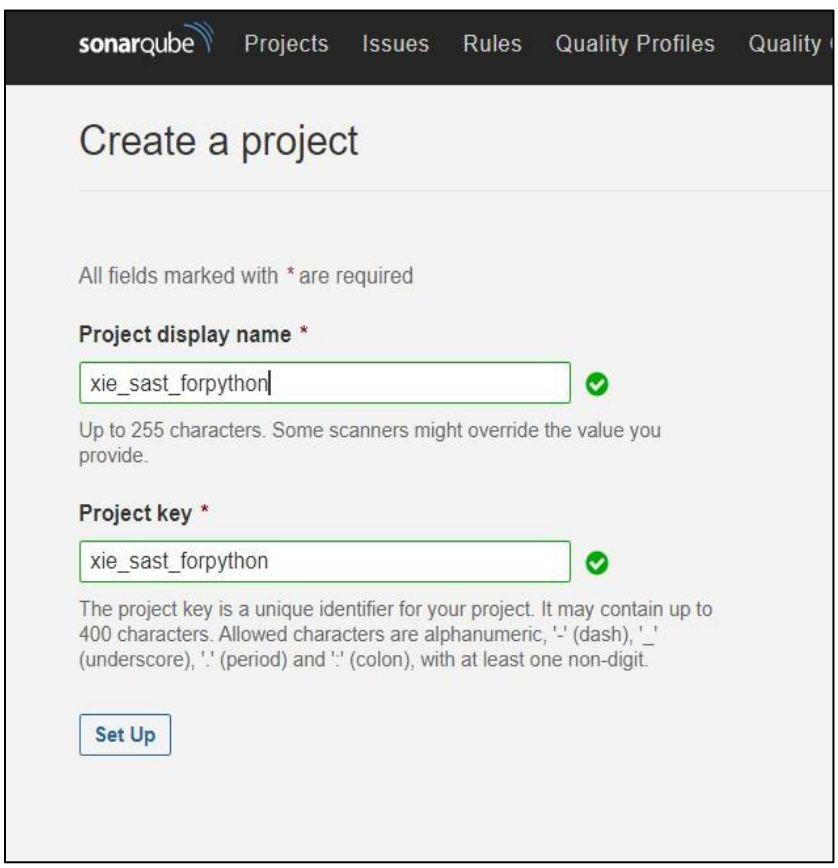




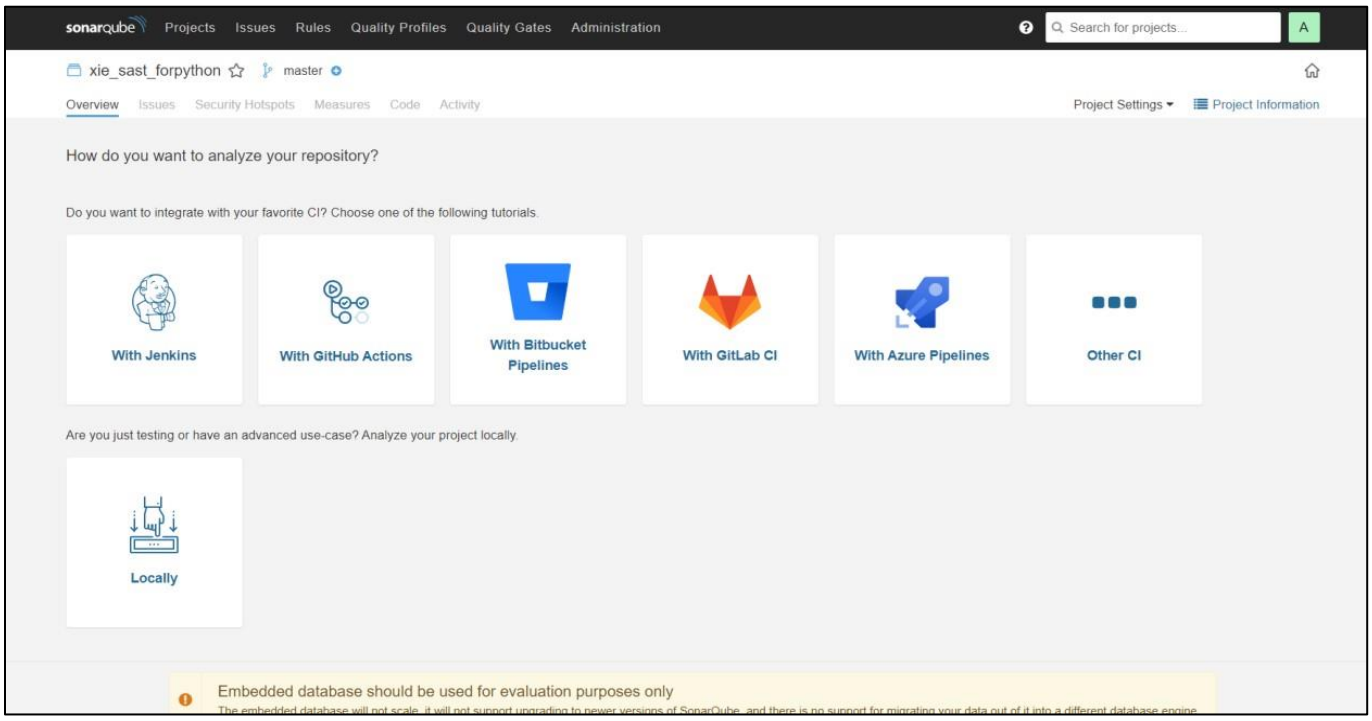
Step 10: Click on “Add Project” and select “Manually”.



Step 11: Now create a Project by giving a Project Key and Display Name.



Step 12: Therefore, the Project is created.



Conclusion: From this experiment it is concluded that, we have learnt the concepts and analyzed the Static Analysis SAST and learned to integrate Jenkins SAST to SonarQube. The ability of SAST tools to catch security problems early in the development process means that even in deadline-driven environments, developers don't need to constantly worry about following best practices. Hence, we have successfully achieved the Lab Outcome One and Four (LO1 and LO4). Also, we have achieved PO1, PO2, PO3, PO4, PO5, PO9, PO10 and PO12 from this experiment.