

## **Experiment No. 8**

**Aim:** To Construct a Jenkins CICD Pipeline with SonarQube/GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web/Java/Python application. **(LO1, LO4)**

### **Theory:**

#### **What is Jenkins Pipeline?**

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment. Pipeline provides an extensible set of tools for modelling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

### **Pipeline Concept:**

**The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax.**

- **Pipeline:** A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it. Also, a pipeline block is a key part of Declarative Pipeline syntax.
- **Node:** A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline. Also, a node block is a key part of Scripted Pipeline syntax.

- **Stage:** A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g., "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress.
- **Step:** A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, that typically means the plugin has implemented a new step.

### **What is Continuous Integration, Continuous Delivery, and Continuous Deployment?**

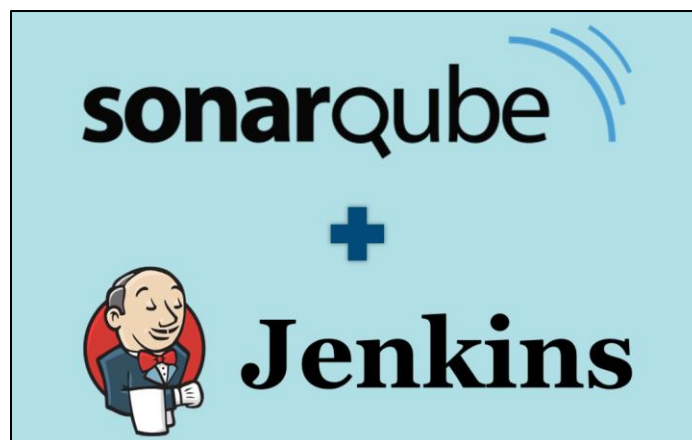
**Continuous integration** is a software development method where members of the team can integrate their work at least once a day. In this method, every integration is checked by an automated build to search the error.

**Continuous delivery** is a software engineering method in which a team develops software products in a short cycle. It ensures that software can be easily released at any time.

**Continuous deployment** is a software engineering process in which product functionalities are delivered using automatic deployment. It helps testers to validate whether the codebase changes are correct, and it is stable or not.

### **SonarQube Integration:**

- SonarQube Integration is an open-source static code analysis tool that is gaining tremendous popularity among software developers. It enables software professionals to measure code quality, identify non-compliant code, and fix code quality issues. The SonarQube community is quite active and provides continuous upgrades, new plug-ins, and customization information on a regular basis. Further, it is a healthy practice to periodically run SonarQube on the source code to fix code quality violations and reduce the technical debt.
- SonarQube enables developers to track code quality, which helps them to ascertain if a project is ready to be deployed in production. Further, it allows developers to continuously inspect the code, perform automatic reviews and run analysis to find code quality issues.



- Furthermore, SonarQube provides a lot of other features, including the ability to record metrics, evolution graphs etc. It has inherent options to perform automated analysis and continuous integration utilizing tools such as Jenkins, Hudson, etc. In this blog, we will explore the process of creating pipeline scripts for SonarQube integration. Here are the steps.

### **Static analysis:**

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program. The process provides an understanding of the code structure and can help ensure that the code adheres to industry standards. Static analysis is used in software engineering by software development and quality assurance teams. Automated tools can assist programmers and developers in carrying out static analysis. The software will scan all code in a project to check for vulnerabilities while validating the code.

Static analysis is generally good at finding coding issues such as:

- Programming errors
- Coding standard violations
- Undefined values
- Syntax violations
- Security vulnerabilities

The static analysis process is also useful for addressing weaknesses in source code that could lead to buffer overflows -- a common software vulnerability.

### **Static Analysis in Continuous Integration:**

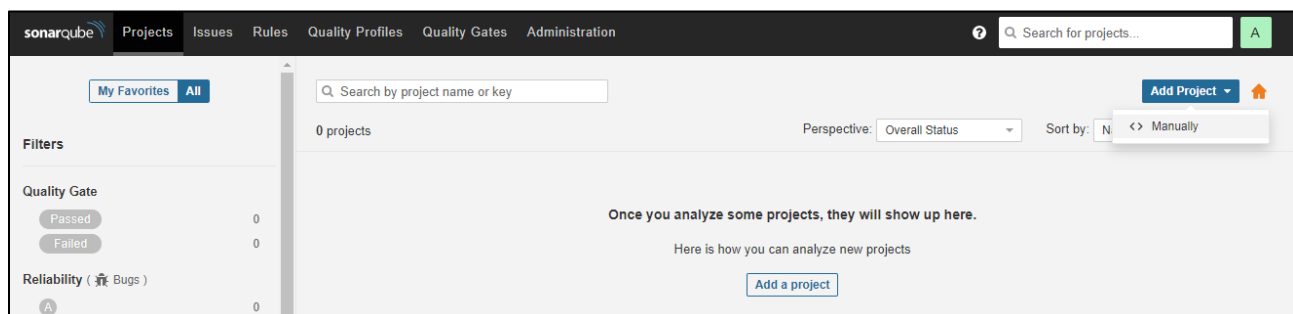
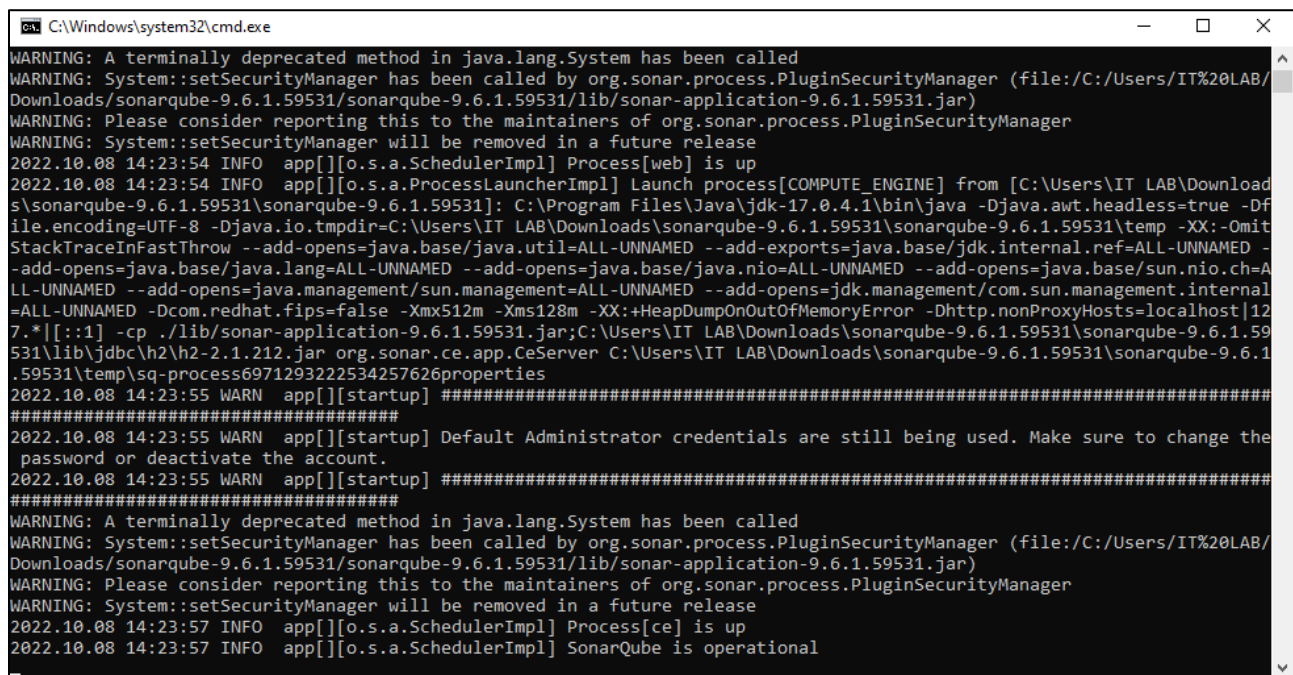
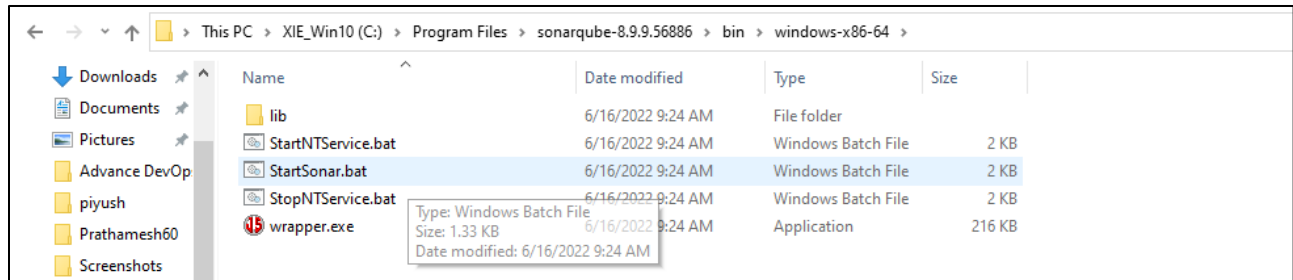
- Static Analysis is often performed during the Continuous Integration (CI) process to generate a report of compliance issues which can be reviewed to receive an objective view of the code-base over time.
- Some people use Static Analysis as an objective measure of their code quality by configuring the static analysis tool to only measure specific parts of the code, and only report on a subset of rules.
- The objectivity is provided by the rules used since these do not vary in their evaluation of code over time. Clearly, the combination of rules used and their configuration is a subjective decision and different teams choose to use different rules at different times.
- Having the Static Analysis performed in CI is useful but might delay the feedback to the programmer. Programmers don't receive feedback when coding, they receive feedback later when the code is run through the Static Analysis tool. Another side-effect of running the Static Analysis in CI is that the results are easier to ignore.

Name: Harsh Dalvi  
Roll No: 13

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

## Steps to perform the Experiment:

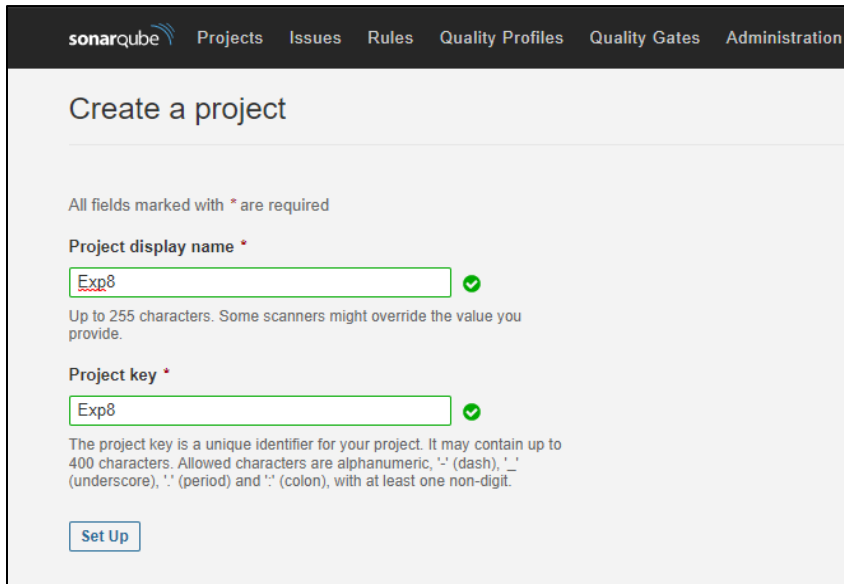
**Step 1:** Start the SonarQube and then create a new project manually.



Name: Harsh Dalvi  
Roll No: 13

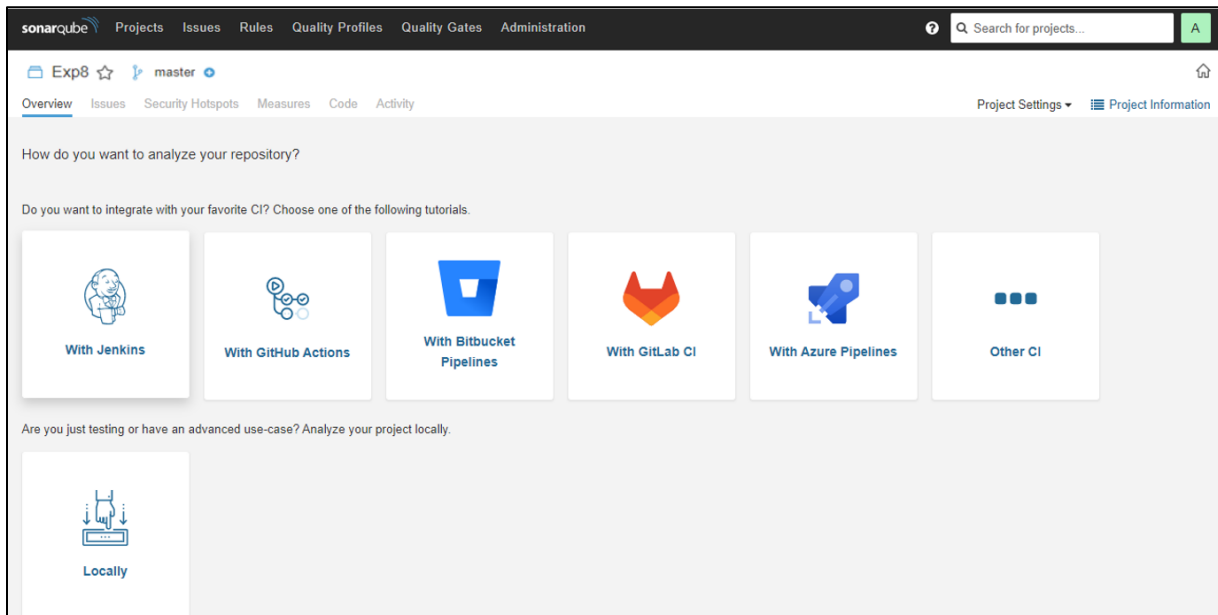
Subject: Advance DevOps , Sem: SEM V  
Class / Batch: TE-IT / Batch B

**Step 2:** Provide the project display name and its key name will be same then click on the setup.



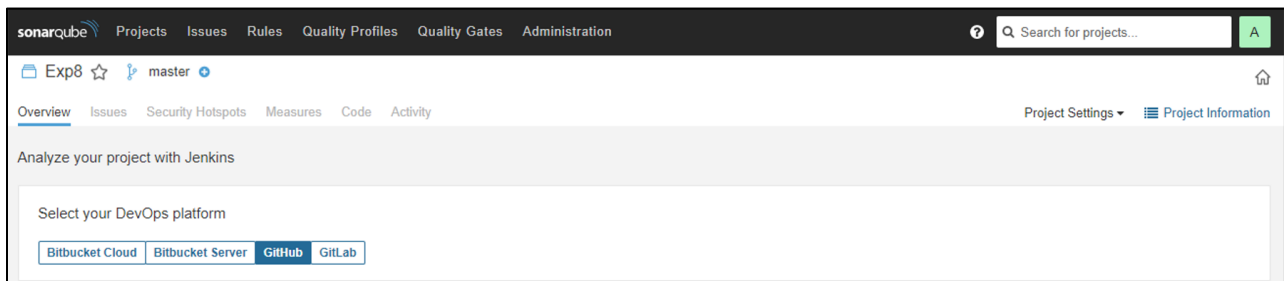
The screenshot shows the 'Create a project' page in SonarQube. The navigation bar at the top includes 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. The main heading is 'Create a project'. Below it, a note states 'All fields marked with \* are required'. There are two input fields: 'Project display name \*' and 'Project key \*', both containing the text 'Exp8'. Each field has a green checkmark icon to its right. Below the 'Project key' field, there is a detailed note: 'The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '\_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.' At the bottom of the form is a blue button labeled 'Set Up'.

**Step 3:** In the overview section Select “with Jenkins” option



The screenshot shows the 'Overview' page for a project named 'Exp8' in SonarQube. The navigation bar is the same as in Step 2. Below the navigation bar, there are tabs for 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity'. The 'Overview' tab is selected. The page asks 'How do you want to analyze your repository?' and 'Do you want to integrate with your favorite CI? Choose one of the following tutorials.' There are six options: 'With Jenkins', 'With GitHub Actions', 'With Bitbucket Pipelines', 'With GitLab CI', 'With Azure Pipelines', and 'Other CI'. The 'With Jenkins' option is highlighted with a blue border. Below these options, there is a section titled 'Are you just testing or have an advanced use-case? Analyze your project locally.' with a 'Locally' option.

**Step 4:** Select the DevOps platform as GitHub.

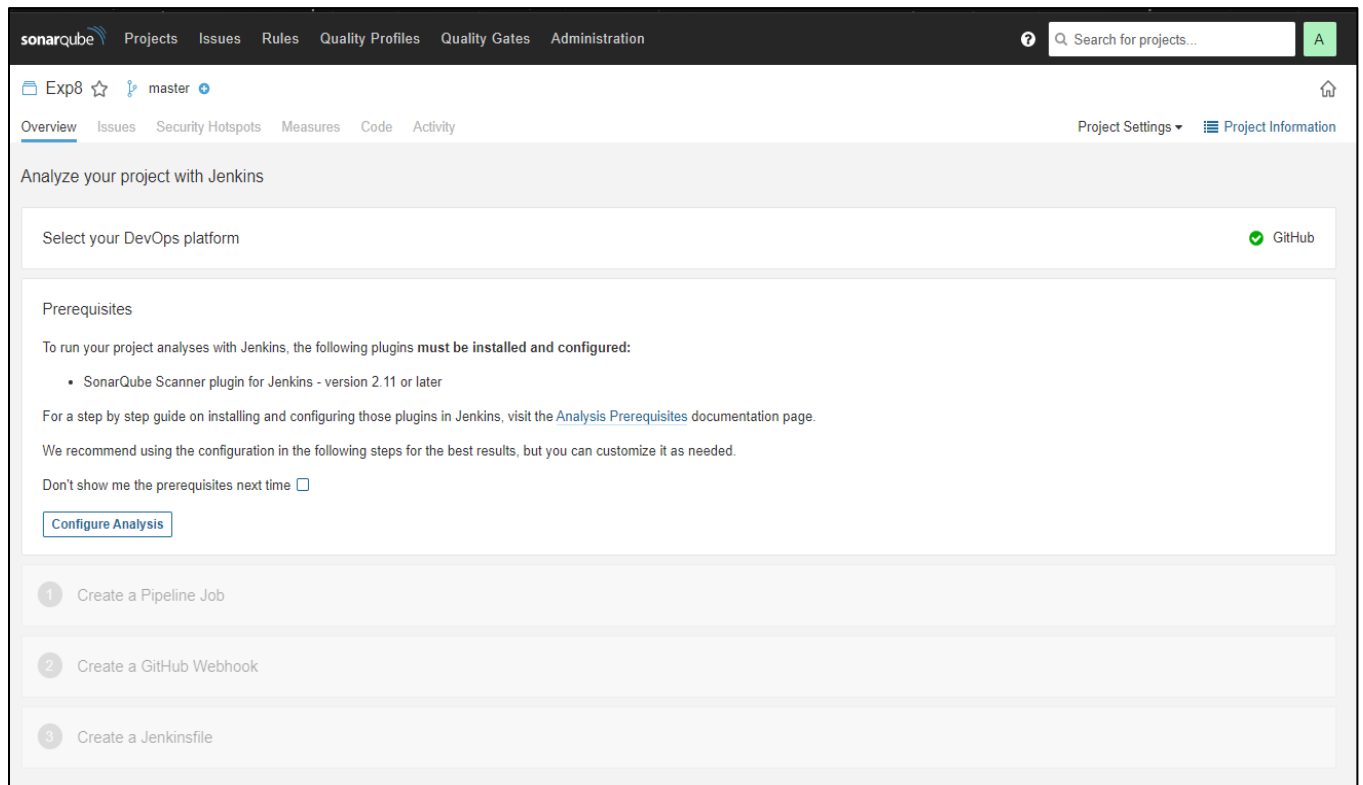


The screenshot shows the 'Analyze your project with Jenkins' page in SonarQube. The navigation bar is the same as in Step 2. Below the navigation bar, there are tabs for 'Overview', 'Issues', 'Security Hotspots', 'Measures', 'Code', and 'Activity'. The 'Overview' tab is selected. The page asks 'Analyze your project with Jenkins' and 'Select your DevOps platform'. There are four options: 'Bitbucket Cloud', 'Bitbucket Server', 'GitHub', and 'GitLab'. The 'GitHub' option is highlighted with a blue border.

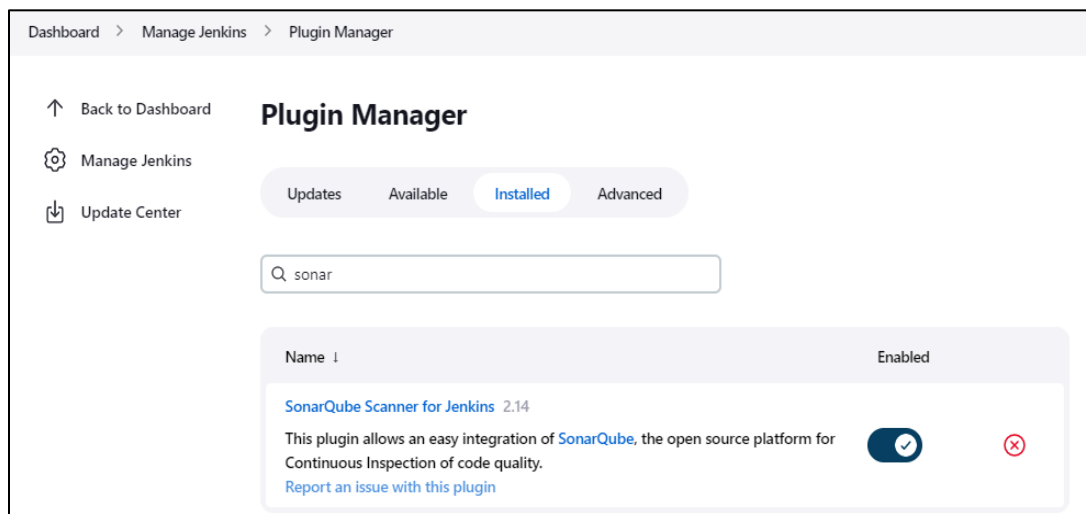
Name: Harsh Dalvi  
Roll No: 13

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

### Step 5: Click on Configure analysis.



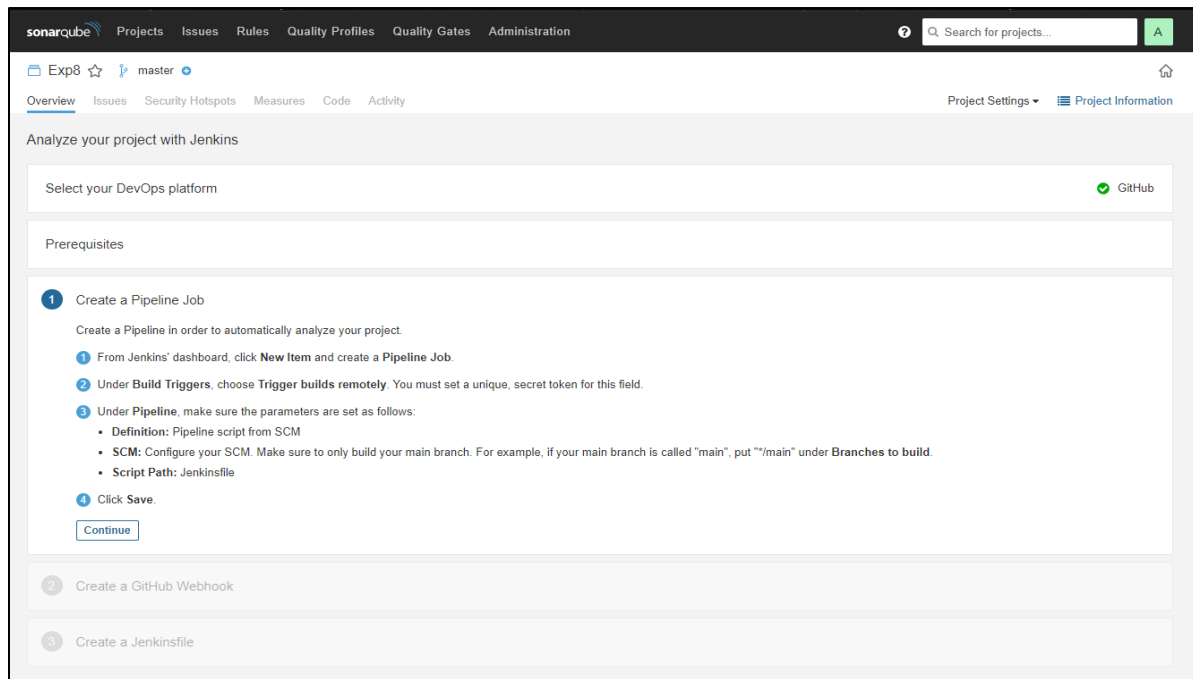
### Step 6: Check in the Jenkins that SonarQube Scanner tool is available or not in the manage plugins.



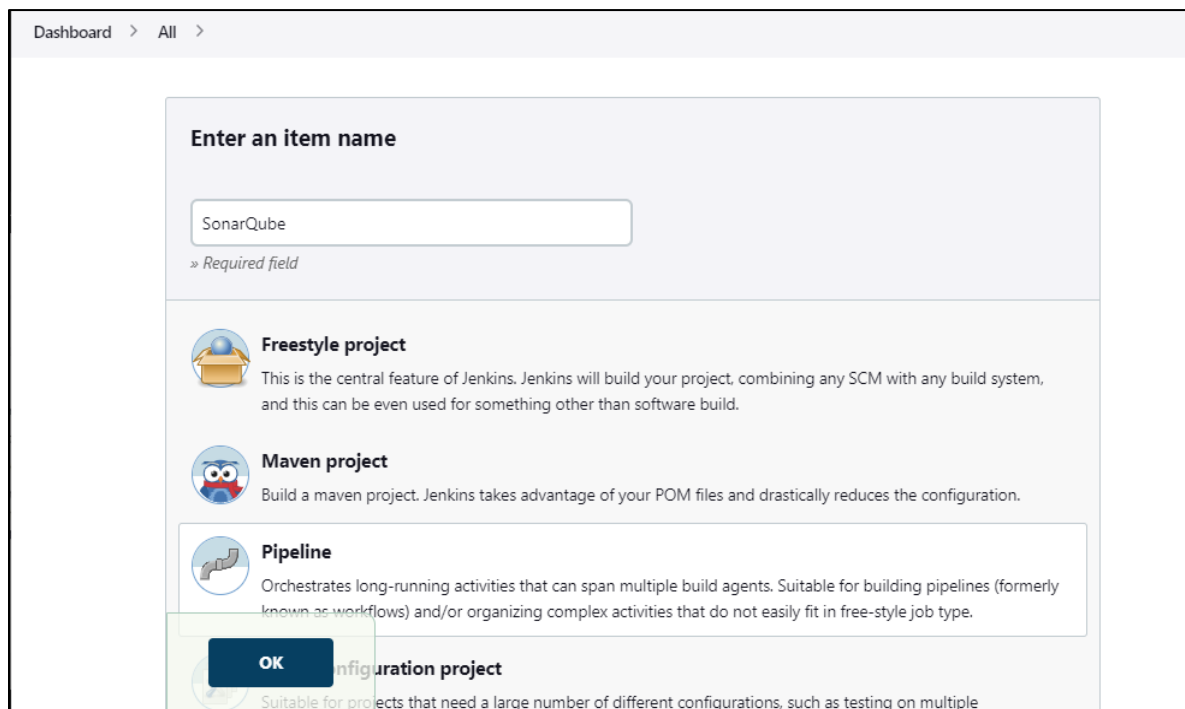
Name: Harsh Dalvi  
Roll No: 13

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

**Step 7:** Follow this steps to create a pipeline in Jenkins.



**Step 8:** Go to Jenkins and create on new item and give the name to your project and select Pipeline and click on OK



Name: Harsh Dalvi  
Roll No: 13

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

**Step 9:** In Build Triggers select Trigger builds remotely (e.g., from scripts) and set a unique token

Dashboard > SonarQube >

### Configuration

- General
- Advanced Project Options
- Pipeline

### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☒ Trigger builds remotely (e.g., from scripts) ?

Authentication Token

`http://localhost:8080/job/SonarQube/build?token=java`

Use the following URL to trigger build remotely: `JENKINS_URL/job/SonarQube/build?token=TOKEN_NAME` or `/buildWithParameters?token=TOKEN_NAME`  
Optionally append `&cause=Cause+Text` to provide text that will be included in the recorded build cause.

**Step 10:** Select SCM

Pipeline

Definition

Pipeline script from SCM

SCM ?

None

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

**Jenkins**

Dashboard > SonarQube >

### Pipeline SonarQube

- Status
- Changes
- Build Now
- Configure
- Delete Pipeline
- Full Stage View
- Rename
- Pipeline Syntax

### Recent Changes

[Recent Changes](#)

### Stage View

No data available. This Pipeline has not yet run.

### Permalinks

Build History

Filter builds...

No builds

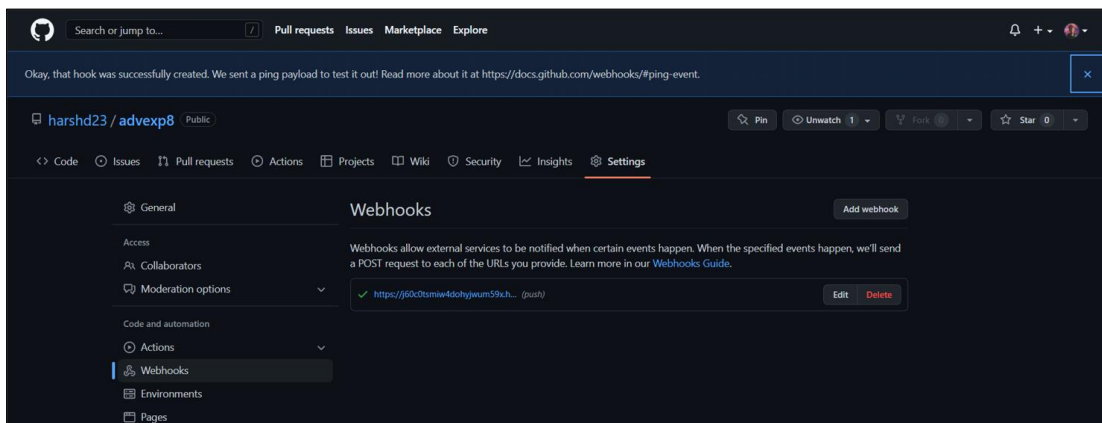
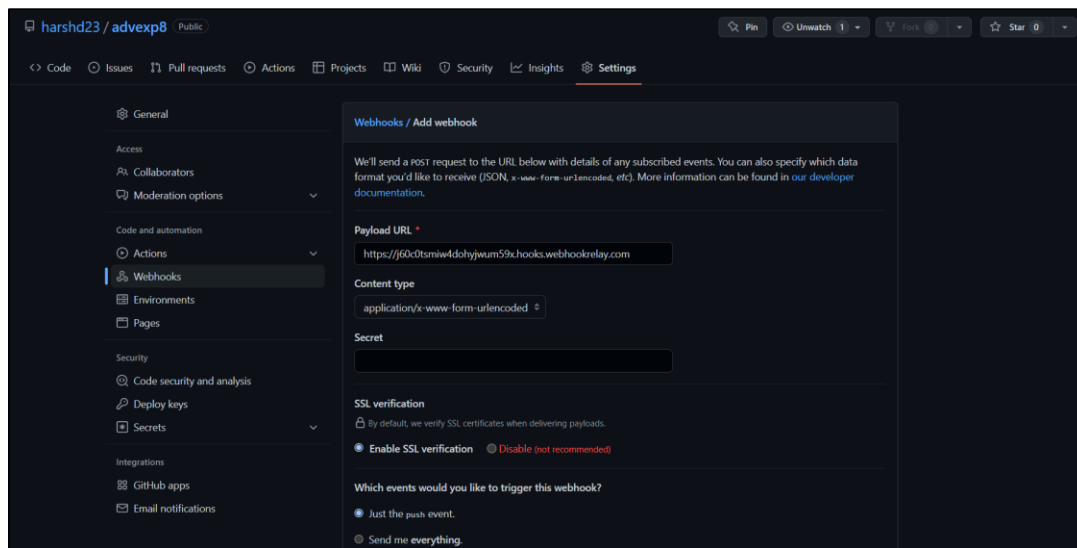
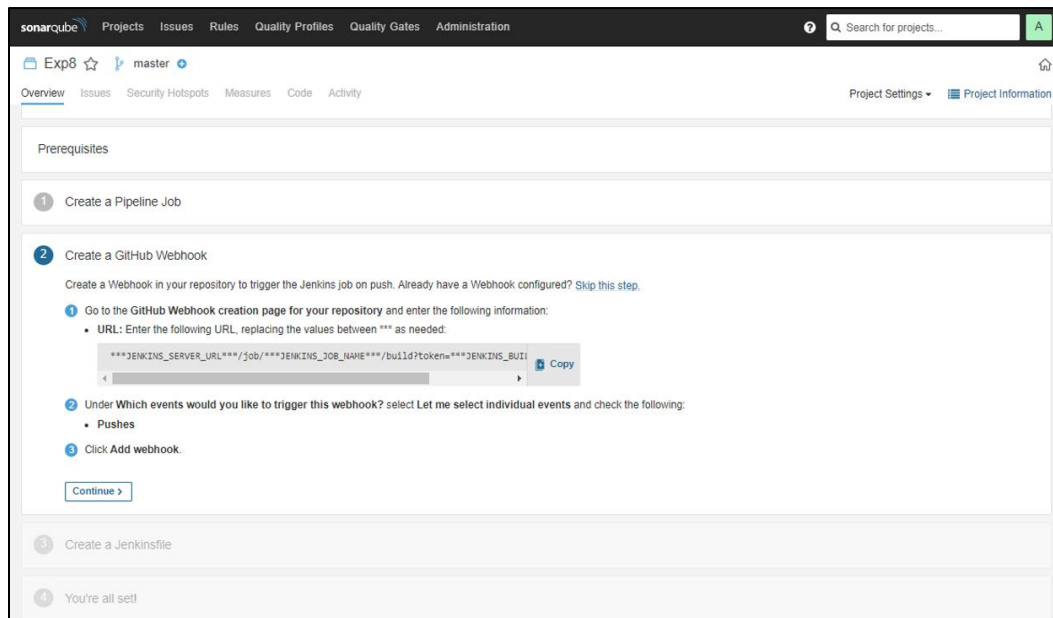
[Atom feed for all](#) [Atom feed for failures](#)



Name: Harsh Dalvi  
Roll No: 13

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

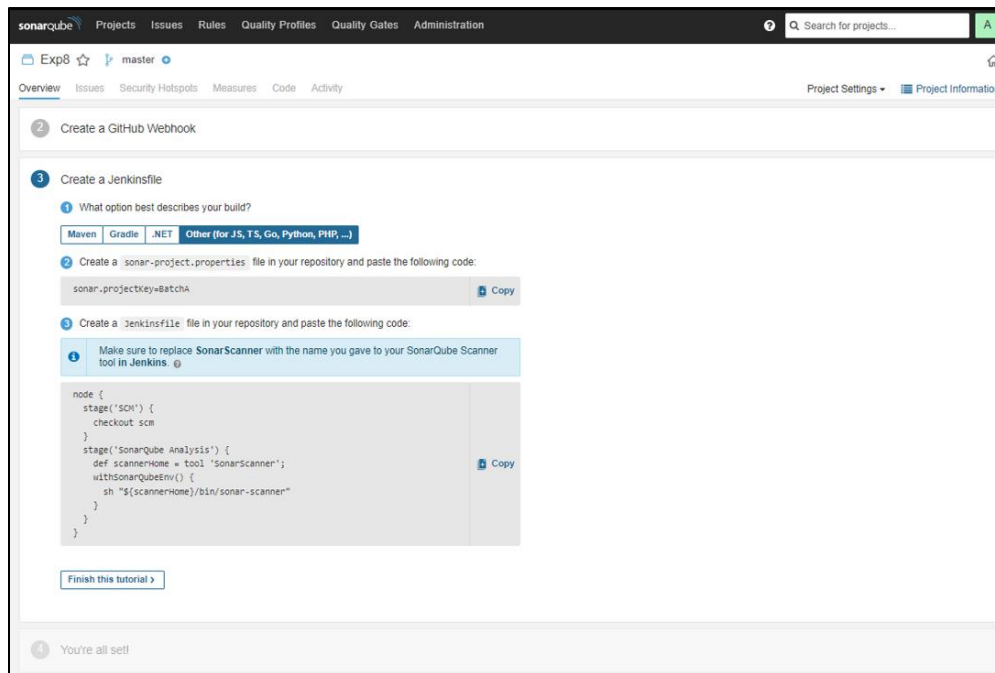
**Step 11:** Create a webhooks in your repository which is created in the GitHub.



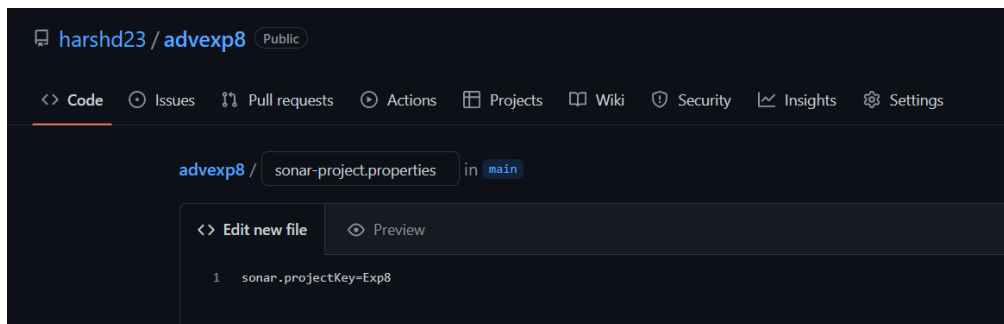
Name: Harsh Dalvi  
Roll No: 13

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

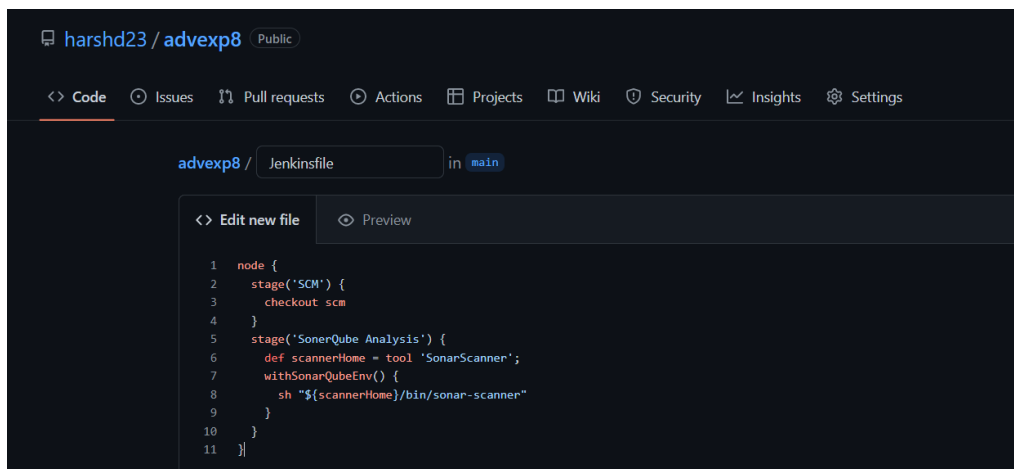
**Step 12:** Now, follow the steps given below to create a Jenkinsfile



Add one file in your repository.

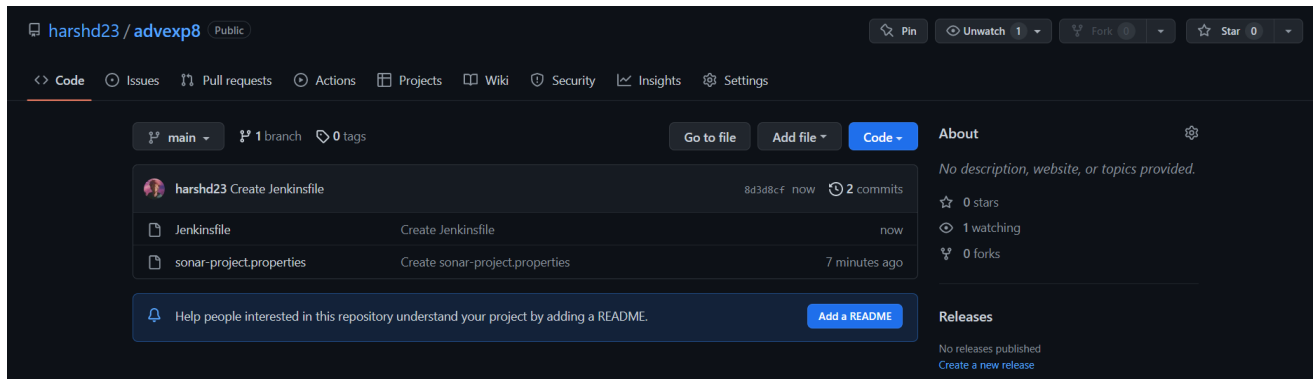


Then Jenkins file and write the below content.

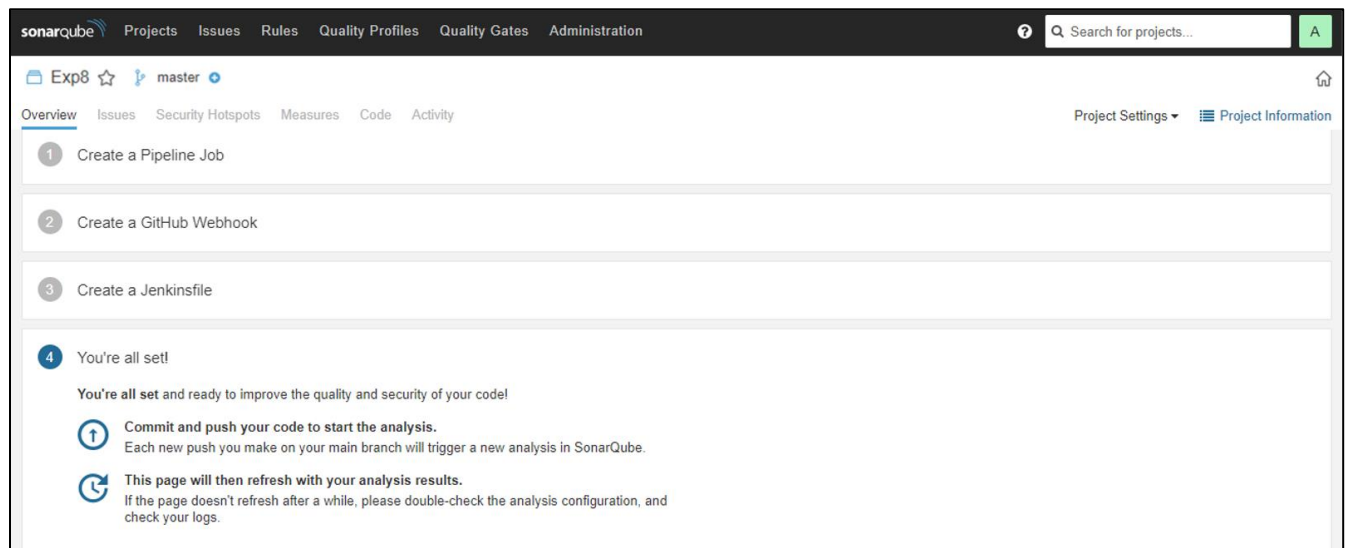


Name: Harsh Dalvi  
Roll No: 13

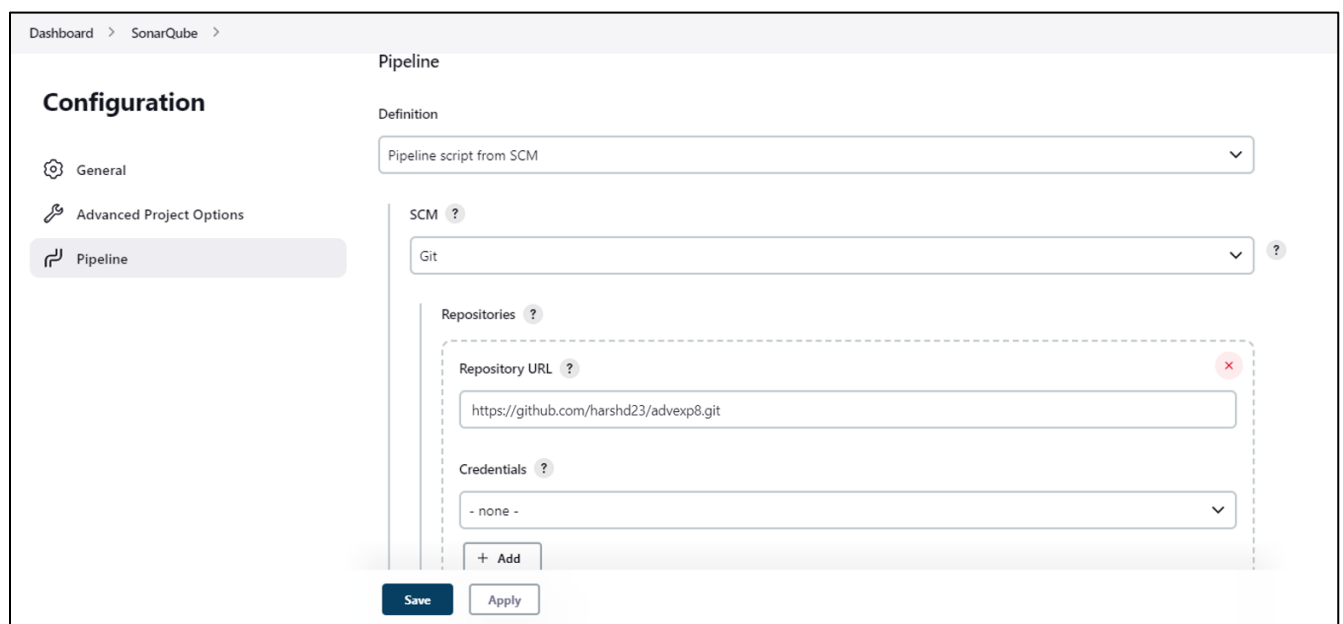
Subject: Advance DevOps , Sem: SEM V  
Class / Batch: TE-IT / Batch B



**Step 13:** After all the steps are completed, the last step i.e. It indicates that the project is configured.



**Step 14:** Now configure the Jenkins created project by providing the URL of the repository.



Name: Harsh Dalvi  
Roll No: 13

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

Dashboard > SonarQube >

### Configuration

- General
- Advanced Project Options
- Pipeline

Branch Specifier (blank for 'any') ? ✕

Add Branch

Repository browser ?

(Auto) ▼

Additional Behaviours

Add ▼

Script Path ?

Jenkinsfile

Save Apply

Apply and Save.

✓ Saved

### Configuration

- General
- Advanced Project Options
- Pipeline

Branch Specifier (blank for 'any') ? ✕

Add Branch

Repository browser ?

(Auto) ▼

Additional Behaviours

Add ▼

Script Path ?

Jenkinsfile

Save Apply

Name: Harsh Dalvi  
Roll No: 13

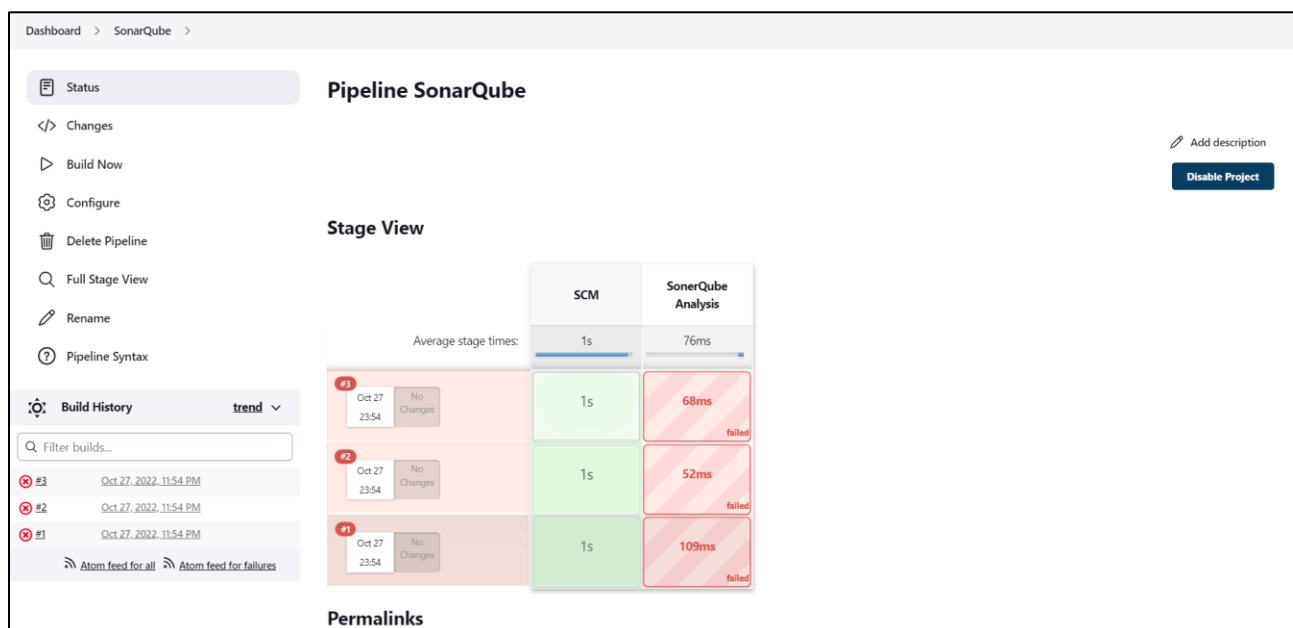
Subject: Advance DevOps , Sem: SEM V  
Class / Batch: TE-IT / Batch B

Then build now



The screenshot shows the Jenkins Console Output for a pipeline named 'Pipeline SonarQube'. The output details the process of cloning a Git repository, checking out a specific revision, and performing a static analysis using SonarQube. The analysis results show a failed status with a duration of 109ms.

```
Started by user admin
Obtained Jenkinsfile from git https://github.com/harshd23/advexp8.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\jenkins\workspace\SonarQube
[Pipeline] {
[Pipeline] stage
[Pipeline] { (SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/harshd23/advexp8.git
> git.exe init C:\ProgramData\Jenkins\jenkins\workspace\SonarQube # timeout=10
Fetching upstream changes from https://github.com/harshd23/advexp8.git
> git.exe --version # timeout=10
> git --version # 'git version 2.37.1.windows.1'
> git.exe fetch --tags --force --progress -- https://github.com/harshd23/advexp8.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git.exe config remote.origin.url https://github.com/harshd23/advexp8.git # timeout=10
> git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision 8d3d8cf06ba8da58b6bac300b4f801c707341bf5 (refs/remotes/origin/main)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f 8d3d8cf06ba8da58b6bac300b4f801c707341bf5 # timeout=10
Commit message: "Create Jenkinsfile"
```



The screenshot shows the Jenkins Pipeline SonarQube Stage View. It displays a table with columns for SCM, SonarQube Analysis, and Average stage times. The table shows three stages, all of which failed. The 'Build History' section on the left shows the build status for each stage.

|    | SCM | SonarQube Analysis | Average stage times |
|----|-----|--------------------|---------------------|
| #3 | 1s  | 68ms               | 1s                  |
| #2 | 1s  | 52ms               | 1s                  |
| #1 | 1s  | 109ms              | 1s                  |

Build History:

- #3 Oct 27, 2022, 11:54 PM No Changes
- #2 Oct 27, 2022, 11:54 PM No Changes
- #1 Oct 27, 2022, 11:54 PM No Changes

**Conclusion:** From this experiment it is concluded that, we have successfully constructed a Jenkins CICD pipeline with SonarQube Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web / Java / Python application. 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.