



PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



Academic Year: 2022-23

Class / Branch: TE IT

Subject: Advanced Devops Lab (ADL)

Subject Lab Incharge: Prof. Manjusha Kashilkar

Name: Chirag Jayesh Malde

Semester: V

Moodle ID: 22104186

EXPERIMENT NO. 07

Aim: To understand Static Analysis SAST process and learn to integrate Jenkins SonarQube/GitLab.

Theory:

Static application security testing (SAST) is a way to perform automated testing and analysis of a program's source code without executing it to catch security vulnerabilities early on in the software development cycle. Also referred to as static code analysis, SAST is the process of parsing through the code looking at how it was written and checking for security vulnerabilities and safety concerns.

Because static application security testing tools don't need a running application to perform an analysis, they can be used early and often in the implementation phase of the software development life cycle (SDLC). As a developer is writing code, SAST can analyze it in real-time to inform the user of any rule violations, so you can immediately deal with issues and deliver higher quality applications out of the box while preventing issues at the end of the development process.

Additionally, as SAST helps you audit code and triage issues during implementation, test automation tools can also easily integrate into development ecosystems where continuous integration/continuous delivery (CI/CD) are part of the workflow that helps assure secure, safe, and reliable code during integration, and before it's delivered.

What's the Difference Between SAST and DAST?

While SAST analyses every line of code without running the application, dynamic application security testing (DAST) simulates malicious attacks and other external behaviors by searching for ways to exploit security vulnerabilities during runtime or black box testing.

DAST is particularly useful when catching unexpected vulnerabilities that development teams simply didn't think of. This additional level of insight that DAST brings offers a broad array of security testing to find flaws and prevent attacks like SQL injections, cross-site scripting (XSS), and



PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



other exploits. Remember the 2014 Sony Pictures hack? That could have been prevented with DAST.

Comparing SAST against DAST, each is more effective than the other during different stages of the SDLC. SAST represents the developer's point of view to make sure that all coding procedures follow the appropriate safety standards to ensure the security of an application from the start. DAST, on the other hand, mimics the hacker approach to identify possible user behavior towards the end of development.

Steps:

- 1) Install and configure a Jenkins and SonarQube CI/CD environment using containers.**
- 2) Configure Jenkins with the SonarQube Scanner plugin for automated analysis.**

1) Install and configure a Jenkins and SonarQube CI/CD environment using Docker containers.

Installation of Jenkins

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To take advantage of the latest fixes and features, you can use the project-maintained packages to install Jenkins.

```
manjusha@apsite:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key  
| sudo apt-key add -
```

When the key is added, the system will return OK. Next, append the Debian package repository address to the server's sources.list:

```
manjusha@apsite:~$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```



PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



When both of these are in place, run `update` so that `apt` will use the new repository:

```
manjusha@apsit:~$ apt update
```

Finally, install Jenkins and its dependencies:

```
manjusha@apsit:~$ sudo apt install jenkins
```

Let's start Jenkins using `systemctl`:

```
manjusha@apsit:~$ sudo systemctl start jenkins
```

Since `systemctl` doesn't display output, you can use its `status` command to verify that Jenkins started successfully:

```
manjusha@apsit:~$ sudo systemctl status jenkins
```

If everything went well, the beginning of the output should show that the service is active and configured to start at boot:

Now that Jenkins is running, let's adjust our firewall rules so that we can reach it from a web browser to complete the initial setup.

Opening the Firewall

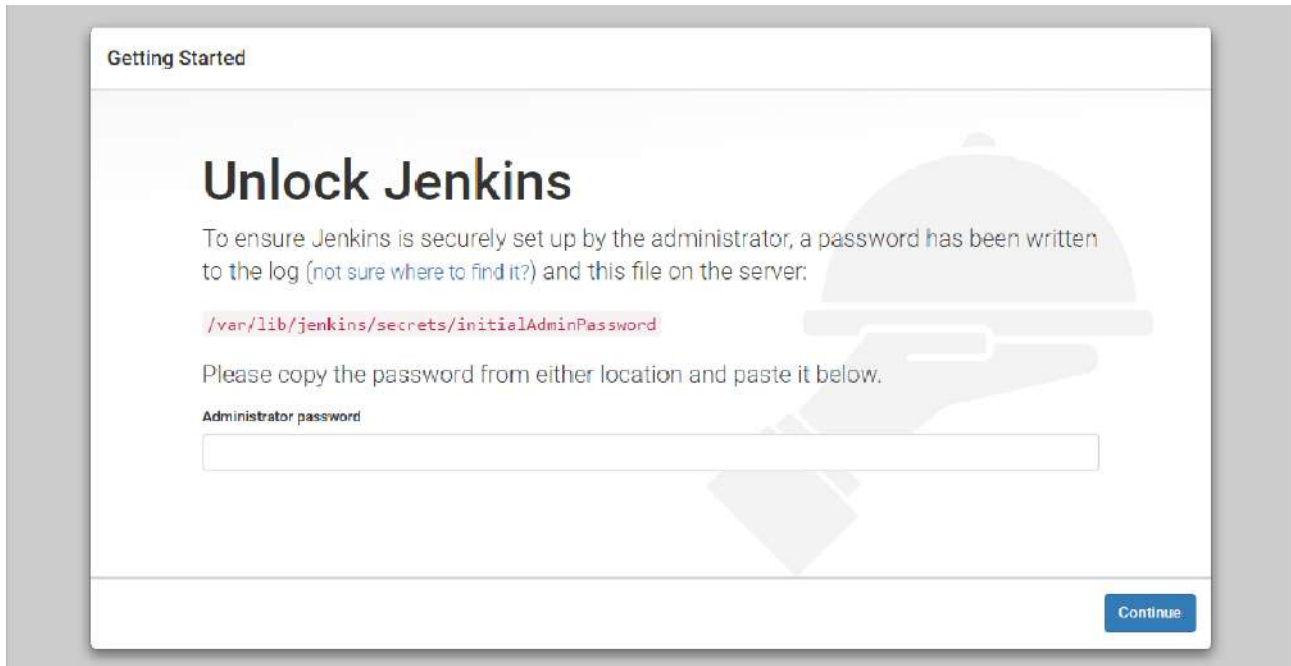
By default, Jenkins runs on port 8080, so let's open that port using `ufw`:

```
manjusha@apsit:~$ sudo ufw allow
```

8080 Setting Up Jenkins

To set up your installation, visit Jenkins on its default port, 8080, using your server domain name or IP address: **`http://your_server_ip_or_domain:8080`**

You should see the Unlock Jenkins screen, which displays the location of the initial password:



In the terminal window, use the cat command to display the password:

manjusha@apsit:~\$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

The next screen presents the option of installing suggested plugins or selecting specific plugins:





PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



We'll click the Install suggested plugins option, which will immediately begin the installation process:

Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	<pre>** Pipeline: Milestone Step ** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) ** Jackson 2 API ** JavaScript GUI Lib: ACE Editor bundle ** Pipeline: SCM Step ** Pipeline: Groovy ** Pipeline: Input Step ** Pipeline: Stage Step ** Pipeline: Job ** Pipeline: Graph Analysis ** Pipeline: REST API ** JavaScript GUI Lib: Handlebars bundle ** JavaScript GUI Lib: Moment.js bundle Pipeline: Stage View ** Pipeline: Build Step ** Pipeline: Model API ** Pipeline: Declarative Extension Points API ** Apache HttpComponents Client 4.x API ** JSch dependency</pre>
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle	
🔄 Pipeline	🔄 GitHub Branch Source	🔄 Pipeline: GitHub Groovy Libraries	✓ Pipeline: Stage View	
🔄 Git	🔄 Subversion	🔄 SSH Slaves	🔄 Matrix Authorization Strategy	
🔄 PAM Authentication	🔄 LDAP	🔄 Email Extension	🔄 Mailer	



PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



Getting Started

Create First Admin User

Username:	<input type="text" value="vishal"/>
Password:	<input type="password" value="*****"/>
Confirm password:	<input type="password" value="*****"/>
Full name:	<input type="text" value="Vishal Badgujar"/>
E-mail address:	<input type="text" value="vsbadgujar@apsit.edu.in"/>

Jenkins 2.289.2

[Skip and continue as admin](#)

[Save and Continue](#)

When the installation is complete, you will be prompted to set up the first administrative user. It's possible to skip this step and continue as admin using the initial password we used above, but we'll take a moment to create the user.

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

After confirming the appropriate information, click Save and Finish. You will see a confirmation page confirming that "Jenkins is Ready!":



Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

Click Start using Jenkins to visit the main Jenkins dashboard:

The screenshot shows the Jenkins Dashboard in a web browser. The dashboard includes a sidebar with navigation links: New Item, People, Build History, Project Relationship, Check File Fingerprint, and Manage Jenkins. The main content area displays a table of build jobs with columns for Status (S), Warnings (W), Name, Last Success, Last Failure, Last Duration, and Coverage. The table lists several build jobs, including 'aaryan213', 'helloworld_06_08_2024', 'mayank', 'mayank_', 'pipeline', 'Sonarqube', and 'SonarQube1'. The 'Build Queue' section shows 'No builds in the queue.' and the 'Build Executor Status' section shows '1 Idle' and '2 Idle'.

S	W	Name	Last Success	Last Failure	Last Duration	Coverage
✖	☁	aaryan213	N/A	1 hr 51 min #11	1.2 sec	▶ n/a
⋮	☀	helloworld_06_08_2024	N/A	N/A	N/A	▶ n/a
✔	☁	mayank	5 days 23 hr #27	5 days 23 hr #26	13 sec	▶ n/a
✔	☀	mayank_	1 mo 10 days #1	N/A	1.2 sec	▶ n/a
⋮	☀	pipeline	N/A	N/A	N/A	▶ n/a
✖	☁	Sonarqube	N/A	26 days #10	0.79 sec	▶ n/a
✖	☁	SonarQube1	N/A	1 hr 57 min #6	0.14 sec	▶ n/a
✔	☀		1 hr 54 min			



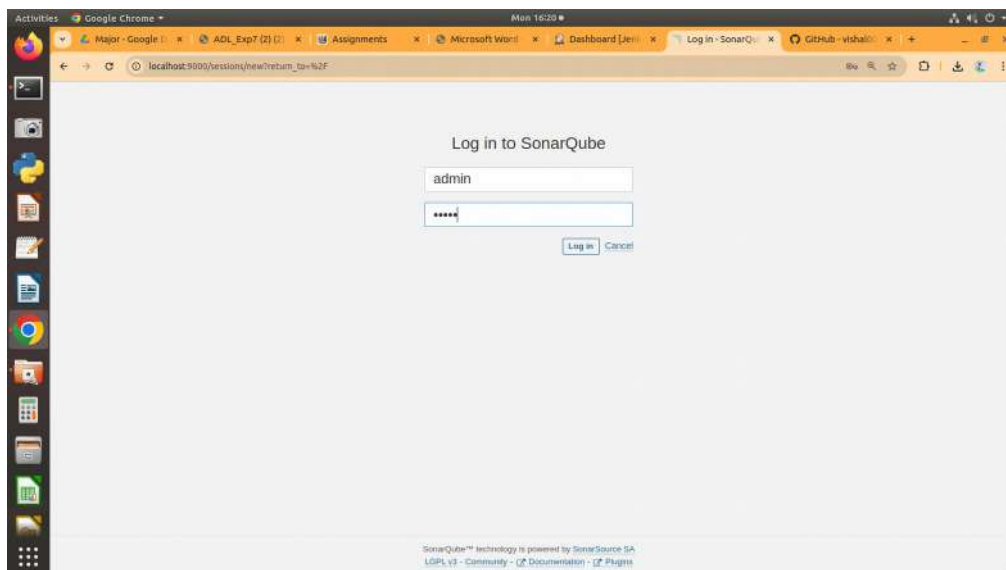
SonarQube Setup

Before proceeding with the integration, we will setup SonarQube Instance. we are using SonarQube Docker Container.

manjusha@apsit:~\$docker run -d -p 9000:9000 sonarqube

```
apsit@apsit-HP-280-Pro-G6-Microtower-PC:~$ sudo docker run -d -p 9000:9000 sonarqube  
9b8194b9a4766719e8e51f4294fb674845c2c2d90463e247218061553e76aa67
```

In the above command, we are forwarding port 9000 of the container to the port 9000 of the host machine as SonarQube is will run on port 9000. Then, from the browser, enter <http://localhost:9000>. After That, you will see the SonarQube is running. Then, login using default credentials (admin:admin).

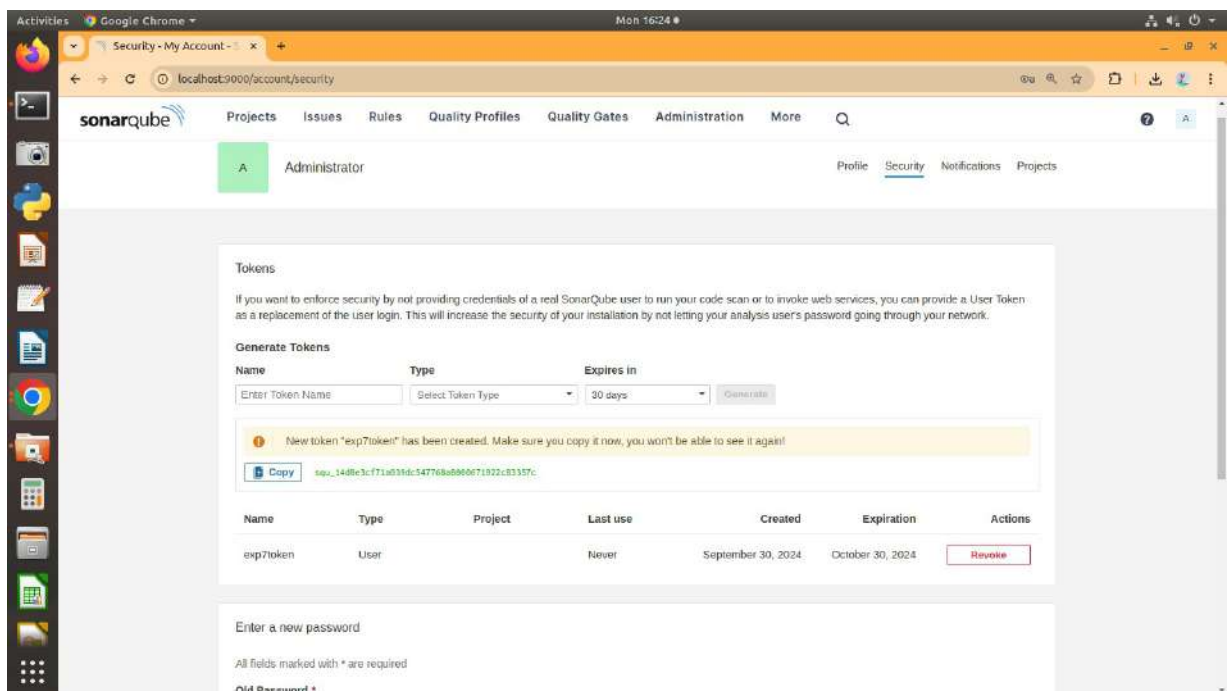




Generate User Token

Now, we need to get the SonarQube user token to make connection between Jenkins and SonarQube. For the same, go to **Administration > User > My Account > Security**, at the bottom of the page you can create new tokens by clicking the Generate Button. Copy the Token and keep it safe.

C96798e9bd081e117189b516c868ddb7d87ee785 **SonarQube**

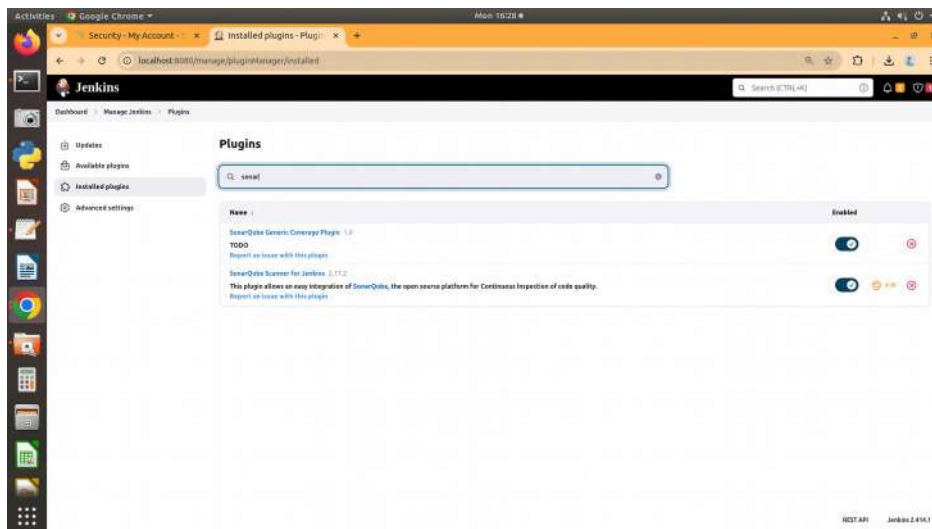




2) Configure Jenkins with the SonarQube Scanner plugin for static code analysis.

Jenkins Setup for SonarQube

Before all, we need to install the SonarQube Scanner plugin in Jenkins. For the same, go to **Manage Jenkins > Plugin Manager > Available**, type SonarQube Scanner then select and install.

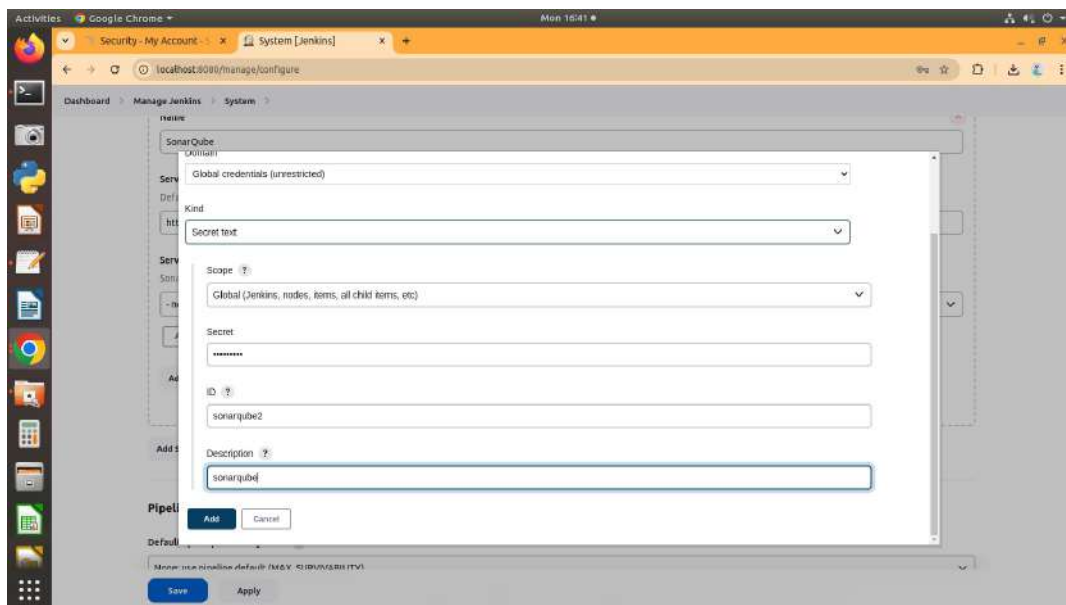


Tool Configuration SonarQube Scanner

Now, we need to configure the Jenkins plugin for SonarQube Scanner to make a connection with the SonarQube Instance. For that, got to **Manage Jenkins > Configure System > SonarQube Server**. Then, Add SonarQube. In this, give the Installation Name, Server URL then Add the Authentication token in the Jenkins Credential Manager and select the same in the configuration.



PARSHVANATH CHARITABLE TRUST'S
A. P. SHAH INSTITUTE OF TECHNOLOGY
Department of Information Technology
(NBA Accredited)



Then, we need to set-up the SonarQube Scanner to scan the source code in the various stage. For the same, go to **Manage Jenkins > Global Tool Configuration > SonarQube Scanner**. Here, give some name of the scanner type and **Add** your choice. In this case, I have selected SonarQube Scanner from Maven Central.



PARSHVANATH CHARITABLE TRUST'S
A. P. SHAH INSTITUTE OF TECHNOLOGY
Department of Information Technology
(NBA Accredited)



Activities Google Chrome Mon 16:44

Security - My Account - x Tools [Jenkins] x +

localhost:8080/manage/configureTools/

Dashboard > Manage Jenkins > Tools

SonarQube Scanner Installations

SonarQube Scanner Installations ^ Edited

Add SonarQube Scanner

SonarQube Scanner

Name

SonarQube

☒ Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 4.6.2.2472

Add Installer

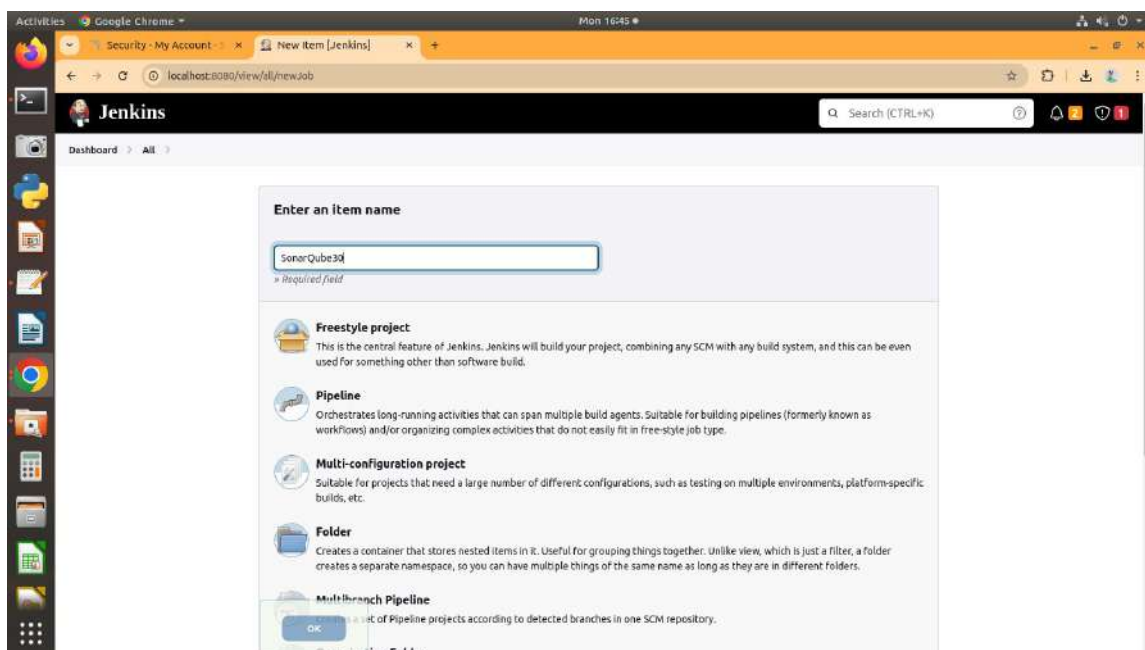
Add SonarQube Scanner

Save Apply



SonarQube Scanner in Jenkins Pipeline

Now, It's time to integrate the SonarQube Scanner in the Jenkins Pipeline. For the same, we are going to add one more stage in the Jenkinsfile called SonarQube and inside that, I am adding the following settings and code.





PARSHVANATH CHARITABLE TRUST'S

A. P. SHAH INSTITUTE OF TECHNOLOGY

Department of Information Technology

(NBA Accredited)



☐ Discard old builds ?

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url ?

Advanced ▾

☐ Pipeline speed/durability override ?

☐ Preserve stashes from completed builds ?

☐ This project is parameterised ?

☐ Throttle builds ?

Build Triggers

Github Configuration in Jenkins Pipeline

Script ?

```
1 node
2 {
3     stage('Cloning from GIT'){
4         git branch: 'main', credentialsId: 'GIT_REPO', url: 'https://github.com/vishal003/jenkins-sonarqube.git'
5     }
6 }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Git Clonning into Jenkins



PARSHVANATH CHARITABLE TRUST'S
A. P. SHAH INSTITUTE OF TECHNOLOGY
Department of Information Technology
(NBA Accredited)



github.com/vishal003/jenkins-sonarqube

Apps Vishal Badgujar Vishal Sahebr... Gmail YouTube BDA - Google... Mumbai Un... WhatsApp Cisco

Search or jump to... Pull requests Issues Marketplace Explore

vishal003 / jenkins-sonarqube
forked from devopshint/jenkins-sonarqube

<> Code Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

This branch is 6 commits ahead of devopshint:main. Contribute Fetch upstream

vishal003 Update README.md 86c34f4 41 minutes ago 16 commits

project	Update sonar-analysis	42 minutes ago
README.md	Update README.md	41 minutes ago

README.md

jenkins-Github-sonarqube CICD Pipeline

Github Repository Contents



PARSHVANATH CHARITABLE TRUST'S
A. P. SHAH INSTITUTE OF TECHNOLOGY
Department of Information Technology
(NBA Accredited)



```
Started by user unknown or anonymous
[Pipeline] Start at Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/SonarQube30
[Pipeline] {
[Pipeline] stage
[Pipeline] { (cloning from GIT)
[Pipeline] git
The recommended git tool is: NONE
Warning: CredentialId "GIT_REPO" could not be found.
Cloning the remote Git repository
Cloning repository https://github.com/vishal003/jenkins-sonarqube.git
> git init /var/lib/jenkins/workspace/SonarQube30 # timeout=10
Fetching upstream changes from https://github.com/vishal003/jenkins-sonarqube.git
> git --version # timeout=10
> git --version # 'git version 2.17.1'
> git fetch --tags --progress -- https://github.com/vishal003/jenkins-sonarqube.git # timeout=10
> git config remote.origin.url https://github.com/vishal003/jenkins-sonarqube.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 80c34f4818e25f7733e50784c2f7639d9804ed90 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 80c34f4818e25f7733e50784c2f7639d9804ed90 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git checkout -b main 80c34f4818e25f7733e50784c2f7639d9804ed90 # timeout=10
```

Successfully Build Github Repository in Jenkins

Pre-requisite required for Integration settings of Jenkins SAST with SonarQube v here successfully, now in order to Integrate of Jenkins CI/CD with SonarQube with sample JAVA program we will implement in next experiment.

Conclusion: Thus we understood how to configure sonar and use it.