Academic Year: 2022-23 **Semester: V**
Class / Branch: TE IT
Subject: Advanced Devops Lab (ADL)
Subject Lab Incharge: Prof. Manasi Choche

EXPERIMENT NO. 07

Aim: To understand Static Analysis SAST process and learn to integrate Jenkins SAST to 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 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 CICD environment using Docker containers.

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

1) Install and configure a Jenkins and SonarQube CICD 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.

```
vishal@apsit:~$ 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:

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

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

```
vishal@apsit:~$ sudo apt update
```

Finally, install Jenkins and its dependencies:

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

Let’s start Jenkins using systemctl:

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

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

```
vishal@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

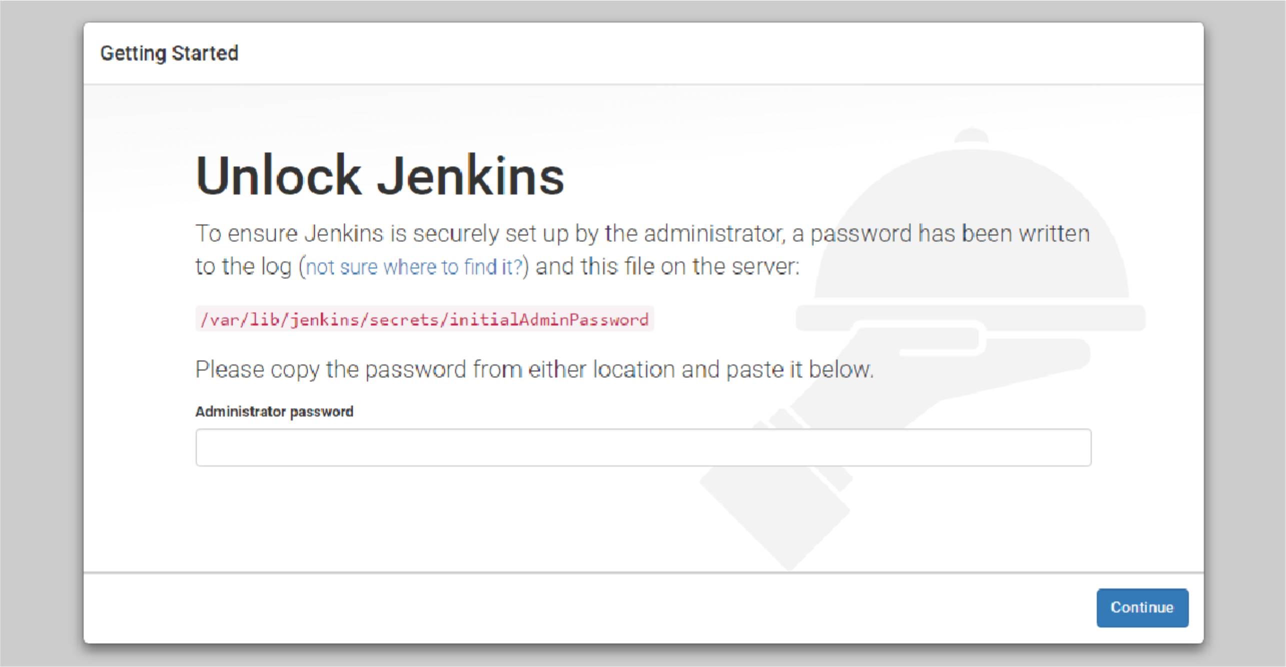
By default, Jenkins runs on port 8080, so let’s open that port using ufw:

```
vishal@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:



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

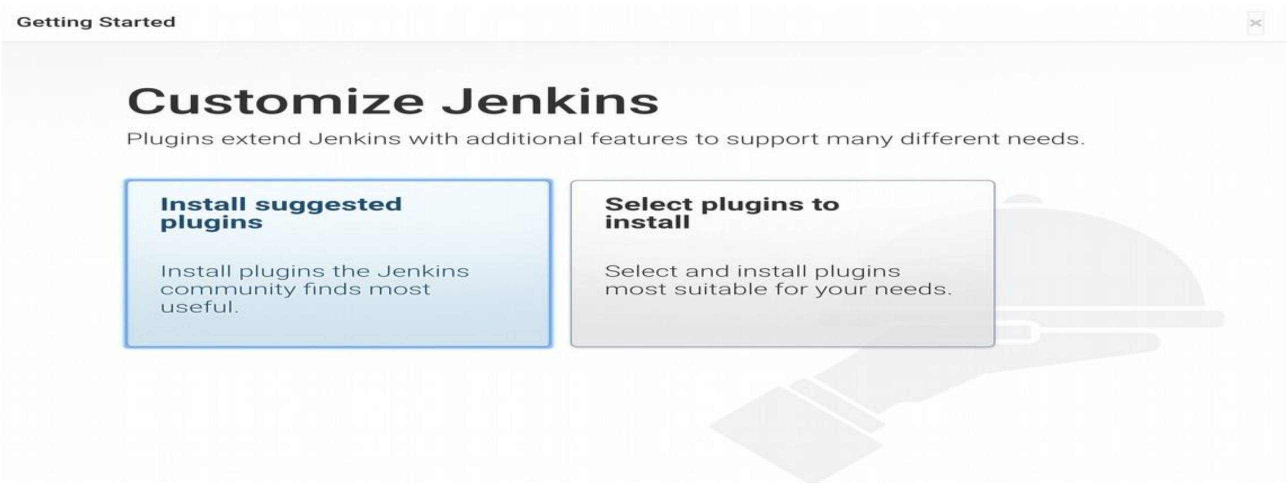
Administrator password

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

vishal@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:



Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

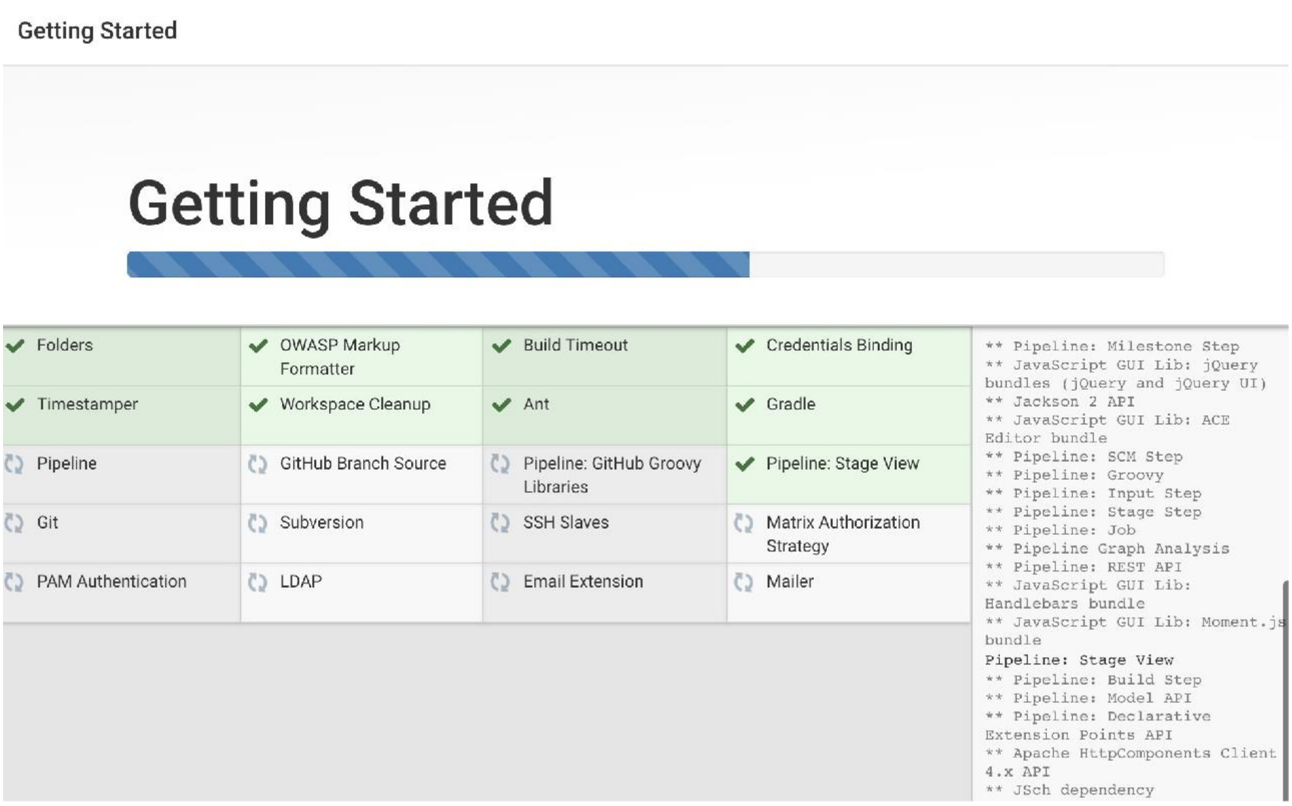
Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

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

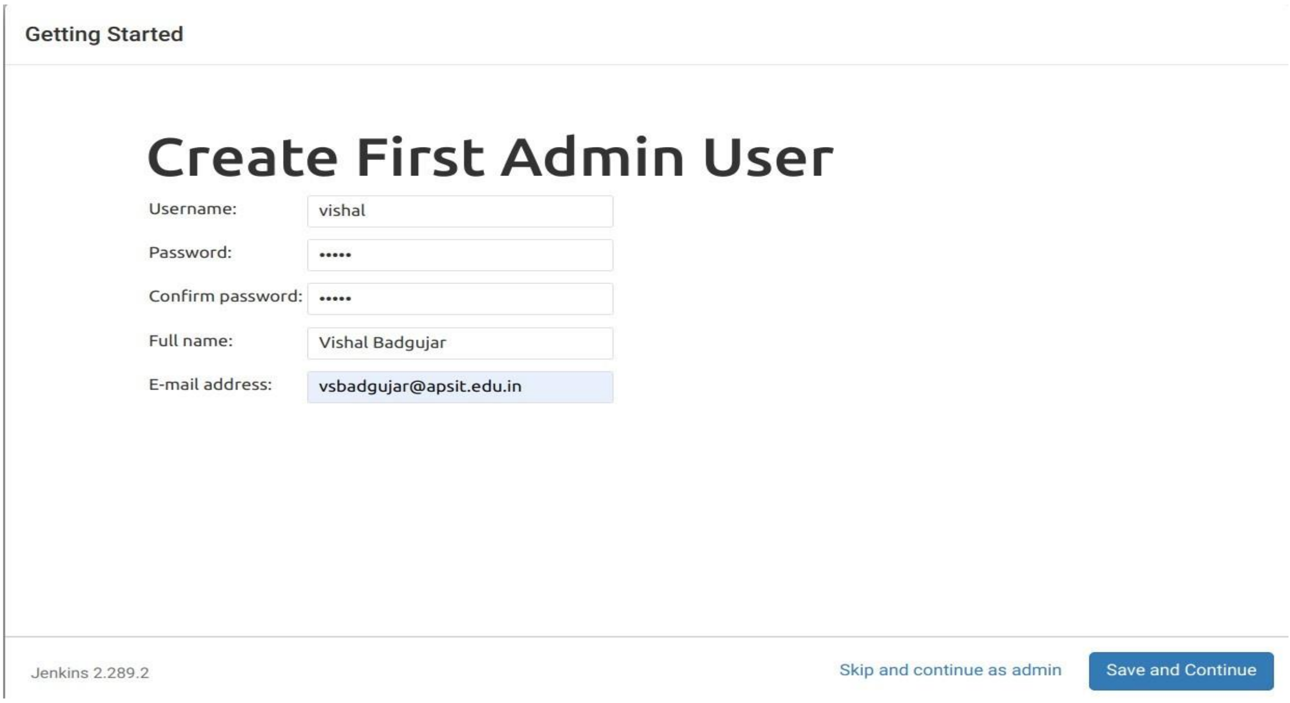


Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ 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

```
** 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
```



Getting Started

Create First Admin User

Username: vishal

Password: •••••

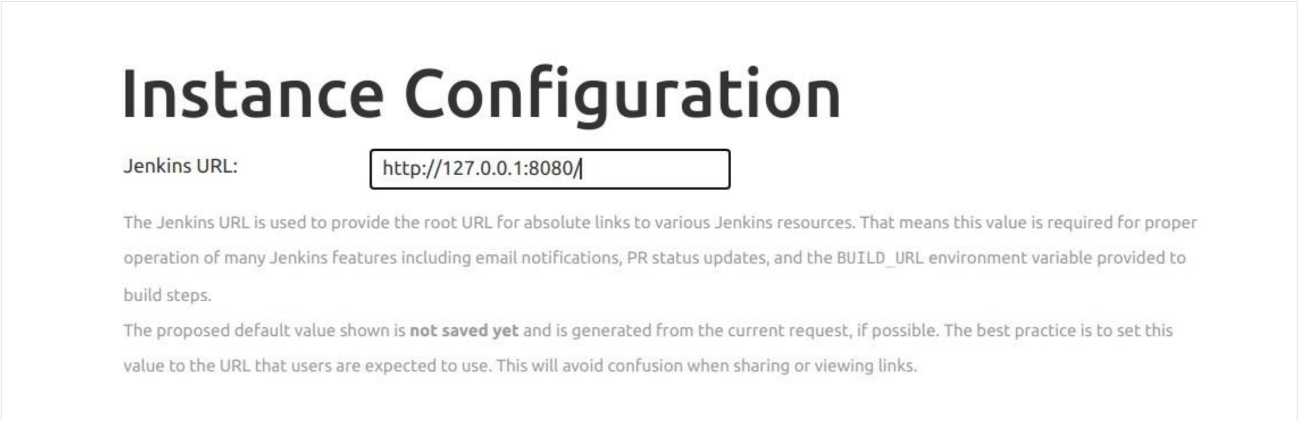
Confirm password: •••••

Full name: Vishal Badgujar

E-mail address: vsbadgujar@apsit.edu.in

Jenkins 2.289.2

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.



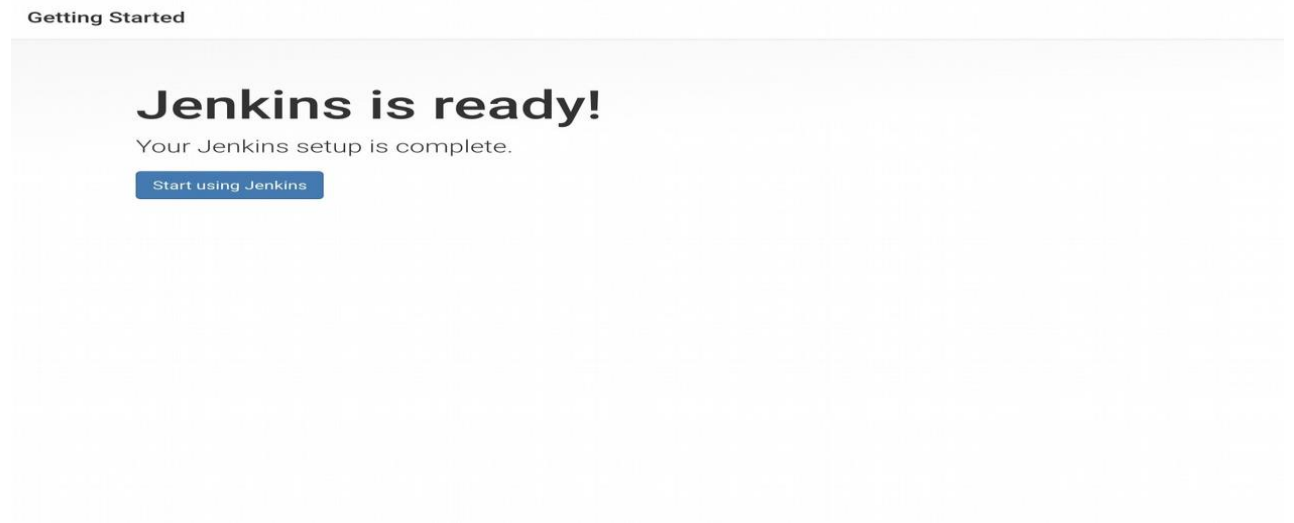
Instance Configuration

Jenkins URL: http://127.0.0.1:8080/

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!”:

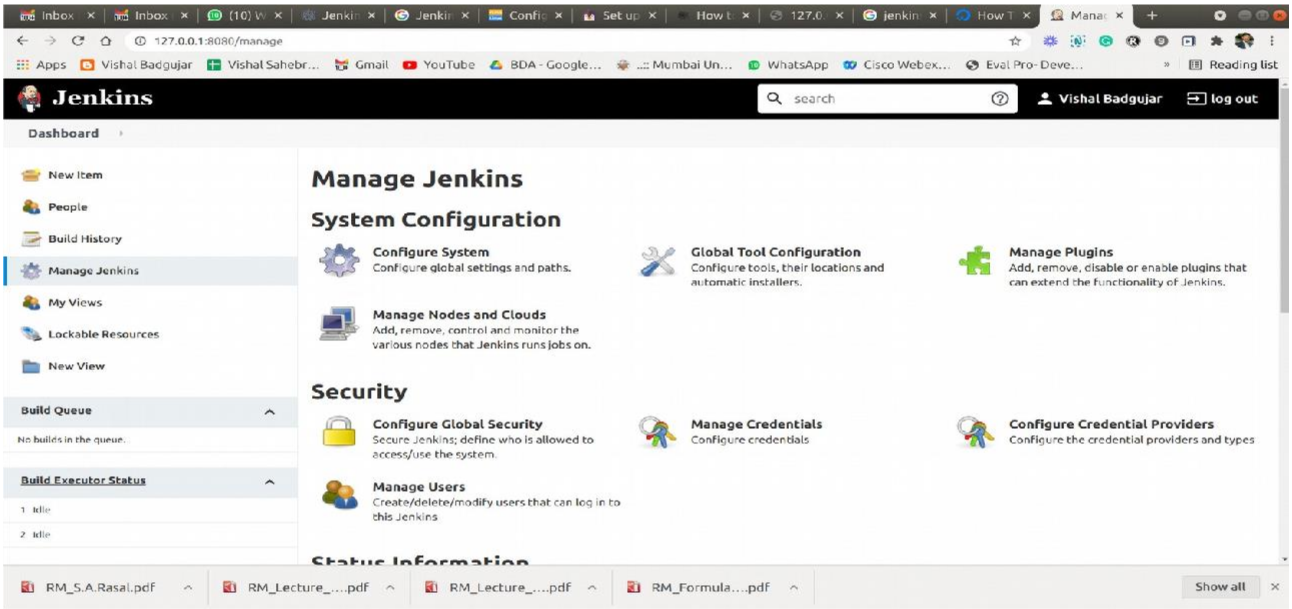


Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

Click Start using Jenkins to visit the main Jenkins dashboard:



No builds in the queue.

1 Idle

2 Idle

Manage Jenkins

System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Manage Credentials
Configure credentials

Configure Credential Providers
Configure the credential providers and types

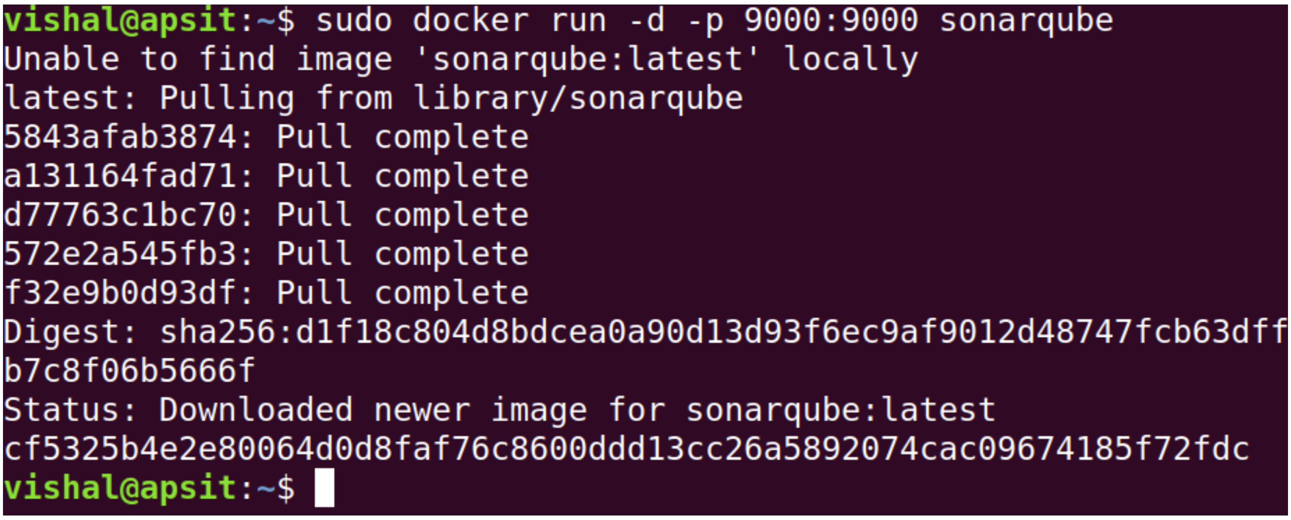
Manage Users
Create/delete/modify users that can log in to this Jenkins

Status Information

SonarQube Setup

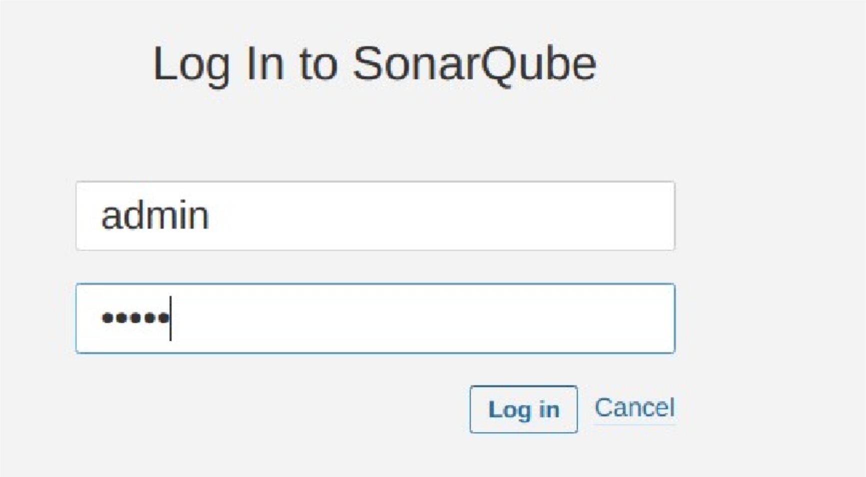
Before proceeding with the integration, we will setup SonarQube Instance. we are using SonarQube Docker Container.

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



```
vishal@apsit:~$ sudo docker run -d -p 9000:9000 sonarqube
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
5843afab3874: Pull complete
a131164fad71: Pull complete
d77763c1bc70: Pull complete
572e2a545fb3: Pull complete
f32e9b0d93df: Pull complete
Digest: sha256:d1f18c804d8bdcea0a90d13d93f6ec9af9012d48747fcb63dff
b7c8f06b5666f
Status: Downloaded newer image for sonarqube:latest
cf5325b4e2e80064d0d8faf76c8600ddd13cc26a5892074cac09674185f72fdc
vishal@apsit:~$
```

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).



Log In to SonarQube

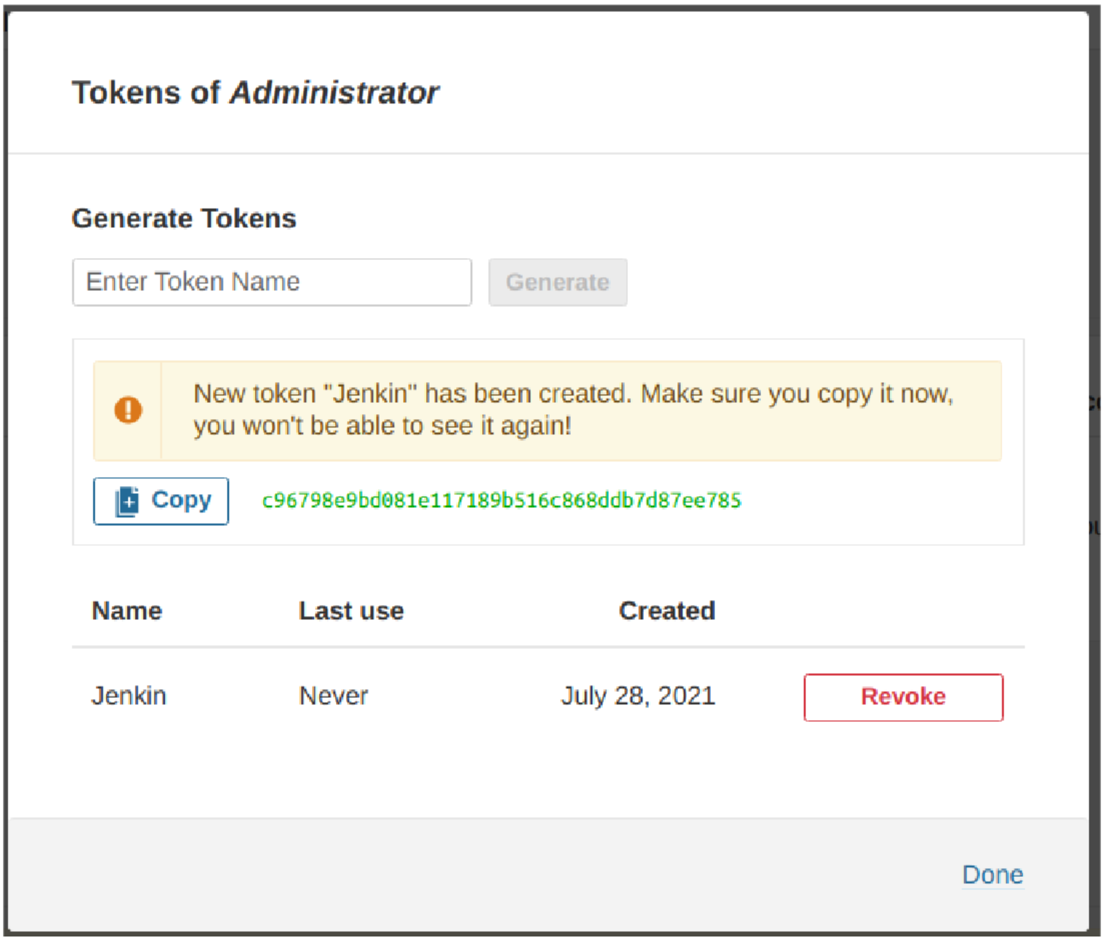
admin

•••••

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** and then, from the bottom of the page you can create new tokens by clicking the Generate Button. Copy the Token and keep it safe.

C96798e9bd081e117189b516c868ddb7d87ee785 SonarQube



Tokens of *Administrator*

Generate Tokens

New token "Jenkin" has been created. Make sure you copy it now, you won't be able to see it again!

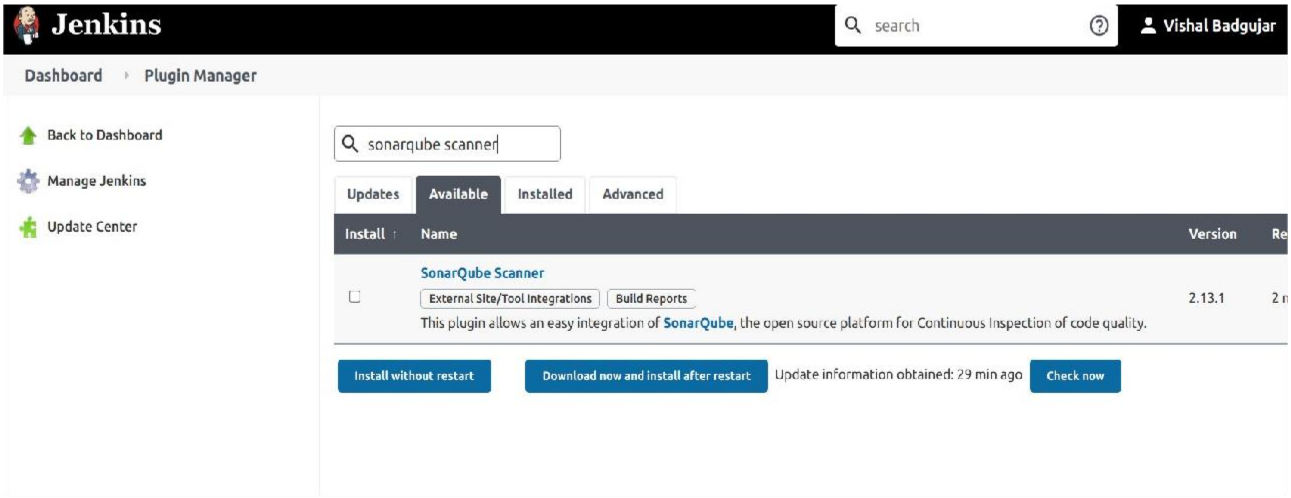
c96798e9bd081e117189b516c868ddb7d87ee785

Name	Last use	Created	
Jenkin	Never	July 28, 2021	Revoke

2) Configure Jenkins with the SonarQube Scanner plugin for automated 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.** From here, type SonarQube Scanner then select and install.

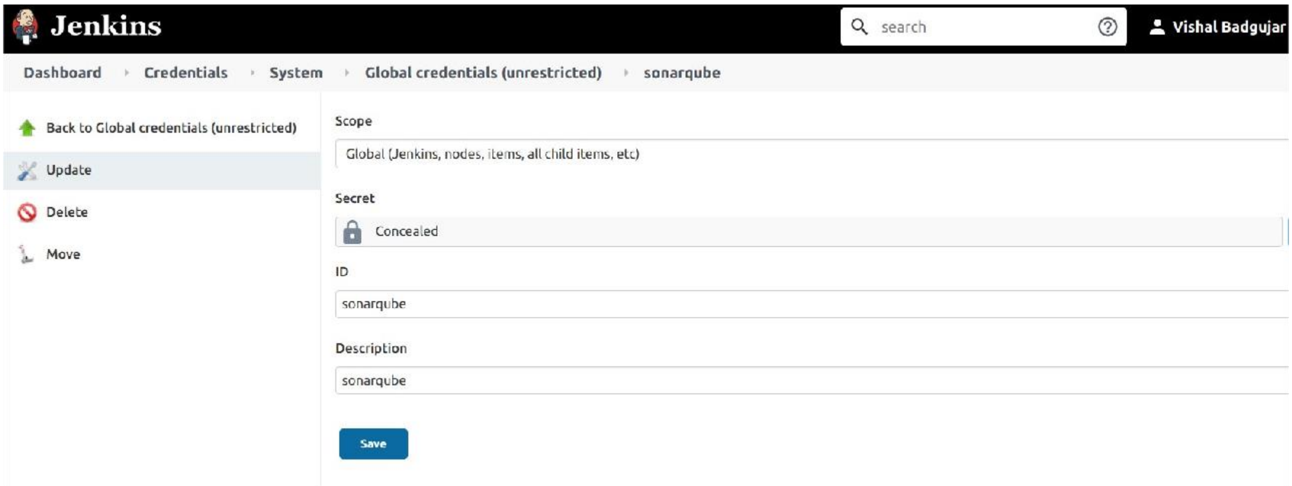


Install	Name	Version	Re
☐	SonarQube Scanner External Site/Tool Integrations Build Reports This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	2.13.1	2 r

Update information obtained: 29 min ago

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.



Scope

Global (Jenkins, nodes, items, all child items, etc)

Secret

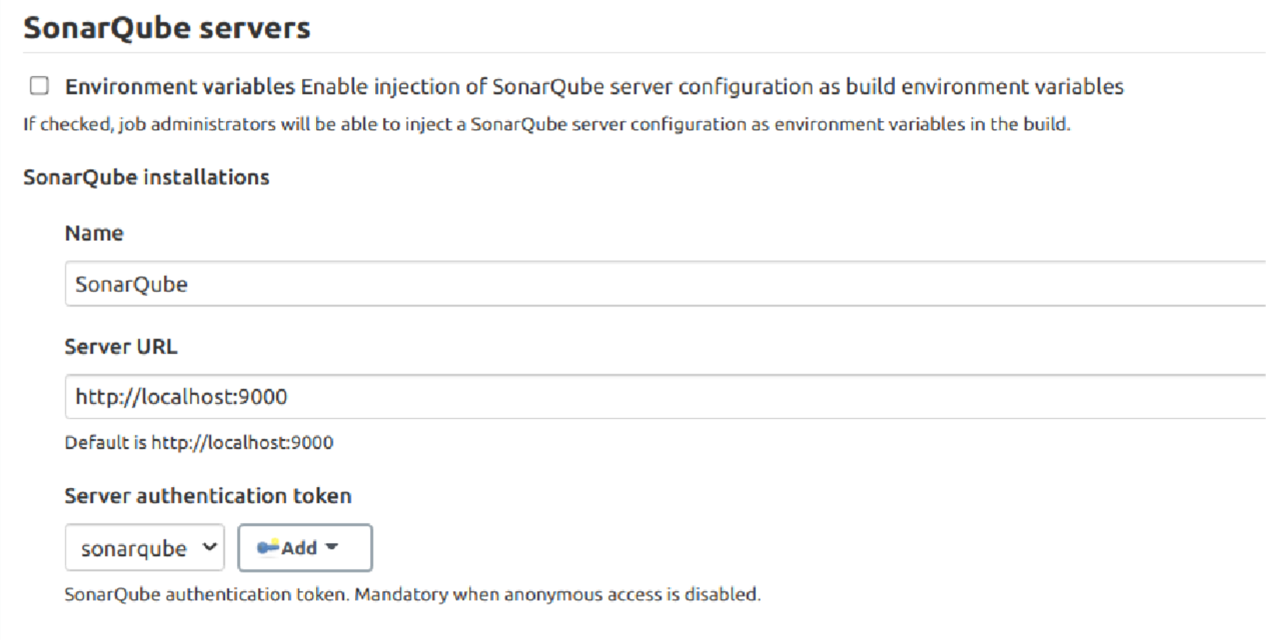
Concealed

ID

sonarqube

Description

sonarqube



SonarQube servers

☐ Environment variables Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

SonarQube

Server URL

http://localhost:9000

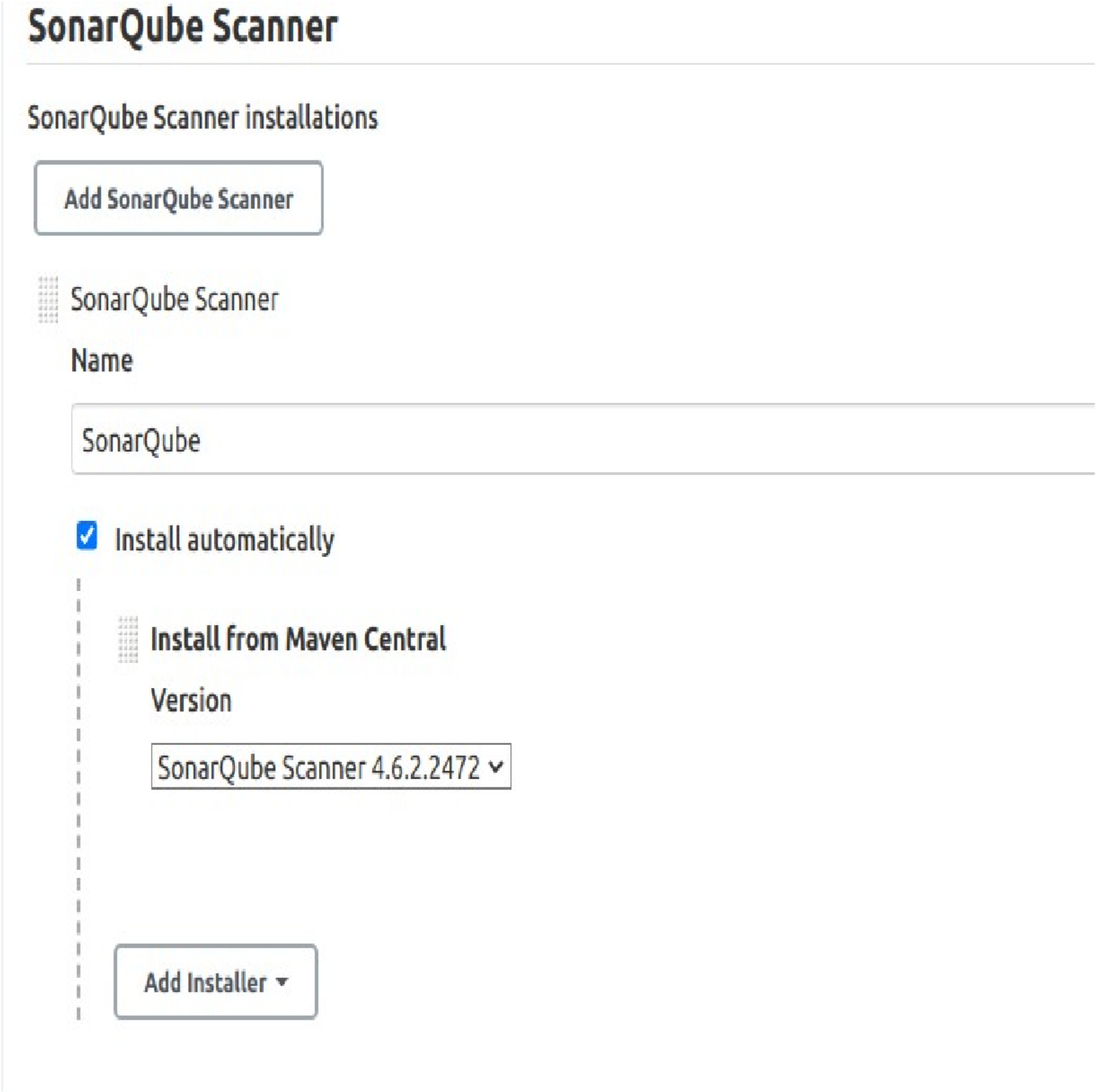
Default is http://localhost:9000

Server authentication token

sonarqube

SonarQube authentication token. Mandatory when anonymous access is disabled.

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**. Then, Click **Add SonarQube Scanner Button**. From there, give some name of the scanner type and **Add Installer** of your choice. In this case, I have selected SonarQube Scanner from Maven Central.



SonarQube Scanner

SonarQube Scanner installations

SonarQube Scanner

Name: SonarQube

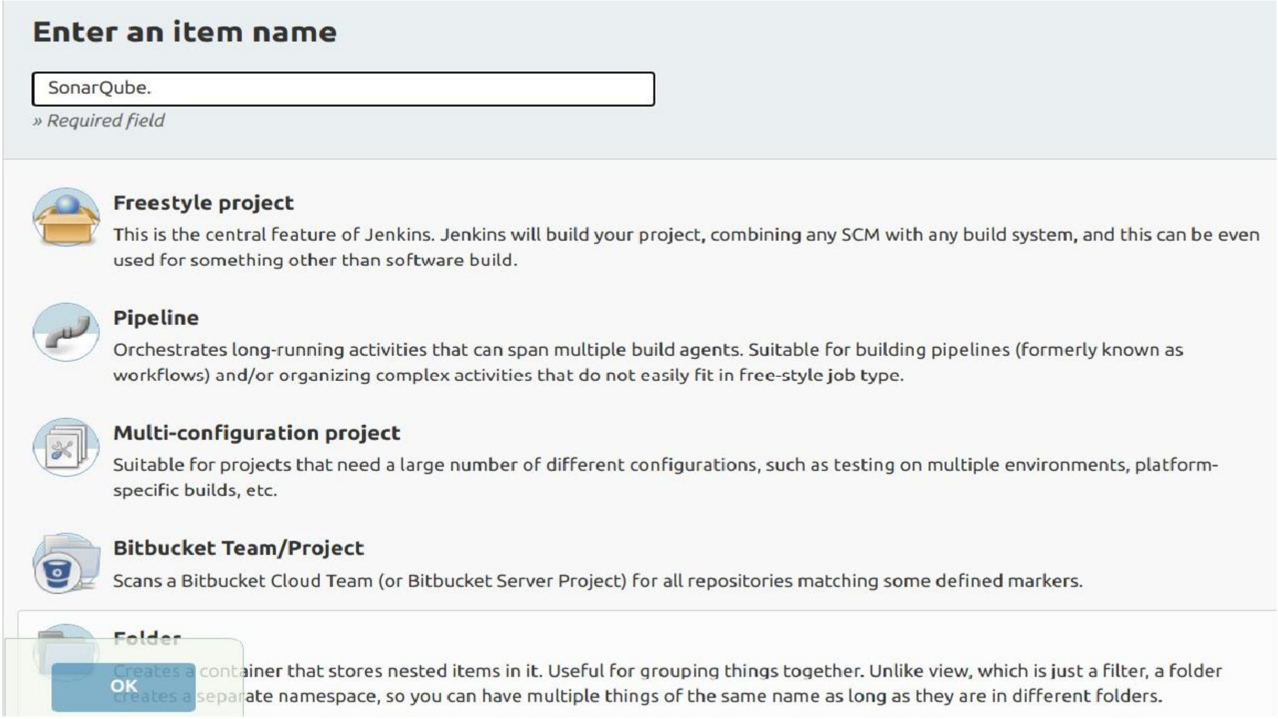
☑ Install automatically

Install from Maven Central

Version: SonarQube Scanner 4.6.2.2472

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.



Enter an item name

SonarQube.

» *Required field*

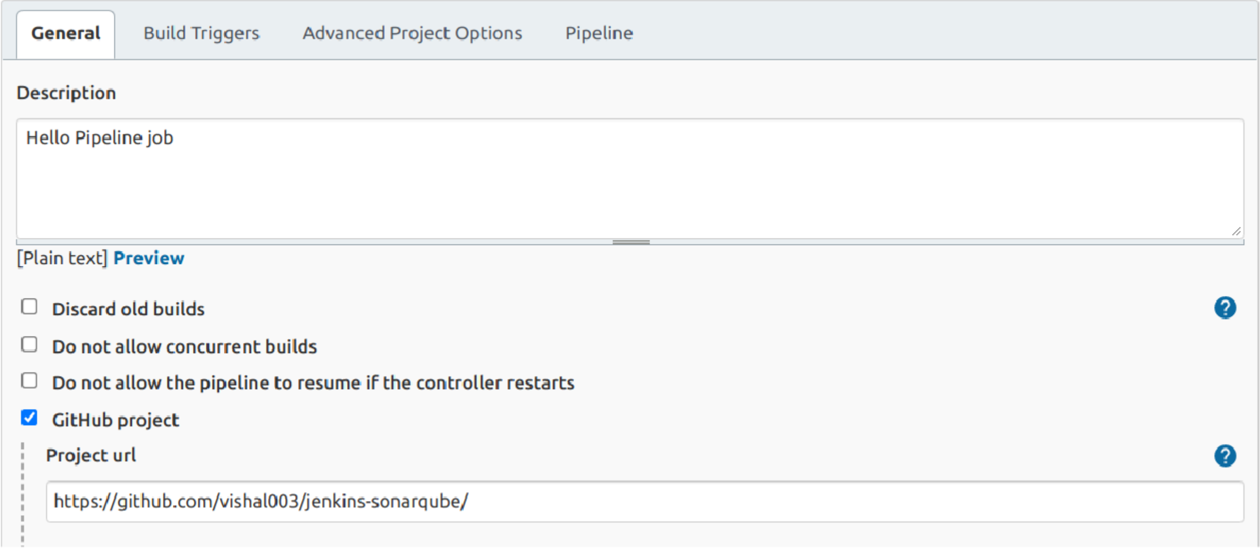
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Bitbucket Team/Project
Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Description

Hello Pipeline job

[Plain text] Preview

☐ Discard old builds

☐ Do not allow concurrent builds

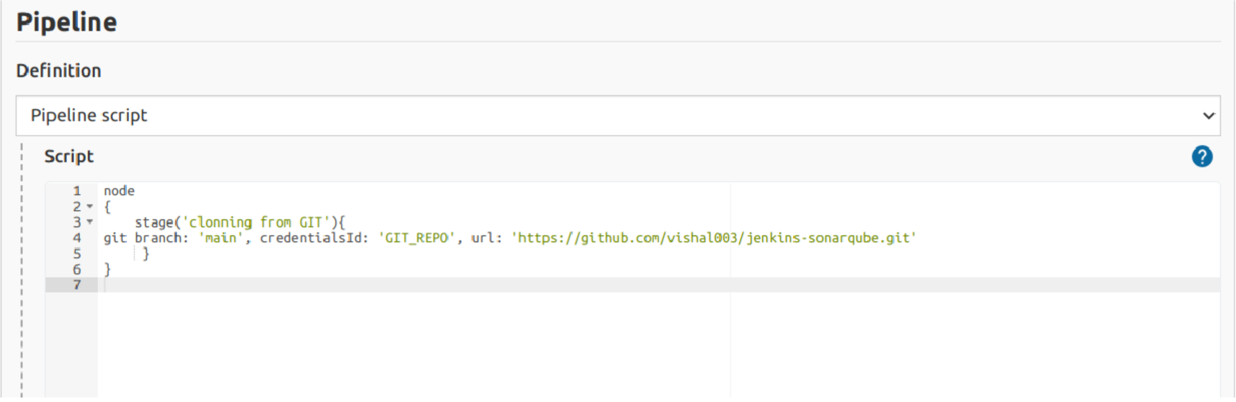
☐ Do not allow the pipeline to resume if the controller restarts

☑ GitHub project

Project url

https://github.com/vishal003/jenkins-sonarqube/

Github Configuration in Jenkins Pipeline



Pipeline

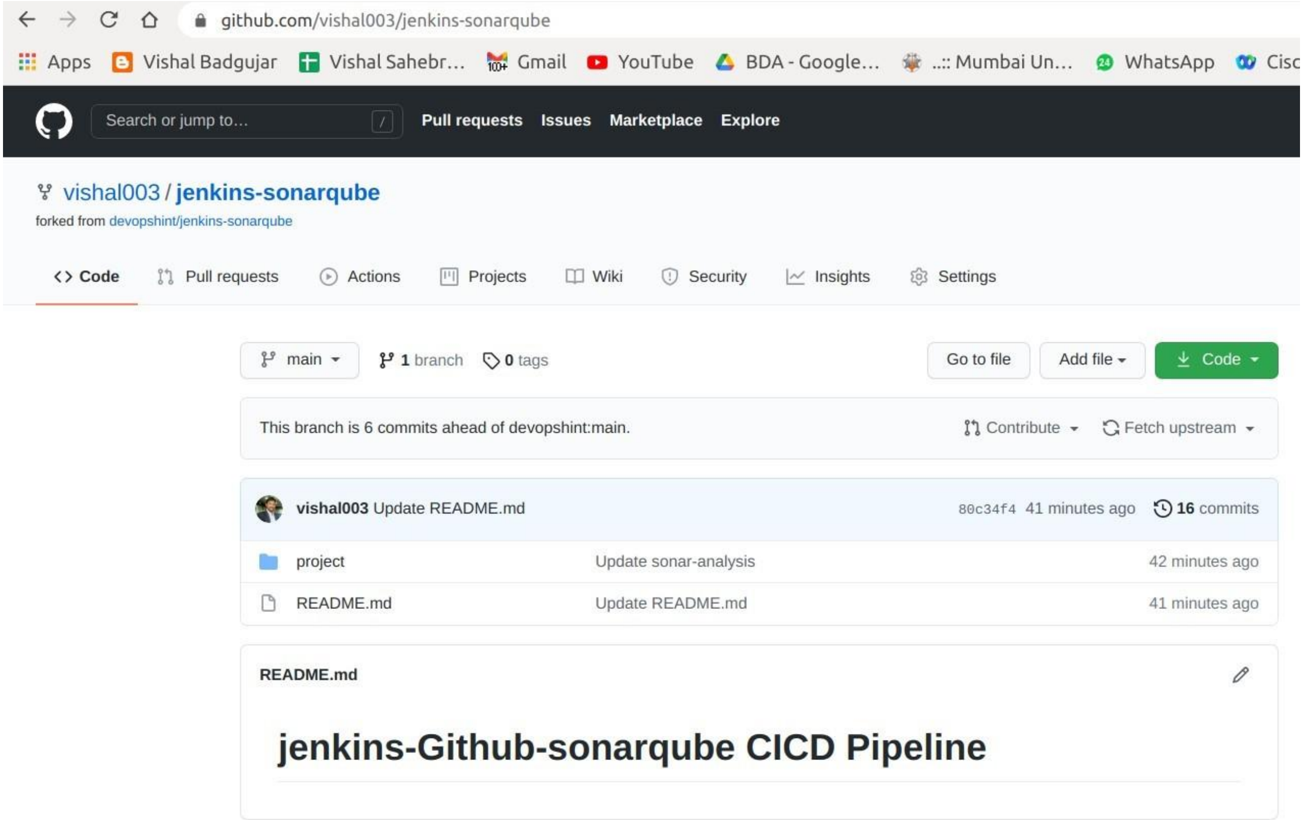
Definition

Pipeline script

Script

```
node
{
    stage('clonning from GIT'){
git branch: 'main', credentialsId: 'GIT_REPO', url: 'https://github.com/vishal003/jenkins-sonarqube.git'
     }
}
```

Git Clonning into Jenkins



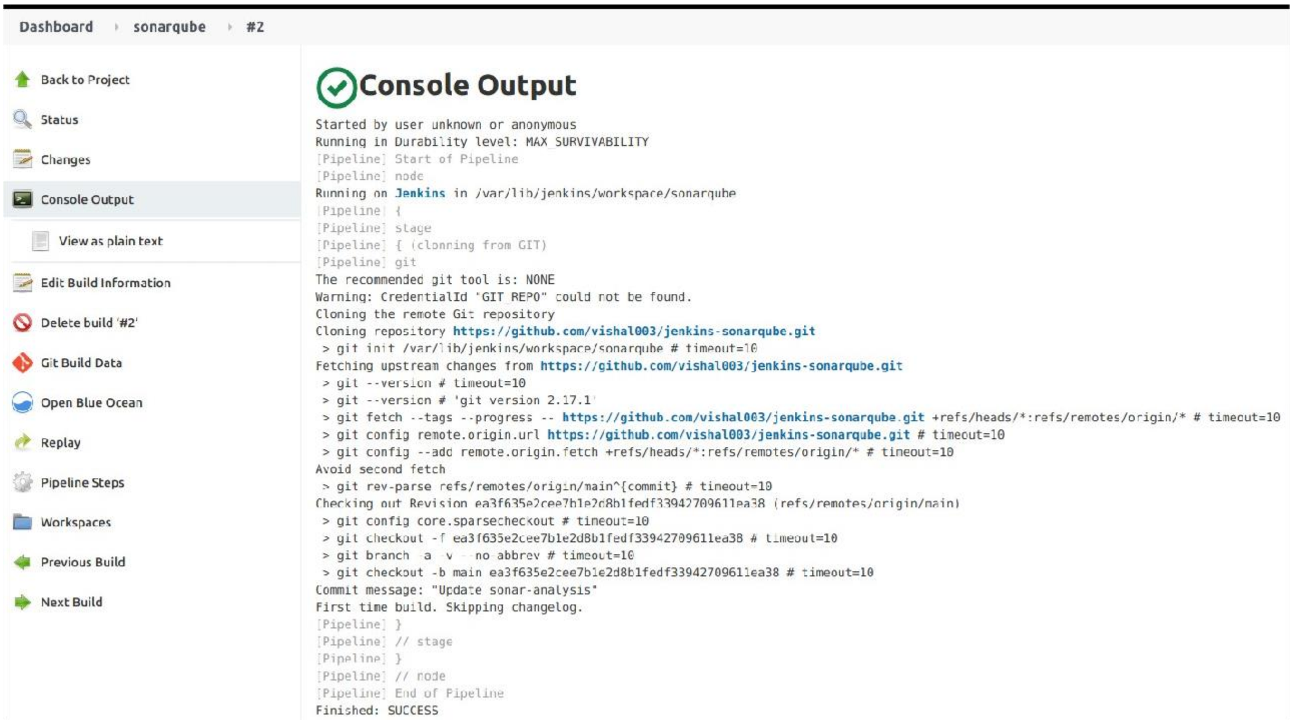
This branch is 6 commits ahead of devopshint:main.

vishal003 Update README.md 80c34f4 41 minutes ago 16 commits

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

jenkins-Github-sonarqube CICD Pipeline

Github Repository Contents



Console Output

```
Started by user unknown or anonymous
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/sonarqube
[Pipeline] {
[Pipeline] stage
[Pipeline] { (clonning 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/sonarqube # 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 +refs/heads/*:refs/remotes/origin/* # 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 ea3f635e2cee7b1e2d8b1fedf33942709611ea38 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f ea3f635e2cee7b1e2d8b1fedf33942709611ea38 # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
 > git checkout -b main ea3f635e2cee7b1e2d8b1fedf33942709611ea38 # timeout=10
Commit message: "Update sonar-analysis"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Successfully Build Github Repository in Jenkins

Pre-requiste required for Integration settings of Jenkins SAST with SonarQube we have done here successfully, now in order to Integrate of Jenkins CICD with SonarQube with the help of sample JAVA program we will implement in next experiment.

Conclusion: Write your own findings.