

**Academic Year: 2022-23 Semester: V Class / Branch: TE IT**

**Subject: Advanced Devops Lab (ADL)**

**Subject Lab Incharge: Prof. Manjusha Kashilkar**

**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

**Department of Information Technology | APSIT**

****

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.

**manjusha@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:

**manjusha@apsit:~$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >**

**/etc/apt/sources.list.d/jenkins.list'**

**Department of Information Technology | APSIT**

****

When both of these are in place, run update so that aptwill use the new repository: **manjusha@apsit:~$ sudo 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**

**8080Setting 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:

**Department of Information Technology | APSIT**

****

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

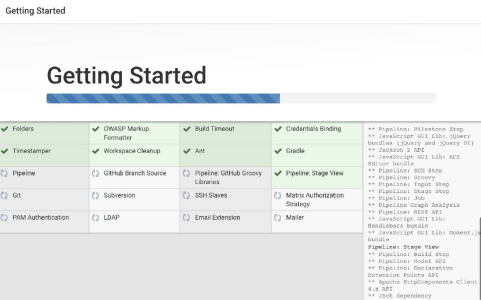
**manjusha@apsit:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

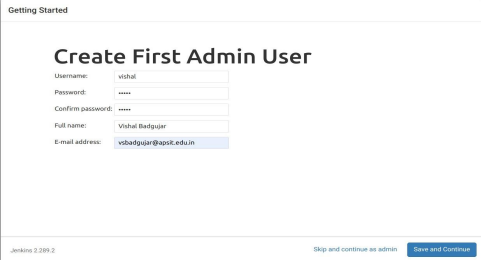
Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

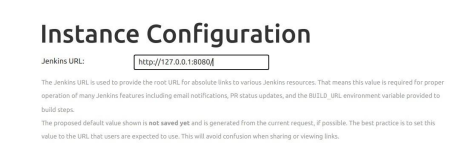
The next screen presents the option of installing suggested plugins or selecting specific plugins:**Department of Information Technology | APSIT**

****

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

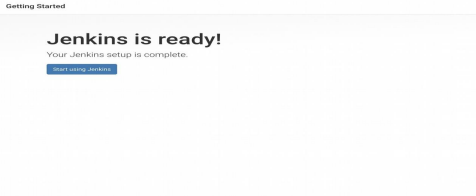
**Department of Information Technology | APSIT**

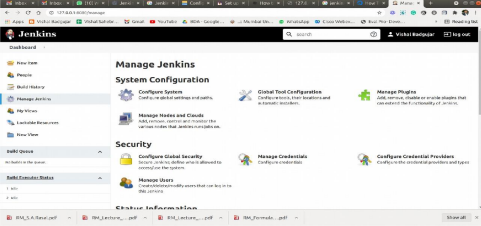
****When the installation is complete, you will be prompted to set up the first administrative user. It’s possible to skip this step and continue as admin using the initial password we used above, but we’ll take a moment to create the user.



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

**Department of Information Technology | APSIT**

****Click Start using Jenkins to visit the main Jenkins dashboard:

**Department of Information Technology | APSIT**

****

**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**

****

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).



**Department of Information Technology | APSIT**

****

**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**

**Department of Information Technology | APSIT**

****

**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.



**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.

**Department of Information Technology | APSIT**

****

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.

**Department of Information Technology | APSIT**

**Department of Information Technology | APSIT**

****

**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.

**Department of Information Technology | APSIT**

**Github Configuration in Jenkins Pipeline**

**Git Clonning into Jenkins**

**Department of Information Technology | APSIT**

**Github Repository Contents**

**Department of Information Technology | APSIT**

****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.**

**Department of Information Technology | APSIT**