Zomato Clone App with DevSecOps CI/CD

Capstone Project

Personal Information

Full Name:	Stanley A
Email Address:	antony.stanley@cprime.com
Phone Number:	9941127755
Location:	Chennai, TamilNadu

Professional Information

Company Name:	CPrime
Job Title:	Apprentice

Github URL

https://github.com/stanleycprime/Zomato-Clone-App-with-DevSecOps-CI-CD

Description

This project implements a DevSecOps CI/CD pipeline for a Zomato Clone App using Jenkins, Docker, SonarQube, Trivy, and OWASP Dependency Check to automate code scanning, containerization, and deployment on an AWS EC2 instance.

Resources

https://mrcloudbook.com/zomato-clone-app-with-devsecops-ci-cd/

Zomato Clone App with DevSecOps CI/CD

Overview

This DevSecOps project implements a secure CI/CD pipeline for a **Zomato Clone Node.js app** on an **Ubuntu 22.04 AWS EC2 instance**. It uses **Jenkins** for orchestration, **SonarQube** and **OWASP Dependency-Check** for static code analysis, and **Trivy** for container vulnerability scanning. Docker is used to containerize the application, which is then pushed to **DockerHub** and deployed locally on the instance. Jenkins automates the entire process from code checkout to deployment. The setup ensures early detection of security issues and code quality enforcement. Finally, the EC2 instance is terminated to optimize resource usage and cost.

Tools and Services

1. GitHub

GitHub is a cloud-based platform for version control and collaboration, built around Git. It allows developers to host and manage code repositories, track changes, and collaborate on projects. In this project, GitHub is used to store and retrieve the Zomato Clone source code, enabling CI/CD automation.

---

2. Node.js

Node.js is a JavaScript runtime built on Chrome's V8 engine, used to run JavaScript code outside the browser. It powers server-side applications and supports scalable development. In your project, Node.js runs the backend of the Zomato Clone app and manages dependencies using npm.

---

3. Amazon EC2 (Elastic Compute Cloud)

EC2 is a web service from AWS that provides resizable compute capacity in the cloud. It allows you to launch virtual machines (instances) for hosting applications. Here, an Ubuntu EC2 instance is used as the base server for setting up Jenkins, Docker, SonarQube, and the deployment environment.

---

4. PuTTY & Pageant

PuTTY is a free SSH and telnet client for Windows used to remotely access servers. Pageant is its companion tool that holds private SSH keys. In your setup, PuTTY connects to the EC2 instance, while Pageant manages your SSH key for secure authentication.

---

5. Jenkins

Jenkins is an open-source automation server used to automate parts of software development related to building, testing, and deploying. It supports plugins to integrate with almost any tool. In this project, Jenkins orchestrates the CI/CD pipeline, running scans, builds, and deployments automatically.

---

6. Jenkins Plugins

Jenkins plugins extend the platform's capabilities. Plugins like **NodeJS**, **JDK**, **SonarQube Scanner**, **Docker Pipeline**, and **OWASP Dependency Check** are added to enable integrations with tools such as SonarQube, Docker, and static analysis scanners, making Jenkins a powerful DevSecOps orchestrator.

---

7. Docker

Docker is a containerization platform that allows packaging applications with their dependencies into isolated units (containers). It simplifies deployment and scalability. In your pipeline, Docker is used to run SonarQube, build Docker images of the app, and deploy containers from those images.

---

8. SonarQube

SonarQube is an open-source platform for continuous inspection of code quality. It performs automatic reviews with static code analysis to detect bugs, vulnerabilities, and code smells. In this pipeline, it is used to ensure code quality of the Node.js app before build and deployment stages.

---

9. Trivy

Trivy is a simple and comprehensive vulnerability scanner for containers and other artifacts. It checks for OS vulnerabilities and misconfigurations. Here, Trivy is integrated into the Jenkins pipeline to scan Docker images, adding a security layer before pushing images to Docker Hub.

---

10. OWASP Dependency-Check

OWASP Dependency-Check is a software composition analysis (SCA) tool that identifies known vulnerabilities in project dependencies. It checks for CVEs in third-party libraries. In this project, it's integrated into Jenkins to scan the application’s node_modules and display detailed security reports in the pipeline.

---

11. SonarQube Webhooks

Webhooks in SonarQube enable integration with external systems by sending build status updates. In this project, a webhook sends code analysis results back to Jenkins, allowing the pipeline to decide whether to proceed based on the SonarQube Quality Gate results.

---

12. DockerHub

DockerHub is a cloud-based container registry that lets you store and distribute Docker images. It integrates with Jenkins to push container images after a successful build. The Zomato Docker image is pushed here, enabling easy deployment or sharing.

---

13. Jenkins Pipeline

Jenkins Pipeline is a suite of plugins that supports integrating and implementing continuous delivery pipelines as code. You define your build, test, and deploy steps in a Jenkinsfile. In this setup, the Zomato Clone CI/CD process is fully automated through a scripted pipeline.

---

14. Jenkins Credentials

Jenkins securely stores credentials (tokens, passwords, SSH keys) used by pipelines. You use it to store the SonarQube token and DockerHub credentials. These credentials are referenced in the pipeline to authenticate with SonarQube and DockerHub during analysis and image push stages.

---

15. Quality Gates (SonarQube)

A Quality Gate is a set of conditions SonarQube uses to determine whether a project passes or fails the quality check. It can block builds with critical vulnerabilities or low test coverage. You used the default Quality Gate to ensure code quality before deployment.

---

16. Jenkins Plugin - Pipeline Stage View

The Pipeline Stage View plugin provides a visual representation of stages in a Jenkins pipeline. It shows stage-by-stage status of the pipeline including success, failure, or skipped steps. This visual feedback helps quickly identify issues in complex pipelines like yours.

---

17. Jenkins Tool Configuration

Jenkins Tool Configuration allows you to define versions of tools like JDK, Node.js, SonarQube Scanner globally. You configured JDK 17 and Node 16 for consistent builds across jobs. This ensures compatibility between your code and the environment it runs in.

---

18. Ubuntu (on EC2)

Ubuntu is a widely used, Debian-based Linux OS known for its stability and community support. In this project, Ubuntu is the operating system of the EC2 instance, chosen for its ease of package management and compatibility with Jenkins, Docker, and DevSecOps tools.

---

19. Security Groups (AWS EC2)

Security Groups in AWS act like virtual firewalls that control inbound and outbound traffic for EC2 instances. You configure rules to allow specific ports (3000, 3001, 8080, 9000) for the app, Jenkins, and SonarQube, ensuring secure and controlled network access.

---

20. Terminating Instances (AWS EC2)

Terminating an EC2 instance permanently deletes the virtual machine and stops billing. It's a best practice to terminate unused instances, especially large ones (like t2.large) after project completion to save on AWS costs.

Workflow

Step 1 — Launch an Ubuntu (22.04) T2 Large EC2 Instance

An AWS EC2 instance with **Ubuntu 22.04 (t2.large)** is launched to serve as the CI/CD host. It offers enough CPU and memory to run Jenkins, Docker, and SonarQube simultaneously. Security groups are configured to allow SSH, HTTP and HTTPS.

Step 2 — Install Jenkins, Docker, and Trivy. Create SonarQube Container Using Docker

Jenkins is installed for automation, Docker for containerization, and Trivy to scan images for vulnerabilities. A SonarQube container is created using Docker, exposing it on port 9000. This setup ensures quick deployment and avoids manual installation overhead for SonarQube.

Step 3 — Install Jenkins Plugins (JDK, Node.js, SonarQube Scanner, OWASP Dependency Check)

Jenkins plugins for **Java (JDK 17)**, **Node.js**, **SonarQube Scanner**, and **OWASP Dependency-Check** are installed and configured. These tools enable Jenkins to run Java-based scanners, execute Node.js apps, and analyze the codebase for vulnerabilities and quality issues during pipeline execution.

Step 4 — Create a Pipeline Project in Jenkins Using a Declarative Pipeline

A new Jenkins pipeline project is created using a **Declarative Pipeline syntax** in a Jenkinsfile. This file defines all CI/CD steps like code checkout, static analysis, Trivy scan, Docker build, push, and deployment. It ensures the entire workflow is version-controlled and automated.

Step 5 — Install and Configure OWASP Dependency Check

The **OWASP Dependency-Check plugin** is configured in Jenkins to scan the project's Node.js dependencies for known vulnerabilities. It generates an HTML report and fails the pipeline if high-severity vulnerabilities are detected, enforcing dependency-level security checks.

Step 6 — Docker Image Build and Push

Jenkins builds a Docker image of the Zomato Clone app and tags it with the appropriate version. After a successful build and vulnerability scan with Trivy, the image is **pushed to DockerHub** using credentials securely stored in Jenkins.

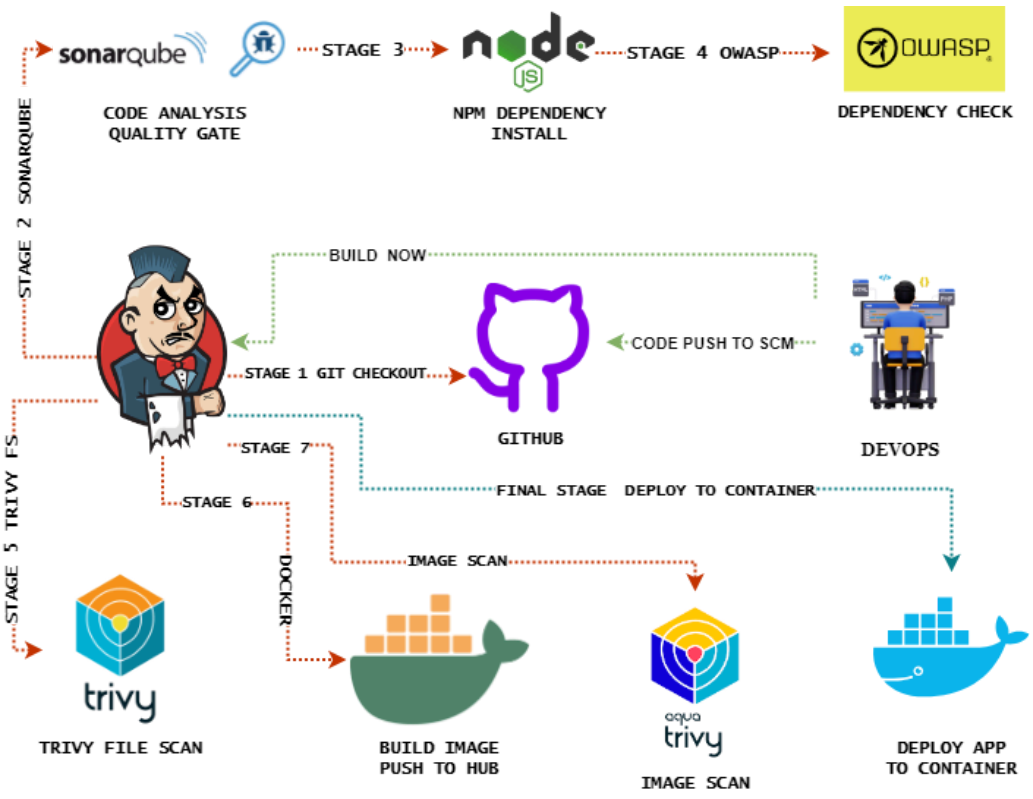
Step 7 — Deploy the Image Using Docker

The pushed Docker image is pulled and deployed on the EC2 instance using Docker. This step validates that the container runs correctly and exposes the Node.js app on ports like 3000 or 3001. This marks the end-to-end DevSecOps deployment cycle.

Step 8 — Terminate the AWS EC2 Instance

After verifying successful deployment, the EC2 instance is **terminated** to stop billing and release resources. This is a best practice to avoid unnecessary charges, especially after running compute-intensive tools like SonarQube on t2.large instances.

Architecture Diagram



Pre Check the Code

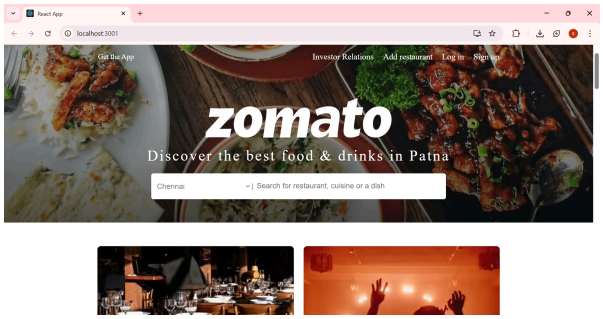
1. Get the code from Github:
   https://github.com/stanleycprime/Zomato-Clone
2. Install the node modules for the Source Code
   npm install

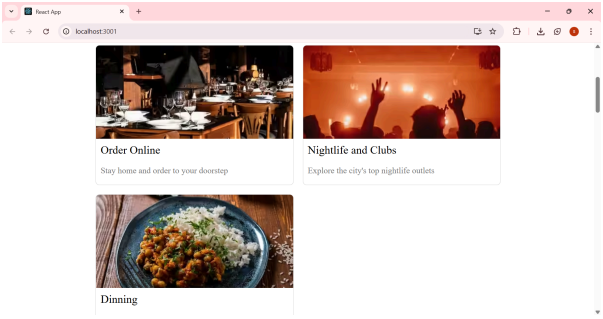
```
C:\Users\A Stanley\Downloads\Zomato-Clone-main\Zomato-Clone-main>npm install
npm warn deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the
compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compa
tibility
npm warn deprecated rollup-plugin-terser@7.0.2: This package has been deprecated and is no longer maintained. Please use @rollup/plug
in-terser
npm warn deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sourcemap-codec instead
npm warn deprecated w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
npm warn deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

added 1506 packages, and audited 1507 packages in 4m

235 packages are looking for funding
  run `npm fund` for details

35 vulnerabilities (3 low, 13 moderate, 17 high, 2 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

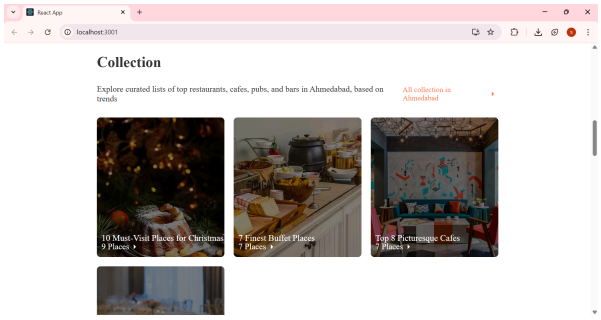
3. Now start running the application
   npm start

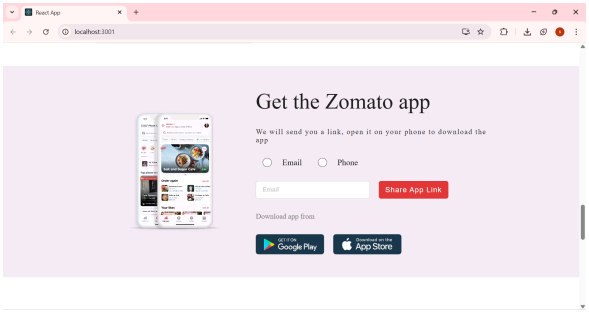
```
C:\Users\A Stanley\Downloads\Zomato-Clone-main\Zomato-Clone-main>npm start

> zomato_clone@0.1.0 start
> react-scripts start
√ Something is already running on port 3000.

Would you like to run the app on another port instead? ... yes
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
(node:15080) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Pl
ease use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:15080) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated.
Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view zomato_clone in the browser.

  Local:            http://localhost:3001
  On Your Network:  http://172.26.0.1:3001

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

4. Open the browser port as mentioned (3001). Here we run as localhost. http://localhost:3001/





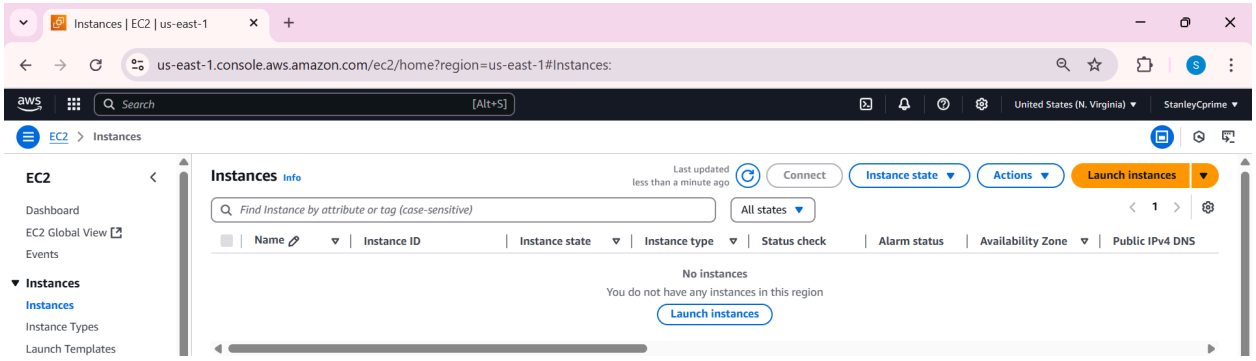




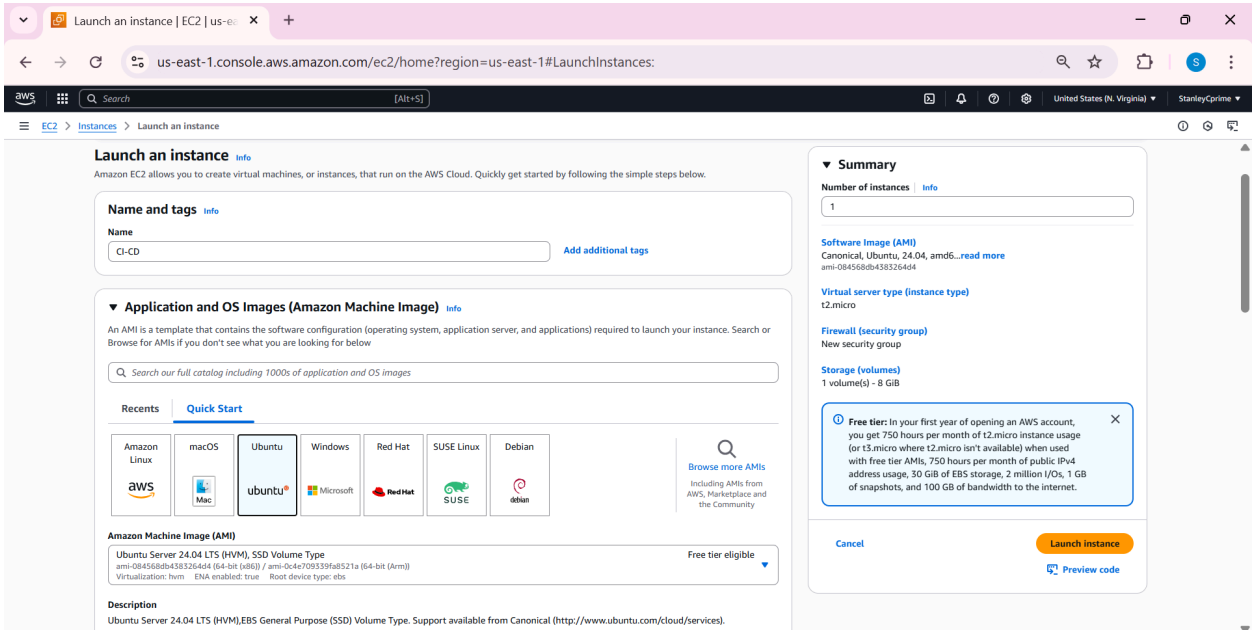
Step by Step Implementation

Step 1 : Launch an Ubuntu - t2.large Instance

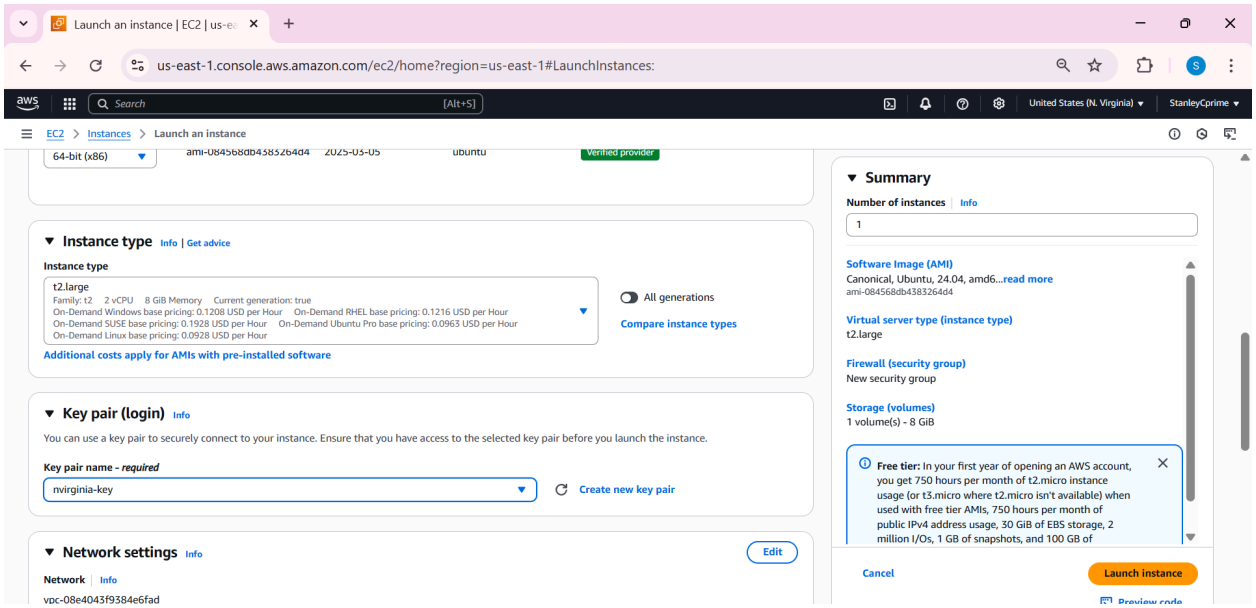
1. Launch an AWS instance by going to AWS EC2 services
2. There click Launch Instance on top right corner



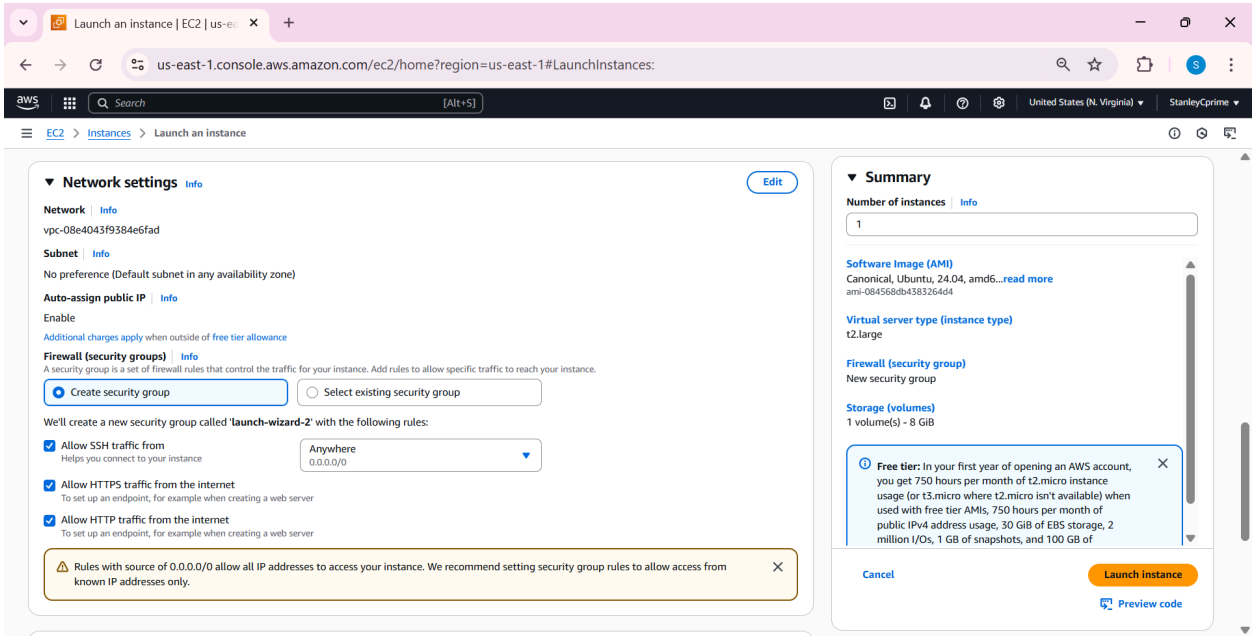
3. Give CI-CD as Instance name and choose Ubuntu OS



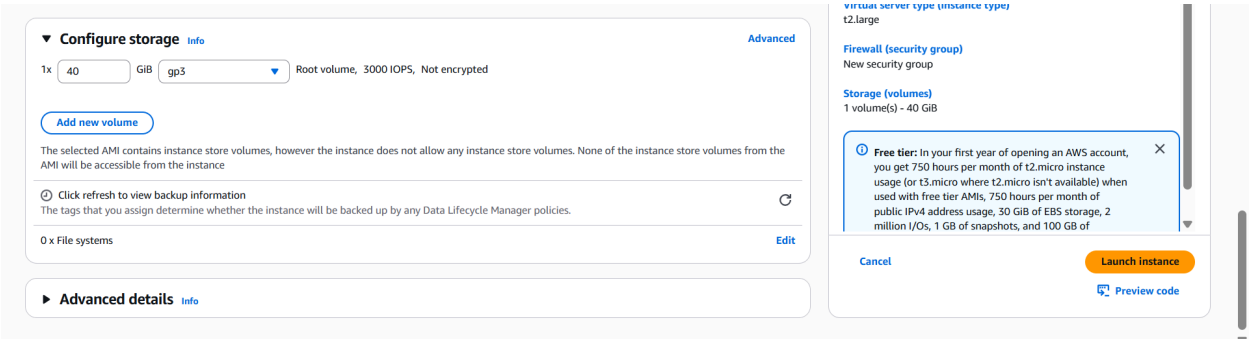
4. Select t2.large as Instance type
5. Choose the already existing key pair (nvirginia-key)



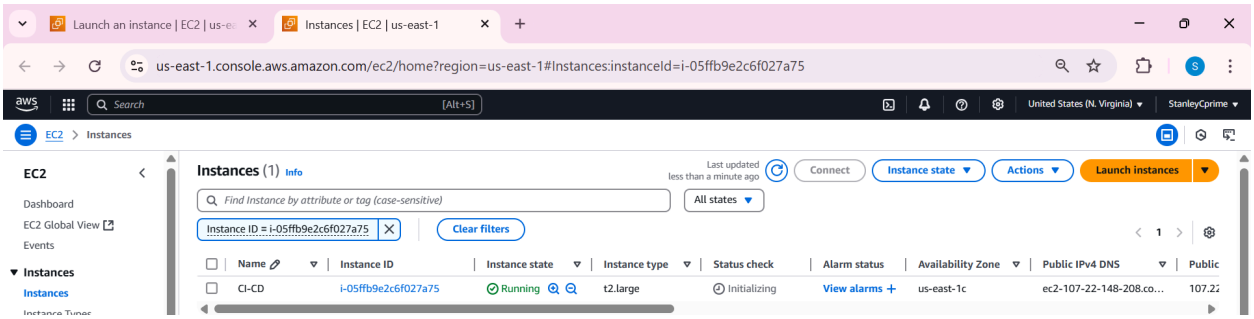
6. Choose default VPC
7. Create a new Security Group (that allows HTTP and HTTPS) (don't allow any unnecessary ports)



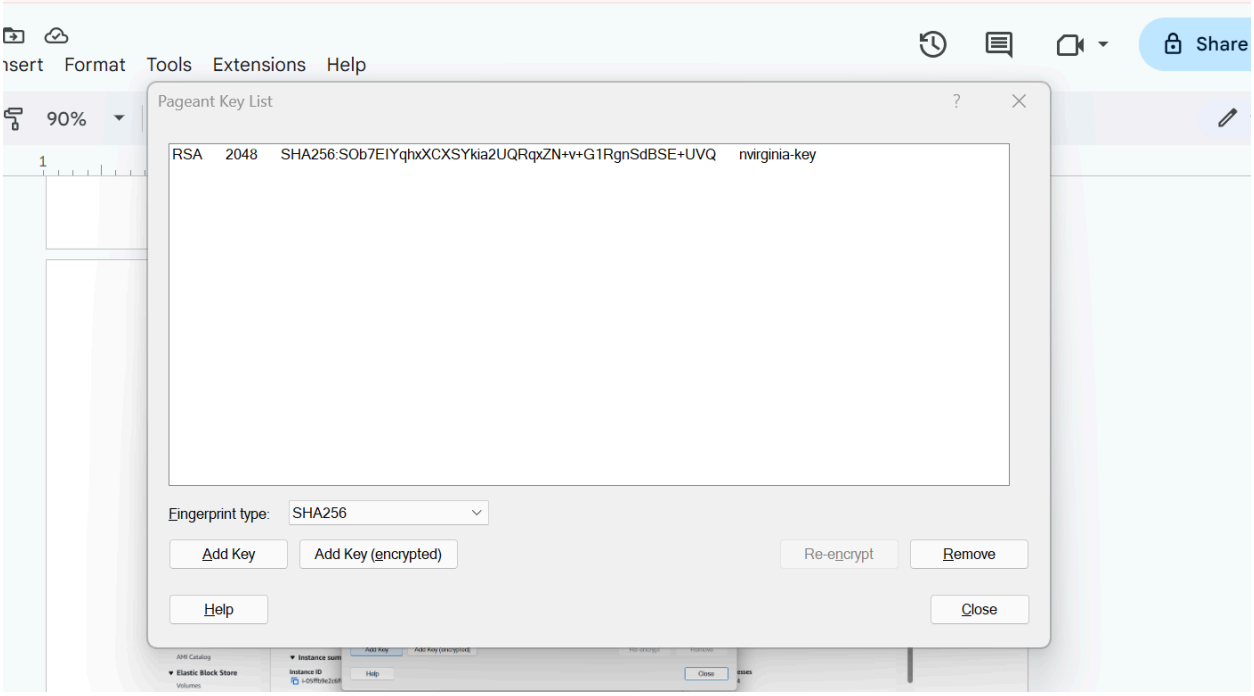
8. Choose storage as 40 GB



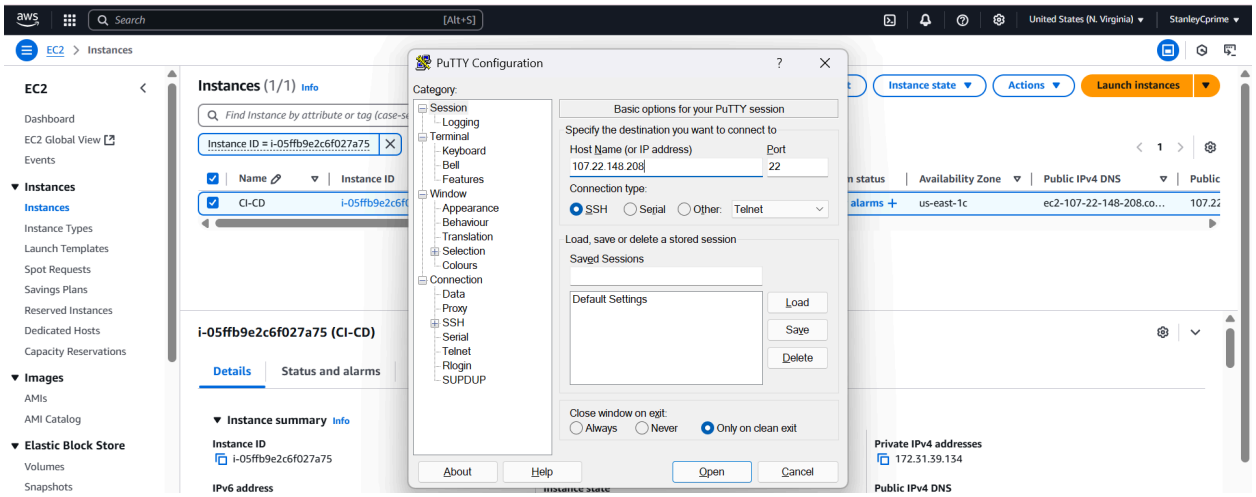
9. Then click Launch Instance



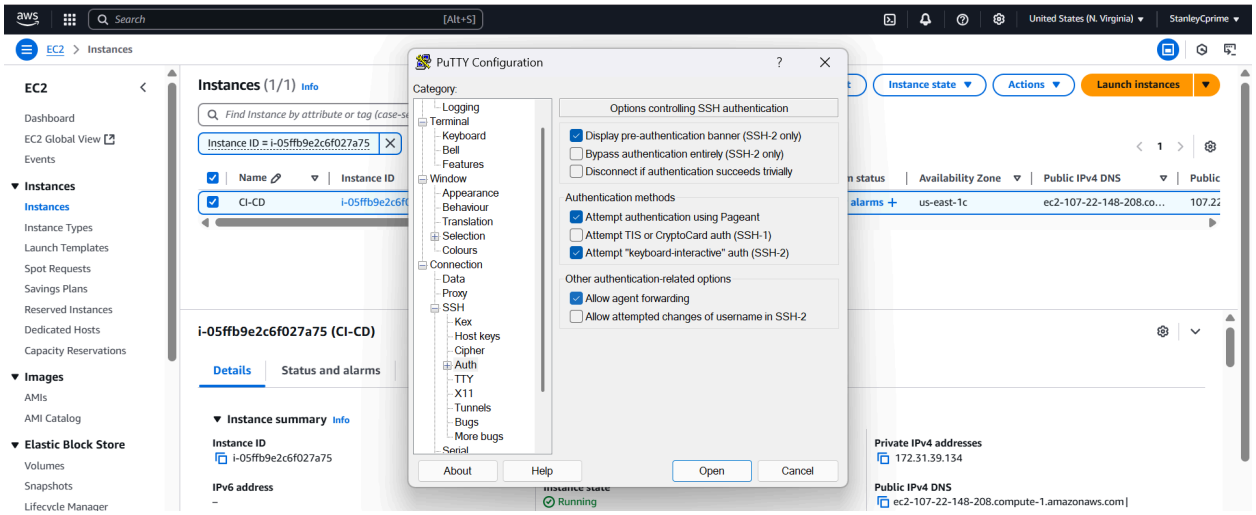
10. Add our instance key in Pageant (nvirginia-key)



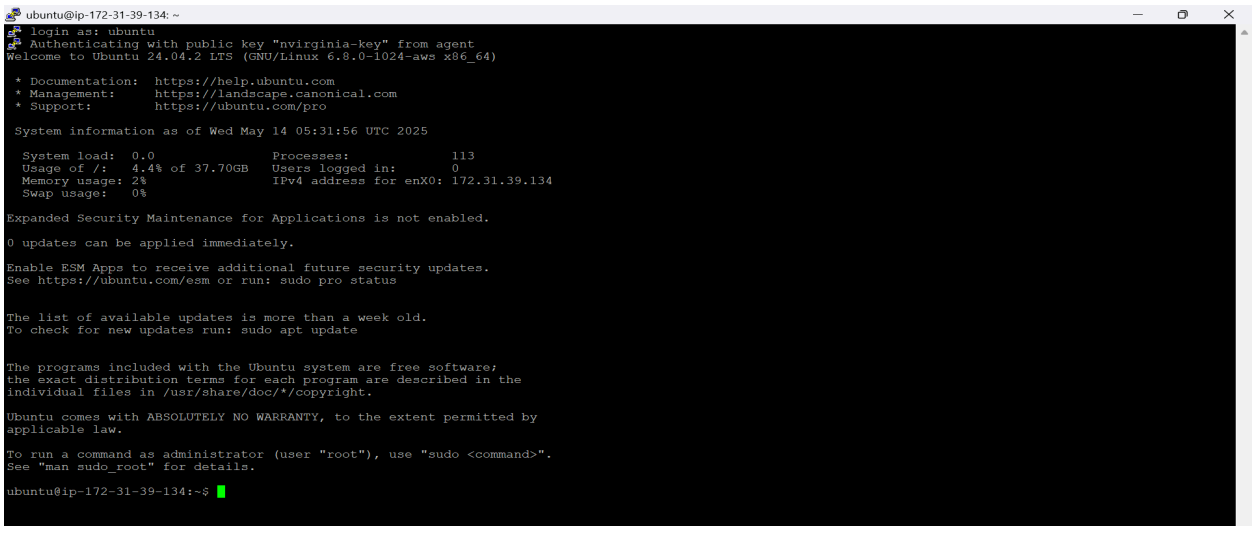
11. Now open Putty and add the IP address of our instance



12. In SSh - Auth - allow Agent Port Forwarding



13. Now type ubuntu in login as user
14. With this our instance is set and connected with putty.

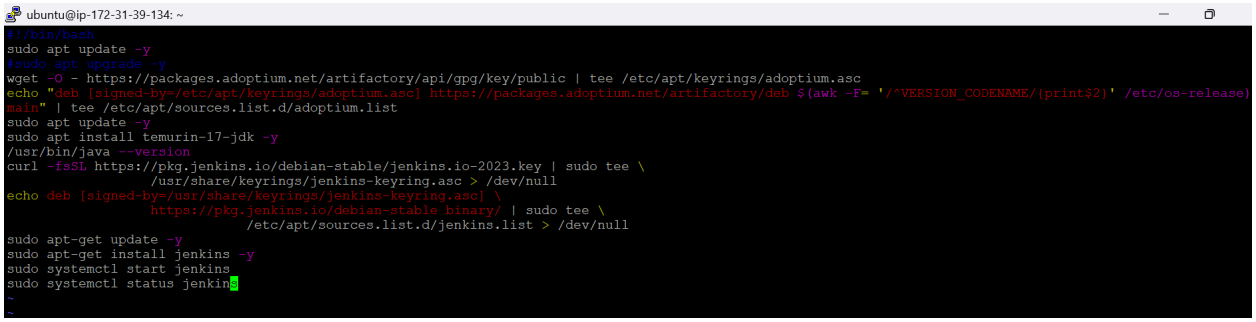


Step 2 : Install Jenkins, Docker and Trivy

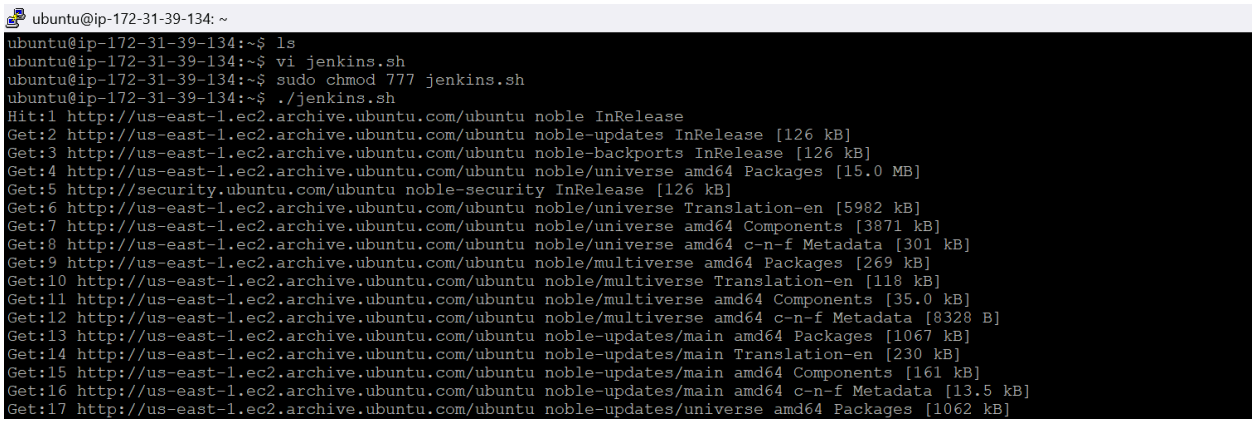
2A : To Install Jenkins

1. Now create a new file in Instance to run and install the dependencies needed for the Jenkins to run

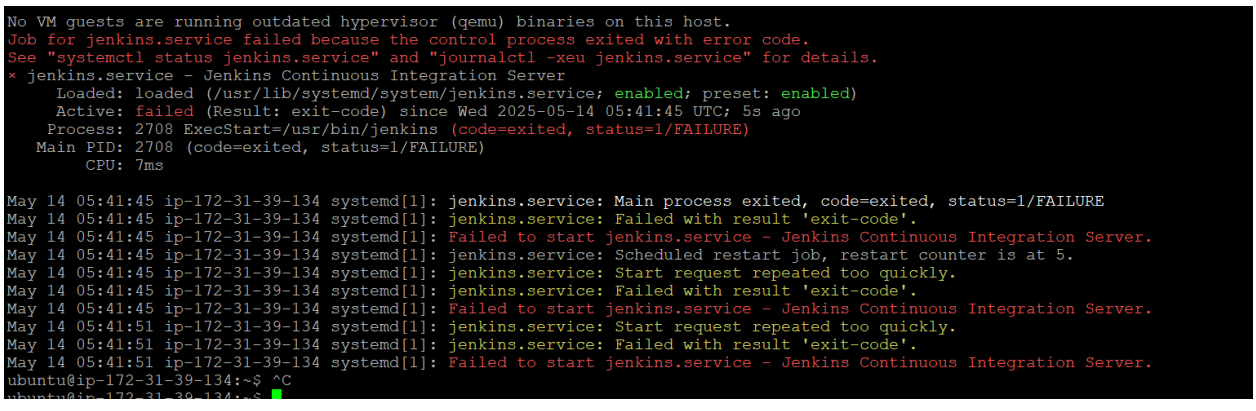
2. Add this code inside the jenkins.sh



3. Now give executable permissions to the file (we give 777 permission)
4. Now run the script and download the packages



Error:



This is because Java is not installed properly

```
ubuntu@ip-172-31-39-134:~$ java -version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless  # version 17.0.14+7-1~24.04, or
sudo apt install openjdk-21-jre-headless  # version 21.0.6+7-1~24.04.1
sudo apt install default-jre              # version 2:1.17-75
sudo apt install openjdk-11-jre-headless  # version 11.0.26+4-1ubuntu1~24.04
sudo apt install openjdk-8-jre-headless   # version 8u442-b06~us1-0ubuntu1~24.04
sudo apt install openjdk-19-jre-headless  # version 19.0.2+7-4
sudo apt install openjdk-20-jre-headless  # version 20.0.2+9-1
sudo apt install openjdk-22-jre-headless  # version 22~22ea-1
ubuntu@ip-172-31-39-134:~$ java --version
Command 'java' not found, but can be installed with:
sudo apt install openjdk-17-jre-headless  # version 17.0.14+7-1~24.04, or
sudo apt install openjdk-21-jre-headless  # version 21.0.6+7-1~24.04.1
sudo apt install default-jre              # version 2:1.17-75
```

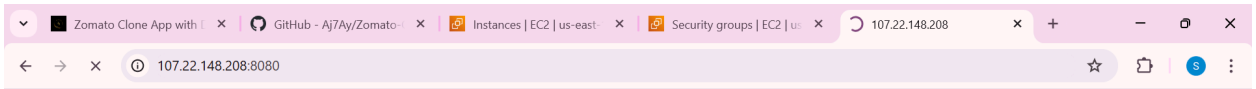
So we try changing the commands in jenkins.sh

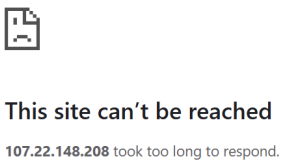
```
ubuntu@ip-172-31-39-134: ~
#!/bin/bash
sudo apt update -y
#sudo apt upgrade -y
wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | tee /etc/apt/keyrings/adoptium.asc
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc] https://packages.adoptium.net/artifactory/deb $(awk -F= '/^VERSION_CODENAME/{print$2}' /etc/os-release)
main" | tee /etc/apt/sources.list.d/adoptium.list
sudo apt update -y
sudo apt install openjdk-17-jdk -y

/usr/bin/java --version
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
               /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
               https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
                       /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
sudo systemctl start jenkins
sudo systemctl status jenkins
```

5. With this script, we also run and enable Jenkins

```
Reading state information... Done
jenkins is already the newest version (2.504.1).
0 upgraded, 0 newly installed, 0 to remove and 83 not upgraded.
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Wed 2025-05-14 05:48:11 UTC; 31ms ago
   Main PID: 6172 (java)
      Tasks: 48 (limit: 9505)
     Memory: 959.6M (peak: 963.7M)
        CPU: 14.879s
     CGroup: /system.slice/jenkins.service
             └─6172 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: Jenkins initial setup is required. An admin user has been created and a password generated.
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: Please use the following password to proceed to installation:
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: 85b0acf8237f4d5f9ddd14e3854b4ec9
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: *************************************************************
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: *************************************************************
May 14 05:48:08 ip-172-31-39-134 jenkins[6172]: *************************************************************
May 14 05:48:11 ip-172-31-39-134 jenkins[6172]: 2025-05-14 05:48:11.533+0000 [id=33]        INFO        jenkins.InitReactorRunner$1#onAttained: Completed in>
May 14 05:48:11 ip-172-31-39-134 jenkins[6172]: 2025-05-14 05:48:11.562+0000 [id=23]        INFO        hudson.lifecycle.Lifecycle#onReady: Jenkins is fully>
May 14 05:48:11 ip-172-31-39-134 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
lines 1-20/20 (END)
```

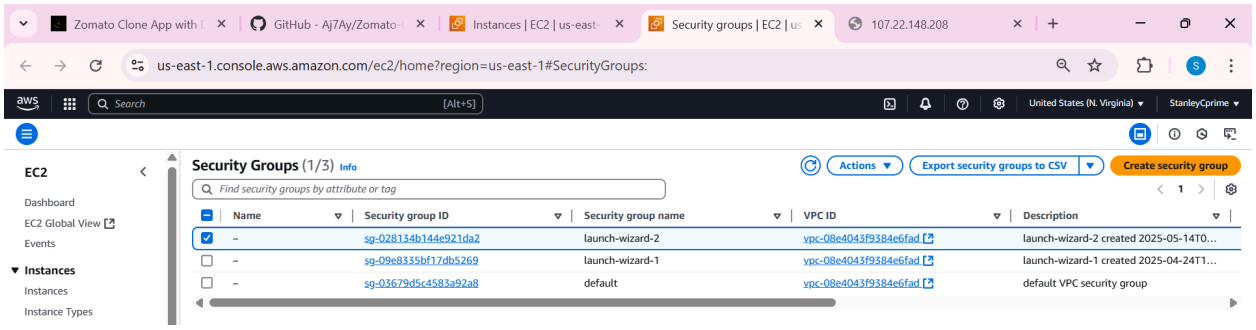
6. Open the browser and open <public-ip>:8080



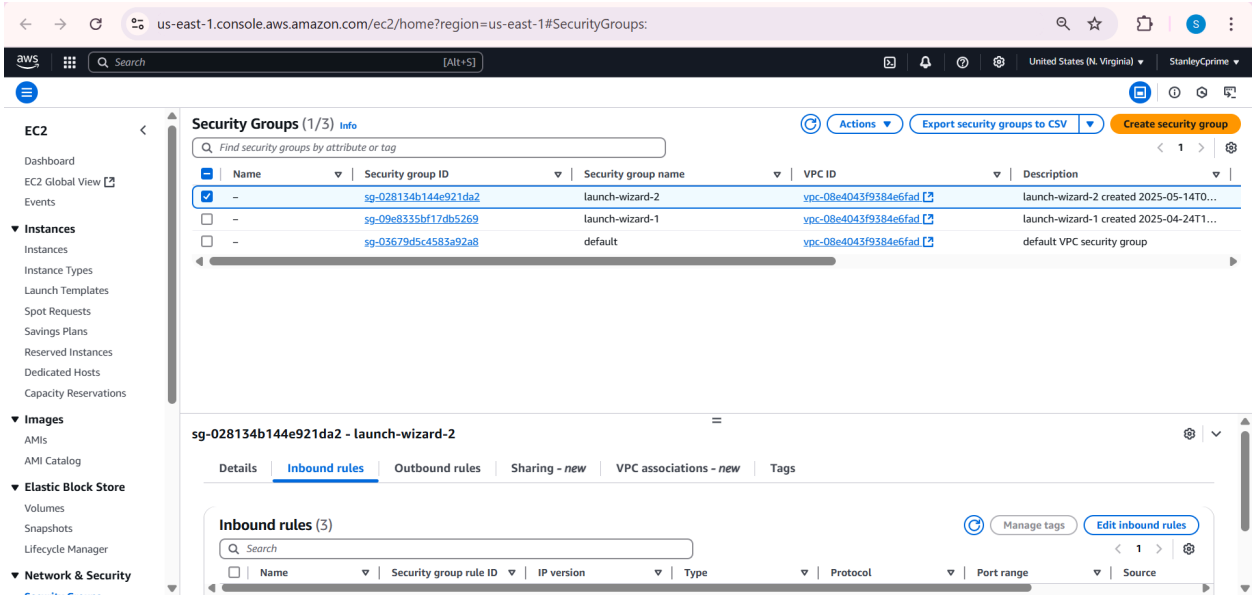


This is due to closed port in Security Group

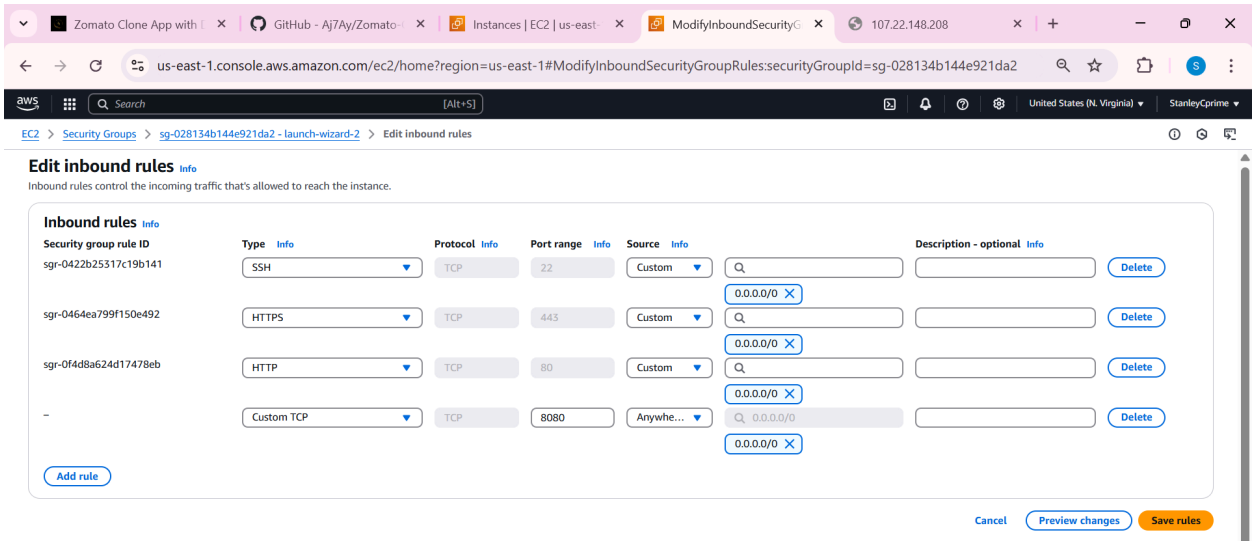
7. So we choose our Security Group



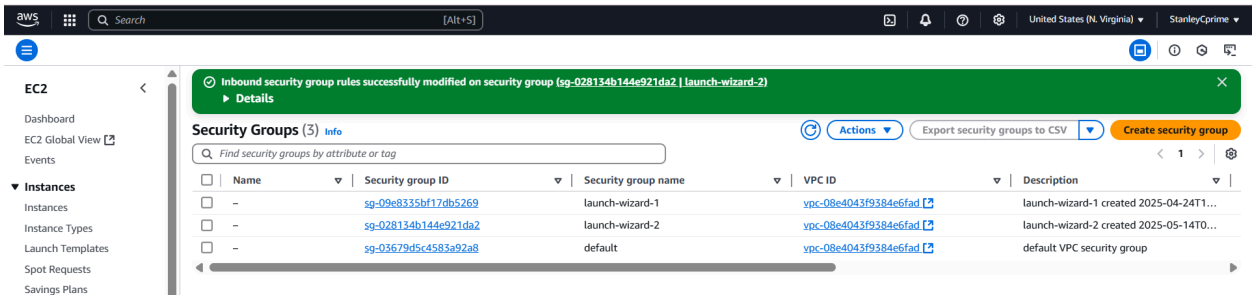
8. Select and Edit the Inbound Rules



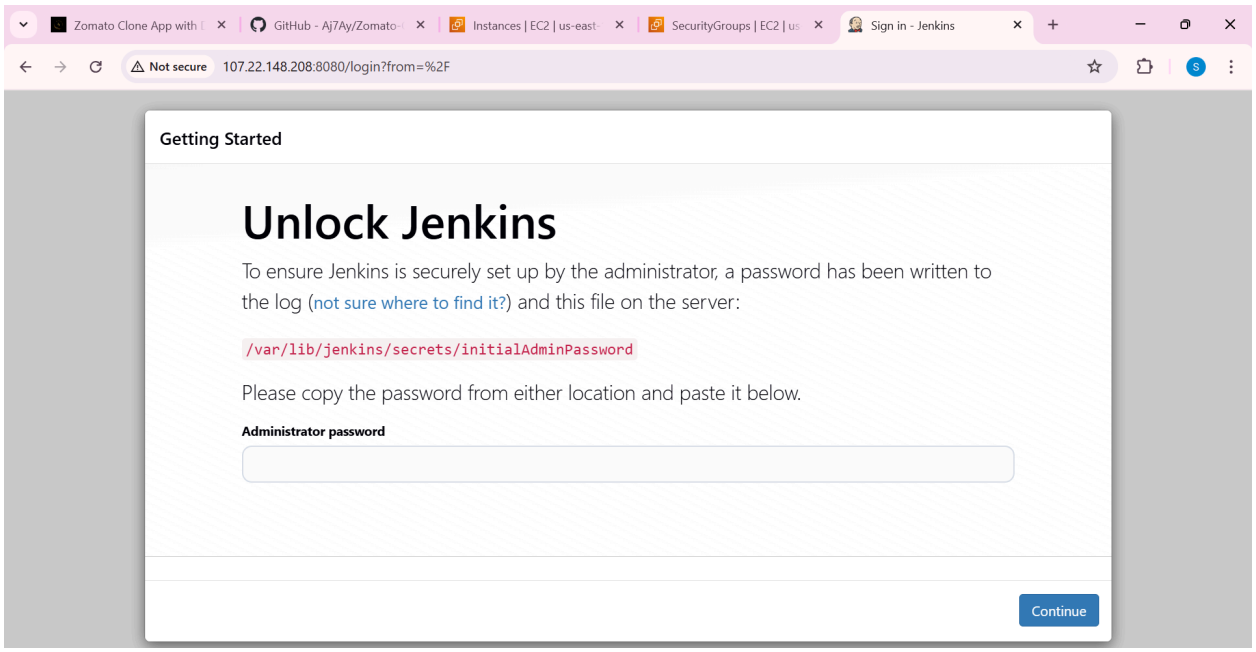
9. Add port 8080 as Custom TCP



10. Click Save



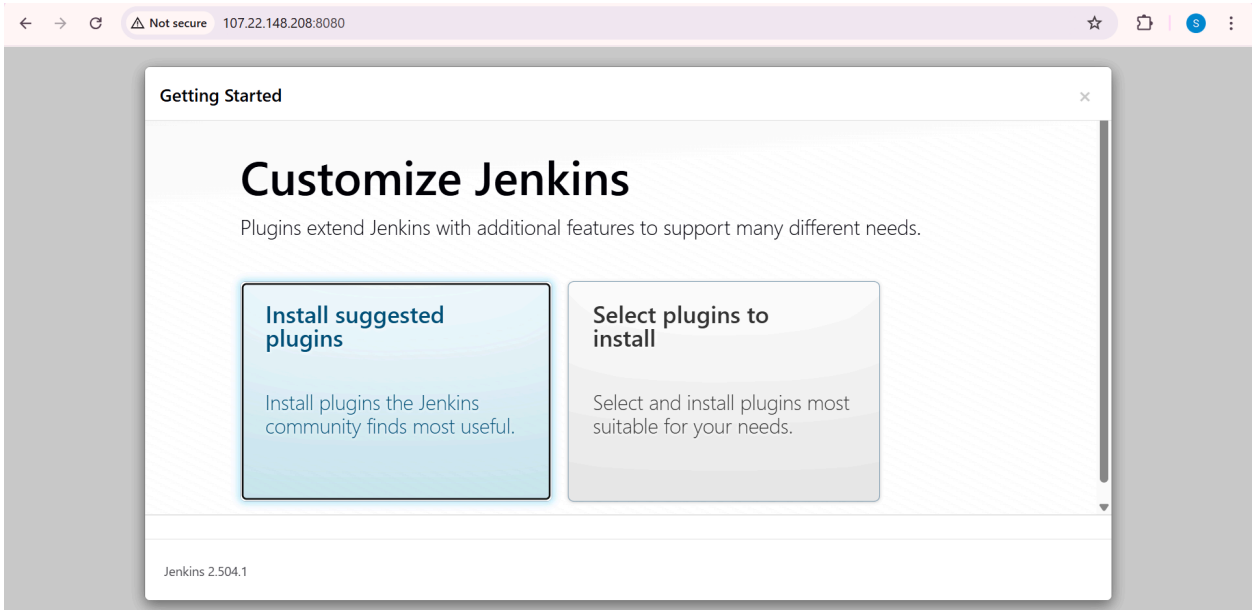
11. Now try opening the jenkins dashboard



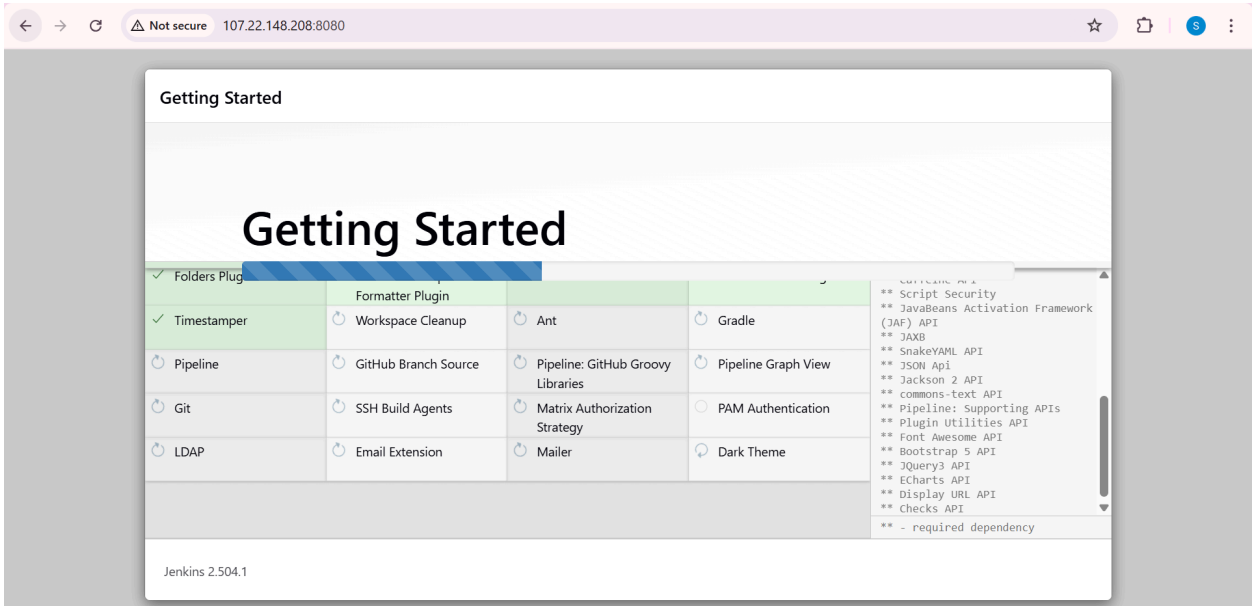
12. Now get the password from instance
    sudo cat /var/lib/jenkins/secrets/initialAdminPassword

```
ubuntu@ip-172-31-39-134:~$ cat /var/lib/jenkins/secrets/initialAdminPassword
cat: /var/lib/jenkins/secrets/initialAdminPassword: Permission denied
ubuntu@ip-172-31-39-134:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
85b0acf8237f4d5f9ddd14e3854b4ec9
ubuntu@ip-172-31-39-134:~$
```

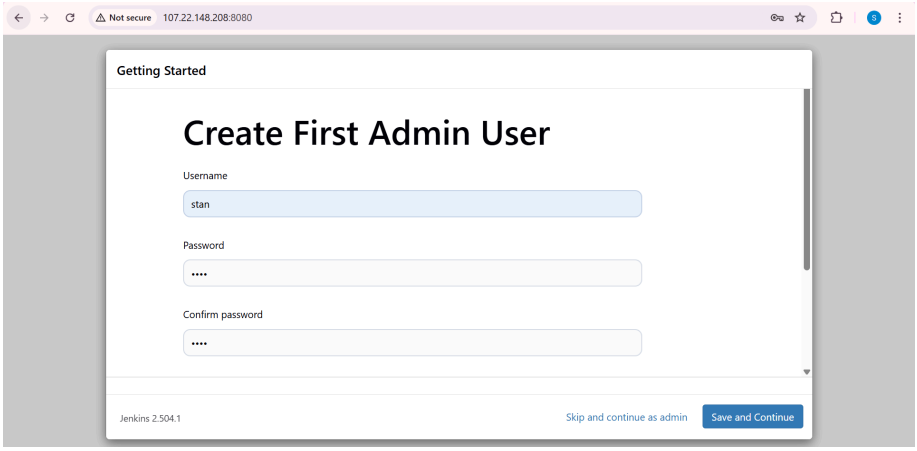
13. Click Install plugins and install necessary packages



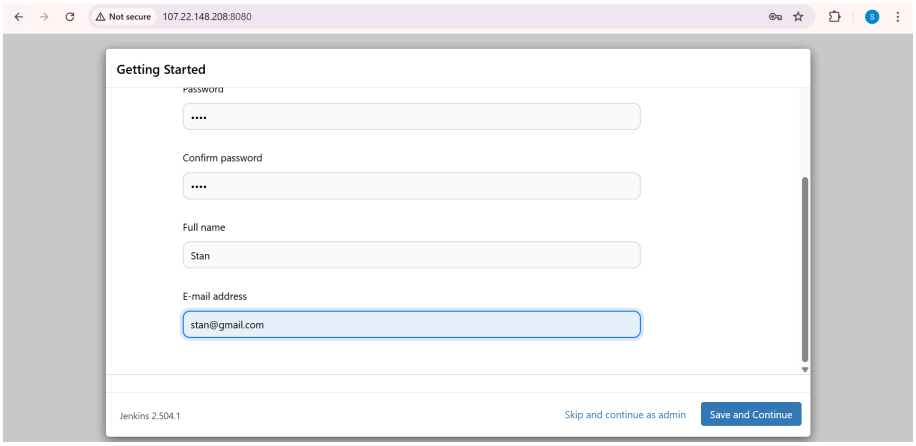
14. Wait till the completion of the Plugins installation



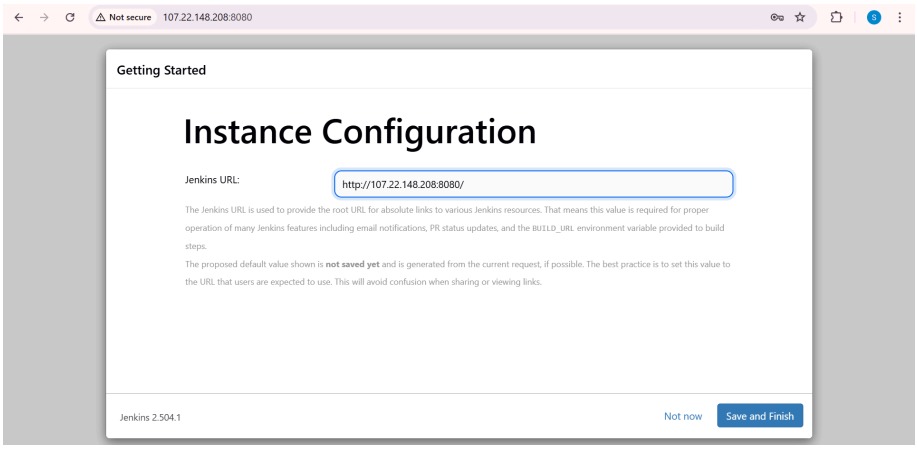
15. Enter your jenkins credentials



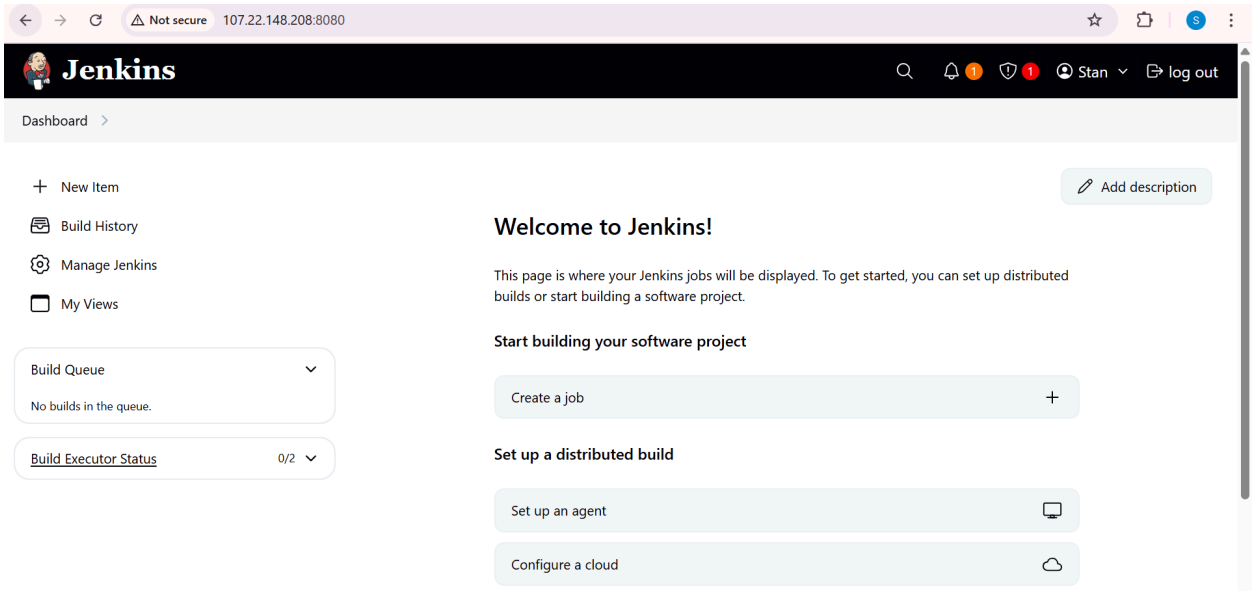
16. Click save and continue



17. With this we see the Jenkins URL



18. This is your dashboard

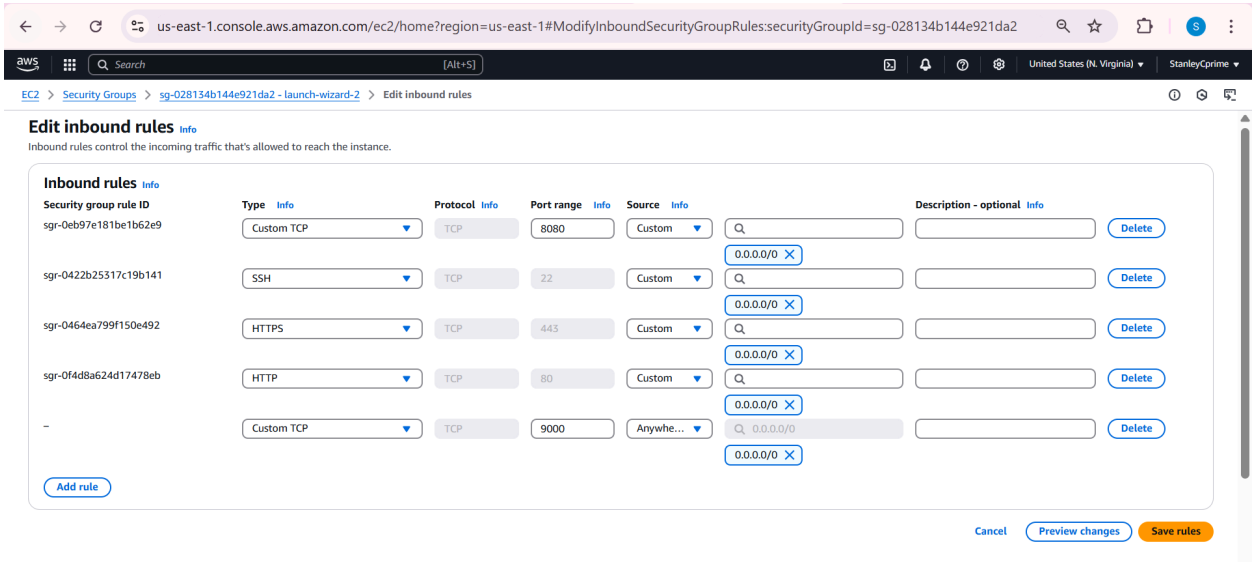


2B : Install Docker

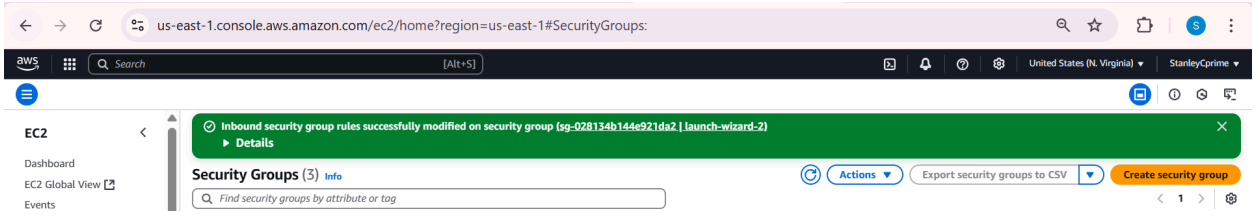
1. Open the instance in putty and install docker
2. Run these commands and install them
3. Also create a group for docker and add the user to it

```
ubuntu@ip-172-31-39-134:~$ sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER
newgrp docker
sudo chmod 777 /var/run/docker.sock
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:6 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 83 not upgraded.
Need to get 78.6 MB of archives.
```

4. Add port 9000 to Security Group to use Docker by choosing the Security Group.
5. Add the port number 9000 as Custom TCP



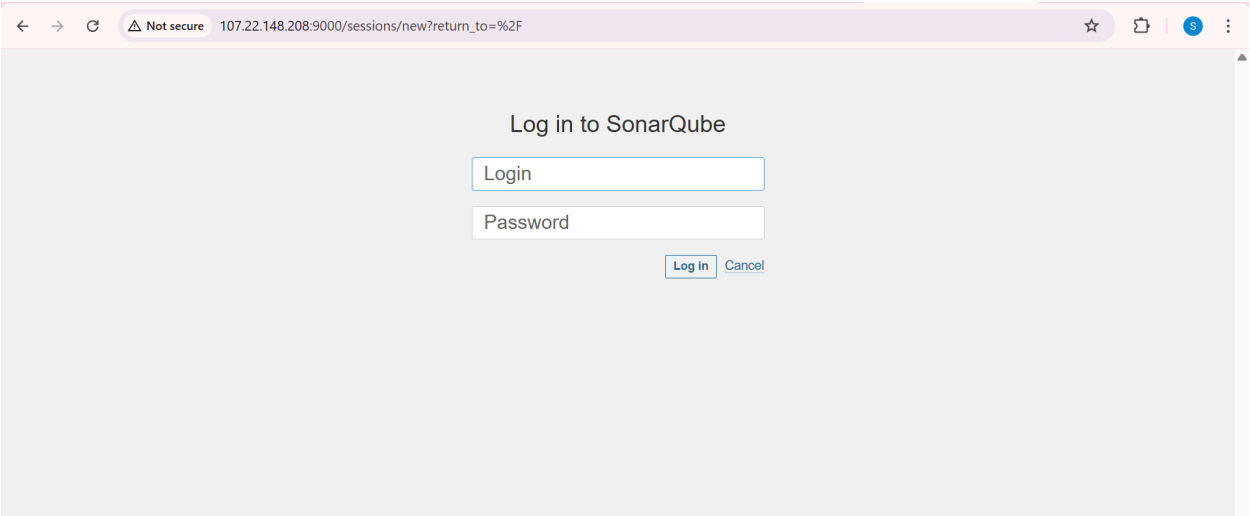
6. Now save the changes made



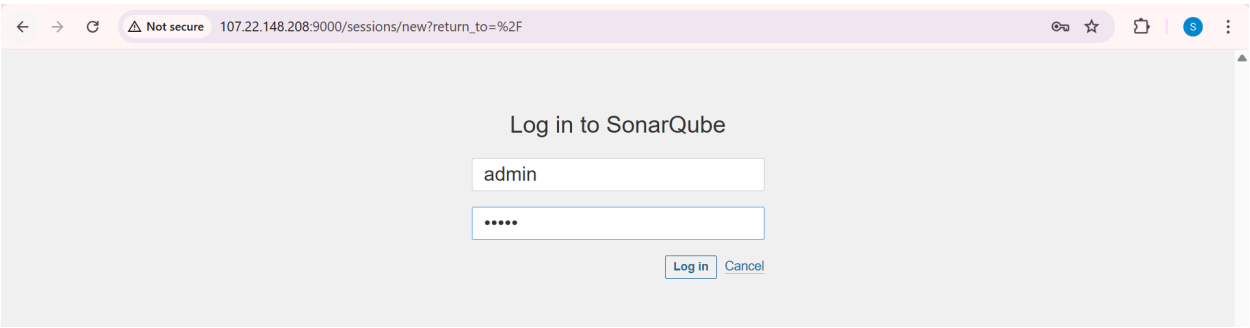
7. Now run and implement SonarQube in the container
(here we run using port 9000)

```
ubuntu@ip-172-31-39-134:~$ docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
215ed5a63843: Pull complete
094bfcb4db7a: Pull complete
4df791be4da6: Pull complete
97a8e80e60c2: Pull complete
7be47dbb02a7: Pull complete
61c8ff67f948: Pull complete
2737e808f9c3: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:93f94e0ea6148cd02d52378049dad85d0f2d6c90c485578317f76addf2af9f3d
Status: Downloaded newer image for sonarqube:lts-community
ed94f18e4fead48d5a18c1d0b526362a5740c70552d2c3b5ab49d6afd23fa34e
ubuntu@ip-172-31-39-134:~$
```

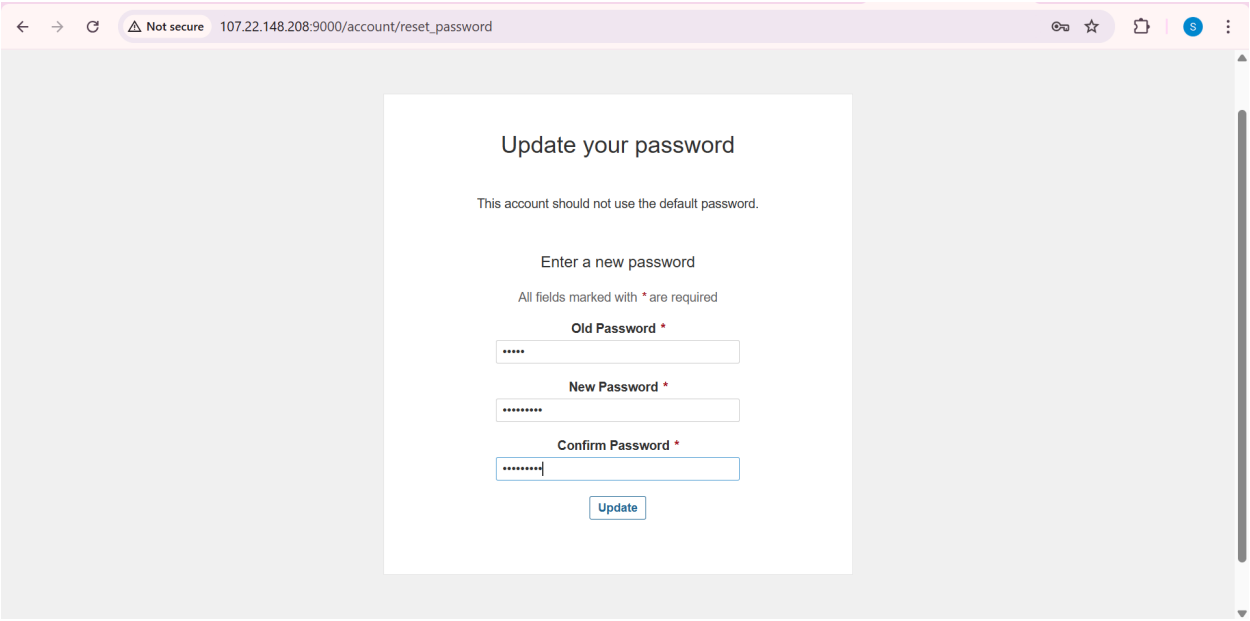
8. Now open the SonarQube Dashboard by using <public-ip>:9000



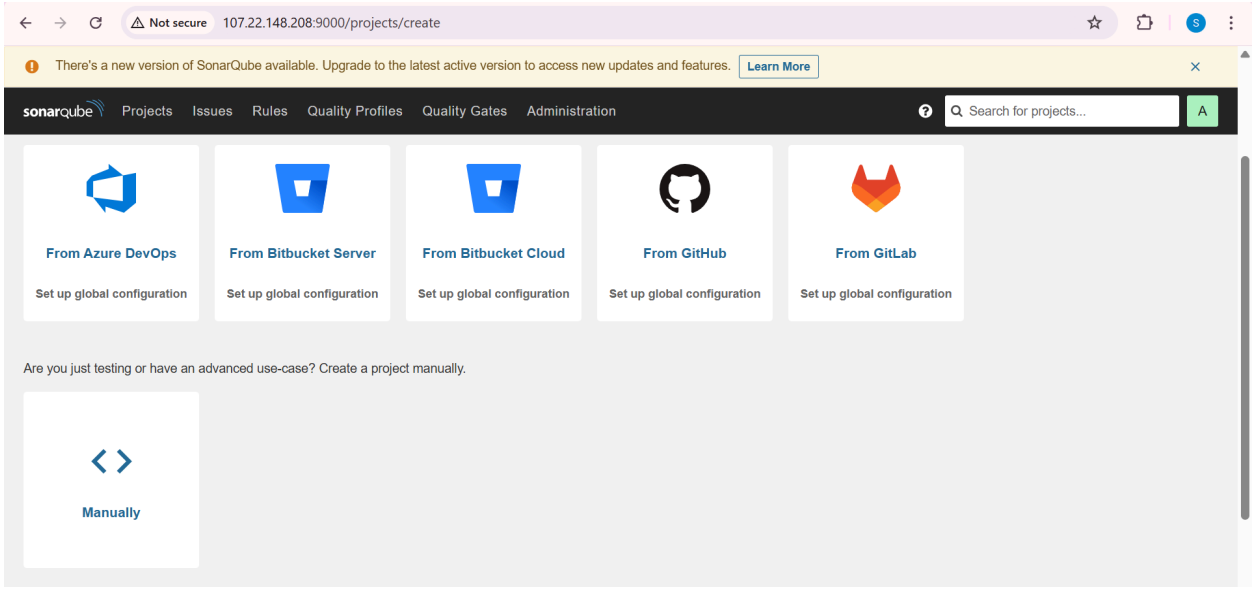
9. Here we pass the credentials as Username - admin, Password - admin



10. Also update and create new password

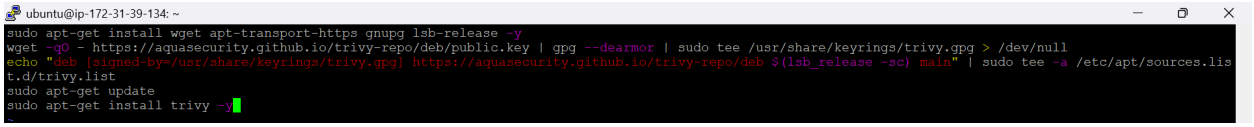


11. This finally shows the SonarQube Dashboard

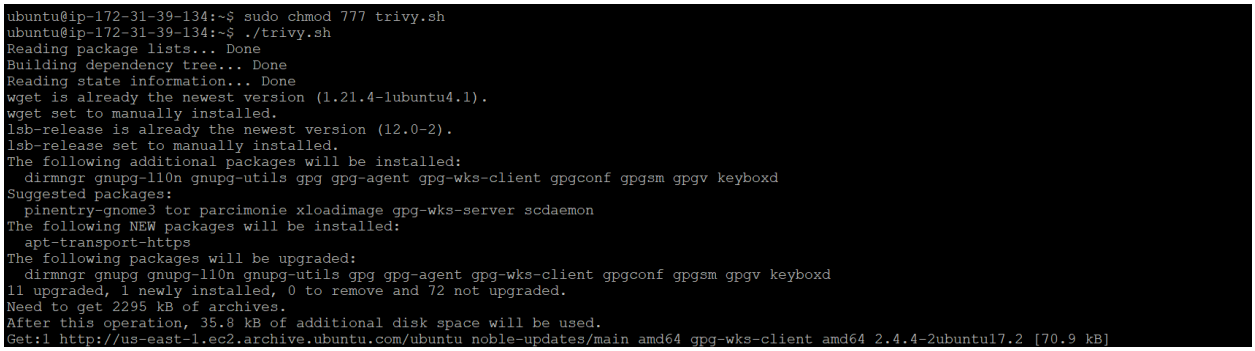


2C : Install Trivy

1. Now create a new file called trivy.sh
2. This file is to run and enable the Trivy package



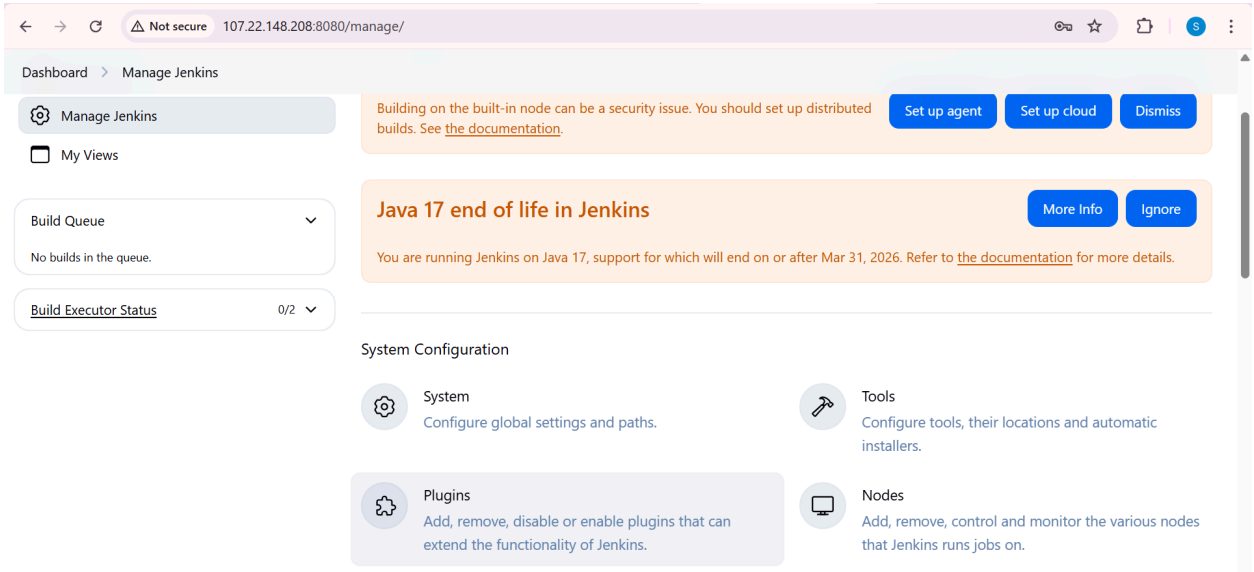
3. Now make the file executable using 777 permissions
4. Now execute the file and execute the commands



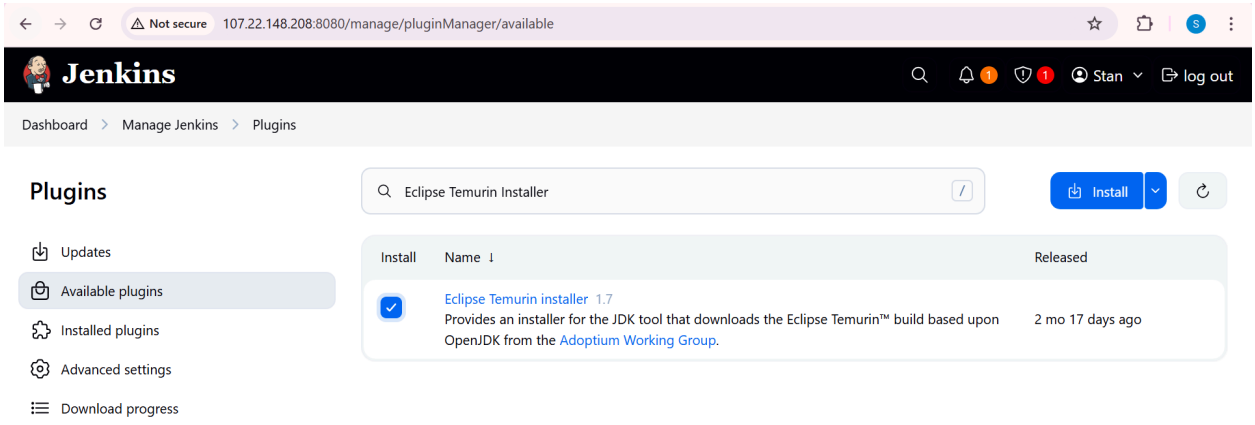
Step 3 : Install Plugins like JDK, Sonarqube Scanner, NodeJs, OWASP Dependency Check

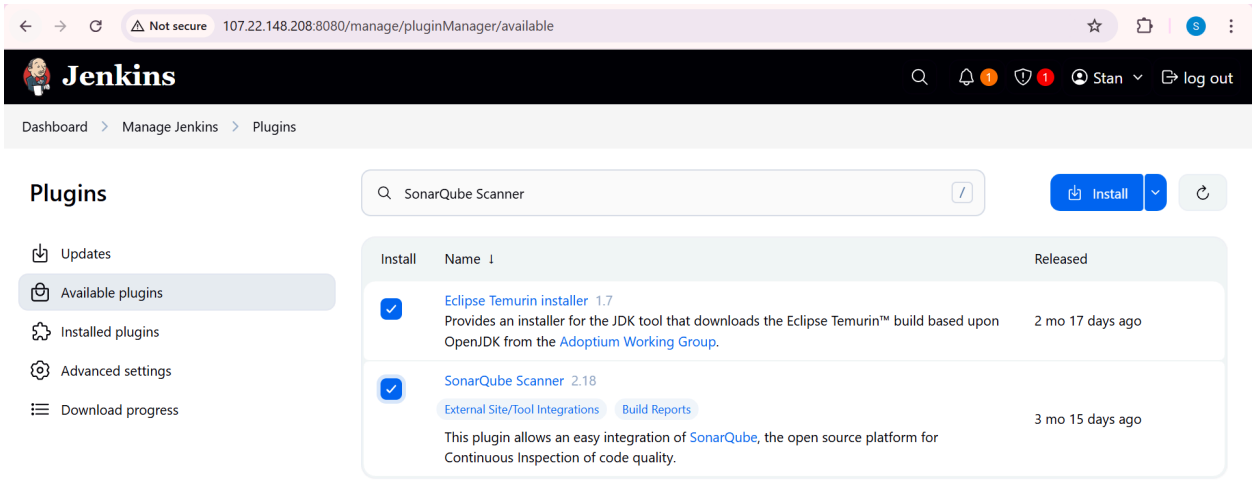
3A : Install Plugin

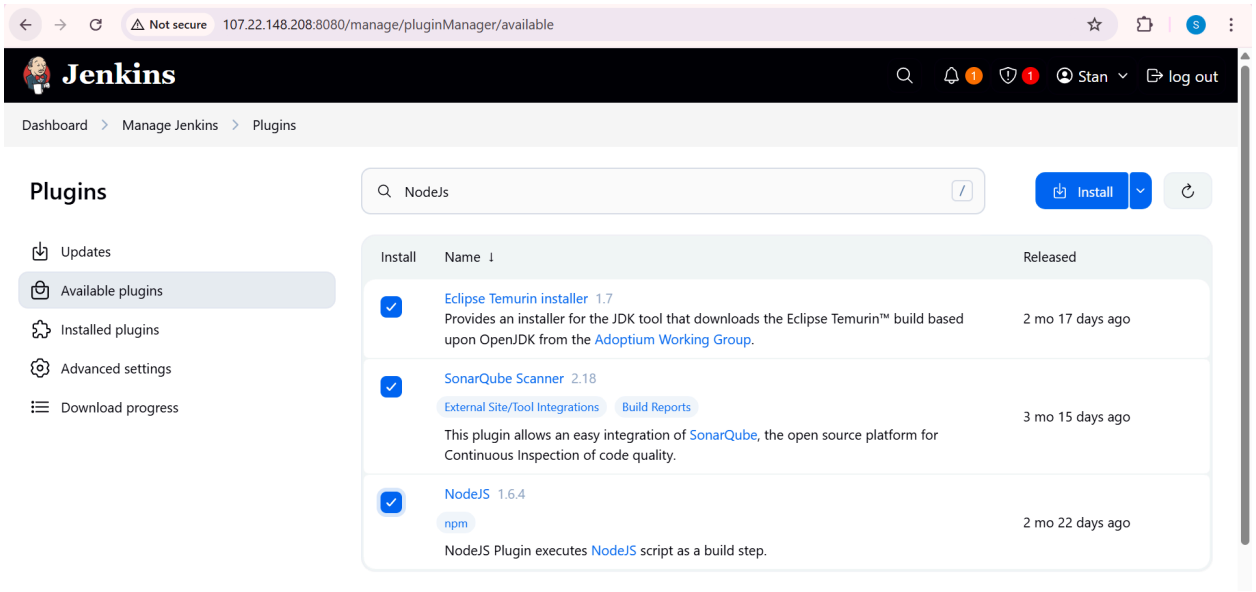
1. Open Jenkins Dashboard
2. Click Manage Jenkins and choose Plugins



3. Now add a plugin from Available Plugins Tab
4. Add plugins such as
    a. Eclipse Temurin installer
    b. SonarQube Scanner
    c. NodeJS



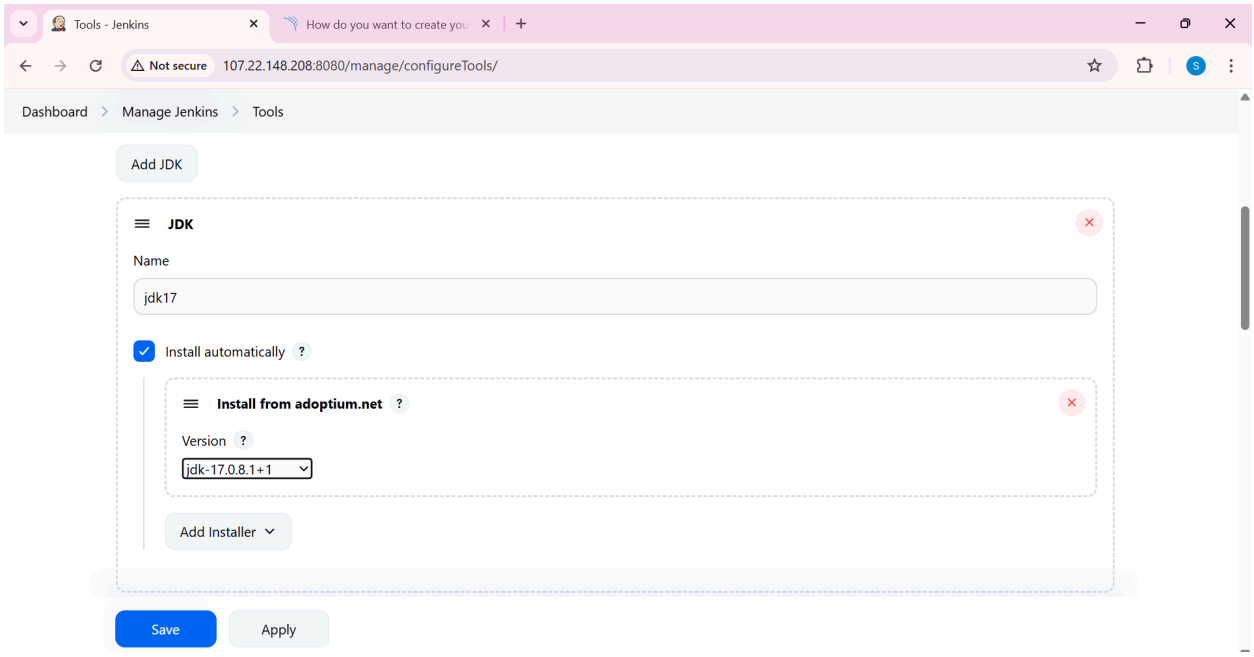




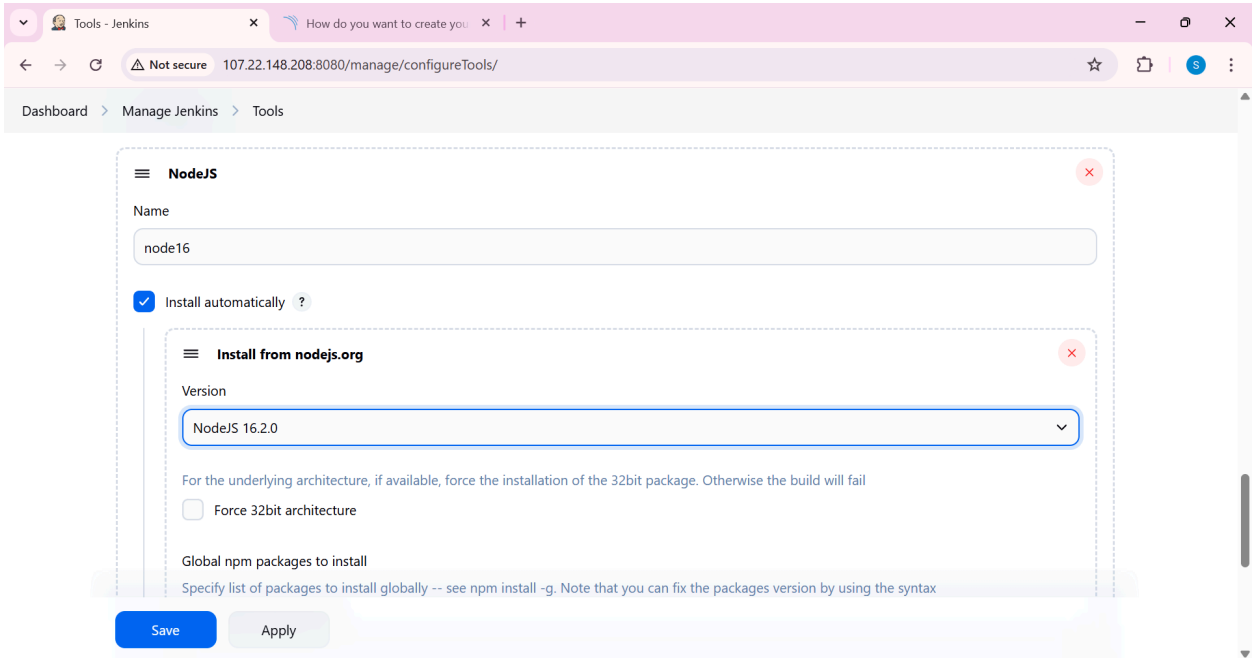
5. Now install them without restart and go back to Dashboard

3B : Configure Java and Nodejs in Global Tool Configuration

1. Now open Jenkins Dashboard and click Manage Jenkins
2. Choose Tools and search for JDK and NodeJs installation
3. Click Add JDK and name as jdk17
4. Choose Install Automatically and select version



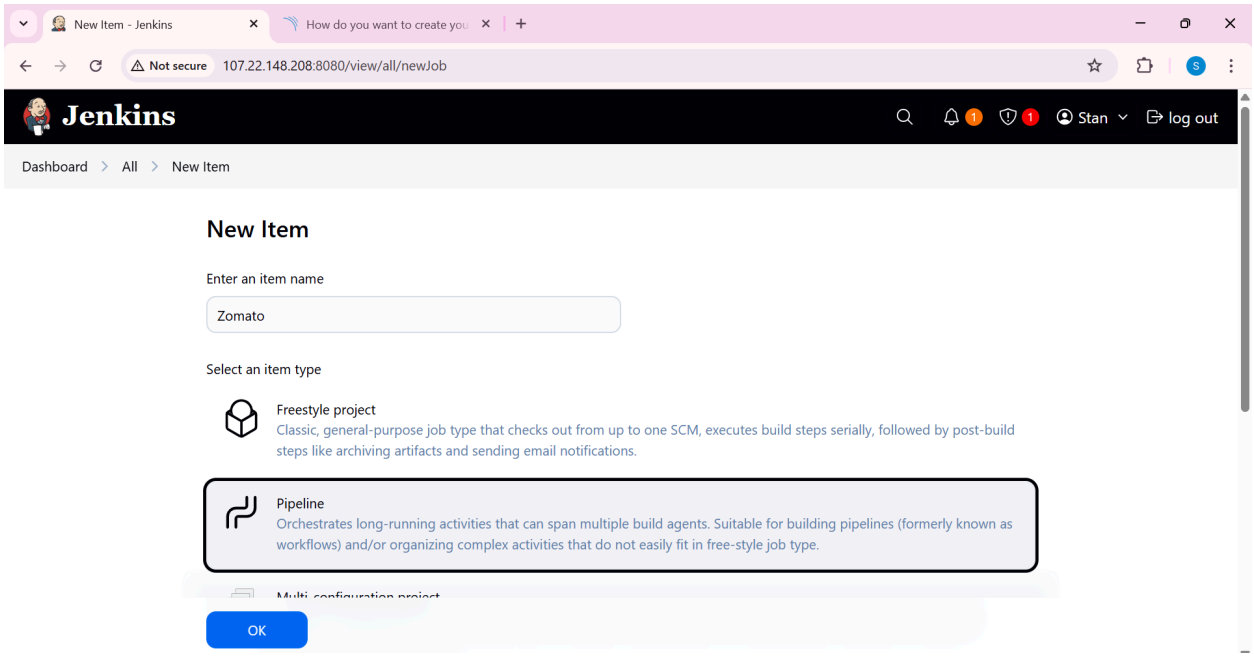
5. Now click add NodeJS
6. Add name as node16 and choose Install Automatically
7. Select the desired version



8. Click Apply and Save

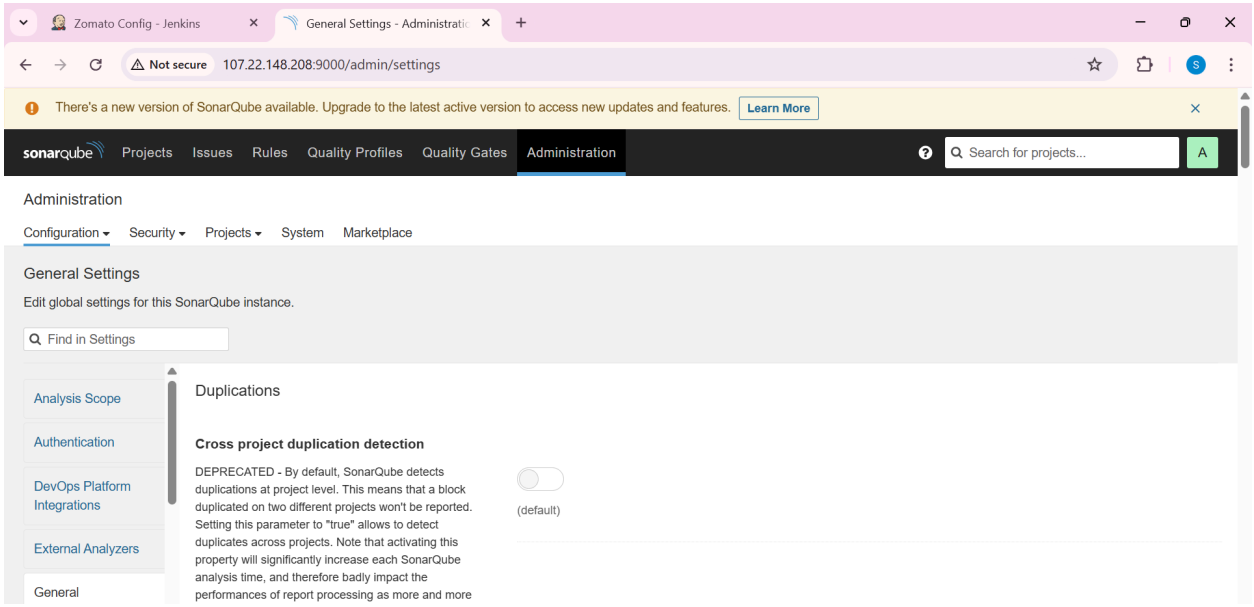
3C : Create a Job

1. In Jenkins Dashboard, click New Item
2. Name it as Zomato and choose Pipeline as Type

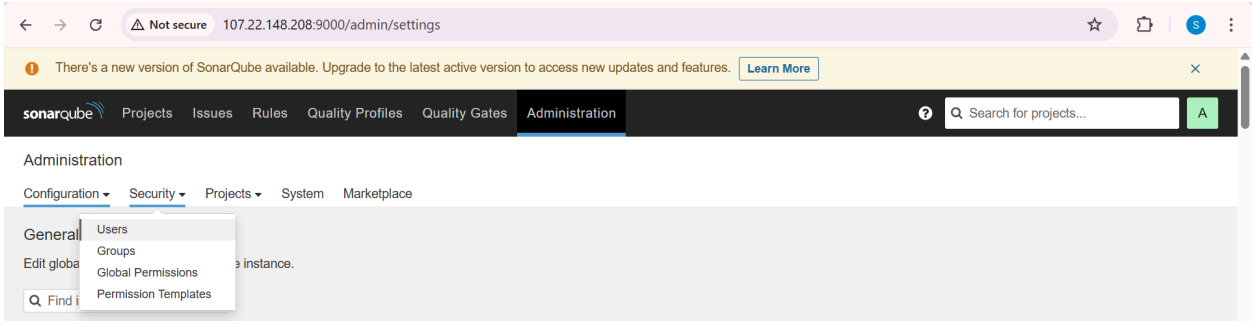


Step 4 — Configure Sonar Server in Manage Jenkins

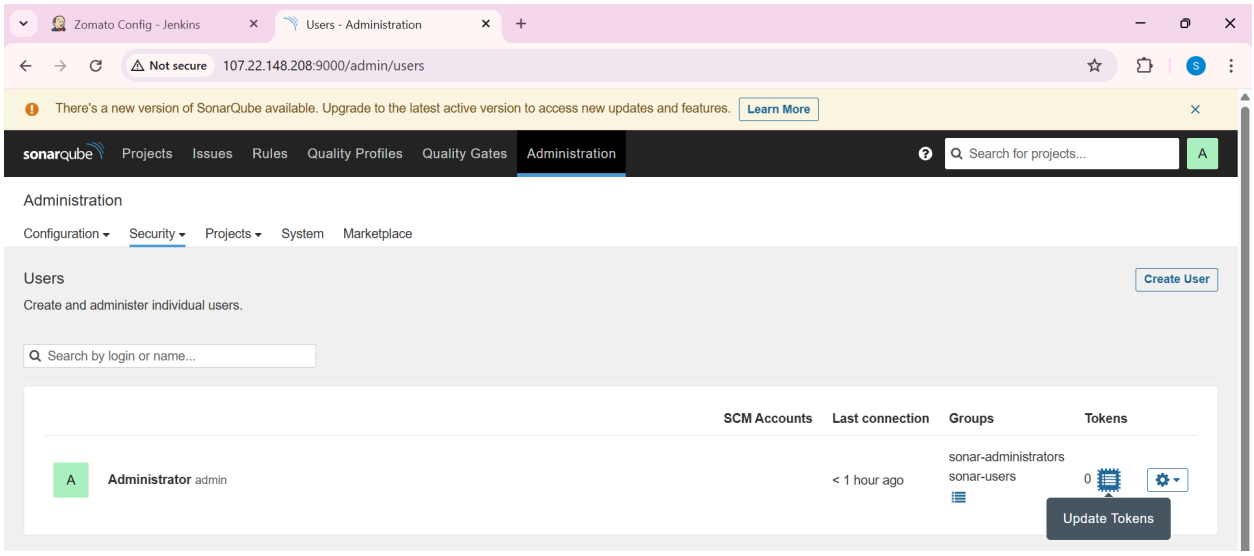
1. Open SonarQube Dashboard using port 9000



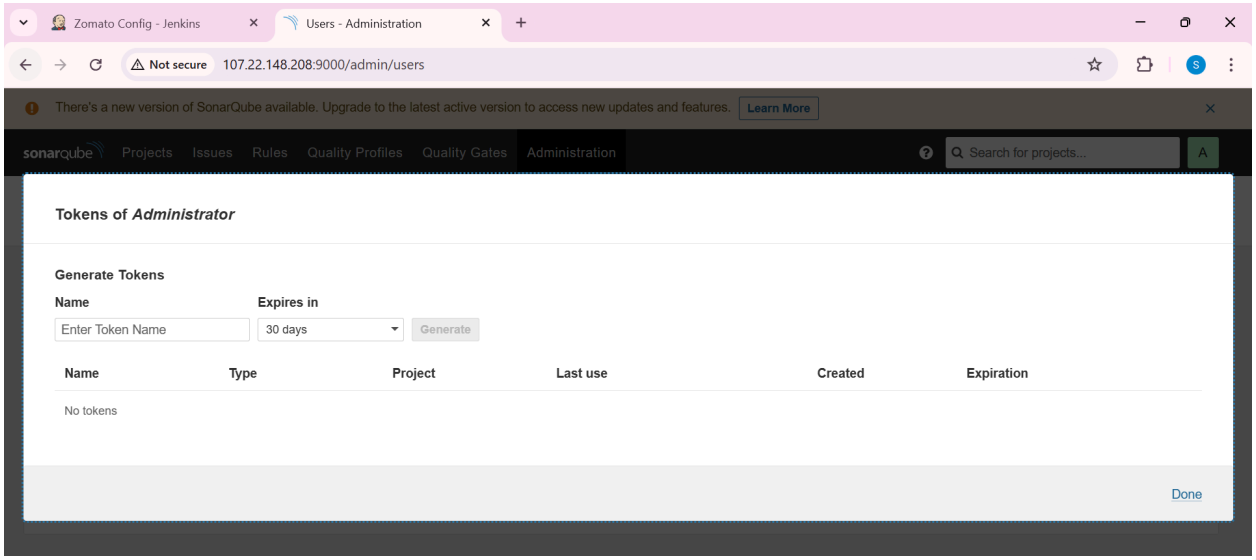
2. Click Administrator in Navigation Bar
3. Within Security Tab, choose Users



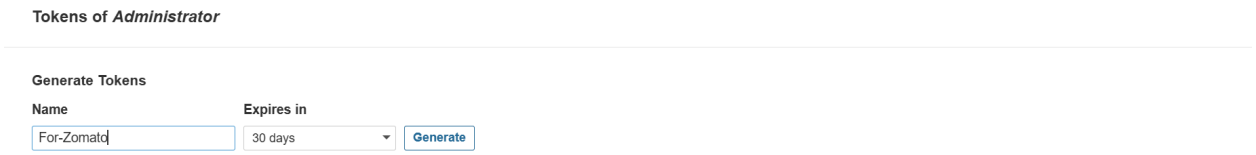
4. Here Administrator will be available. Here click Update Token



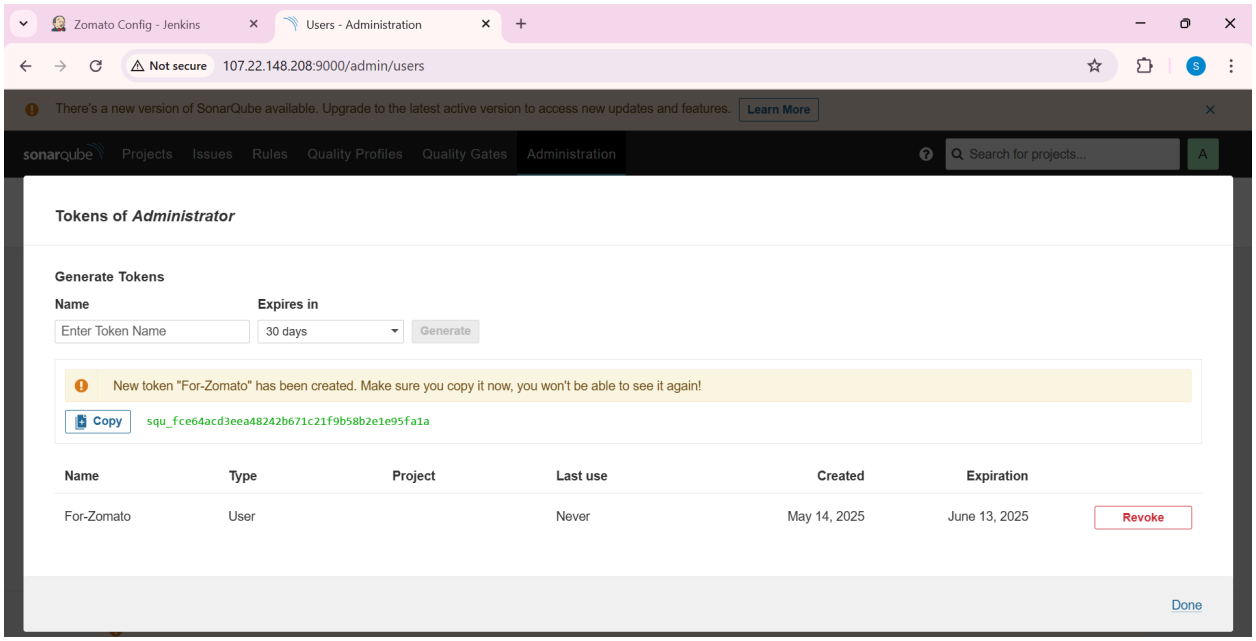
5. Here add any Token name and set the Expiry date



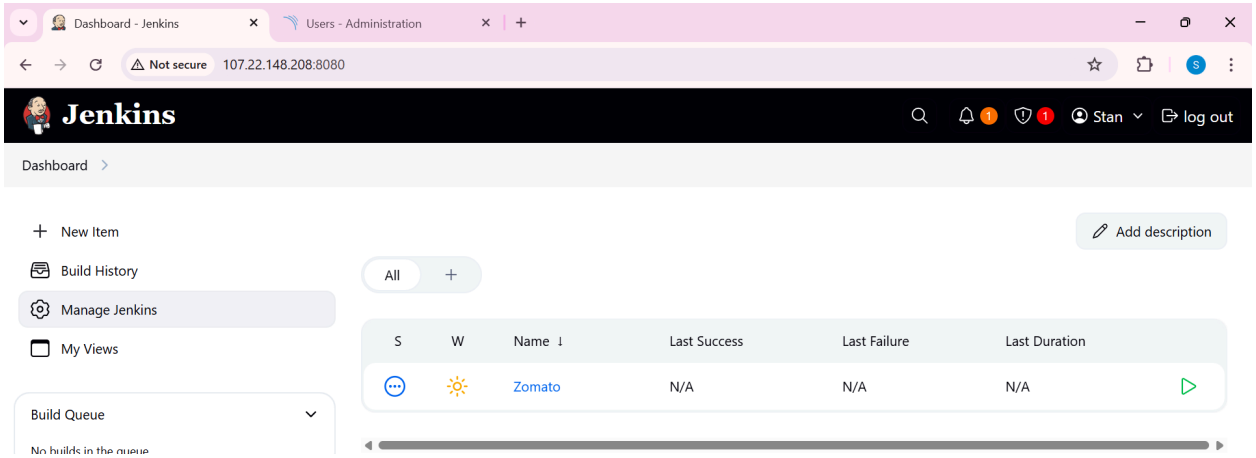
6. We set it as For-Zomato with 30 days to expire



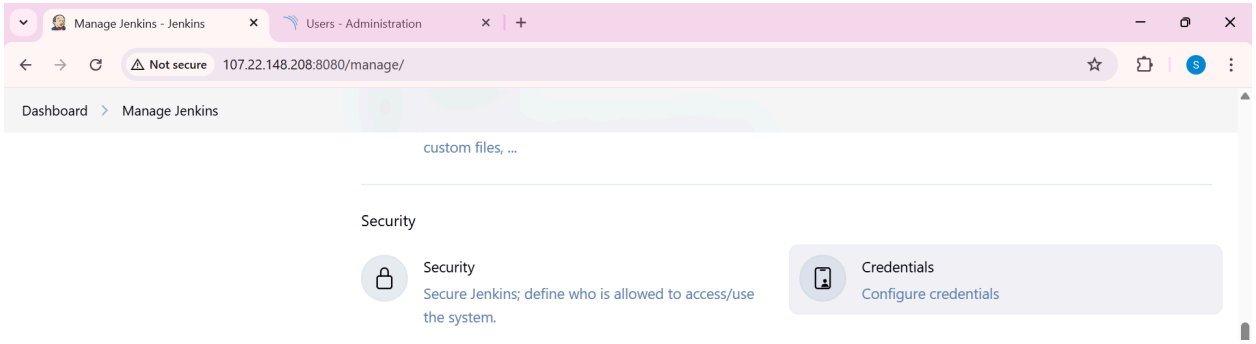
7. Now copy paste the token generated somewhere safe

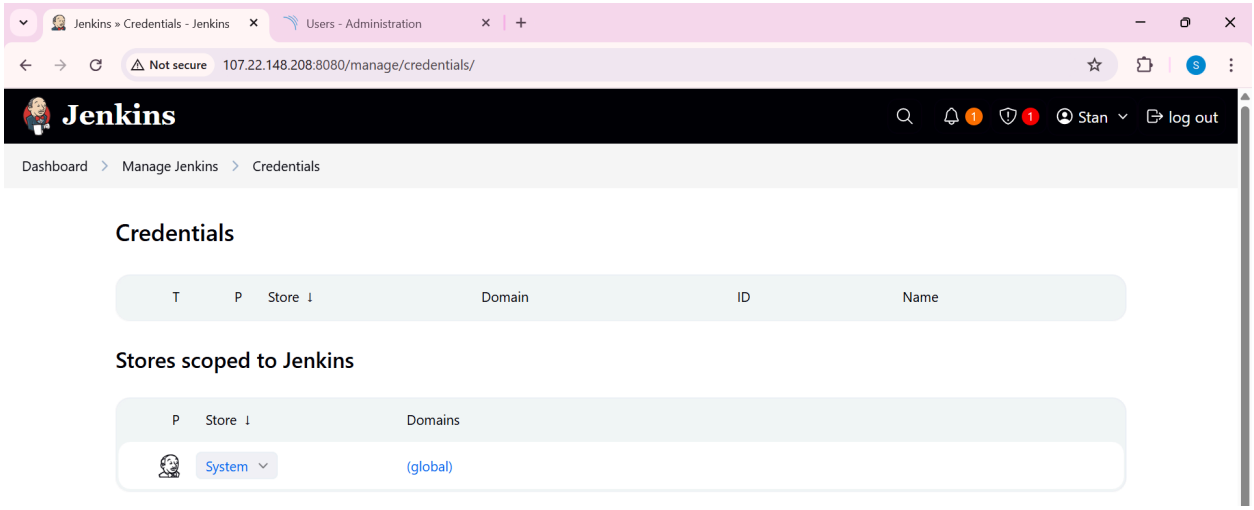


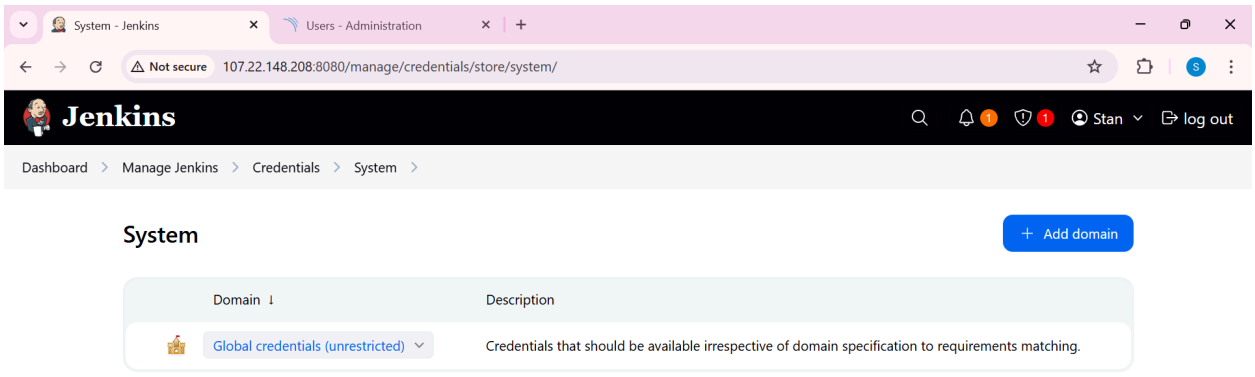
8. Now open the Jenkins Dashboard, and click Manage Plugins



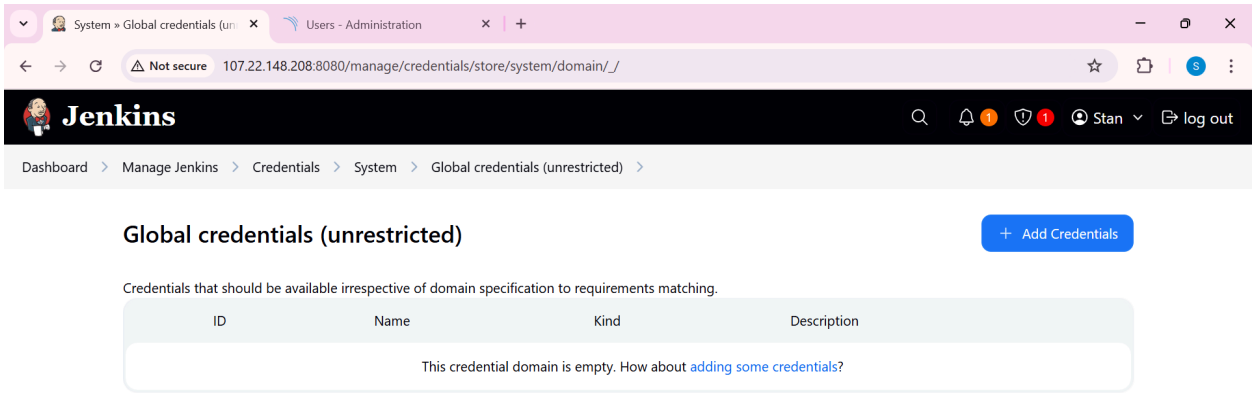
9. Here we choose Credentials - System - Global Credentials



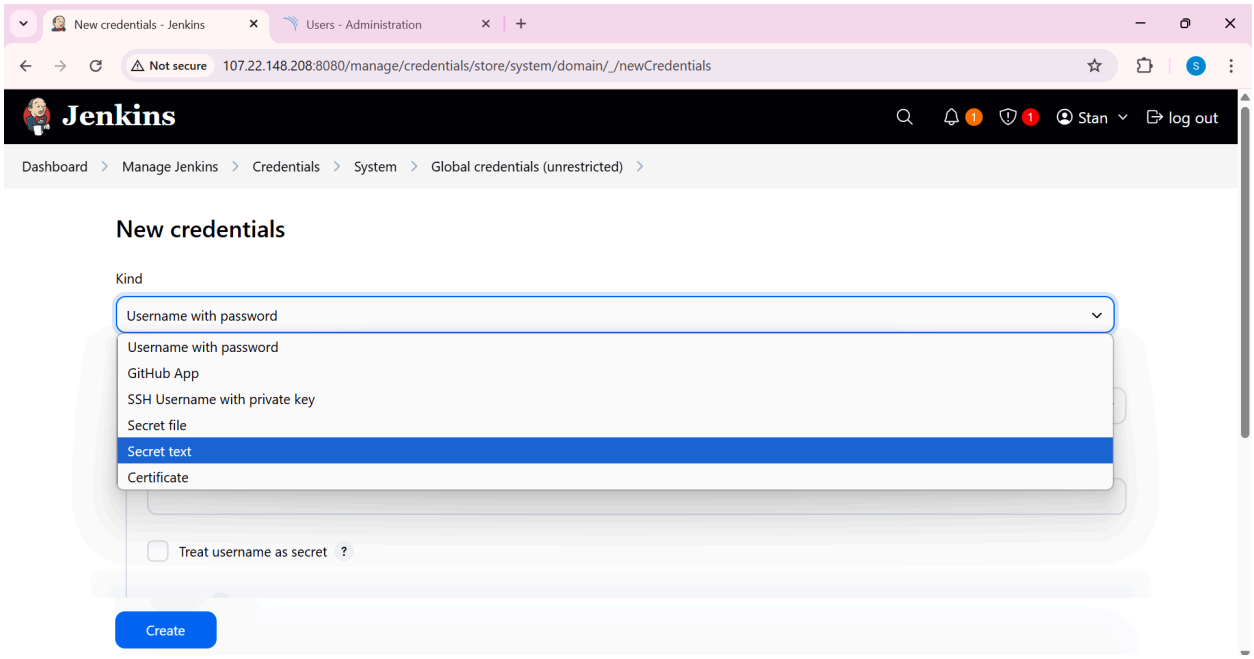




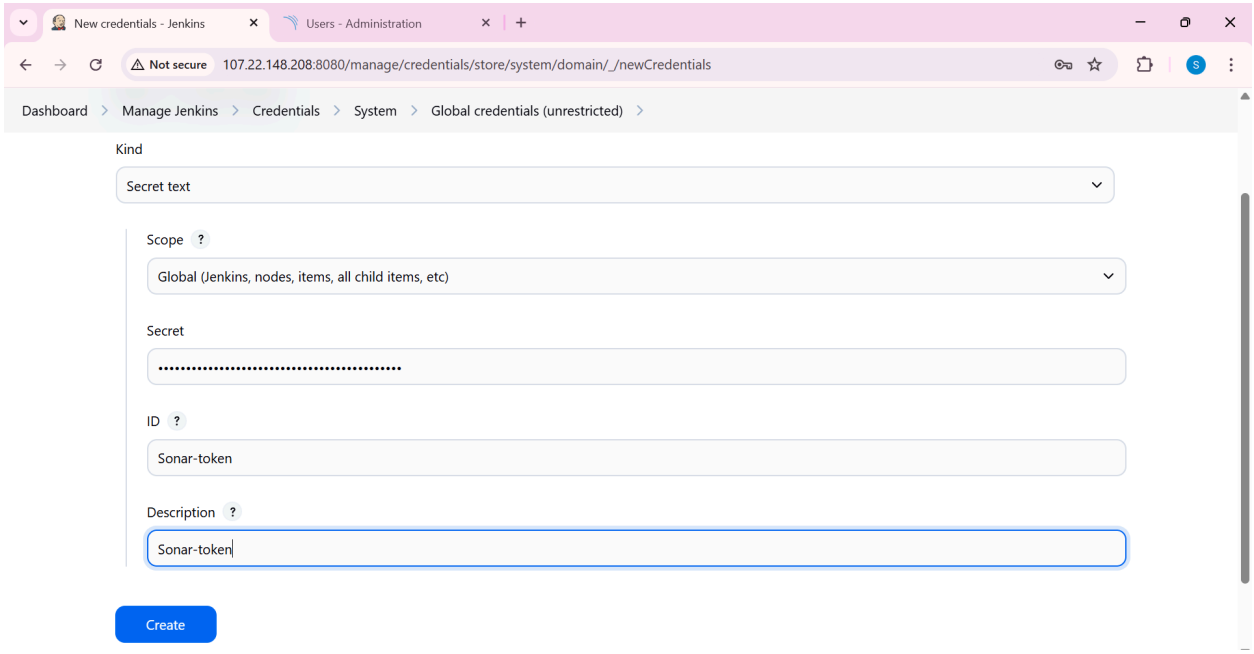
10. Here click Add Credentials



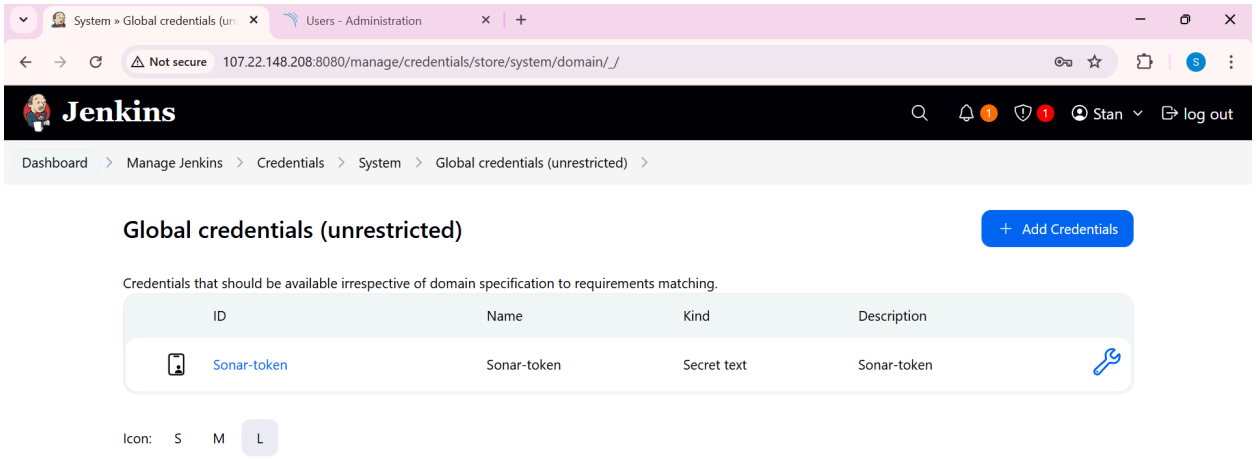
11. Choose Secret Text



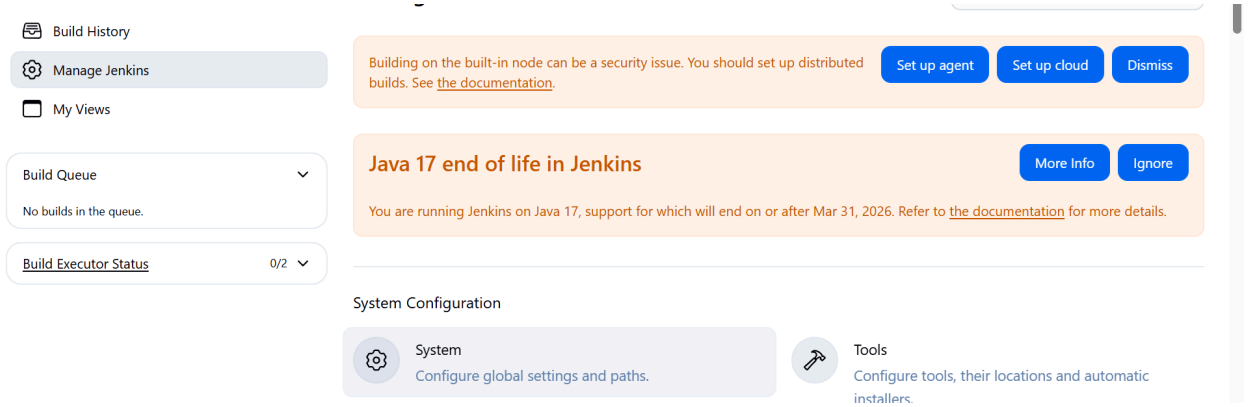
12. Add the token we generated in SonarQube Administrator and add a name to it (here we use Sonar-token) and click Create



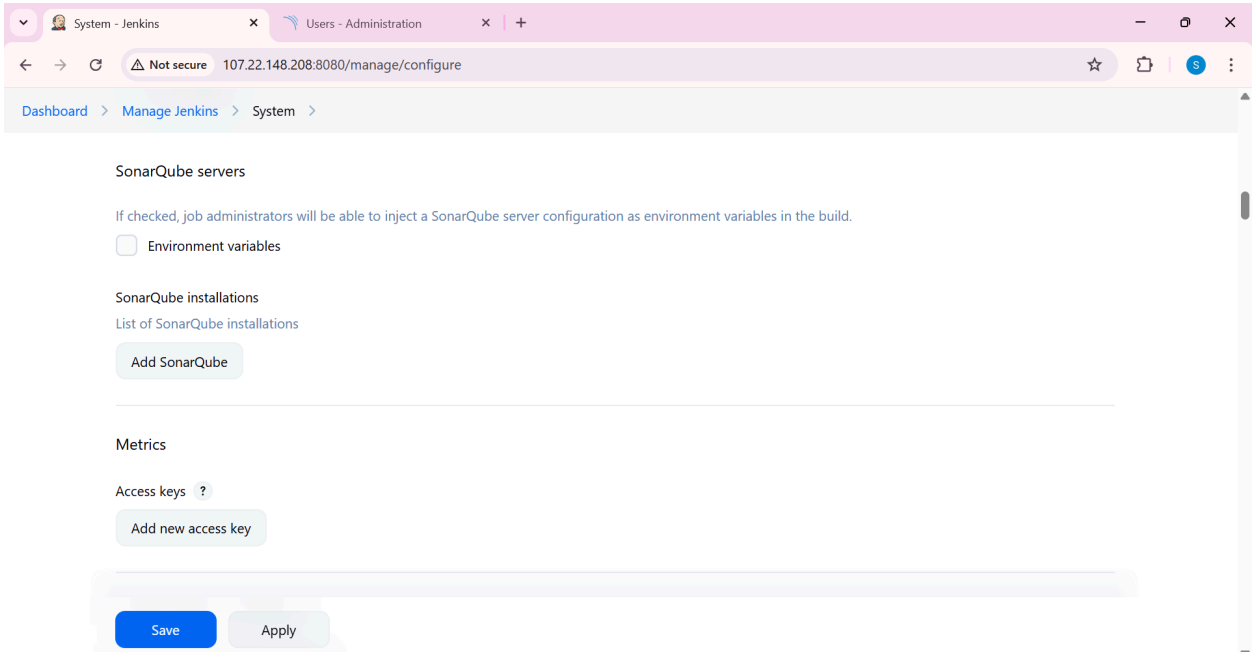
13. Now we see the Credentials available here



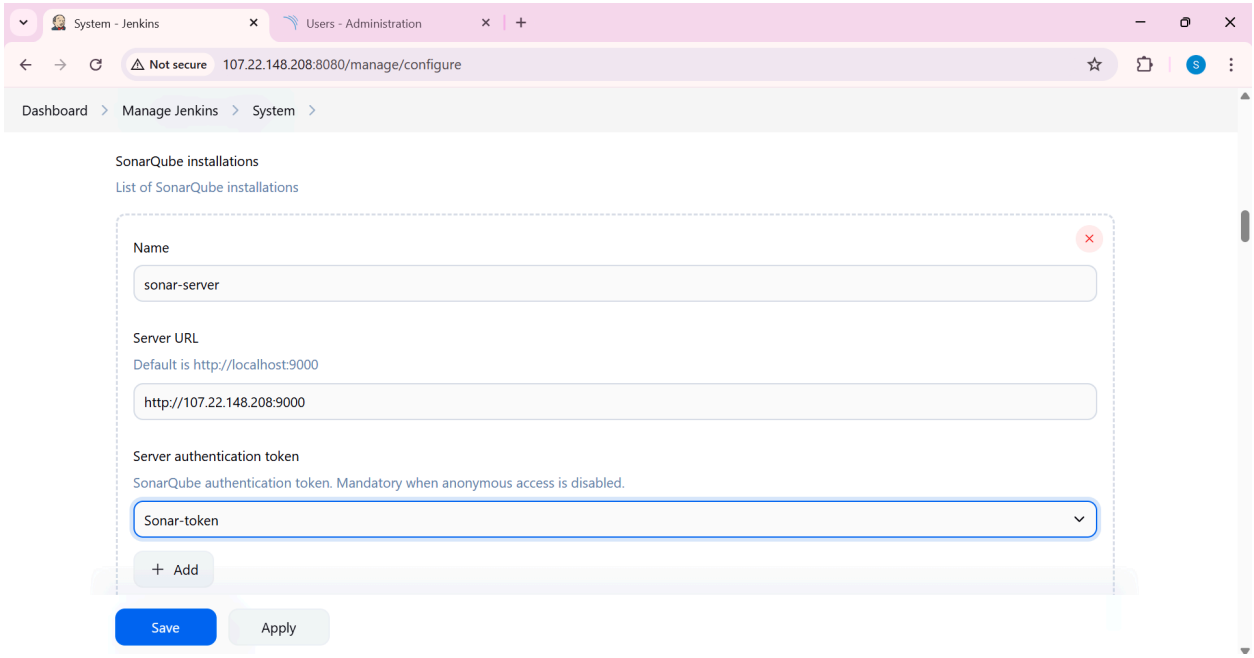
14. Now go back to Jenkins Dashboard



15. Click Manage Jenkins, where we choose System
16. Inside search for SonarQube installations and click Add SonarQube

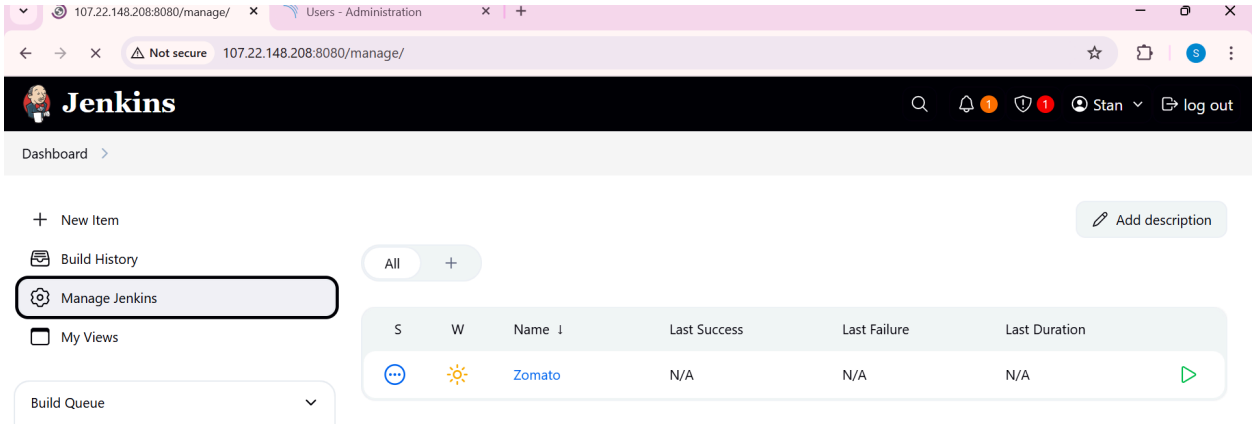


17. Name it as sonar-server and add Url ad <public-ip>:9000
18. Finally select the credentials, we created for SonarQube

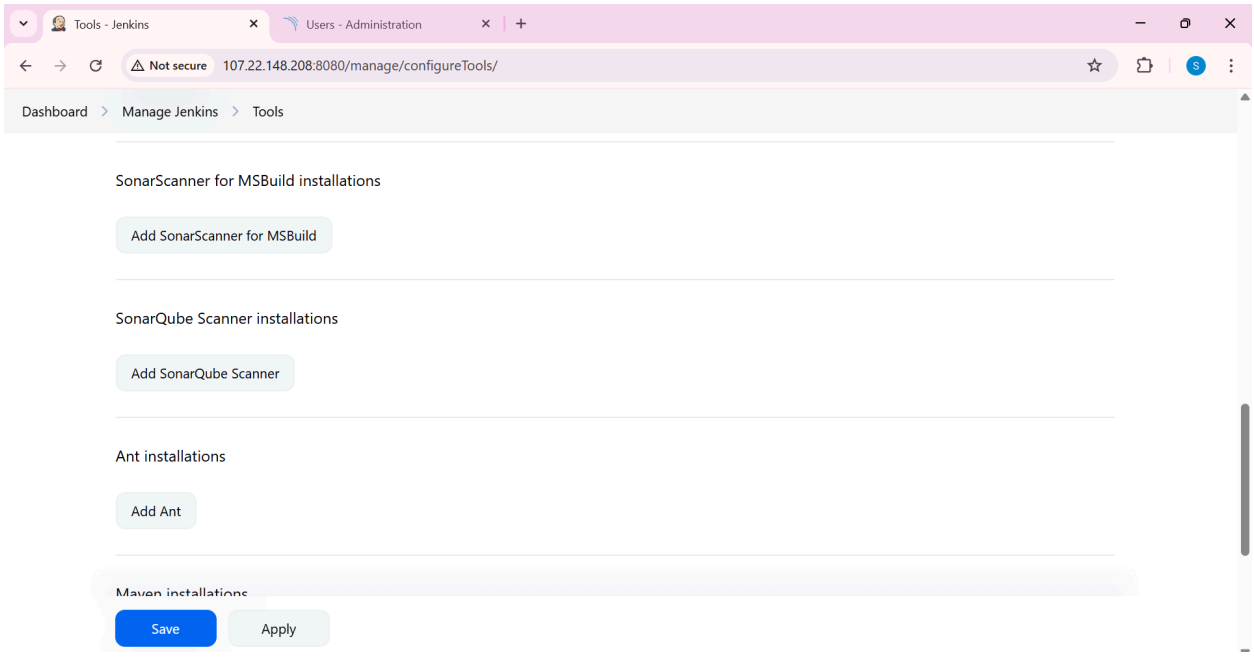


19. Click Apply and Save

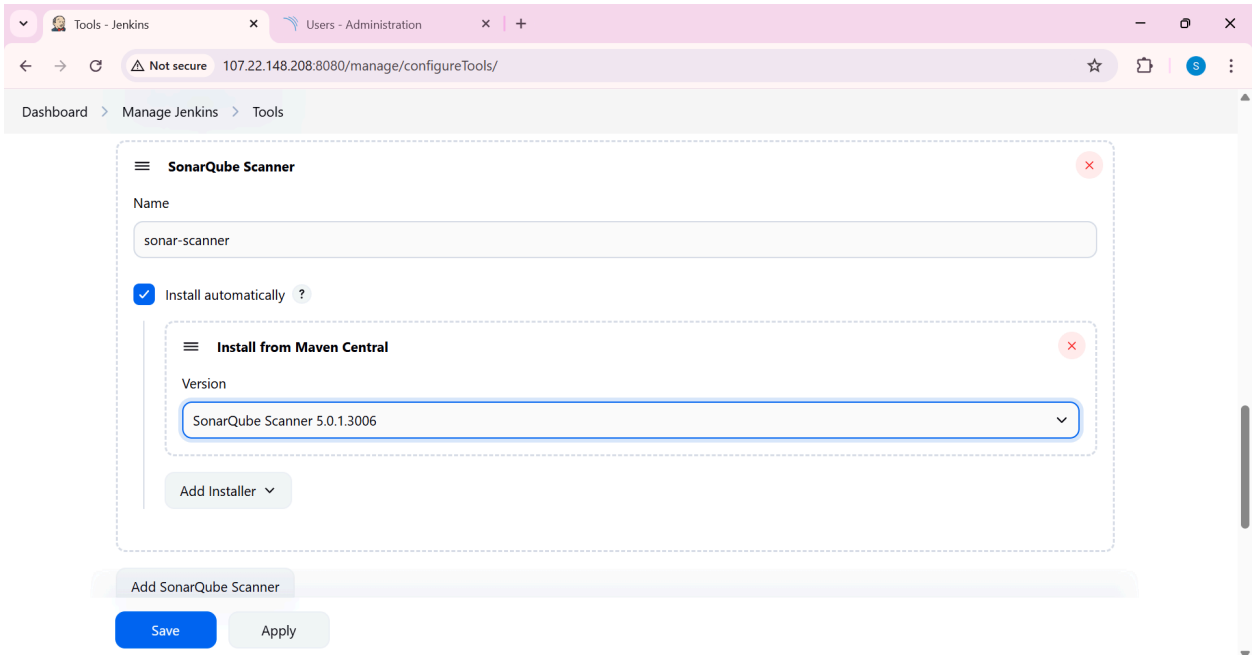
20. Now the the Configure System option is used in Jenkins to configure different server
21. So we use Global Tool Configuration is used to configure different tools that we install using Plugins



22. Choose Tools in Manage Jenkins
23. Search for SonarQube Scanner Installations and click add SonarQube Server



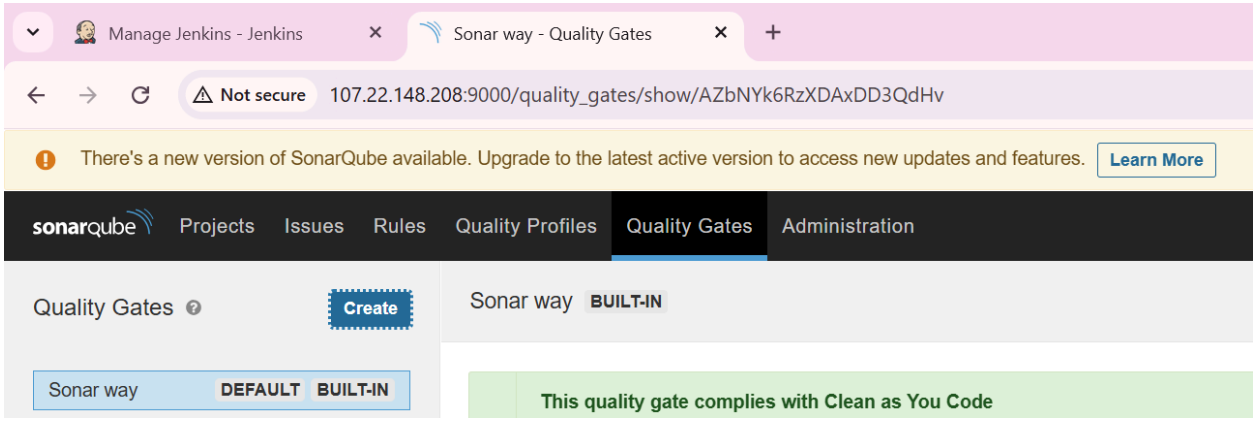
24. Here add name as sonar-scanner and choose Install automatically
25. Select the desired version and click Apply and Save



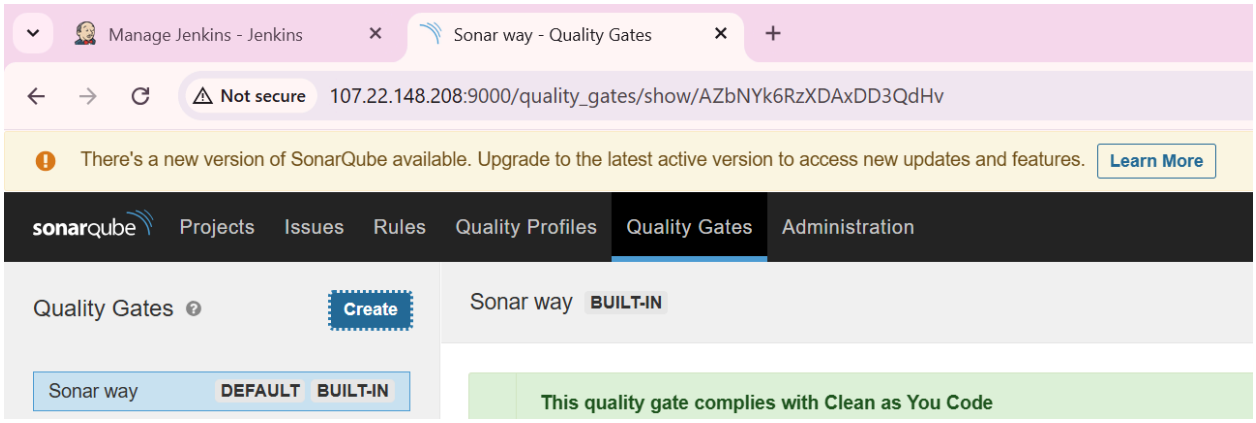
26. Here we can create a Quality Gate or use the Default one

27. To create a Custom Quality Gate

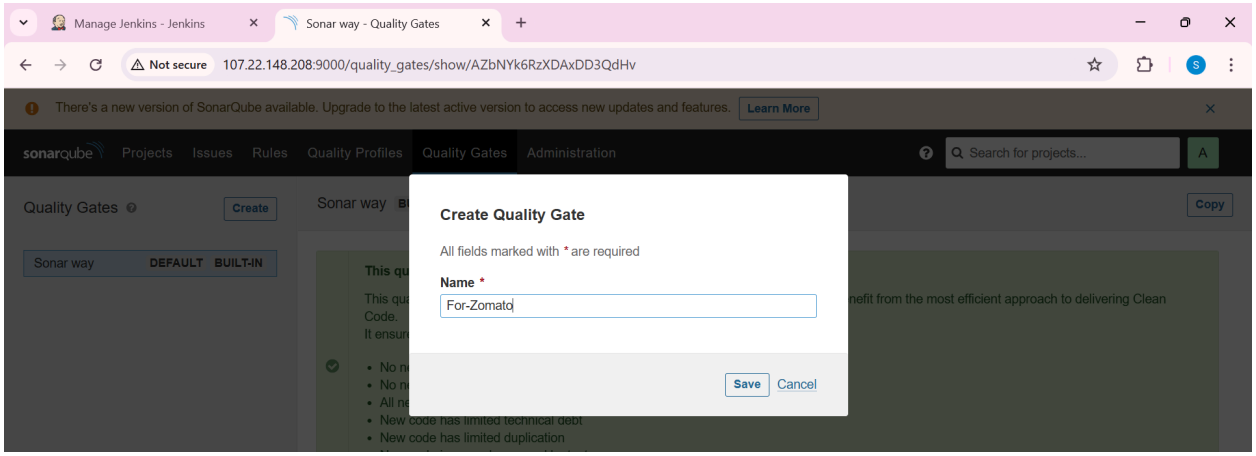
a. Open SonarQube Dashboard and choose Quality Gates



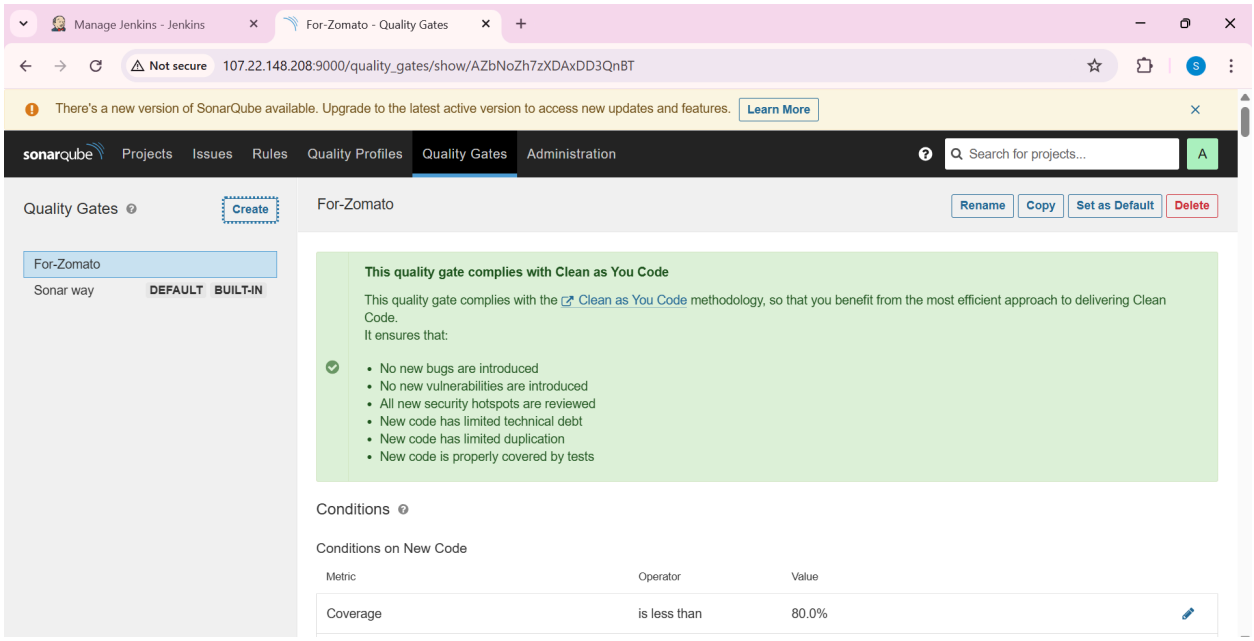
b. Here click Create and see the Pop up



c. Add name as For-Zomato and Click Save

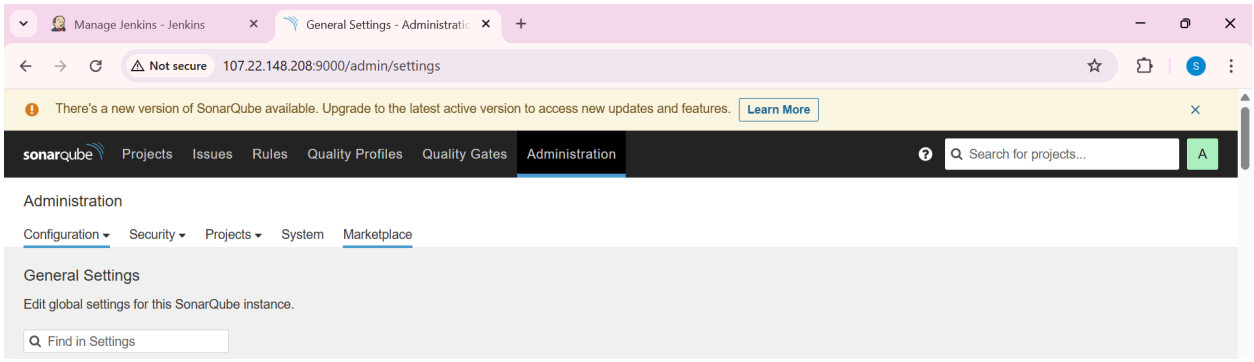


d. We can see the Quality Gate created

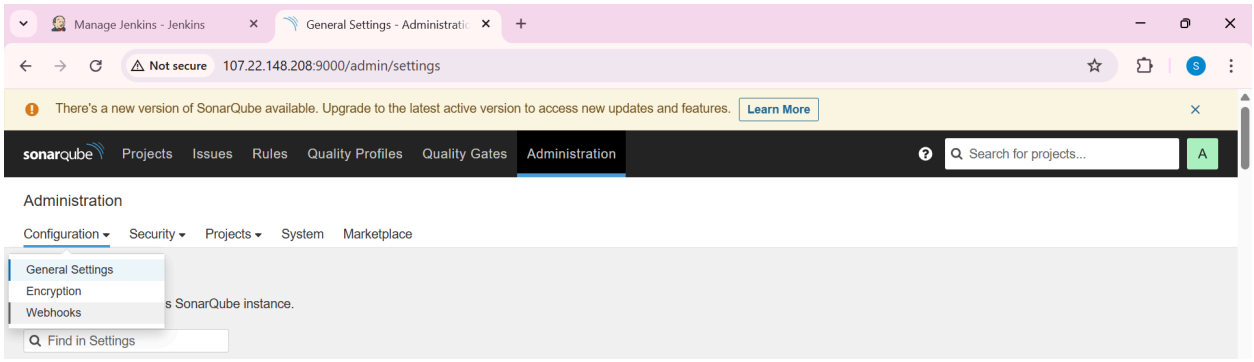


e. But we move forward with Default Quality Gate

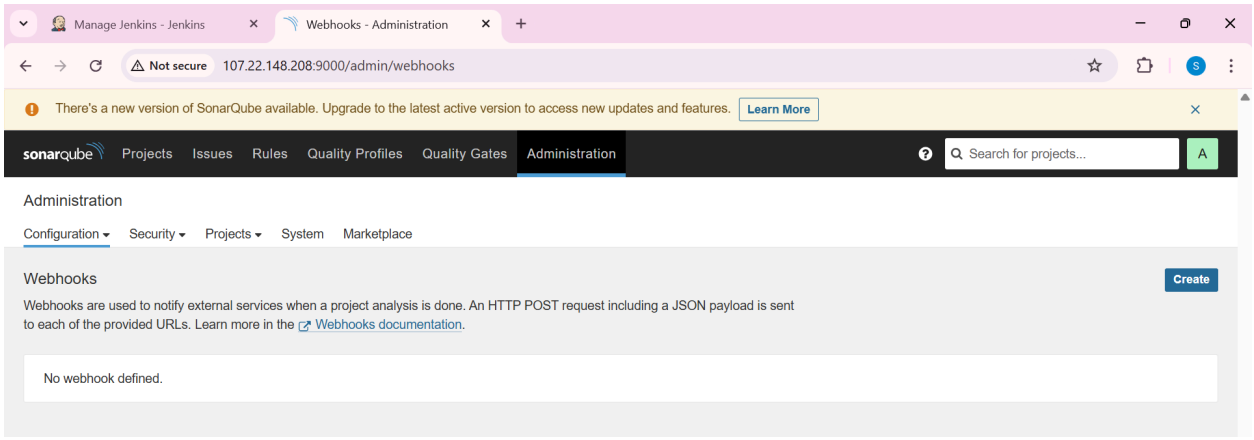
28. Now create Webhook by going to SonarQube Dashboard and choose Administration



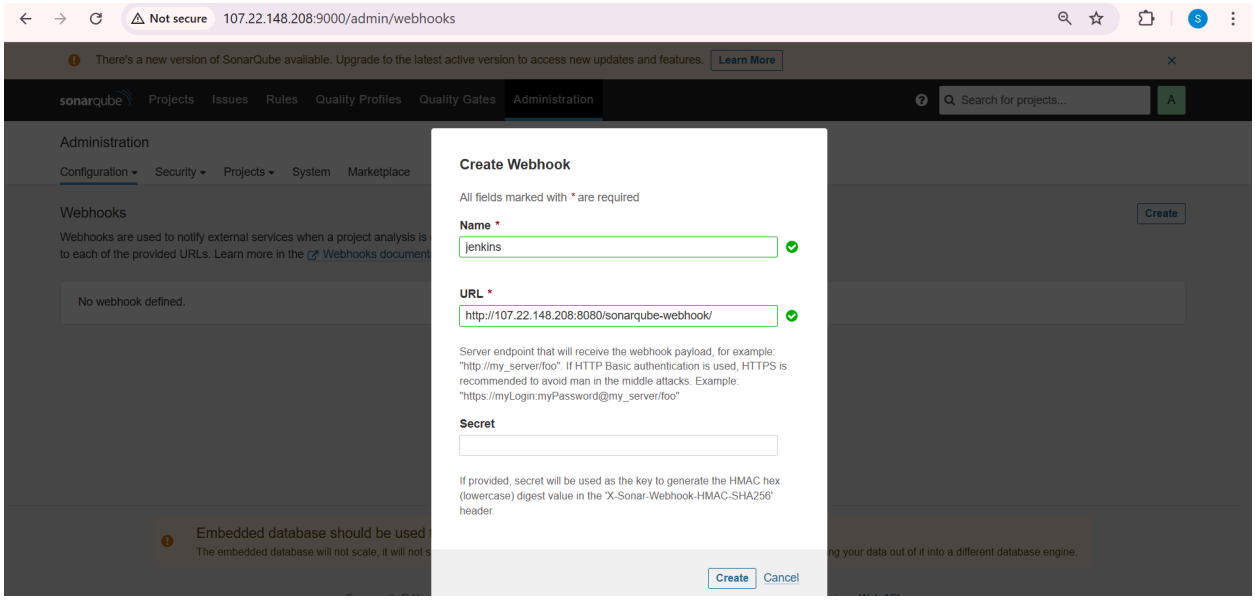
29. In Configuration Tab, by hovering we see the Webhooks option



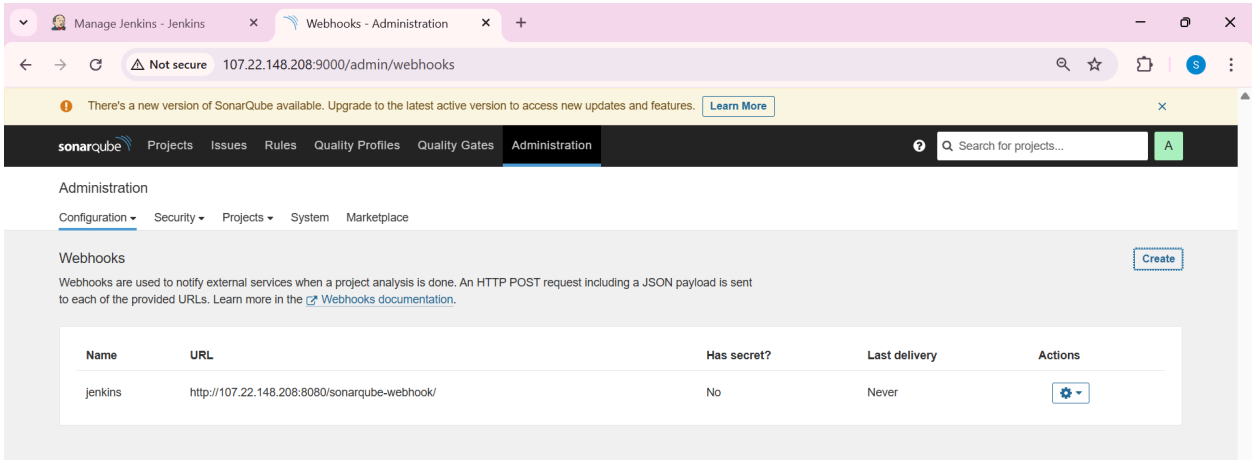
30. Choose Webhook and click Create



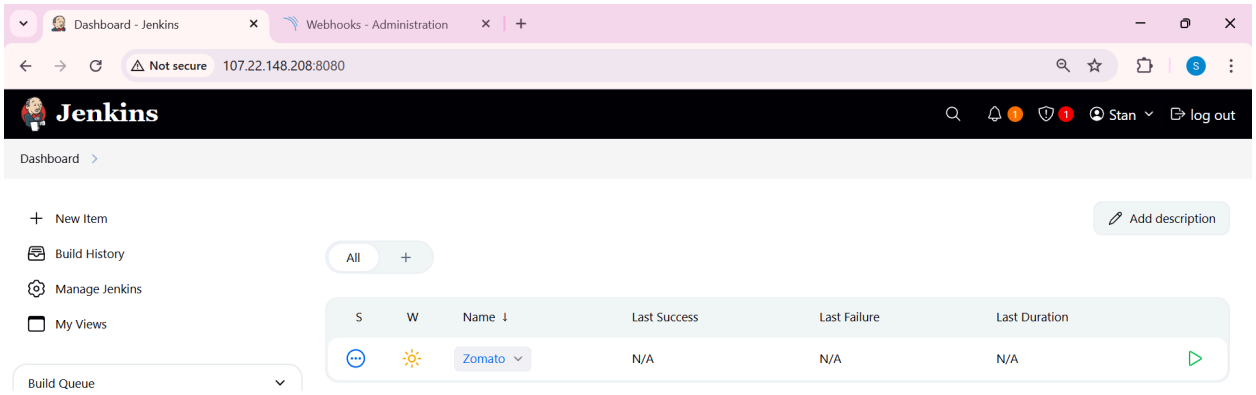
31. In pop-up, add name as jenkins and add URL as http://<public-ip>:8080/sonarqube-webhook



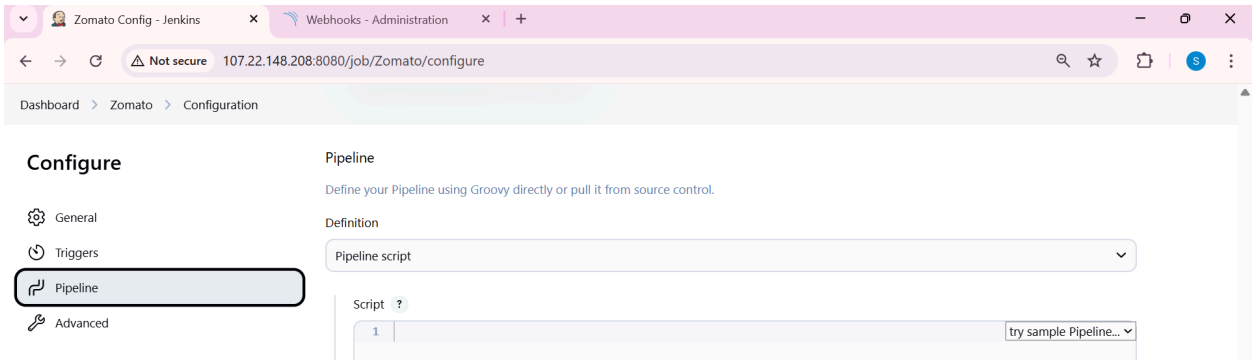
32. Now click create and see the Webhook n Dashboard



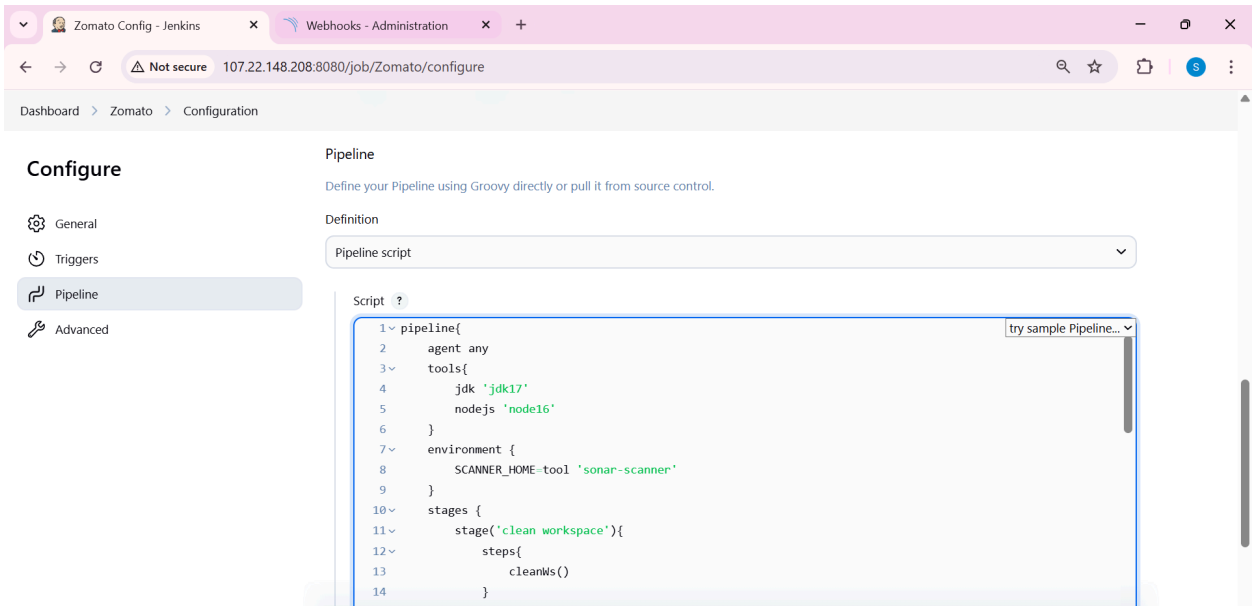
33. Now go back to Jenkins Dashboard, choose the Zomato Job

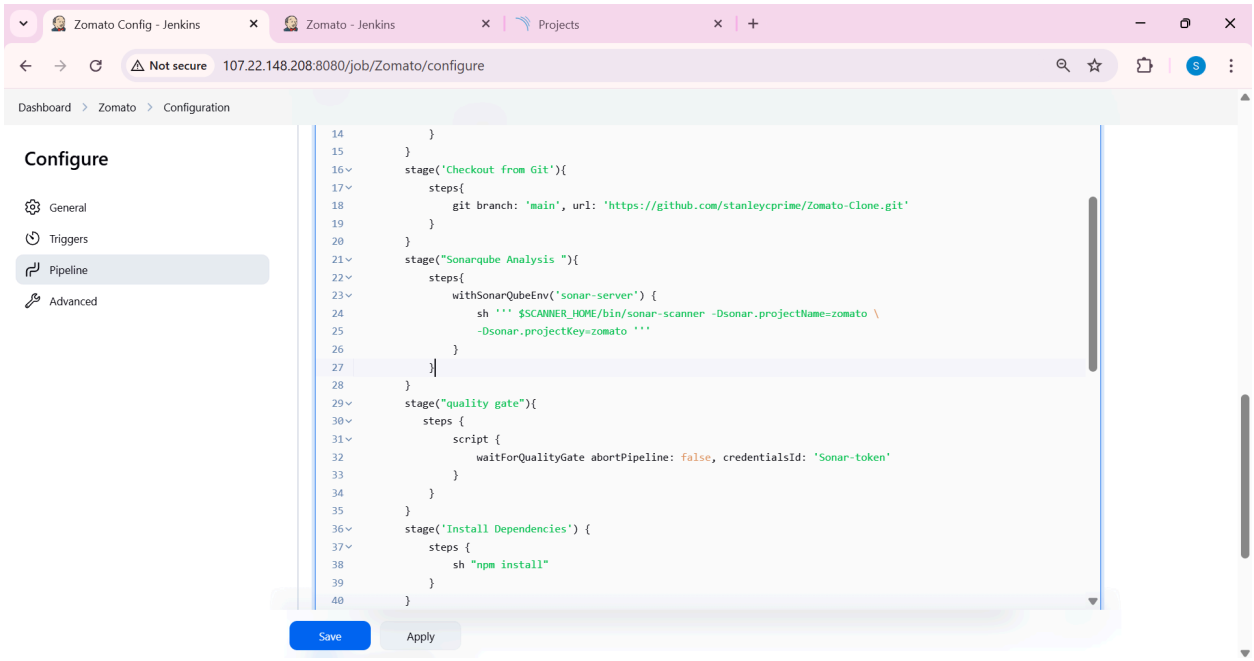


34. Inside Zomato, click Configure Now and search for Pipeline (or choose it)

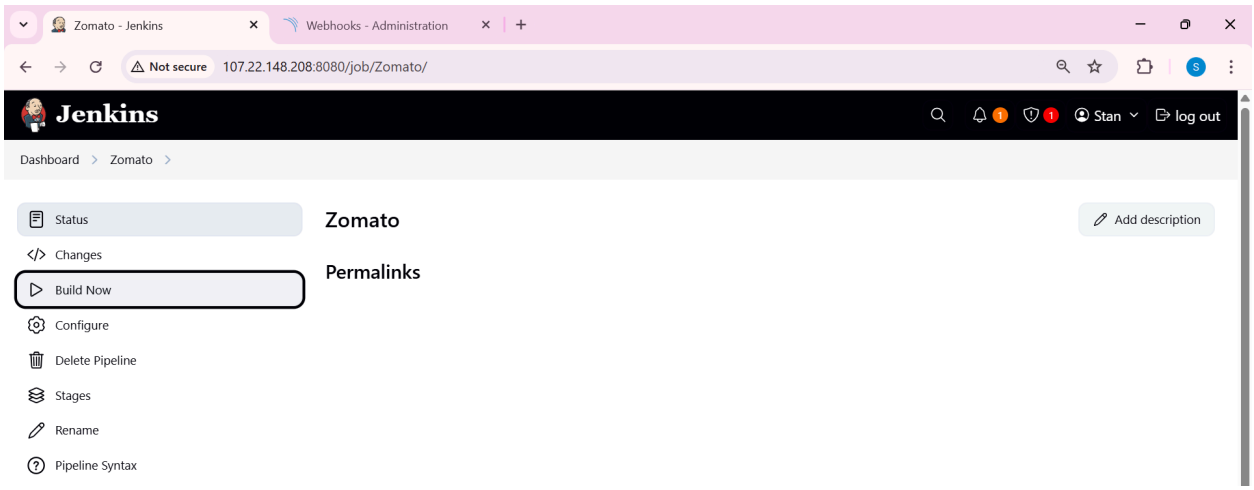


35. Add the pipeline script and press Apply and Save

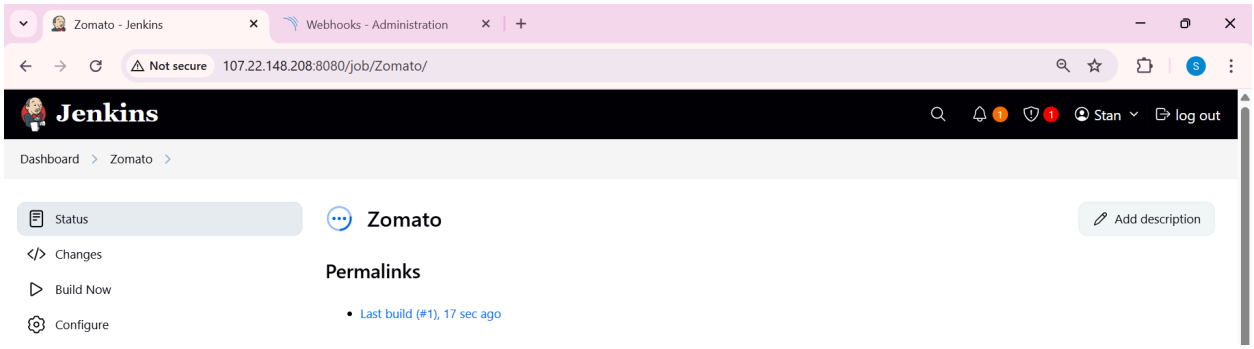




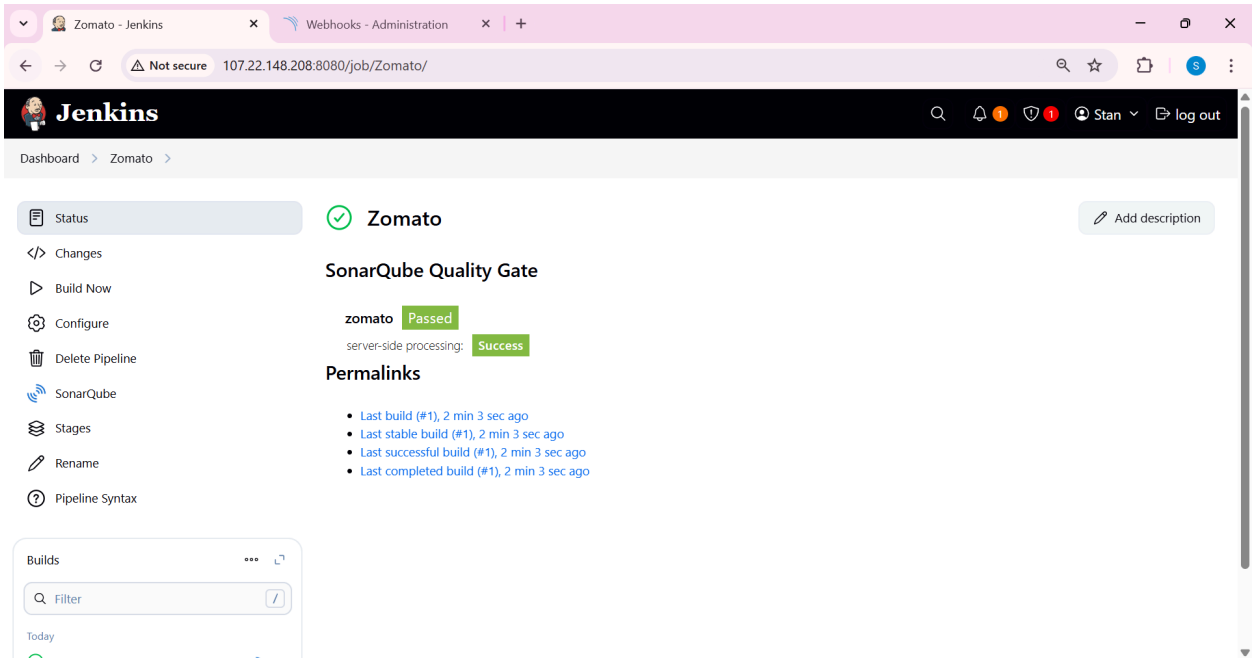
36. Now click Build Now (inside Zomato Job)



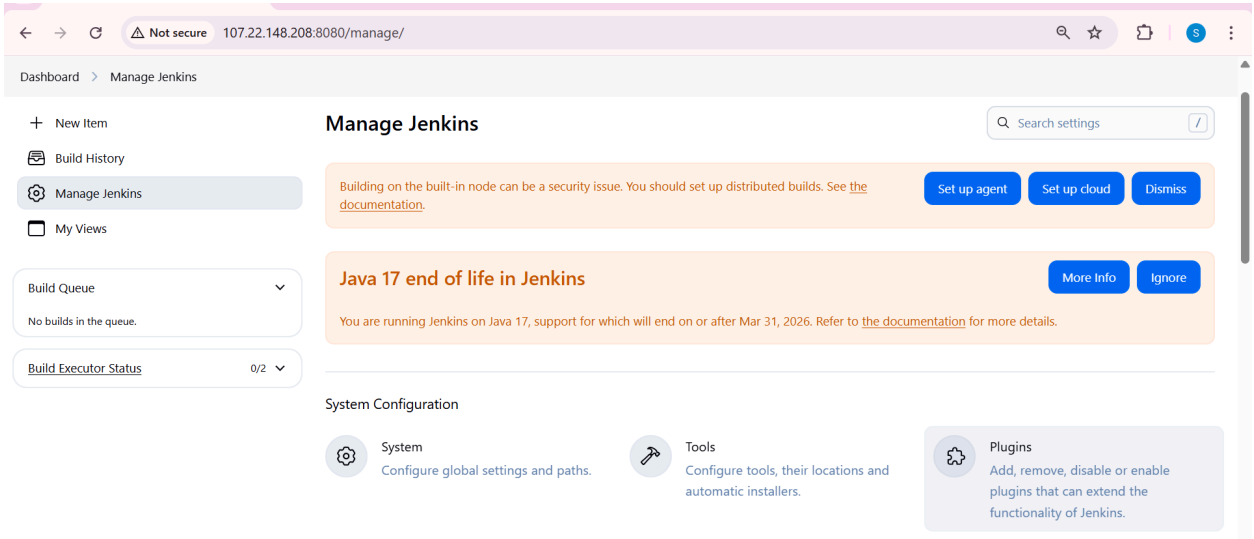
37. Wait for the building



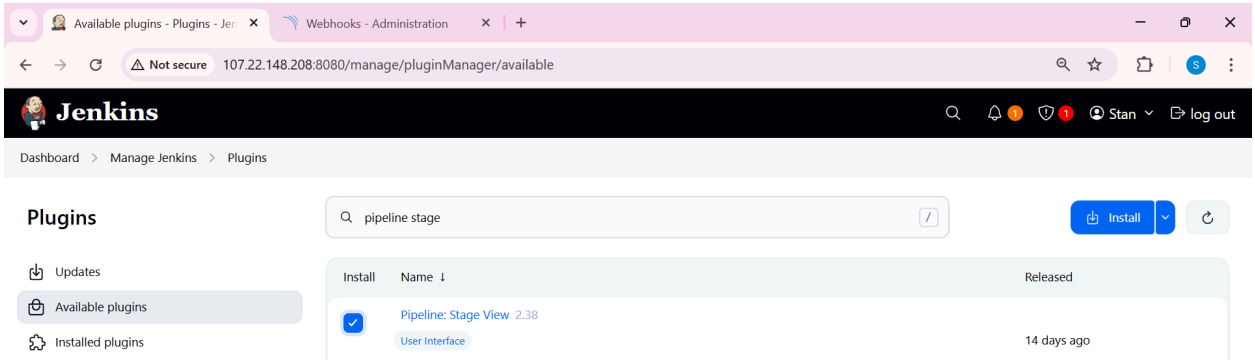
38. After the completion of Build, we see the success message



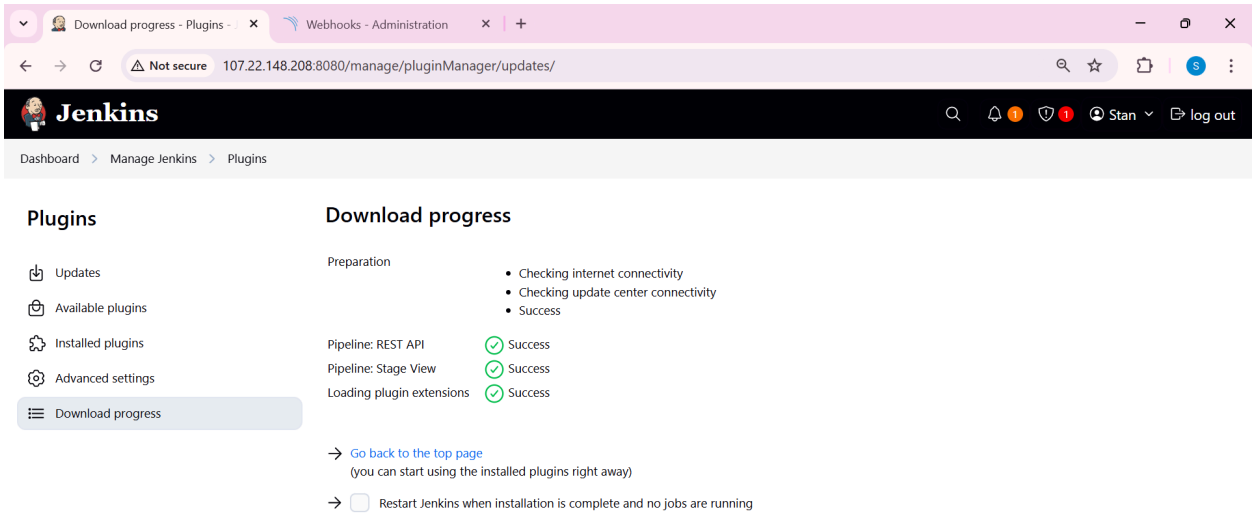
39. But to embed the Pipeline Stage View, do this
    a. In Manage Jenkins, go to Plugins



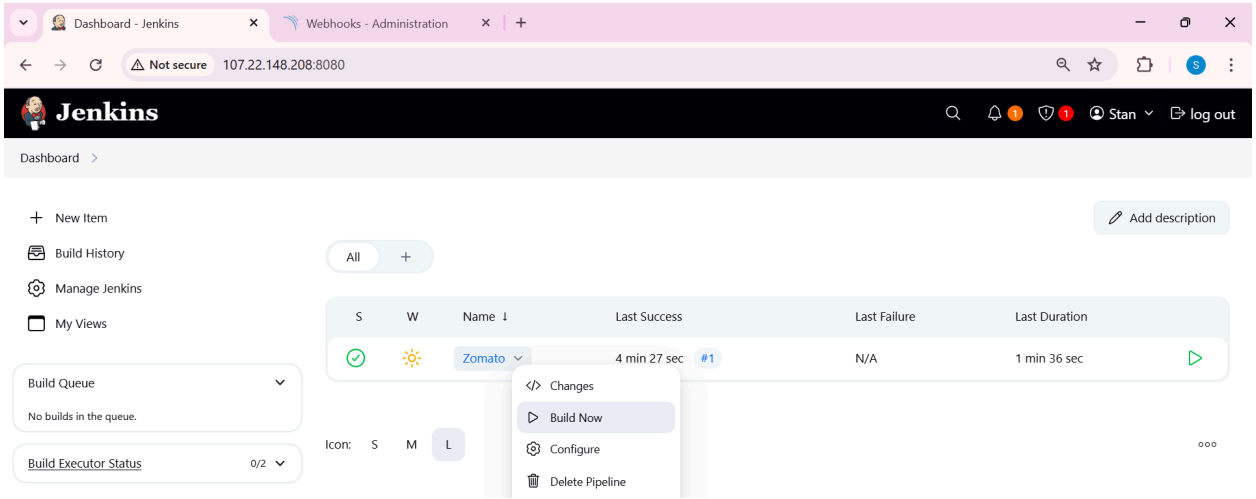
    b. Search for Pipeline Stage View and Install it



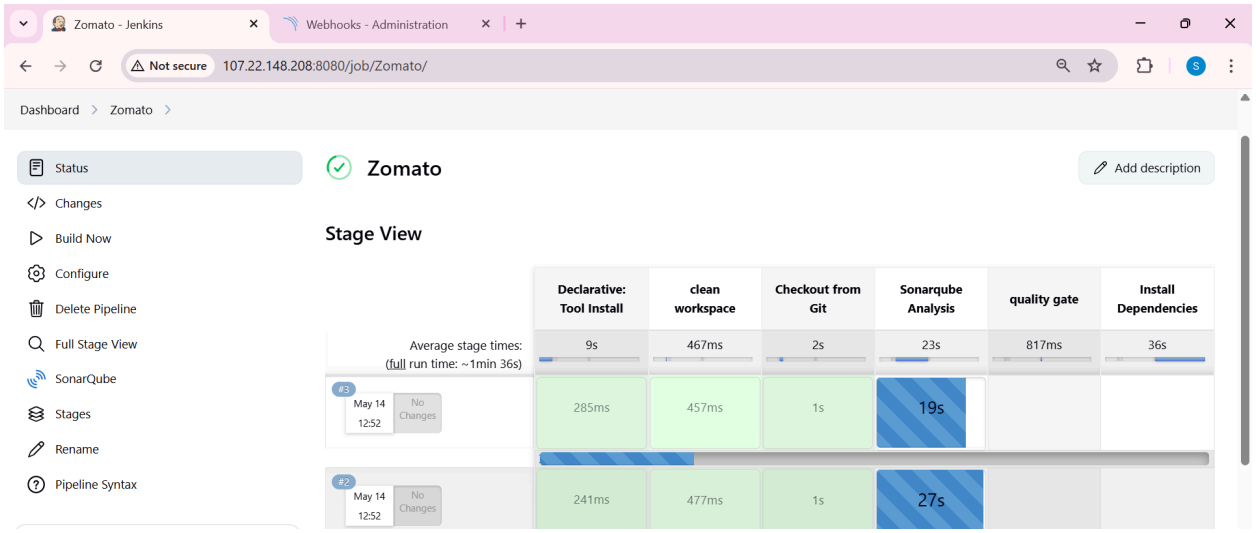
c. Now wait for the completion and go back to Dashboard



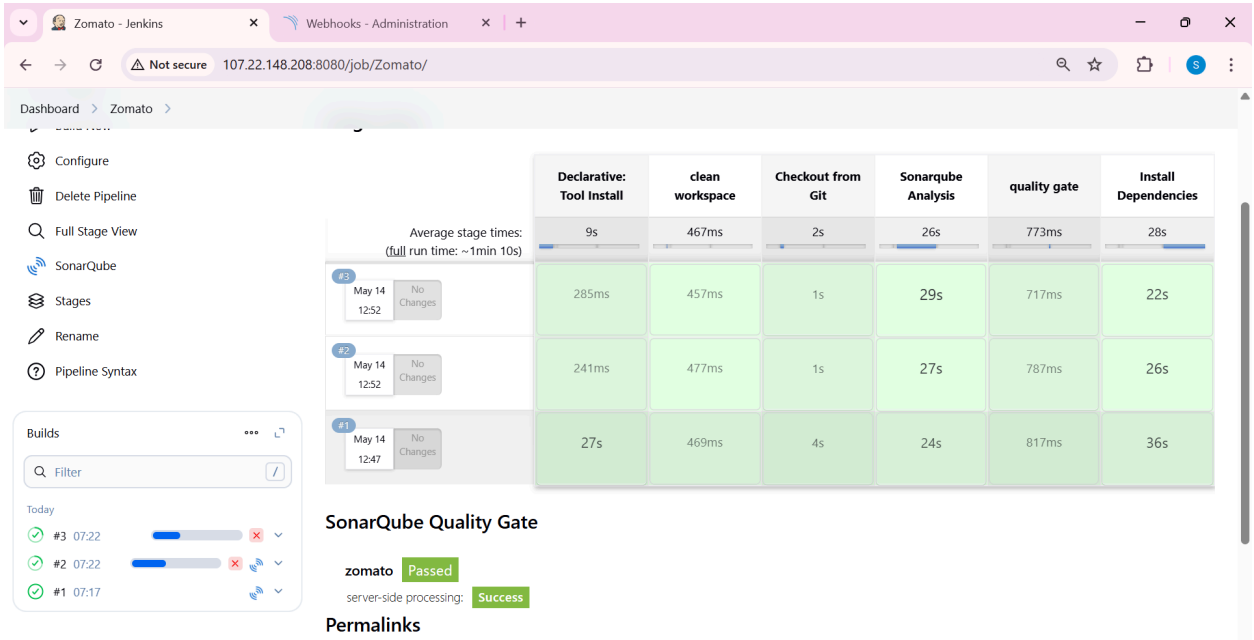
d. Now click Build Now for Zomato Job



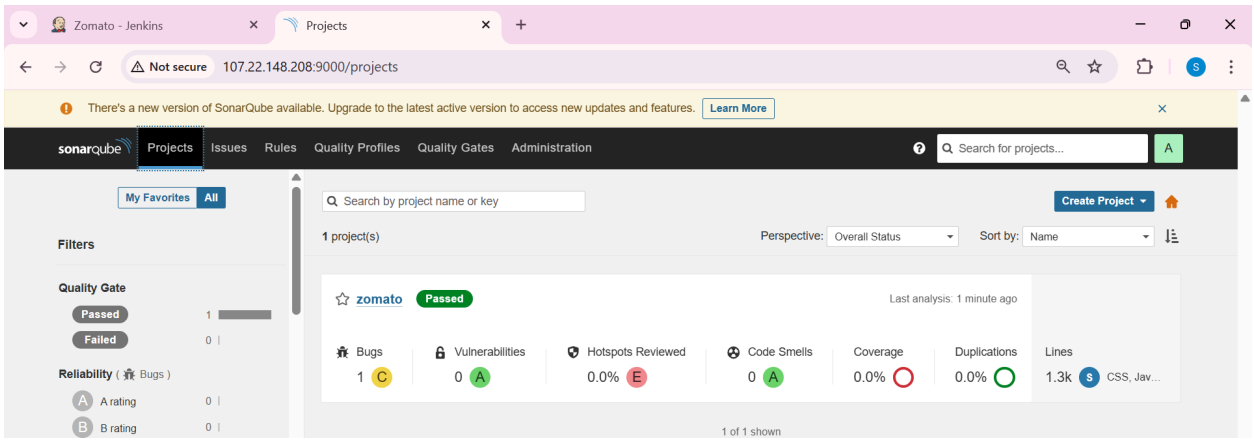
e. We see the Pipeline Stage View (stage wise) inside the Job



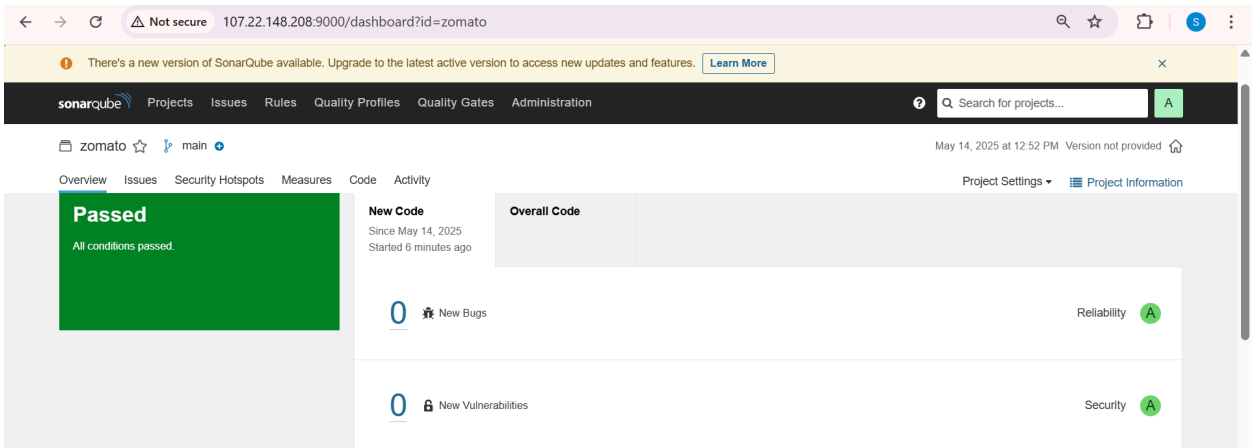
40. Now we see the Build is completed and the Quality Gate is Success



41. In SonarQube Dashboard, go to Projects and see the report

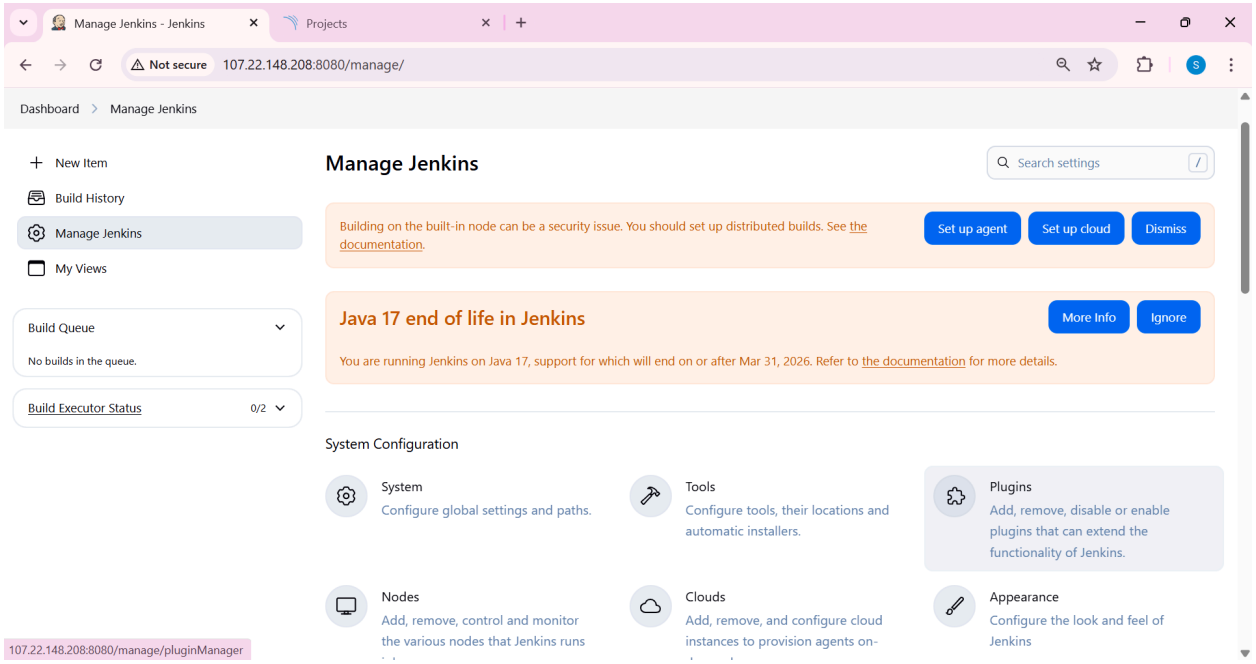


42. Choose the Project - Zomato, for further details

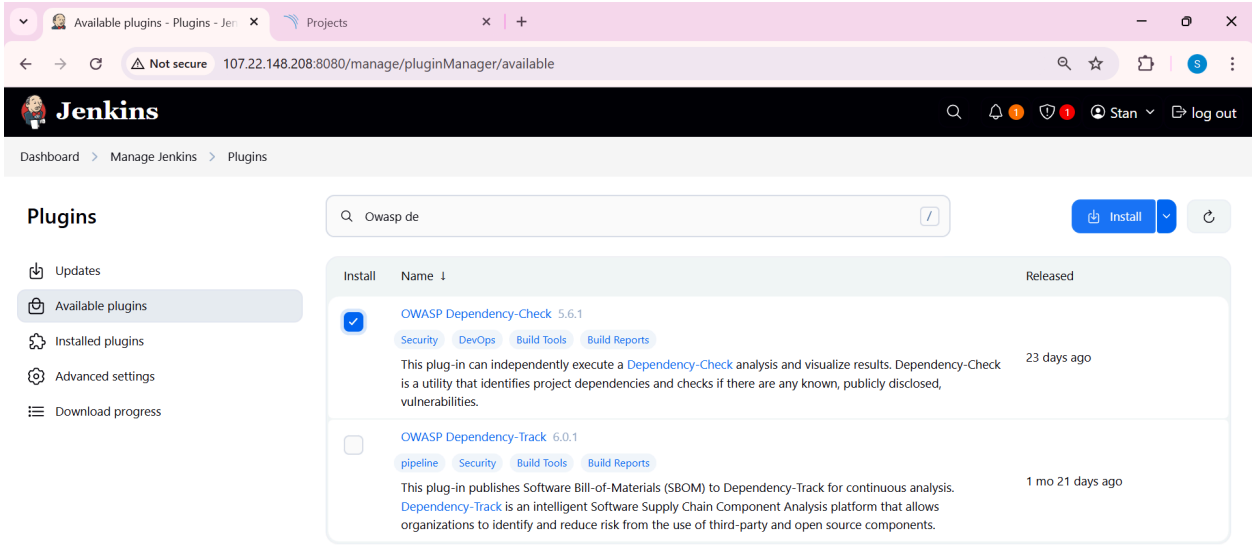


Step 5 : Install OWASP Dependency Check Plugins

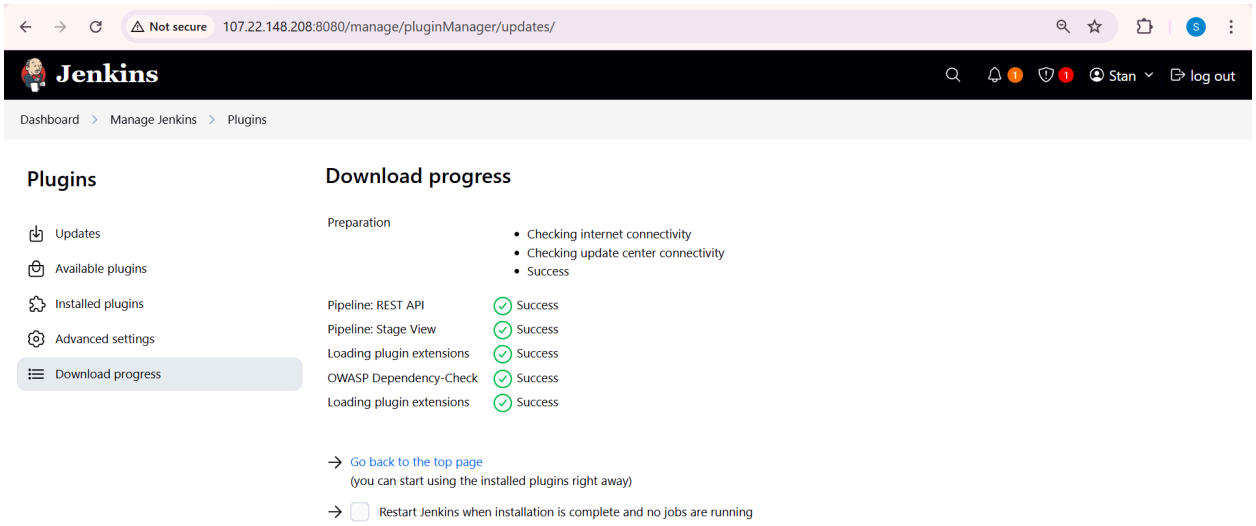
43. In Jenkins Dashboard, go to Manage Jenkins



44. Choose Plugins and install OWASP Dependency Check

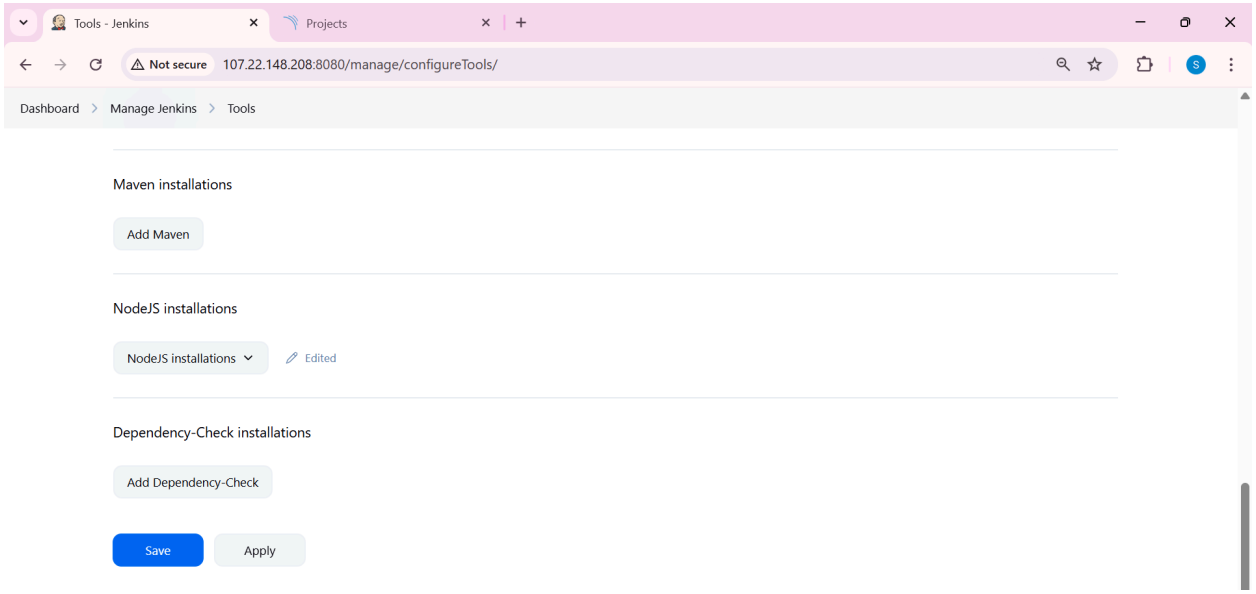


45. Wait for installation to complete and go back to Dashboard

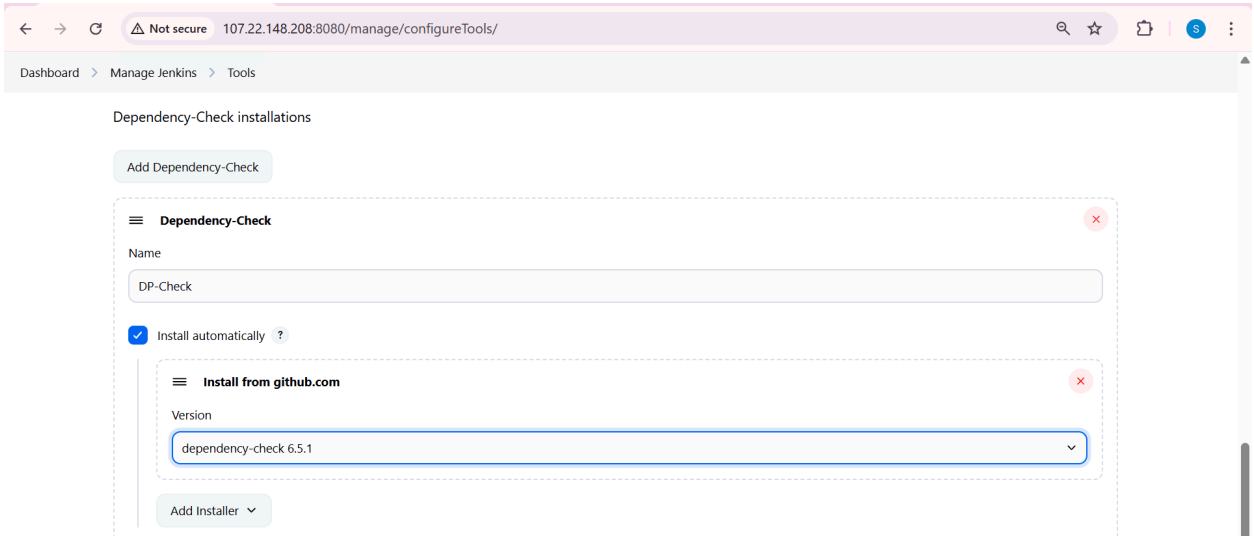


46. In Manage Jenkins, choose Tools

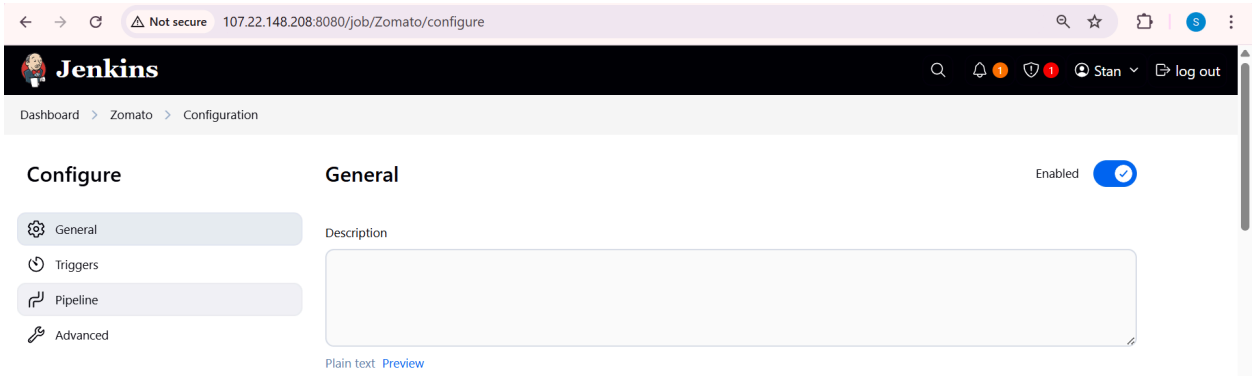
47. Search for Dependency Check Installations and click Add Dependency Check



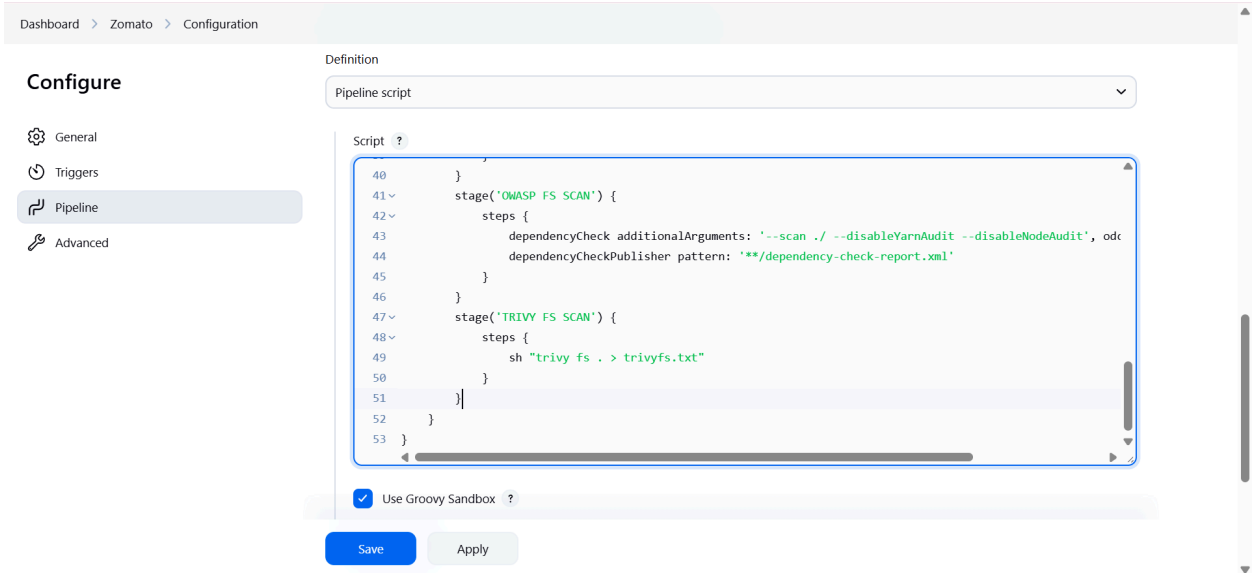
48. Add name as DP-Check and choose Install automatically
49. Select the desired version and click Apply and Save



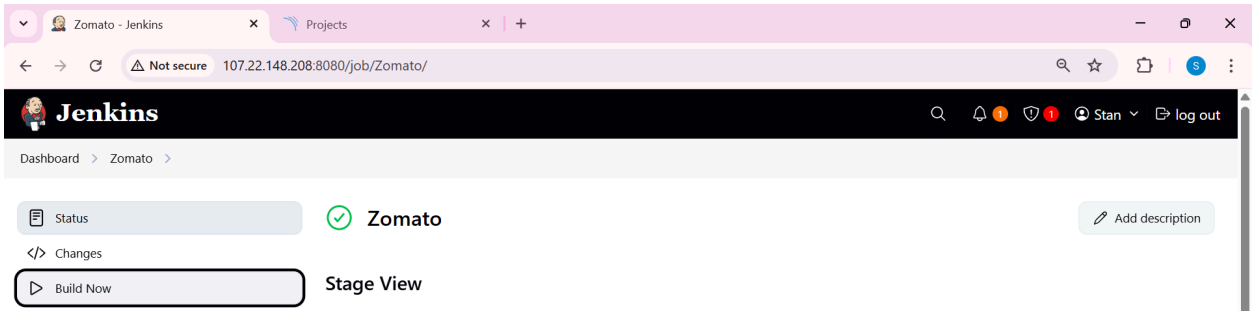
50. Now go back to Jenkins Dashboard and choose the Zomato Job
51. Click Configure and choose Pipeline option (or scroll down)



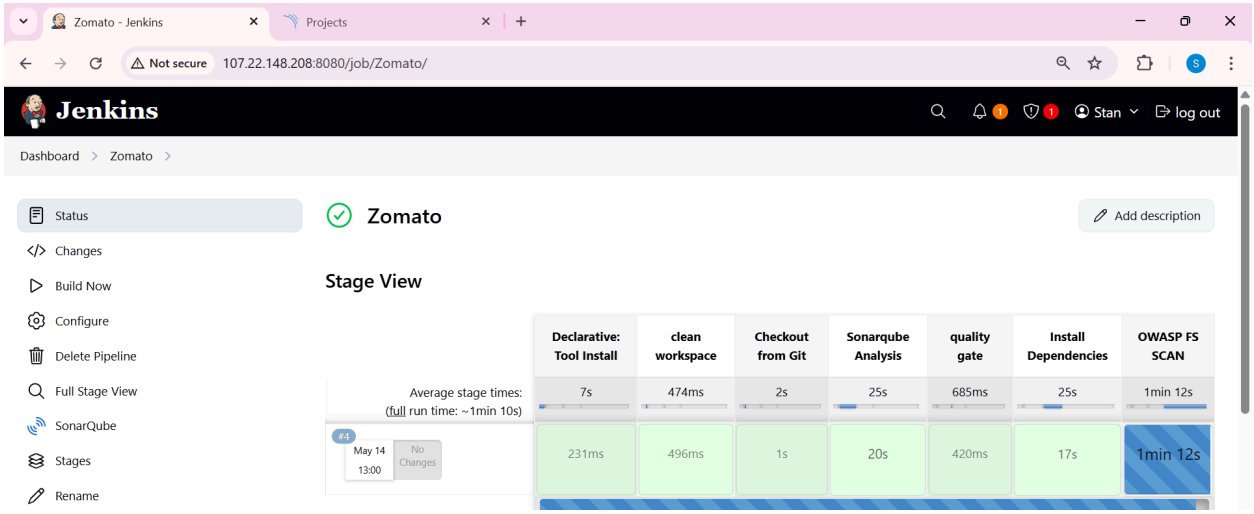
52. Here add code in Pipeline Script to implement OWASP Scan



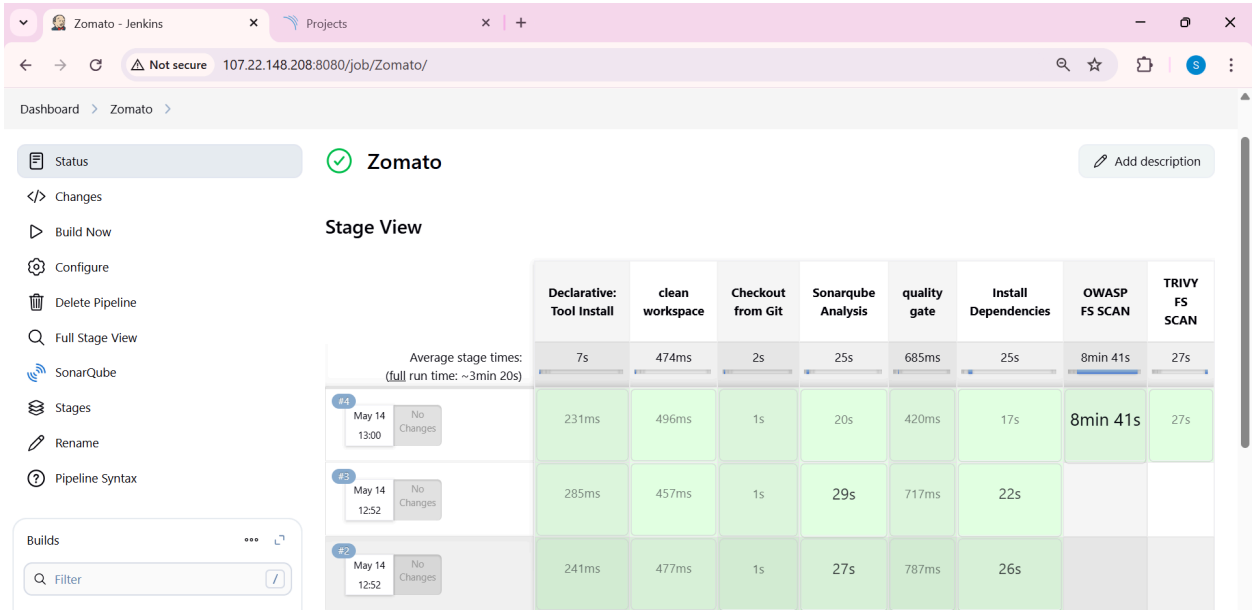
53. Now choose the Zomato Job and click Build Now

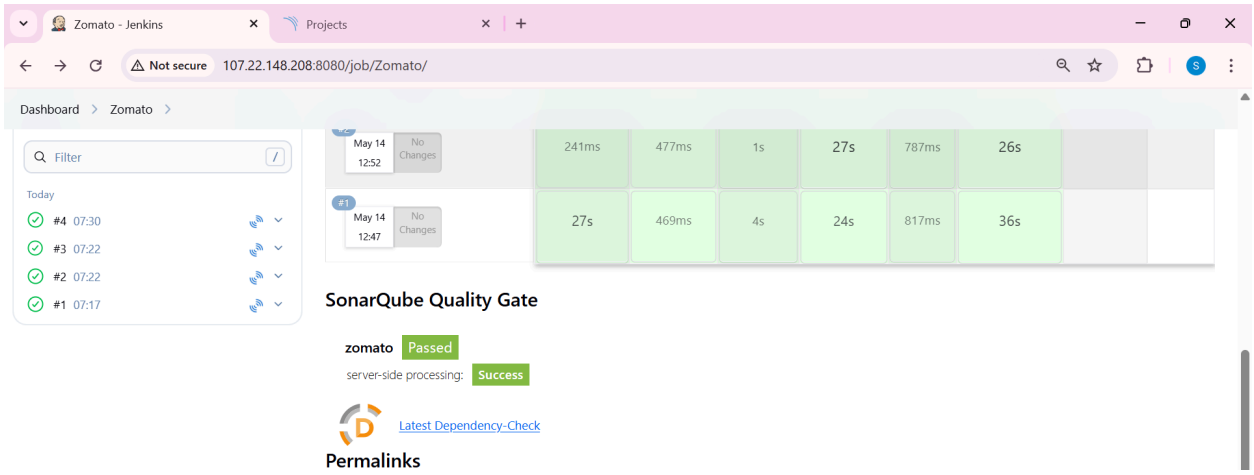


54. Wait for the Build to be completed while viewing the stage progress in Pipeline View

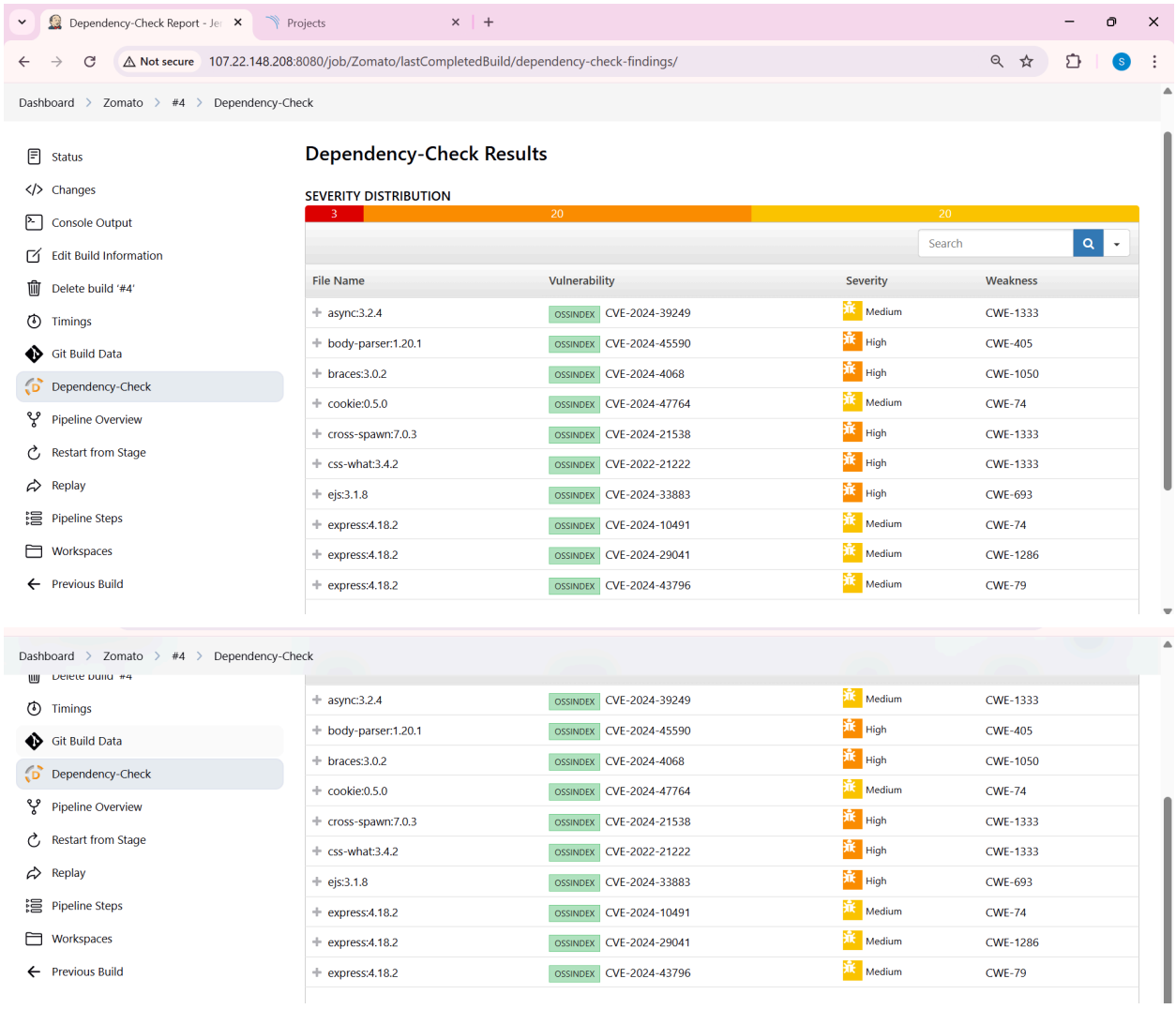


55. Here we see the build is completed





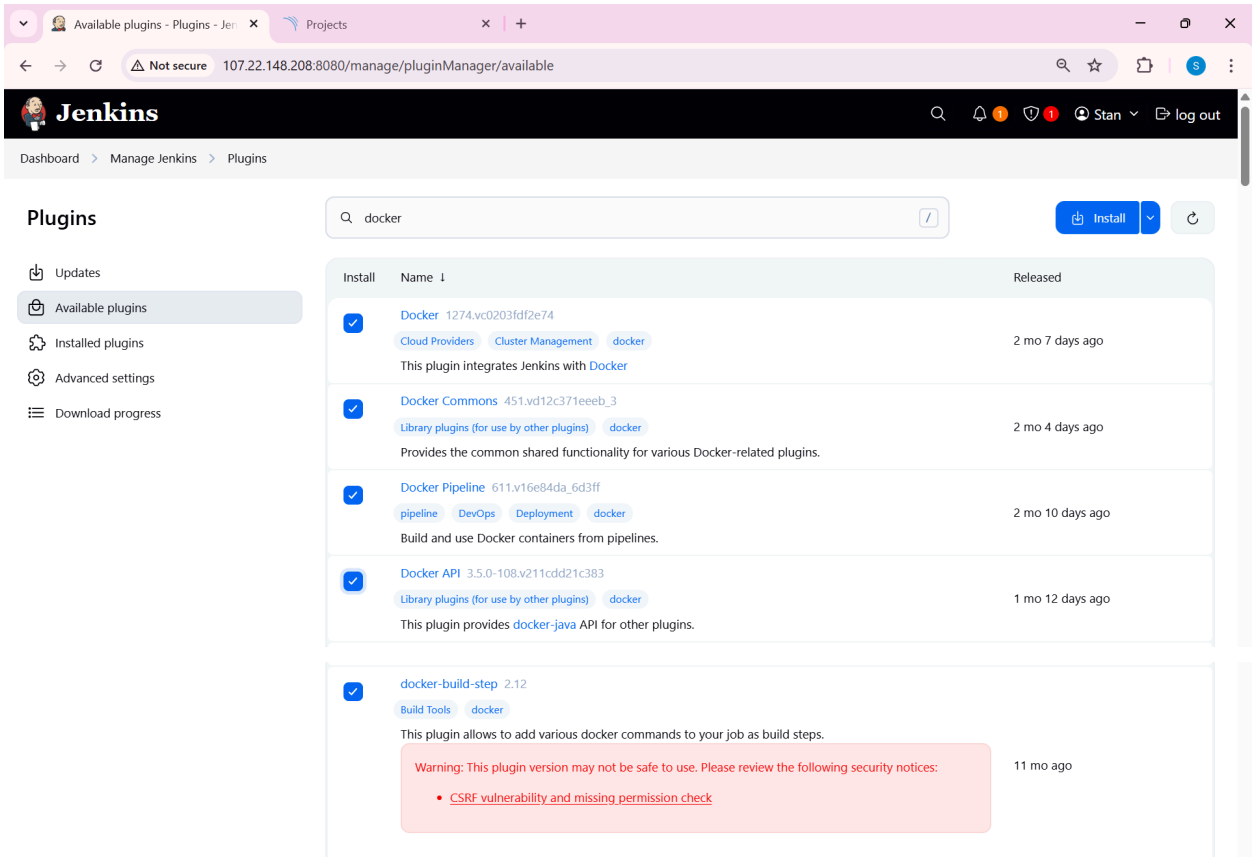
56. Here click the Latest Dependency Check option which shows the Results (Also available in Graph based manner)



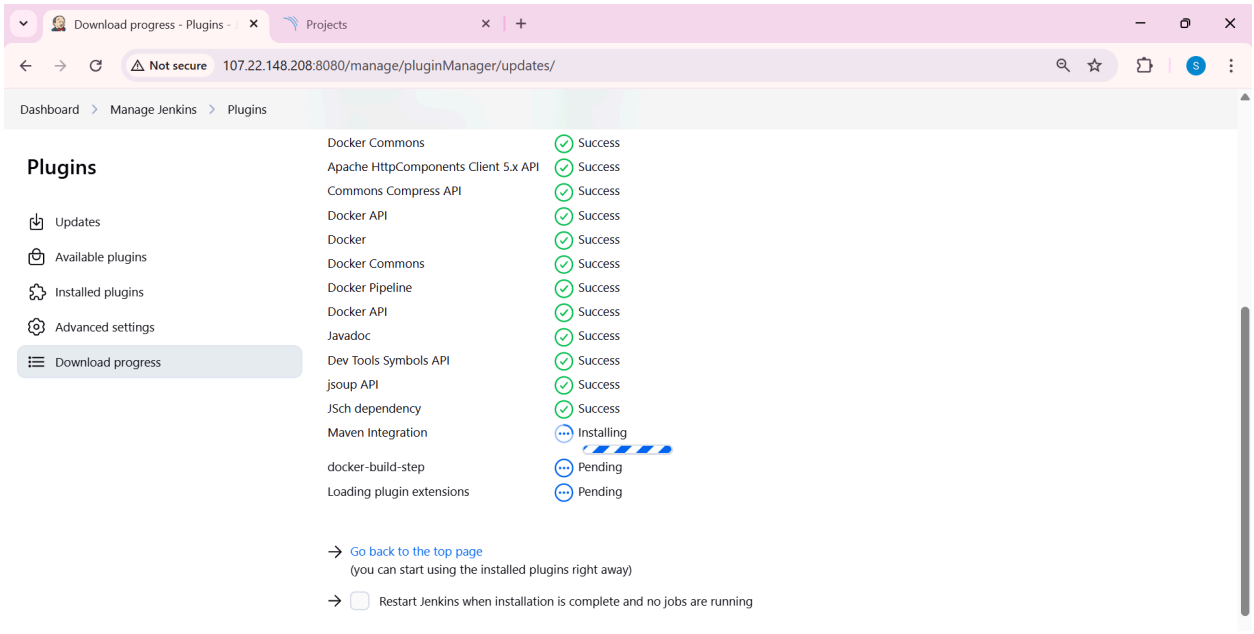
57. It shows Vulnerability and Severity with other details

Step 6 : Docker Image Build and Push

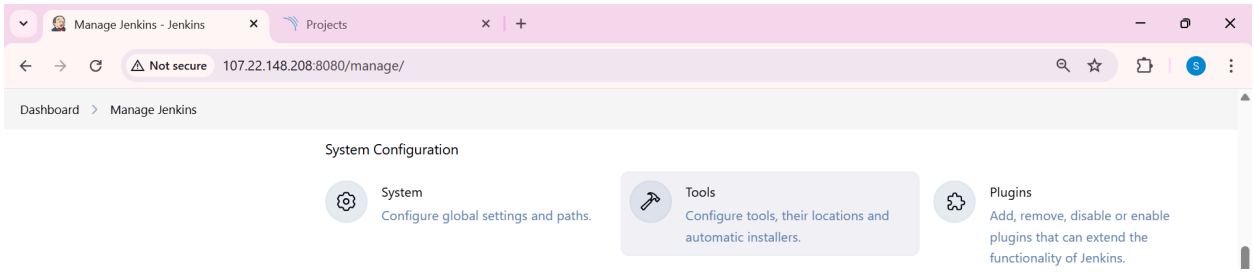
1. In Jenkins Dashboard, choose Plugins in Manage Jenkins
2. Add various plugins such as
    a. Docker
    b. Docker Commons
    c. Docker Pipeline
    d. Docker API
    e. docker-build-step



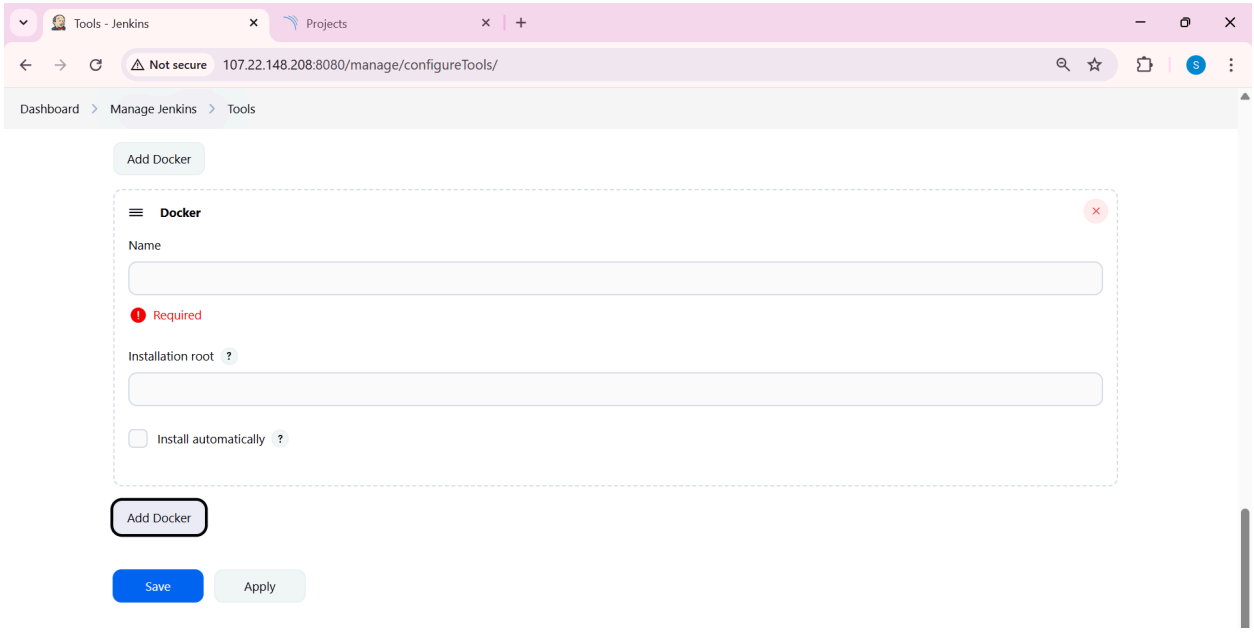
3. Install them and go back to dashboard (once completed, without restart)



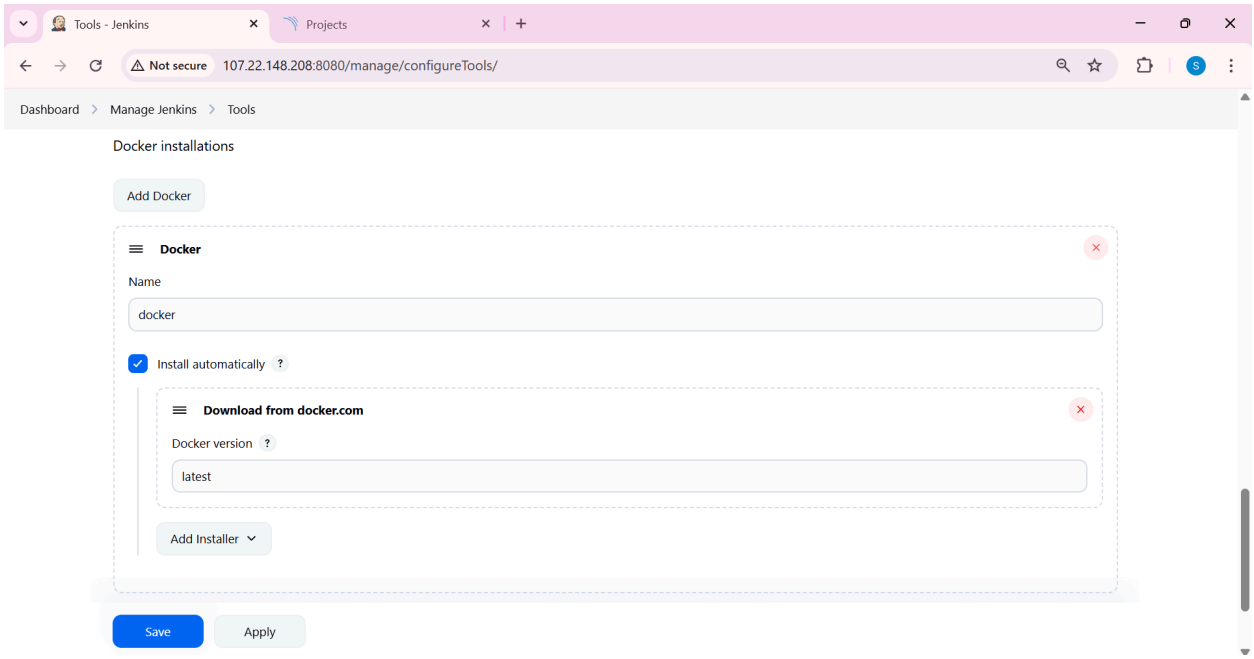
4. Now go to Tools in Manage Jenkins



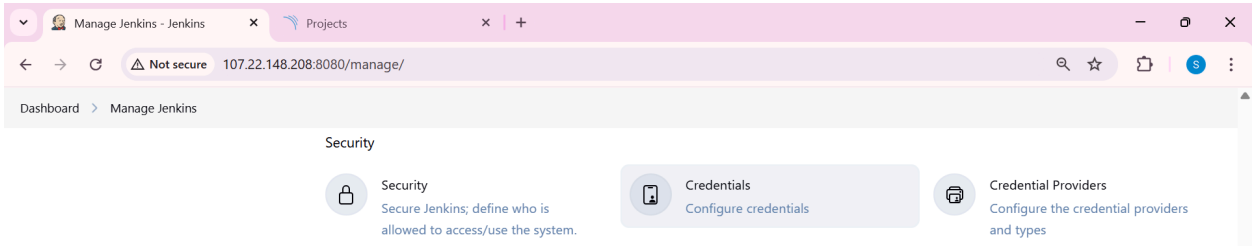
5. Scroll and search for Docker Installations and click Add Docker



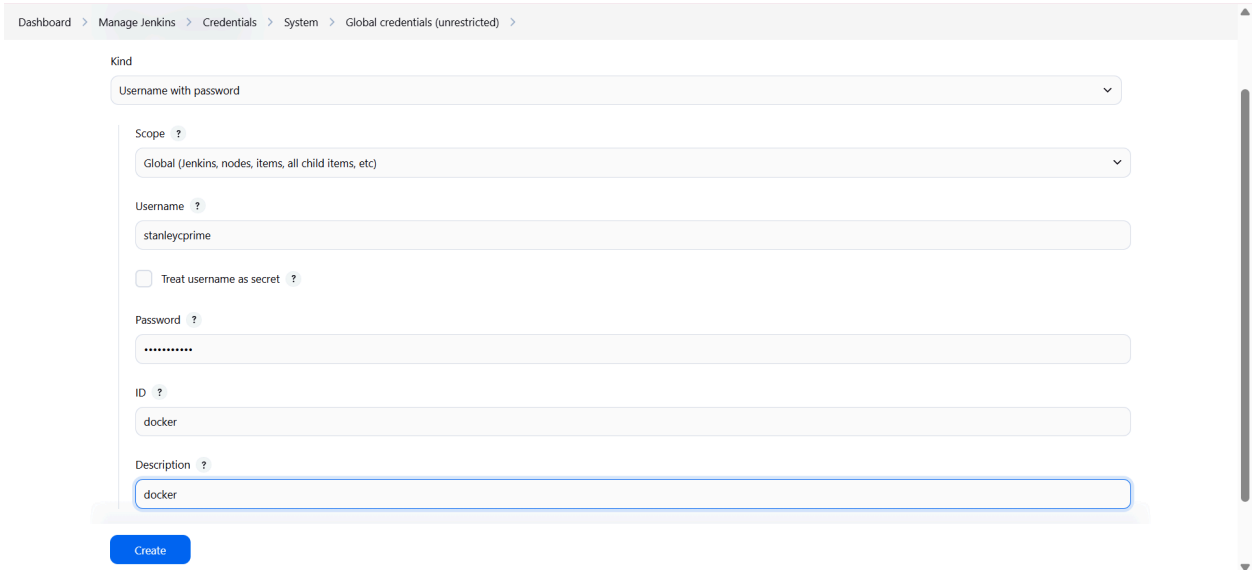
6. Add name as docker and click Install automatically while selecting the desired version



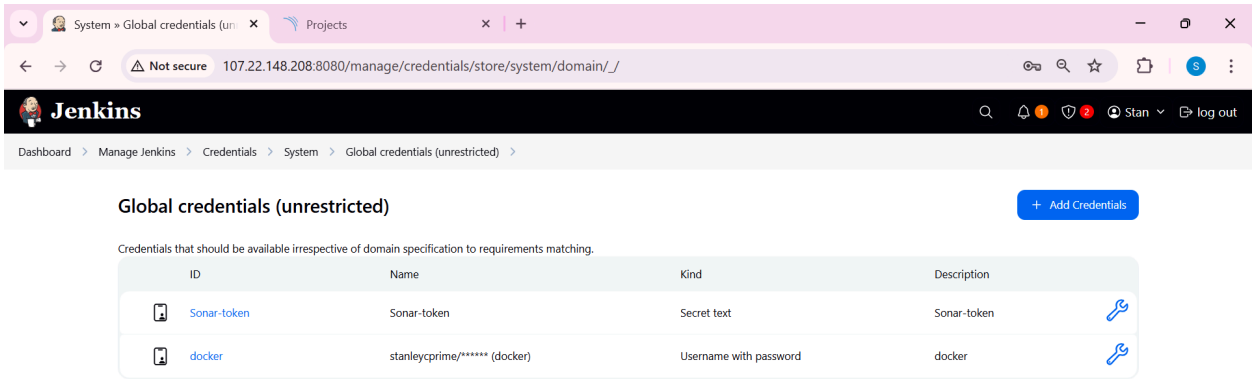
7. Now choose Credentials in Manage Jenkins



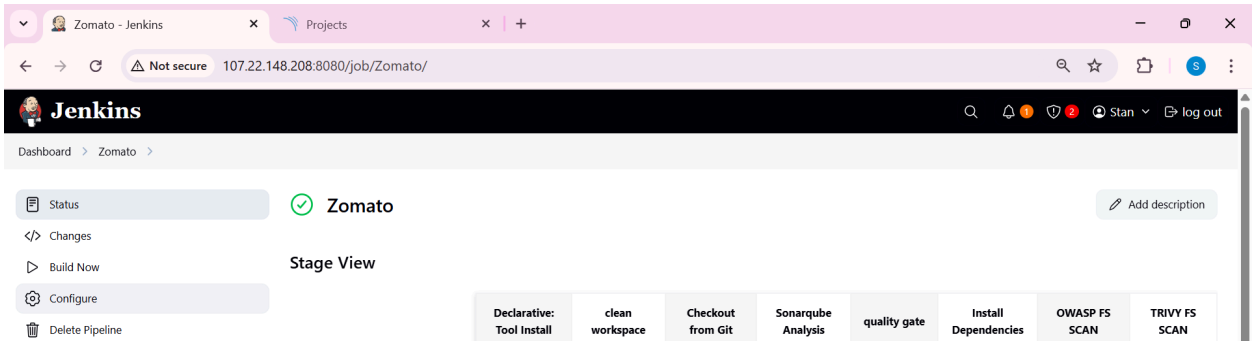
8. Choose Username with password and pass the username and password from DockerHub
9. Add ID and Description as Docker and click Create



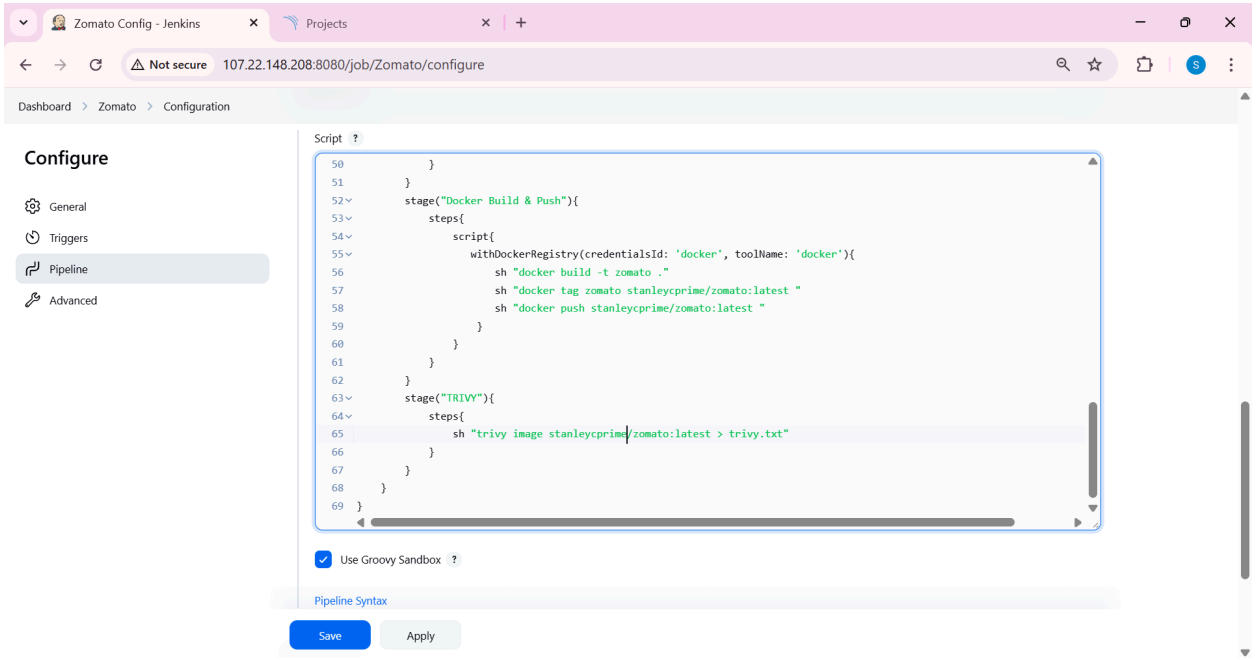
10. Now we see the Credentials we created in Global Dashboard



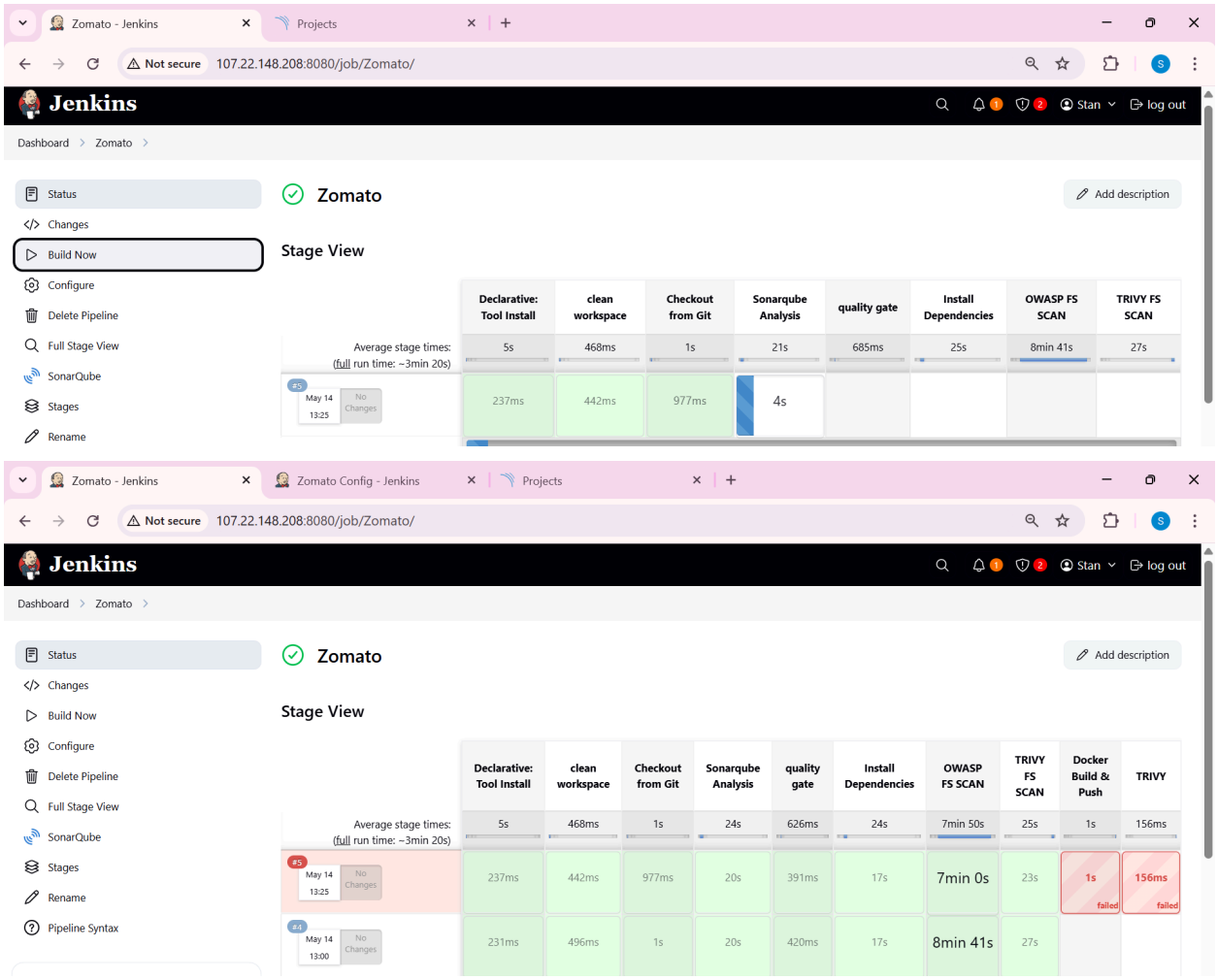
11. Now go back to Jenkins Dashboard and choose the Zomato Job



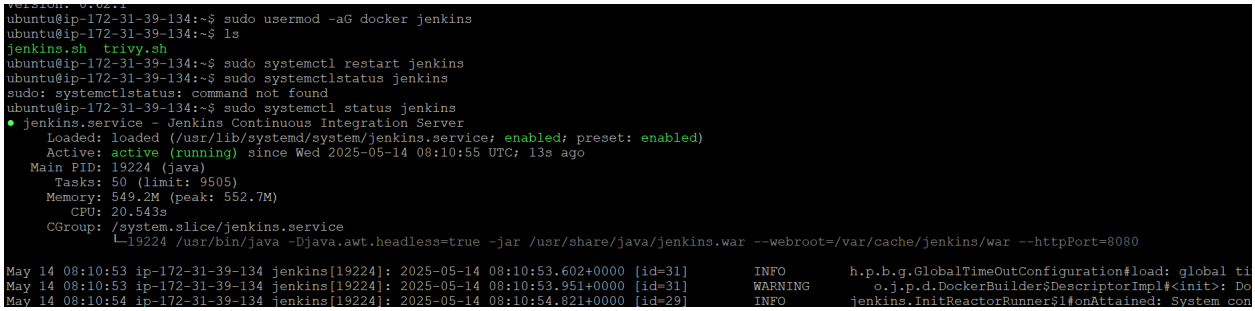
12. Choose Configure and go to Pipeline and add the code to implement Docker Build and Push stage
13. Now click Apply and Save



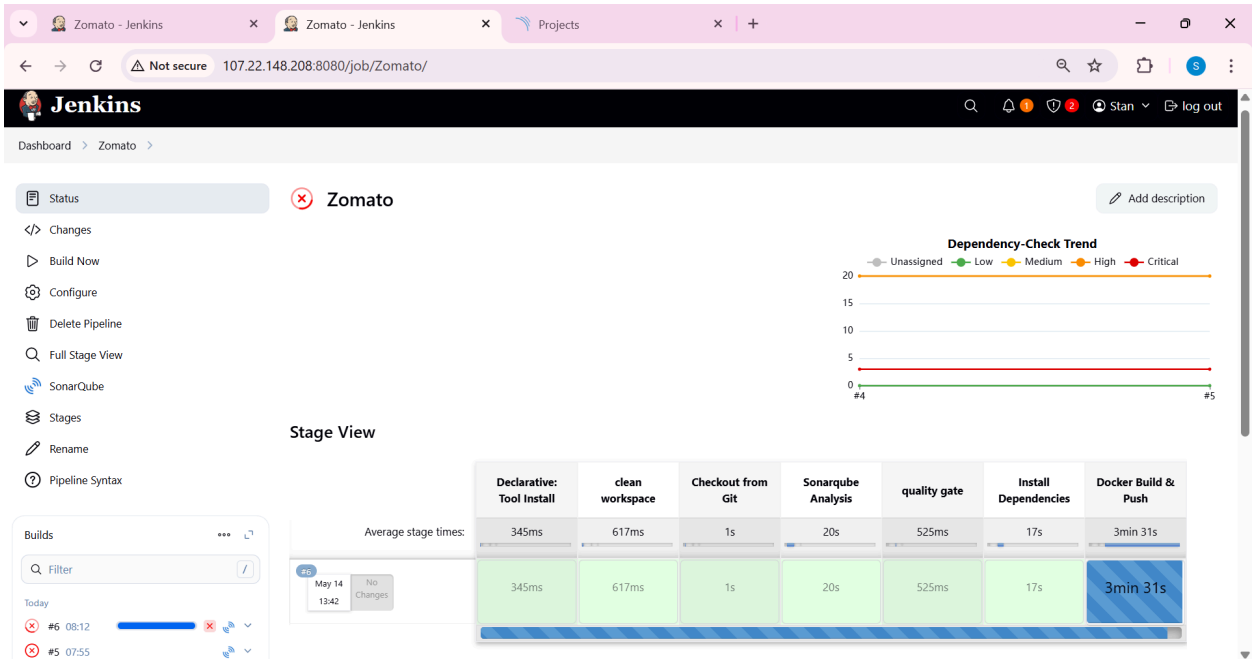
14. Click Build Now for the Zomato Job
15. We see the Pipeline progress here



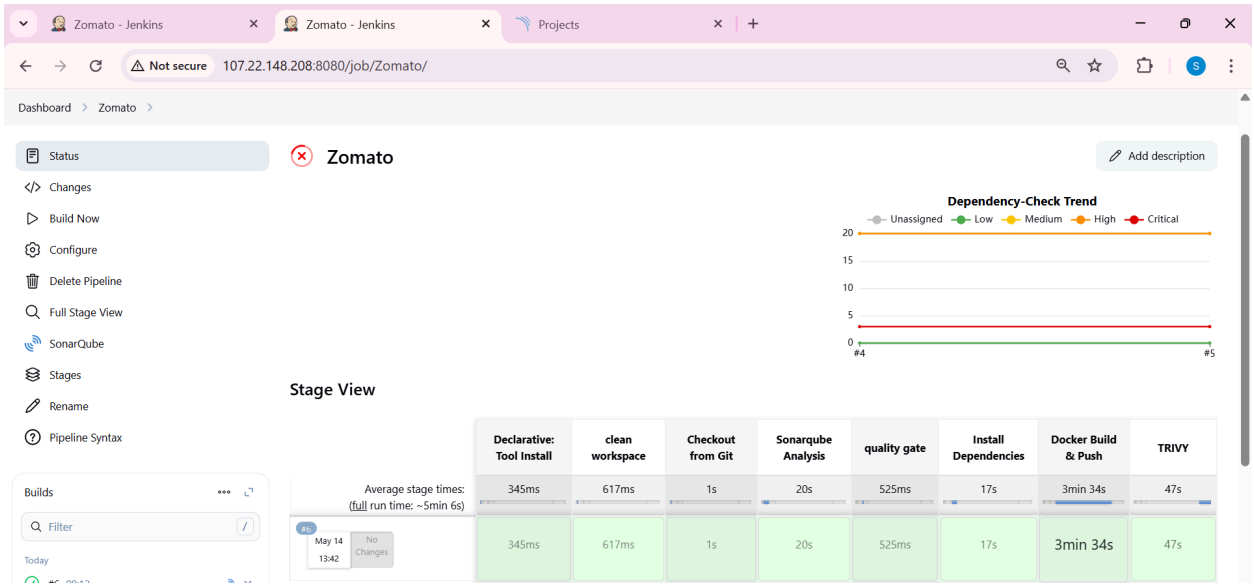
16. Here we encounter an Error
    a. We see the Docker Build and Push stage got failed
    b. This is because we need jenkins to be installed within docker group



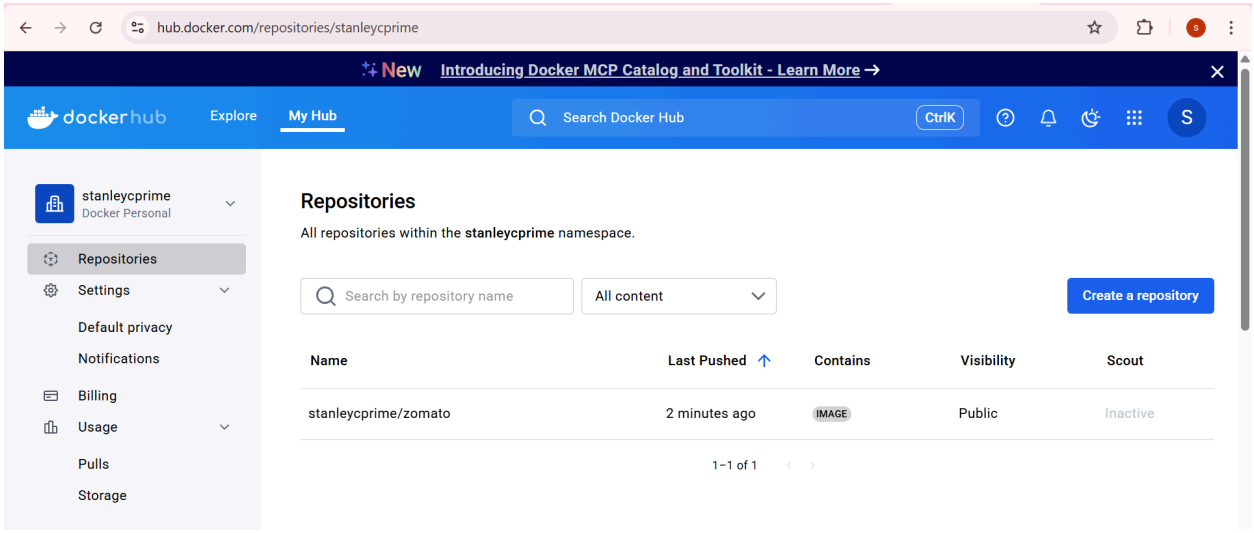
17. Now rebuild the Job with the same Script



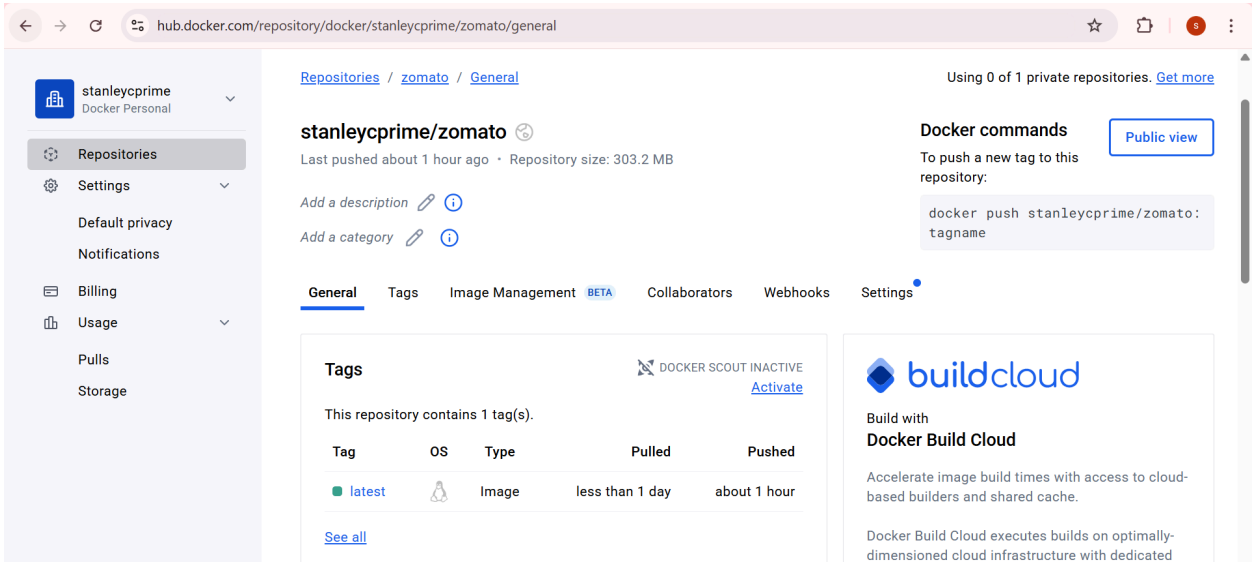
18. Now we can see the build is complete and we can also see the Dependency check Trend Graph embedded during the Build Process



19. We can ensure this by going to Repositories in DockerHub, where we stanleycprime/zomato repository



20. We open them, to see for further details

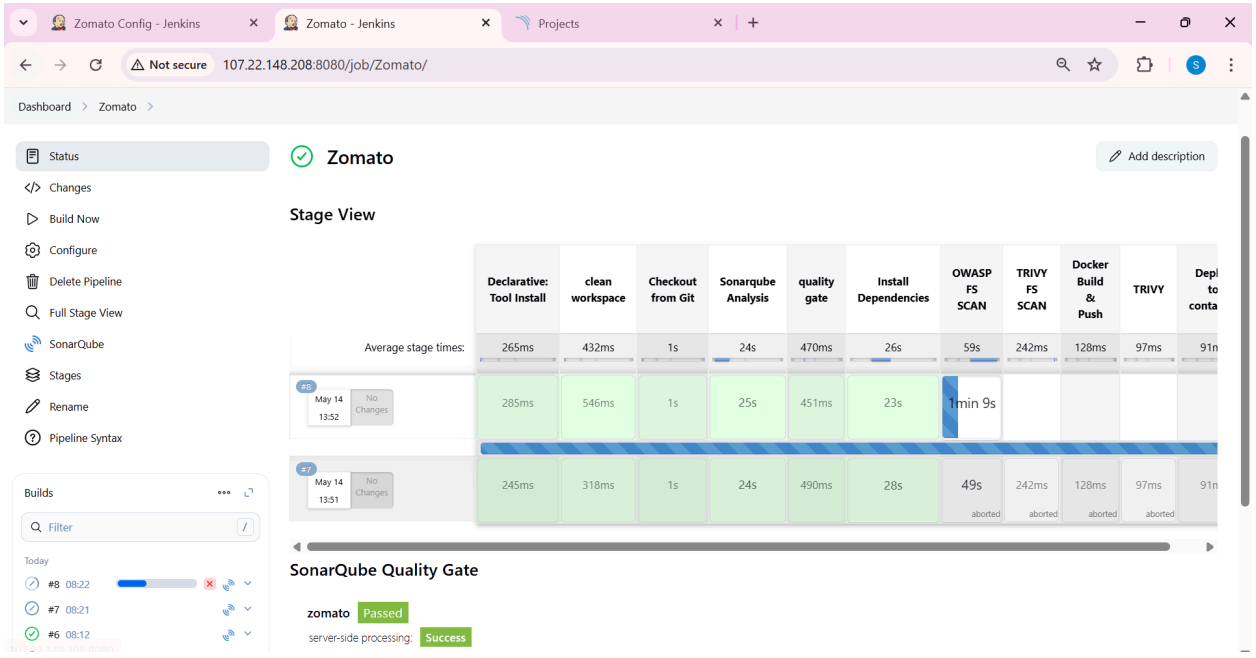


Step 7 : Deploy the image using Docker

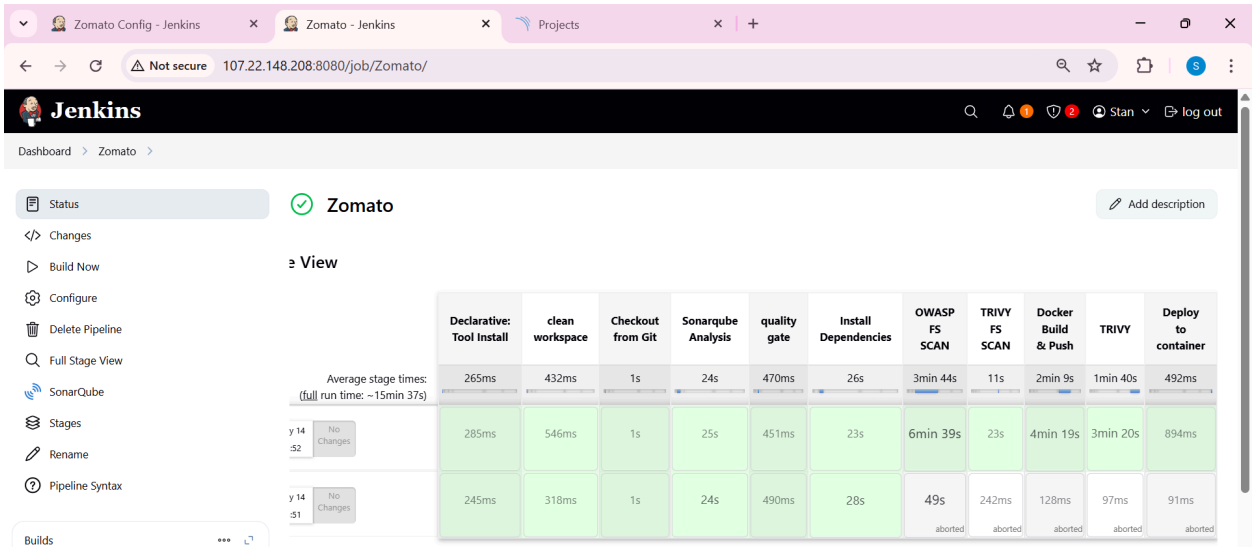
1. To implement the Deployment in pipeline script, we go back to Jenkins Dashboard and choose the Zomato Job
2. Choose Pipeline inside Configure Tab, where we insert the code to deploy to container stage

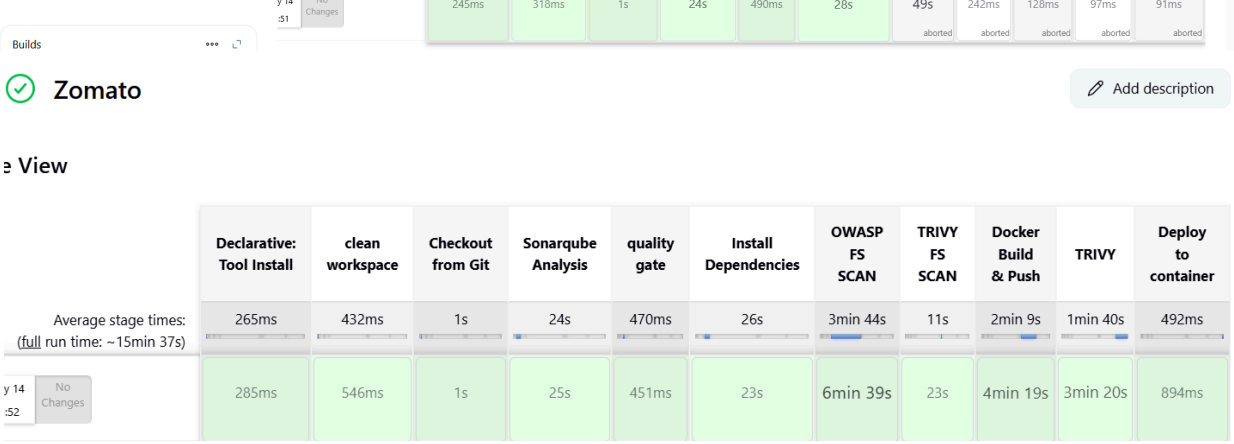
```
        }
        stage("TRIVY"){
            steps{
                sh "trivy image stanleycprime/zomato:latest > trivy.txt"
            }
        }
        stage('Deploy to container'){
             steps{
                    sh 'docker run -d --name zomato -p 3000:3000 stanleycprime/zomato:latest'
            }
        }

    }
}
```

3. Now Build the Zomato Job and see the pipeline progress

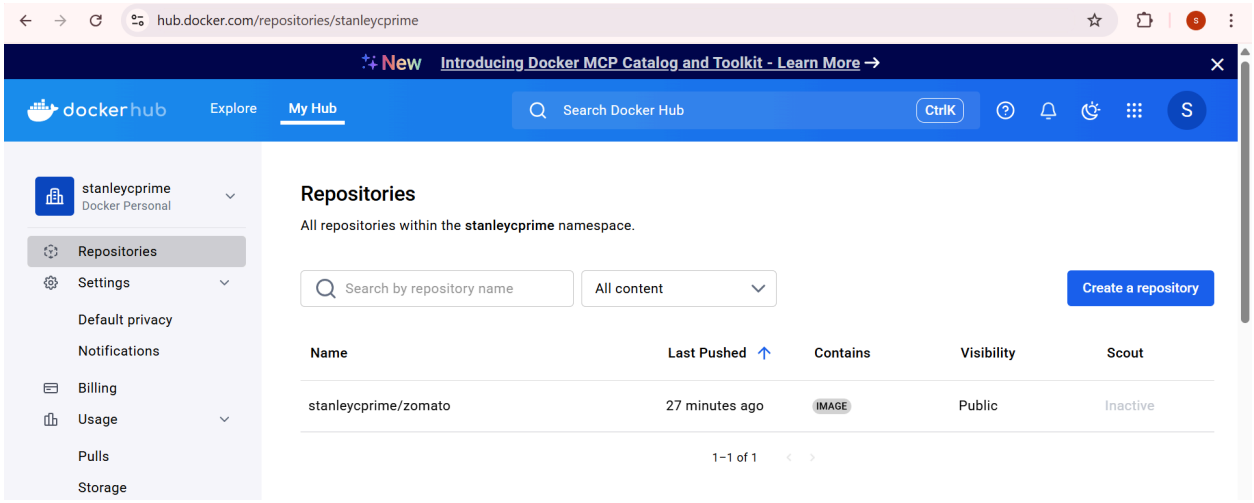


4. Now we see the pipeline is completed

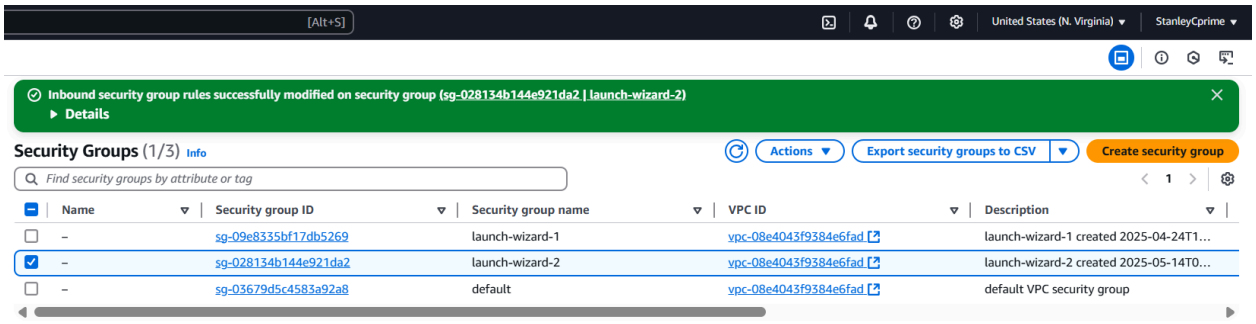


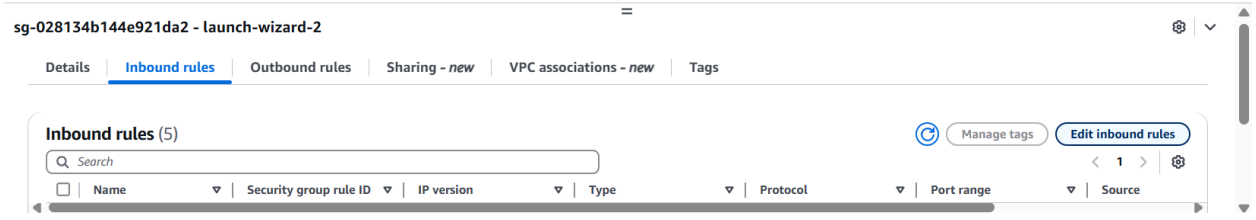


5. Ensure them by opening the DockerHUb repositories

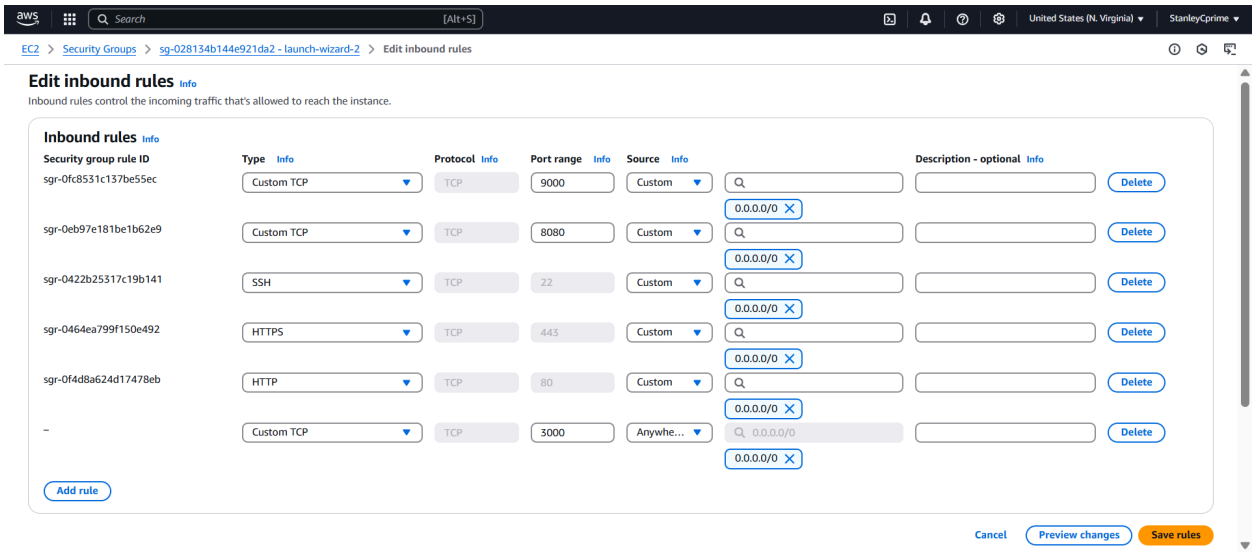


6. To open the Deployed application in Browser, we need to add port 3000 to Security Group. To do this:
    a. Choose the Security Group which we used for this instance
    b. Click Edit Inbound rules

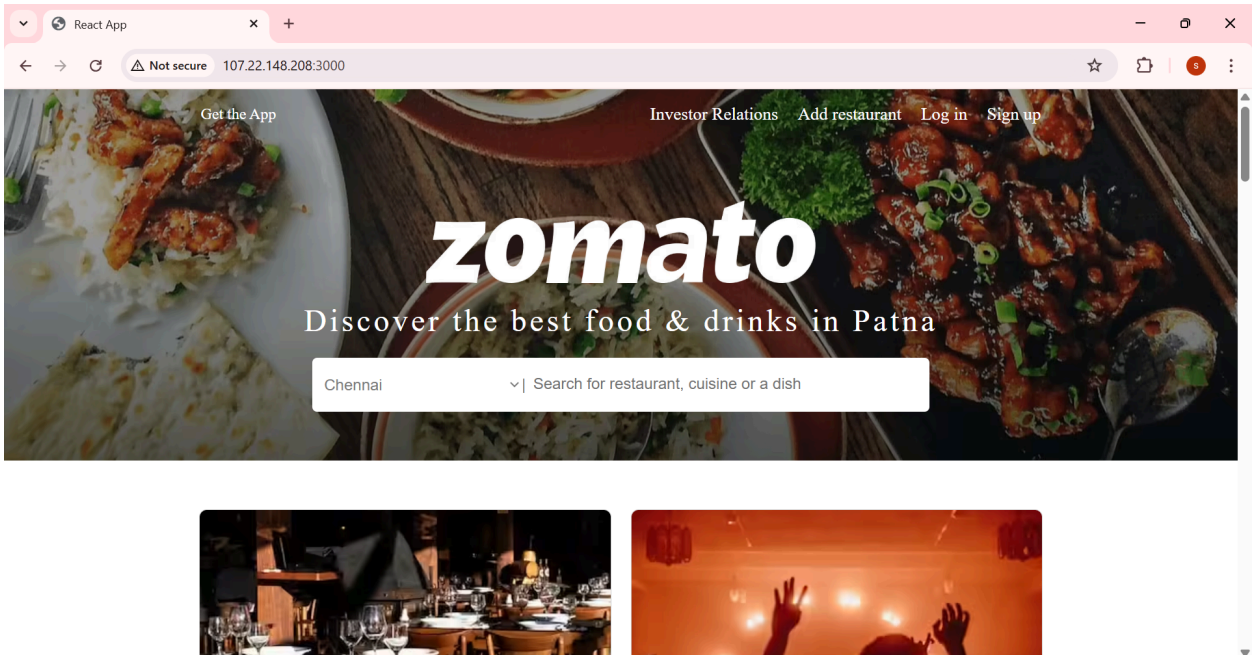


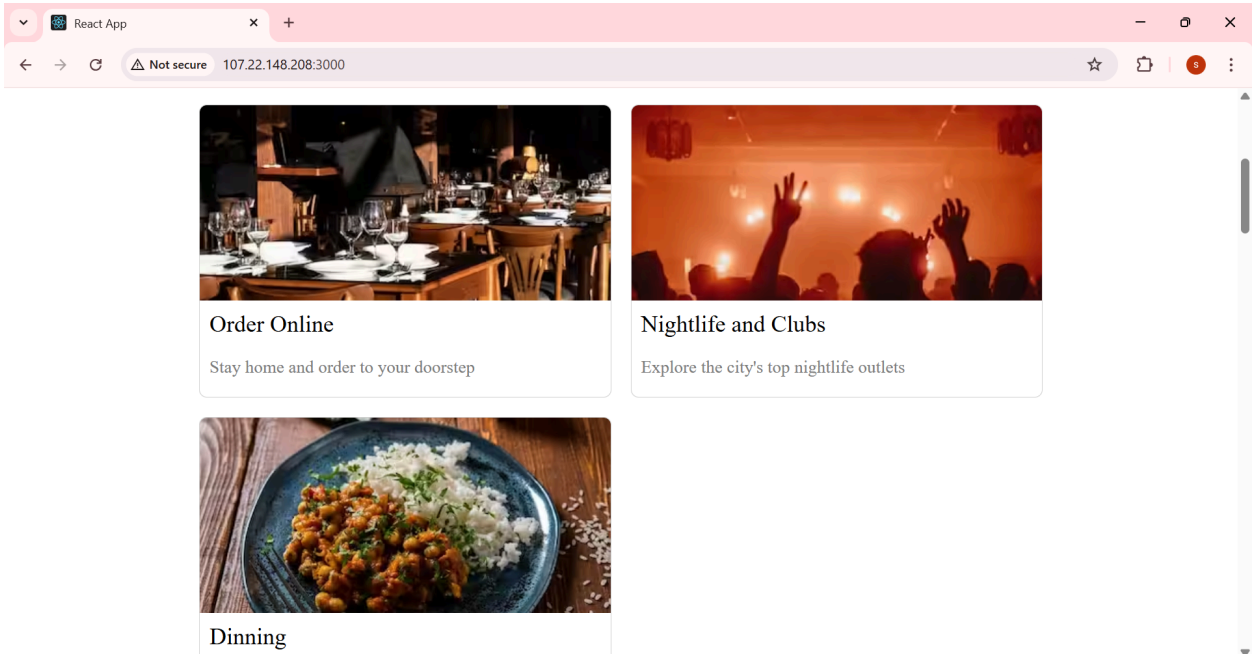


    c. Add port 3000 as Custom TCP with open subnet and click Save rules



7. Now open the Application using http://<public-ip>:3000

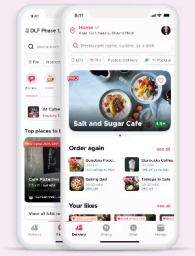




Popular localities in and around Ahmedabad

- Bodakdev — 345 Places
- Setellite — 336 Places
- Gurukul — 83 Places
- Navrangpura — 302 Places
- Vastrapur — 217 Places
- Thaltej — 222 Places
- Prahalad Nagar — 181 Places
- C G Road — 94 Places

See more



Get the Zomato app

We will send you a link, open it on your phone to download the app

Email Phone

Email

Share App Link

Download app from

GET IT ON Google Play

Download on the App Store

Explore options near me

Top Restaurant Chains

Zomato

India English

ABOUT ZOMATO
Who We Are
Blog
Work With Us
Investor Relations
Report Fraud

ZOMAVERSE
Zomato
Blinkit
Feeding India
HyperPure
Zomaland

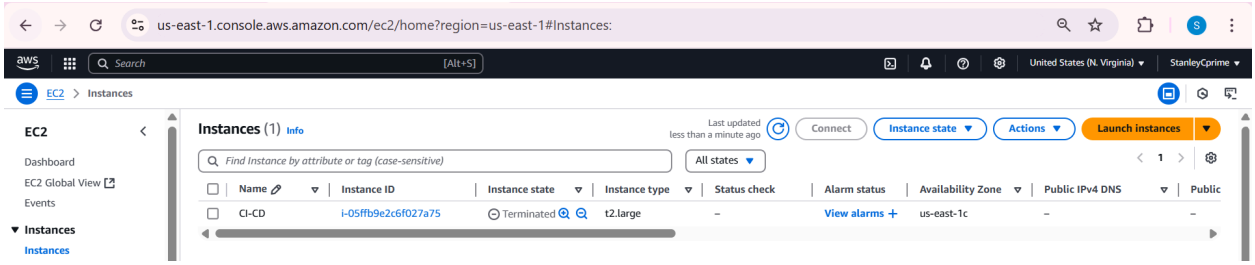
FOR RESTAURANTS
Patner With Us
Apps For You

For Enterprises
Zomato For Work

LEARN MORE
Privecy
Security
Terms
Sitemap

Step 8 : Delete the Instances

1. At first glance, this step may be neglected
2. But the instances we use are heavy and huge
3. So it is mandatory to delete the instances as soon the Project is completed
4. Open EC2 in AWS service provider
5. Select the instance (or resource) you want to delete
6. Click Instance State and choose Terminate Instance
7. Wait for the status to be changed to Terminated



Things we Achieved

End-to-End CI/CD Pipeline Setup:

- Automated the process of code checkout, build, test, scan, containerize, and deploy using Jenkins.

Secure DevSecOps Integration:

- Integrated security at every stage using **SonarQube**, **Trivy**, and **OWASP Dependency-Check** for code and container vulnerability analysis.

Containerization with Docker:

- Packaged the Node.js application into a portable Docker image, ensuring consistent deployment across environments.

DockerHub Image Push:

- Automatically pushed the application image to **DockerHub**, enabling image reuse and version control.

Jenkins Plugin Configuration:

- Installed and configured essential plugins (JDK, NodeJS, SonarQube Scanner, Docker, Dependency Check) for pipeline execution.

Custom Jenkins Declarative Pipeline:

- Defined all CI/CD steps in a version-controlled Jenkinsfile, allowing for easy updates and transparency.

Static Code Quality Checks:

- Enforced **SonarQube Quality Gates** to maintain high code standards and detect code smells, bugs, and security hotspots early.

Dependency Vulnerability Scanning:

- Used **OWASP Dependency Check** to scan third-party packages and libraries for known vulnerabilities.

Container Vulnerability Scanning:

- Leveraged **Trivy** to scan Docker images for OS-level and application-level security risks before deployment.

Real-Time Deployment:

- Deployed the Dockerized Zomato Clone application directly on an EC2 instance, accessible via browser on port 3000.

Cost Optimization:

- Terminated the EC2 instance after deployment testing to avoid unnecessary billing.

Conclusion

The Zomato Clone App deployment with a DevSecOps CI/CD pipeline is a comprehensive process that integrates various tools and services to automate the building, scanning, and deployment of a Node.js application. From launching an AWS EC2 instance to setting up Jenkins, Docker, SonarQube, Trivy, and OWASP Dependency Check, each step is crucial for ensuring the application is built with security and quality in mind.

Key stages in the setup include:

- **Instance Setup**: Launching an Ubuntu EC2 instance and connecting it via SSH using Putty, followed by configuring the necessary security settings and installing required services.

- **Tool Installations**: Installing Jenkins for automation, Docker for containerization, SonarQube for code quality checks, Trivy for vulnerability scanning, and OWASP Dependency Check for assessing application dependencies.

- **CI/CD Pipeline Configuration**: Setting up Jenkins pipelines to handle tasks such as code building, SonarQube analysis, vulnerability scanning, Docker image building, and deployment.

- **Automated Security Scanning**: Leveraging SonarQube and Trivy ensures that security flaws are detected and handled early in the development lifecycle.

- **Docker Image Management**: Building Docker images, pushing them to DockerHub, and deploying them in containers ensures a repeatable and scalable deployment process.

- **Cleaning up Resources**: Finally, after successfully deploying the application, terminating the EC2 instances helps avoid unnecessary costs.

This setup not only automates the entire process but also ensures that security and quality gates are integrated into every step, making the deployment process robust and reliable. The combination of Jenkins, SonarQube, Trivy, and Docker provides a modern DevSecOps approach that is well-suited for managing production-ready applications.
