**Name:** Atif Ansari

**Roll no:** 04

**Class:** D15B

**Case Study Report: Continuous Integration with Static Code Analysis**

**Table of Contents**

1. Introduction
2. Concepts Used
3. Problem Statement
4. Environment Setup
5. Step-by-Step Execution
6. Key Features & Applications
7. Conclusion

**1. Introduction**

This case study details the implementation of a Continuous Integration (CI) pipeline using Jenkins and SonarQube, hosted on AWS Cloud9. The primary goal is to automate the build and testing processes for a Python application (specifically, a calculator.py file) while integrating static code analysis to maintain high code quality.

**2. Concepts Used**

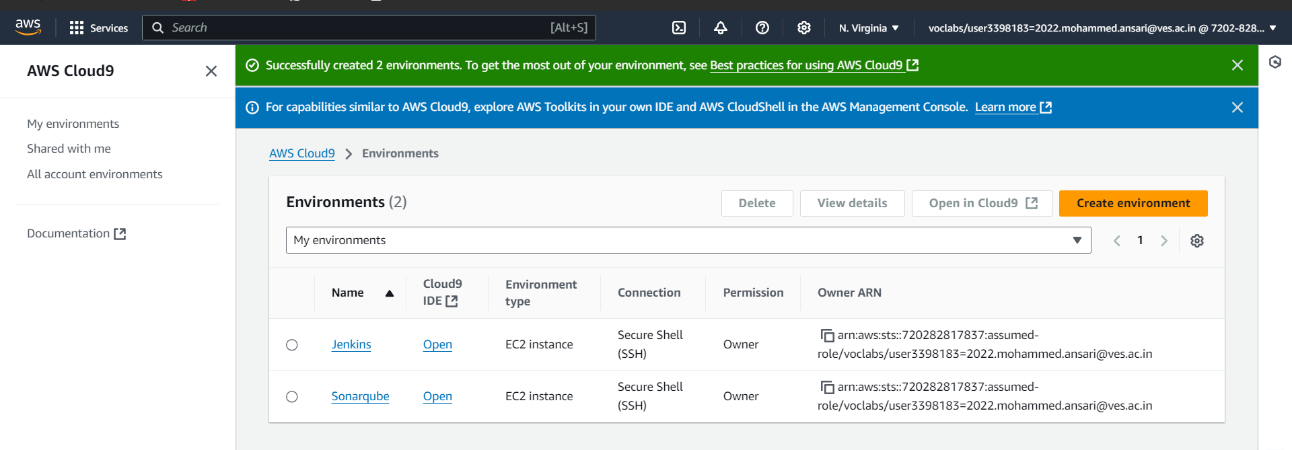
* **Continuous Integration (CI)**: A software development practice where code changes are automatically tested and merged into a shared repository.
* **Jenkins**: An open-source automation server that supports building, deploying, and automating software development processes.
* **SonarQube**: A tool that helps developers manage code quality by performing static code analysis.
* **AWS Cloud9**: A cloud-based IDE that allows developers to write, run, and debug code in a web browser, providing a seamless development experience.

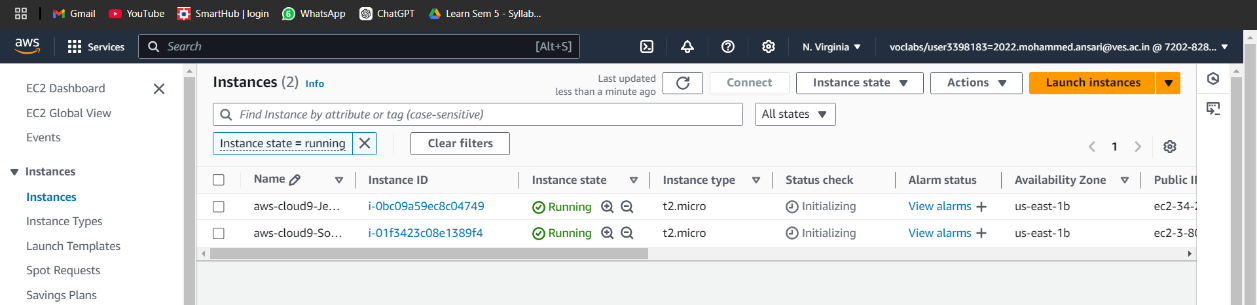
**3. Problem Statement**

The objective of this project was to set up a Jenkins CI pipeline integrated with SonarQube to perform static analysis on a Python application, ensuring code quality and early detection of bugs.

**4. Environment Setup**

1. **Access AWS Management Console**:
   * Log in to your AWS account and navigate to the AWS Management Console.
2. **Create Cloud9 Environment for Jenkins**:
   * Go to the **Cloud9** service and click on **Create environment**.
   * Name the environment **Jenkins**.
   * Configure the instance type and other settings as required, then create the environment.
3. **Create Cloud9 Environment for SonarQube**:
   * Repeat the process to create another Cloud9 environment named **SonarQube**.
   * Ensure both environments are accessible from the AWS console.
4. **Open Both Environments**:
   * Once both environments are created, open them in separate browser tabs to facilitate easier navigation.





**5. Steps of Execution**

**5.1. Setting Up Jenkins**

1. **Connect to Jenkins Instance**:
   * Open the terminal in the **Jenkins** Cloud9 environment and use the SSH command to connect:

**ssh -i your-key.pem ubuntu@your-jenkins-public-ip**

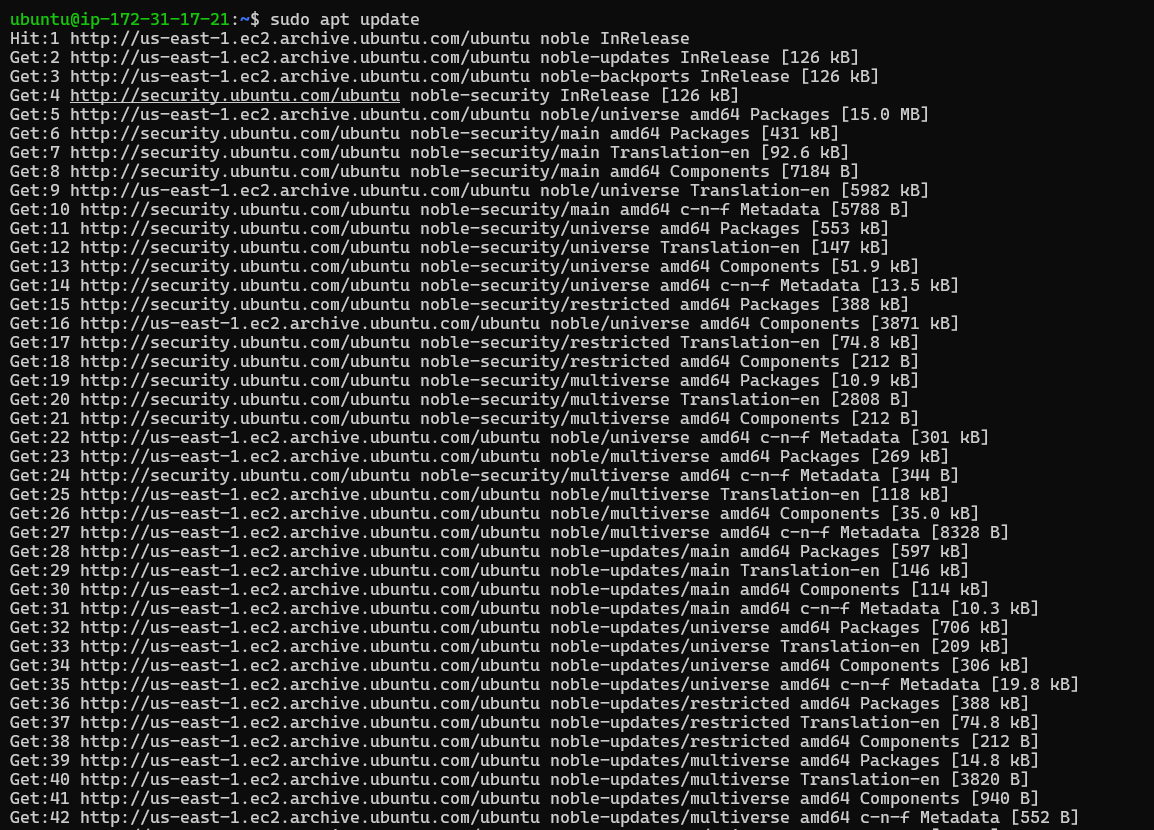
* + Replace **your-key.pem** with your actual PEM file and **your-jenkins-public-ip** with the public IP of your Jenkins instance.

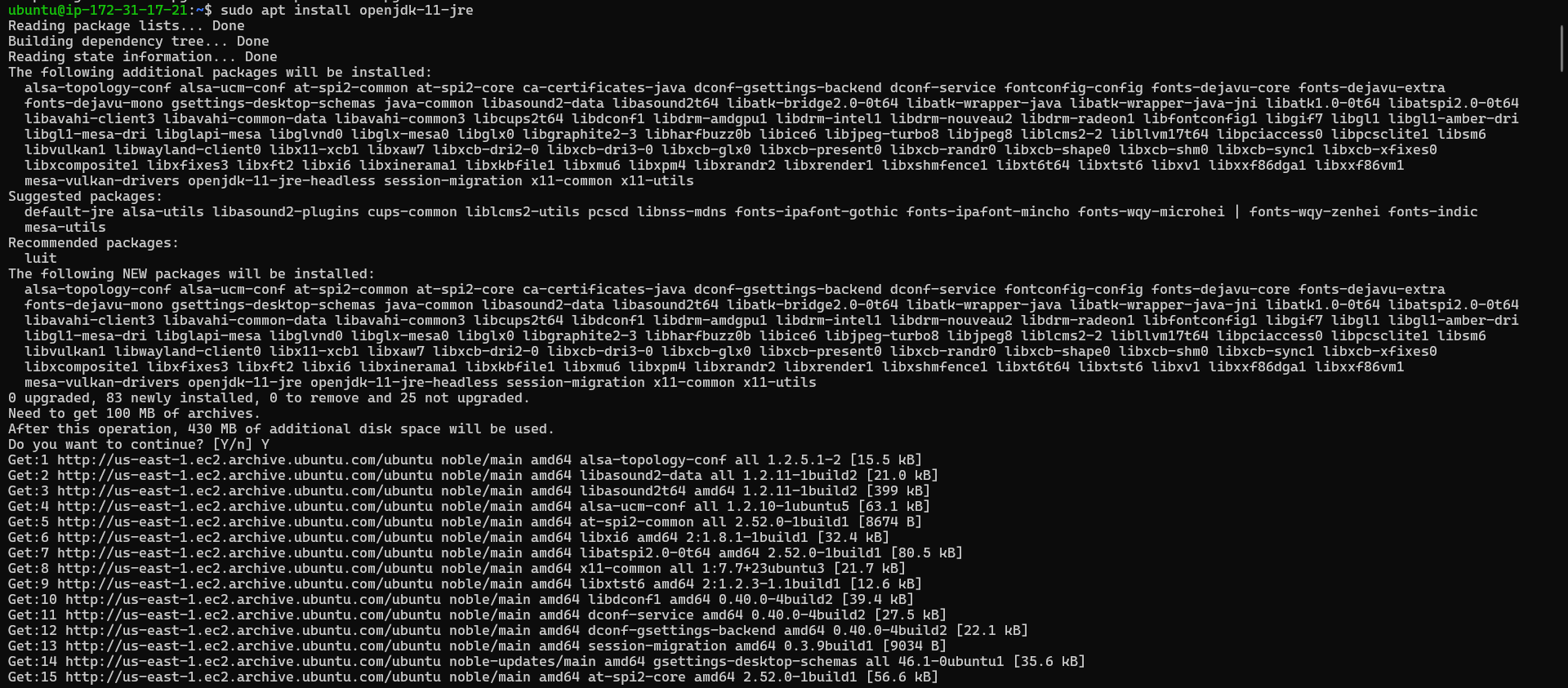


1. **Update Package Lists and Install Java**:
   * Run the following commands to update the package list and install Java:

**sudo apt-get update**

**sudo apt-get install -y openjdk-11-jdk**





1. **Install Jenkins**:
   * Use the commands provided on the Jenkins installation page to install Jenkins on your instance:

**sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \**

**https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key**

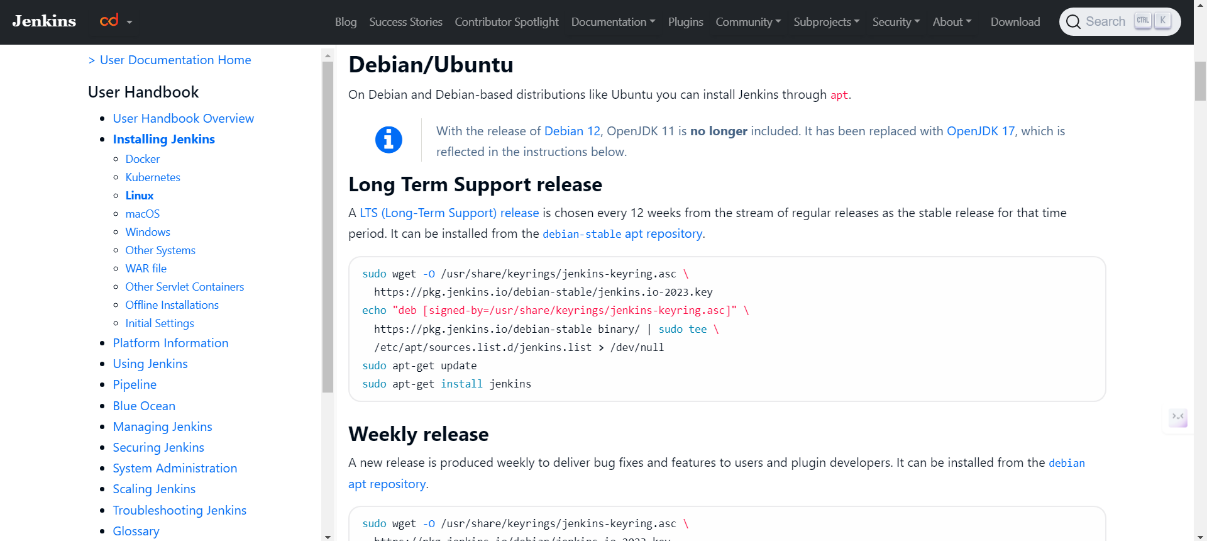
**echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \**

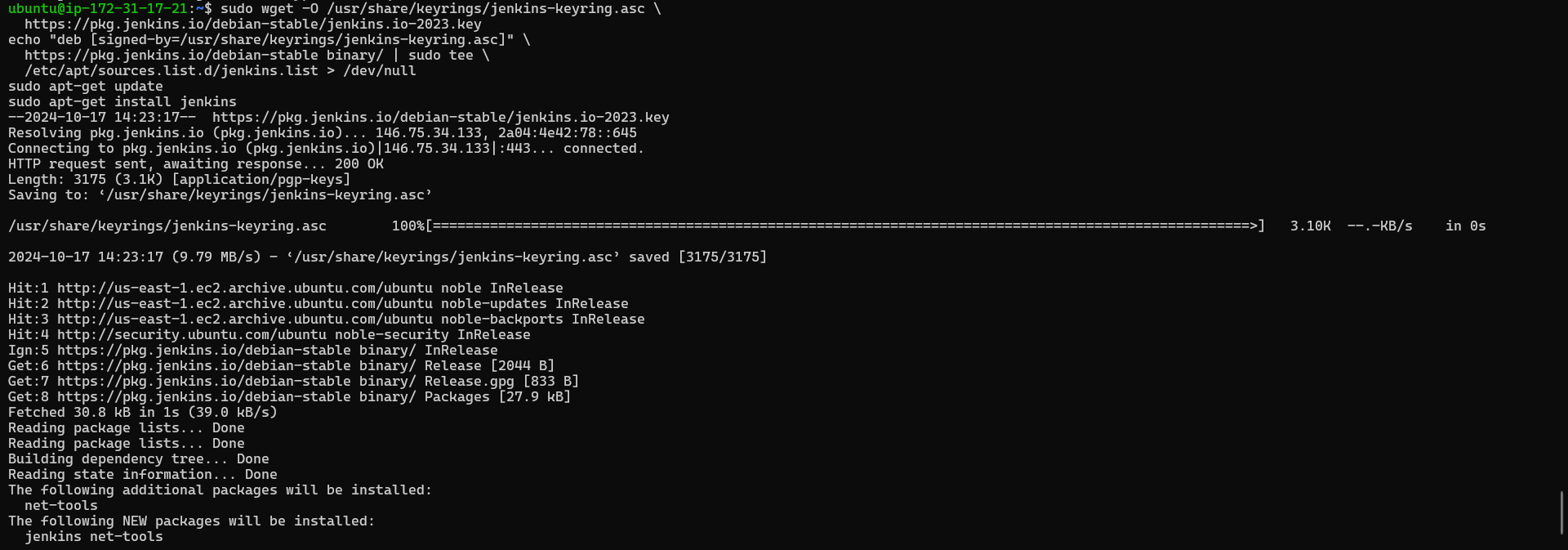
**https://pkg.jenkins.io/debian-stable binary/ | sudo tee \**

**/etc/apt/sources.list.d/jenkins.list > /dev/null**

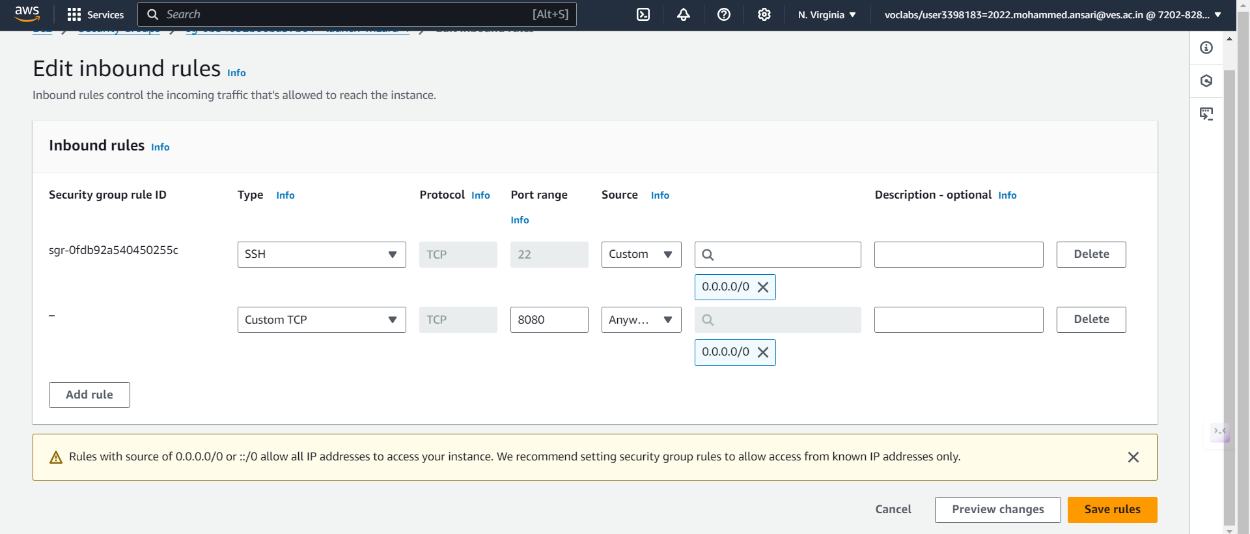
**sudo apt-get update**

**sudo apt-get install -y Jenkins**





1. **Configure Inbound Rules**:
   * Set the inbound rules in your AWS security group to allow traffic on port **8080** to access Jenkins.

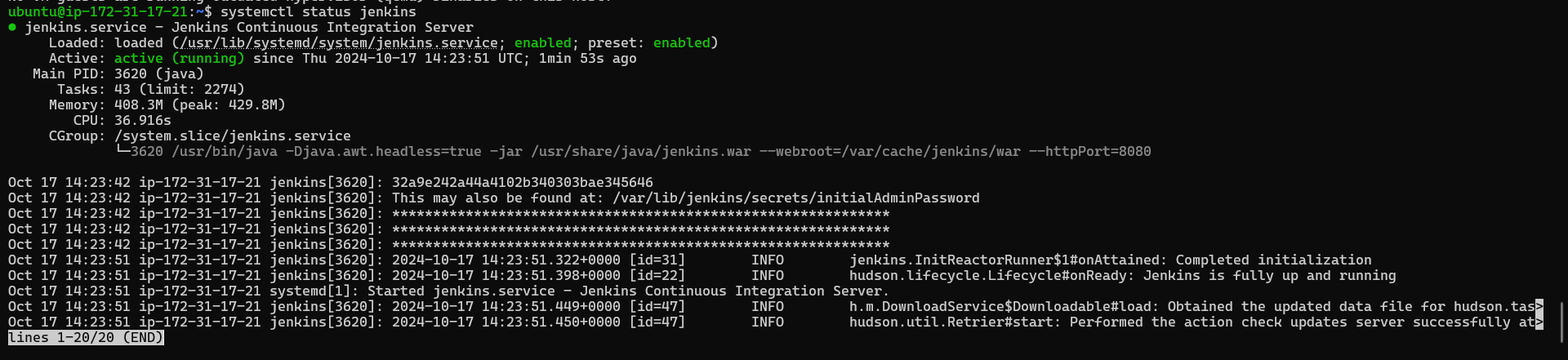


1. **Start Jenkins**:
   * Start the Jenkins service using the following command:

**sudo systemctl start Jenkins**

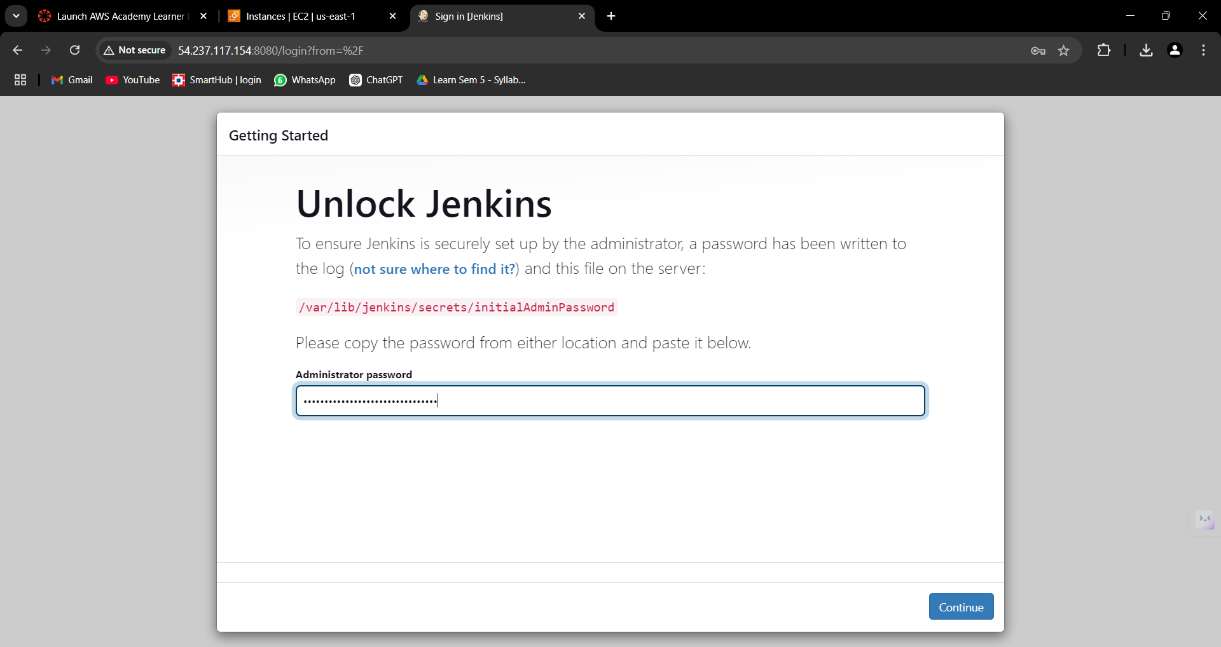
* + Check the status of Jenkins to confirm it is running:

**sudo systemctl status Jenkins**

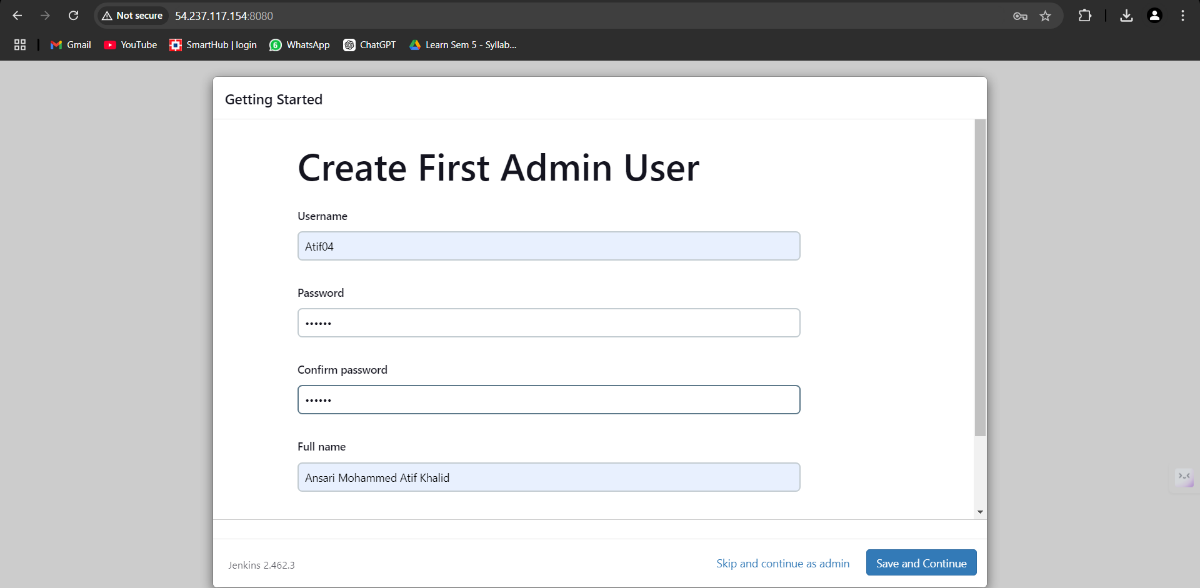


1. **Access Jenkins Web Interface**:
   * Open your web browser and navigate to http://your-jenkins-public-ip:8080 to access the Jenkins interface.
   * Retrieve the initial admin password with:

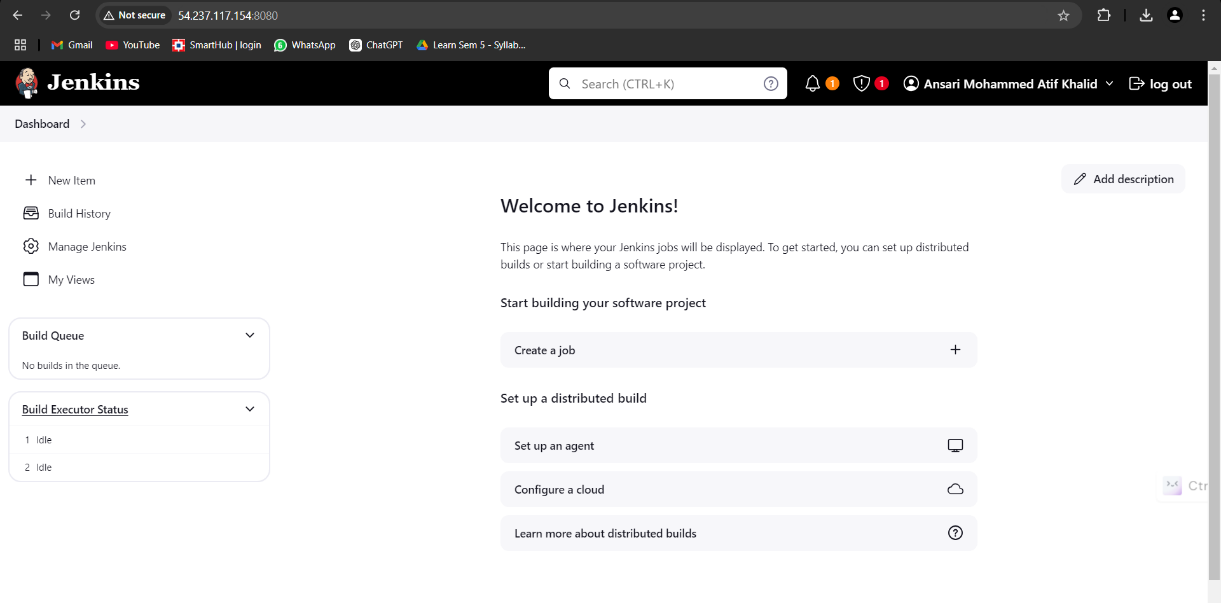
**sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

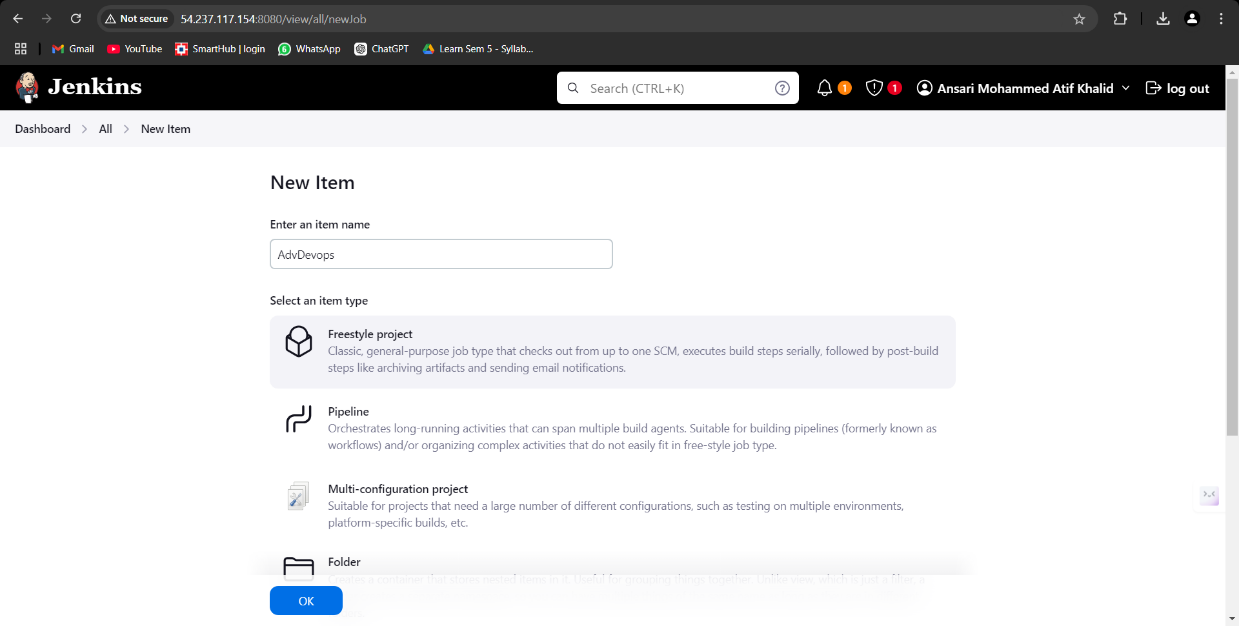


* + Use this password to unlock Jenkins and proceed to set up your Jenkins account.

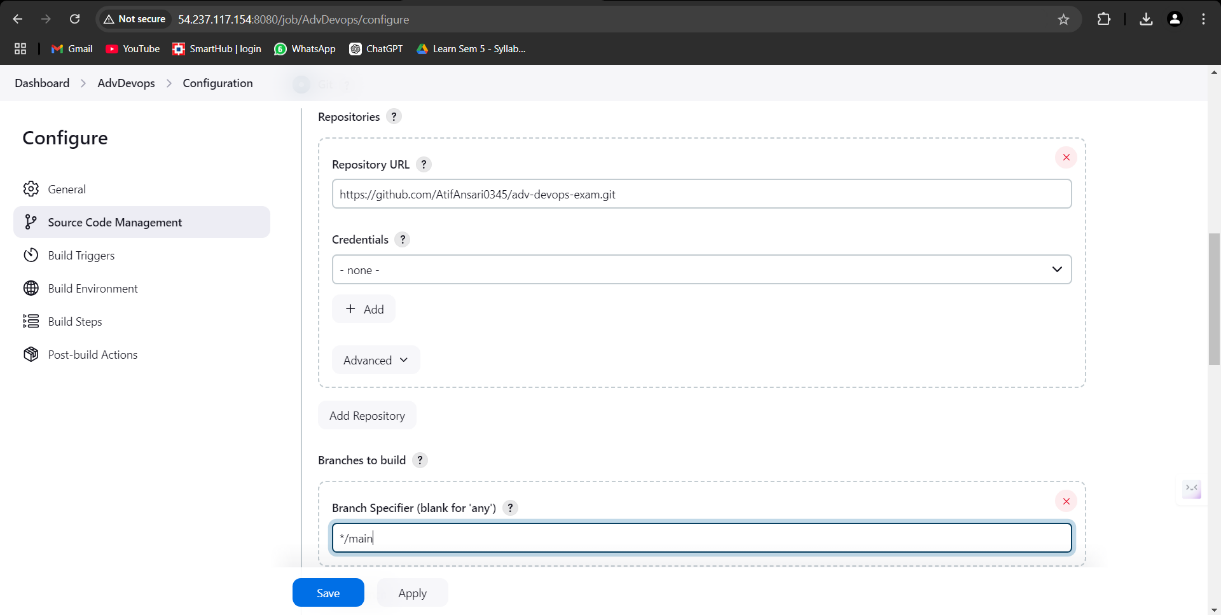


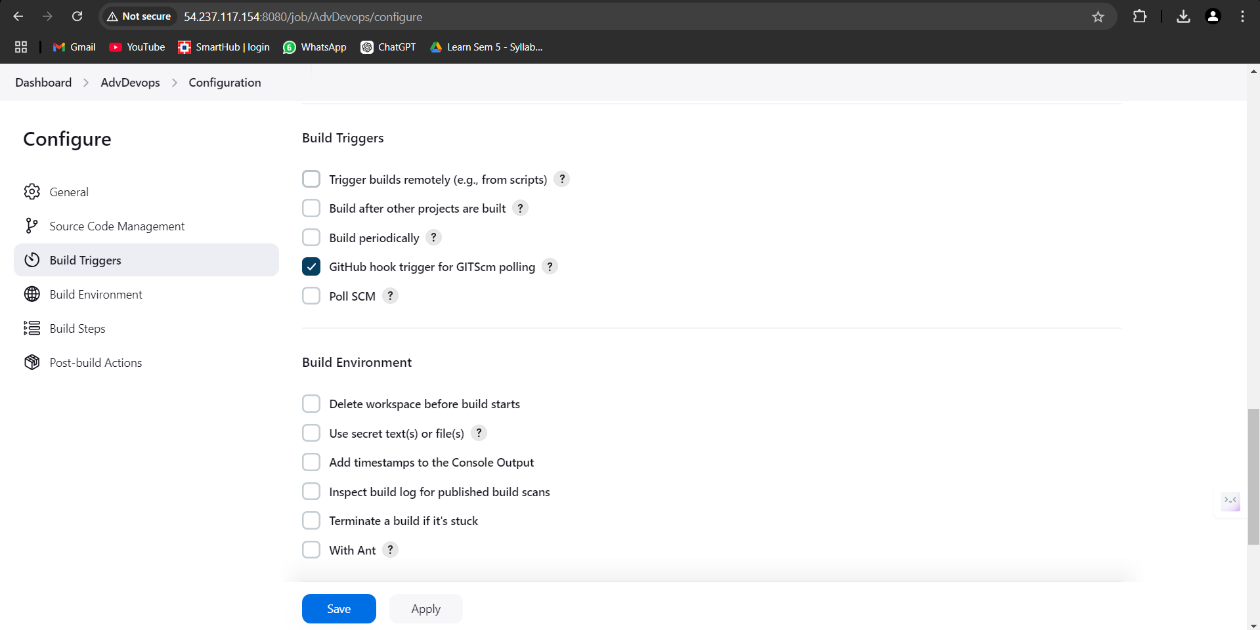
1. **Create a New Jenkins Job**:
   * Click on **New Item** in the Jenkins dashboard, name it **AdvDevops**, and select **Freestyle project**.



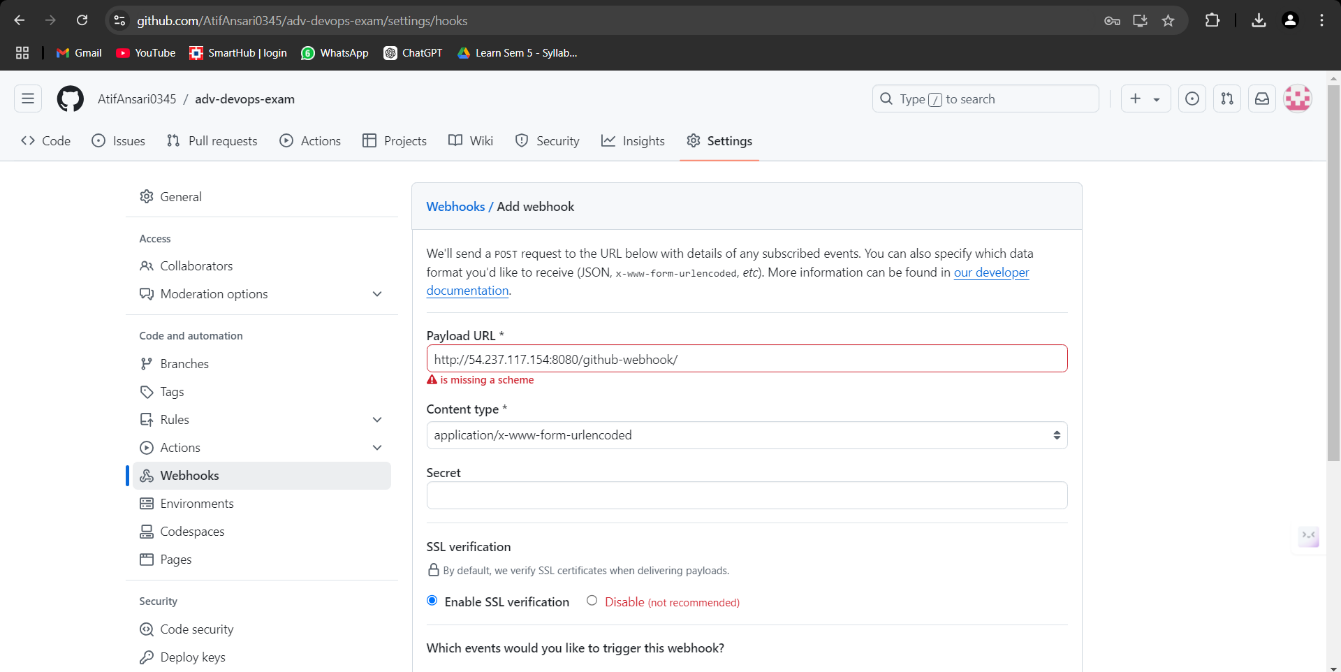


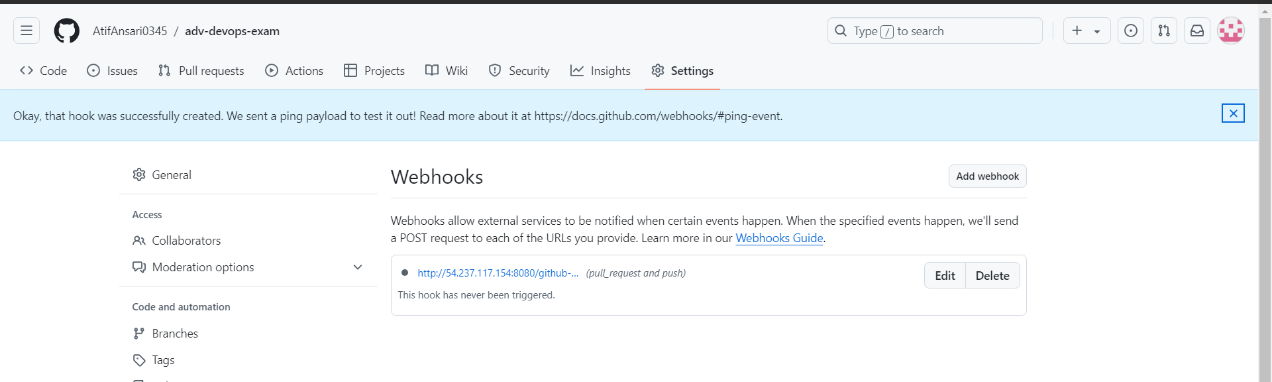
* + In the project configuration, set up your Git repository URL that contains the calculator.py file.

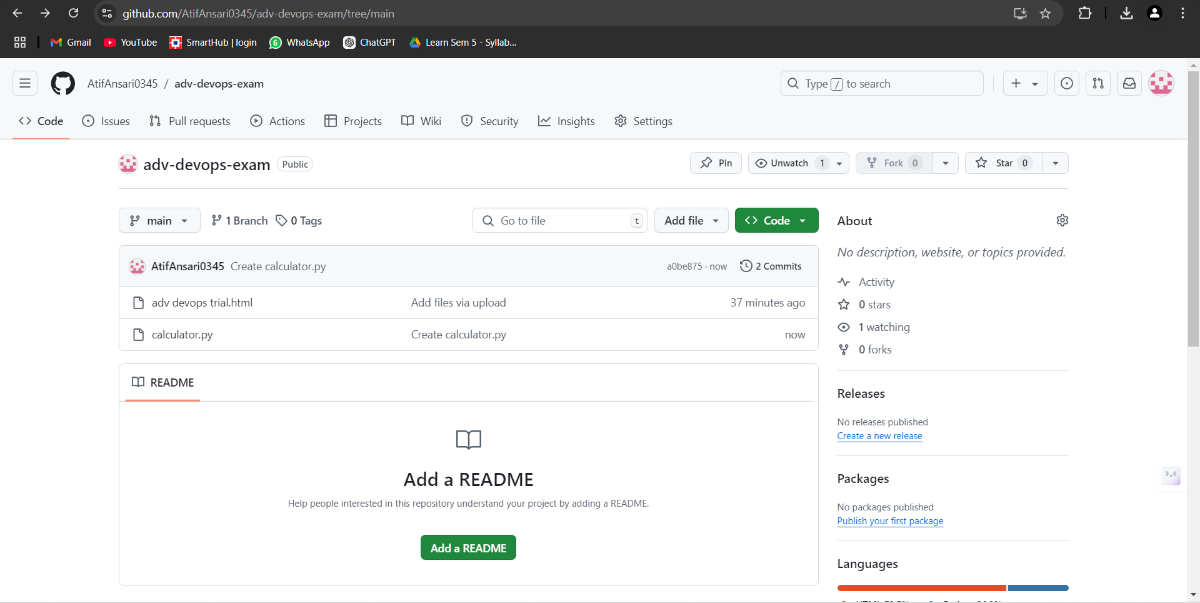


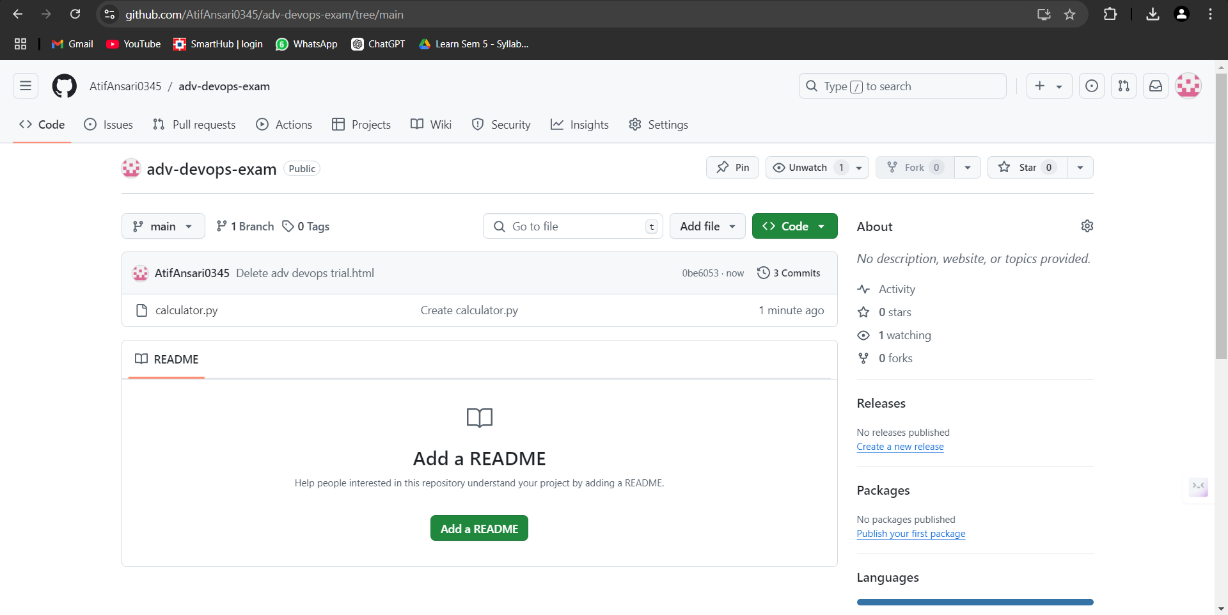


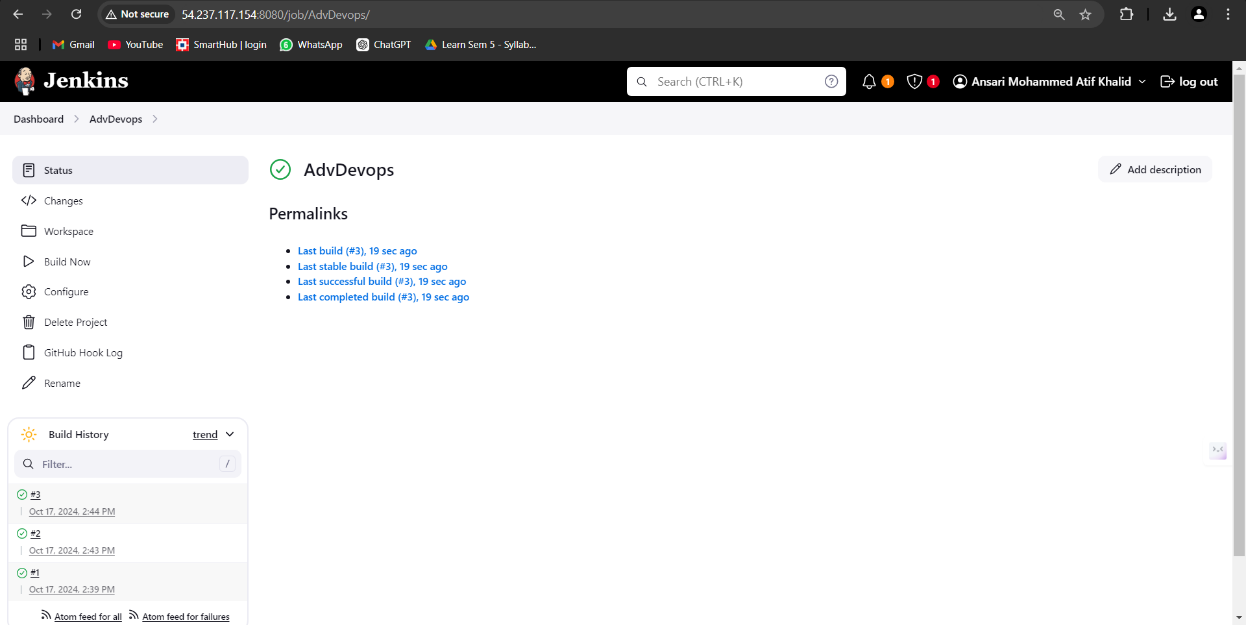
1. **Setting Up Webhooks**:
   * Configure GitHub webhooks to notify Jenkins of changes to the repository. This allows Jenkins to automatically trigger builds and tests whenever changes are pushed to the repository.







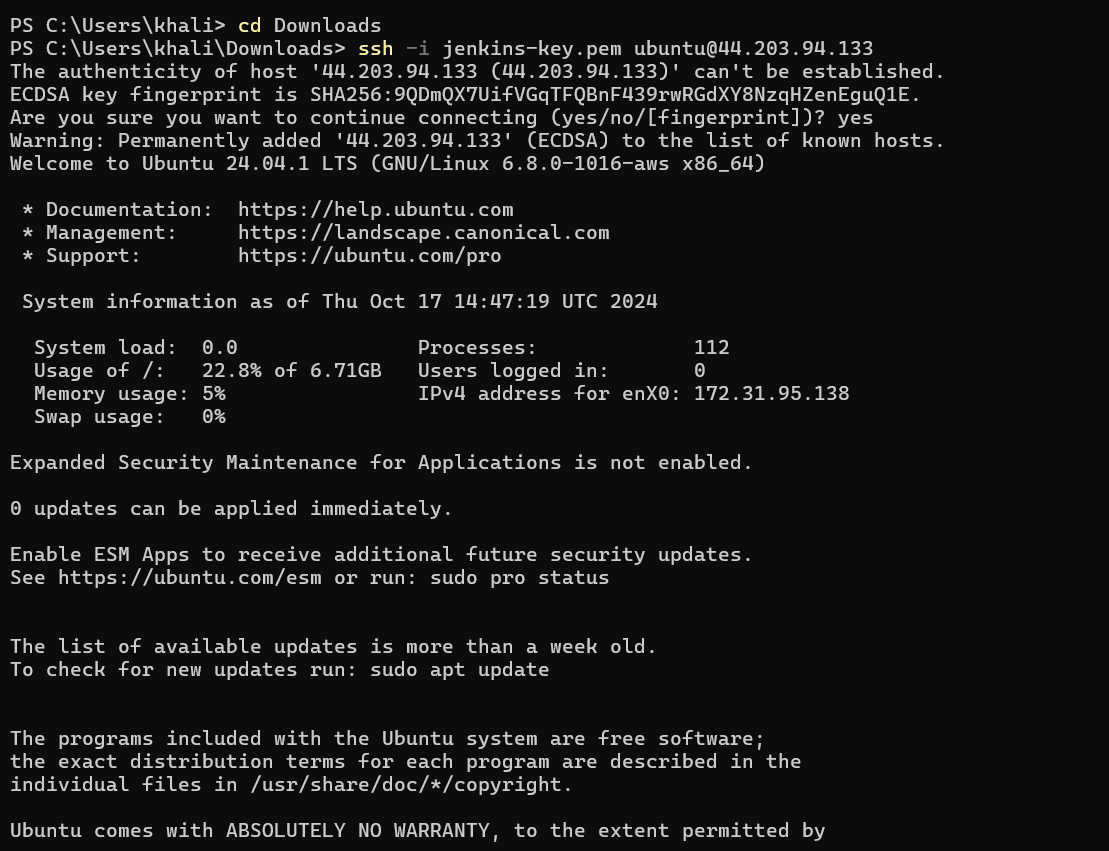




**5.2. Setting Up SonarQube**

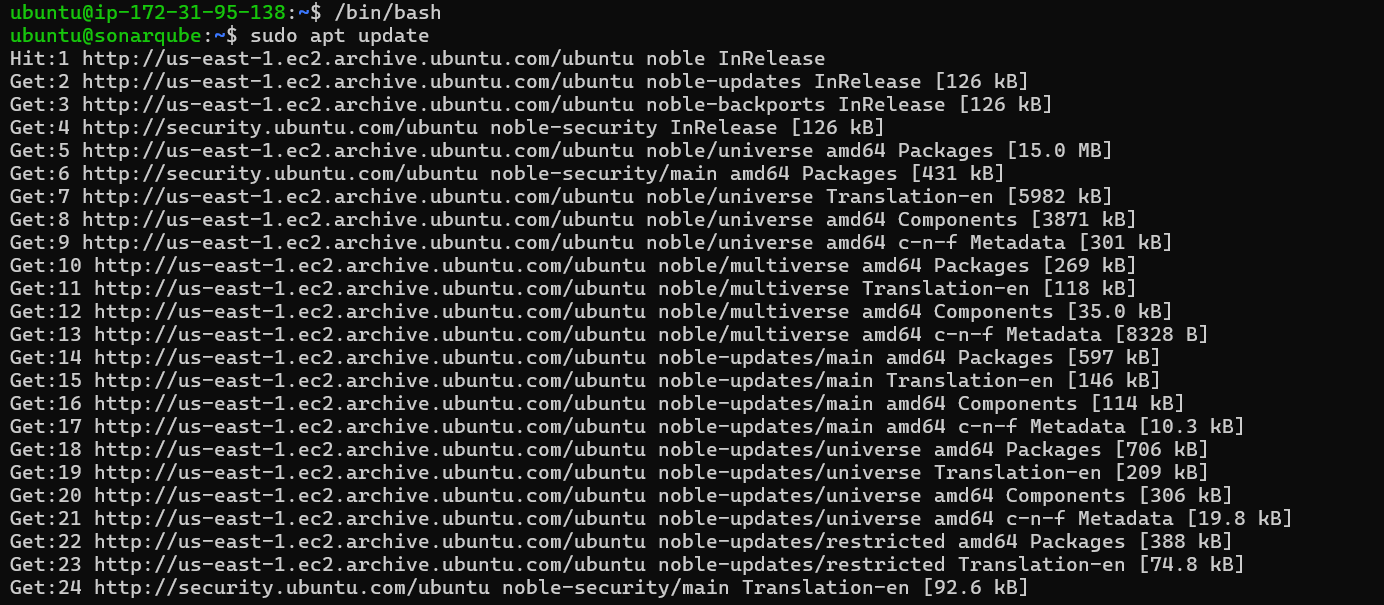
1. **Connect to SonarQube Instance**:
   * Open a new terminal in the **SonarQube** Cloud9 environment and connect using SSH:

**ssh -i your-key.pem ubuntu@your-sonarqube-public-ip**

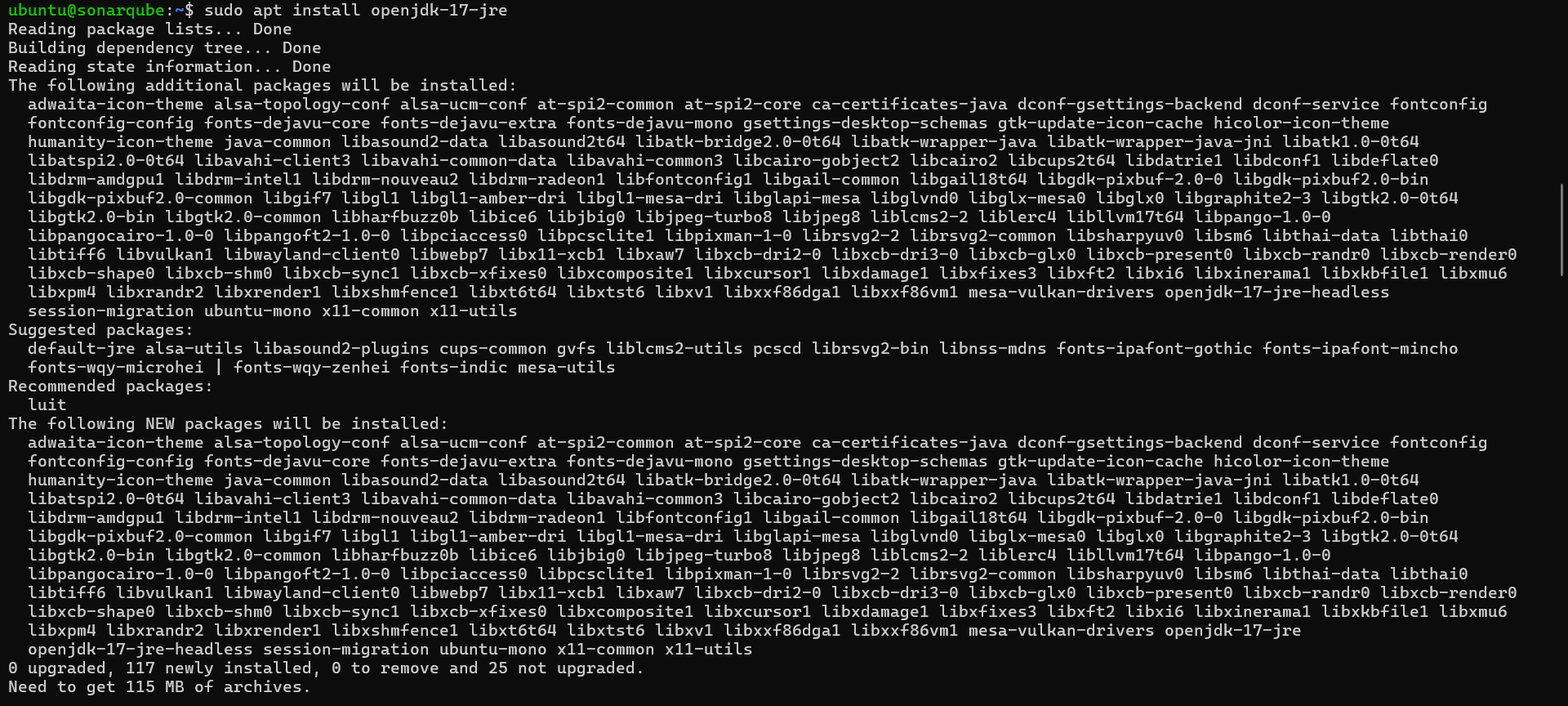


1. **Update Package Lists and Install Java**:
   * Update the package list and install Java as follows:

**sudo apt-get update**



**sudo apt-get install -y openjdk-11-jdk**

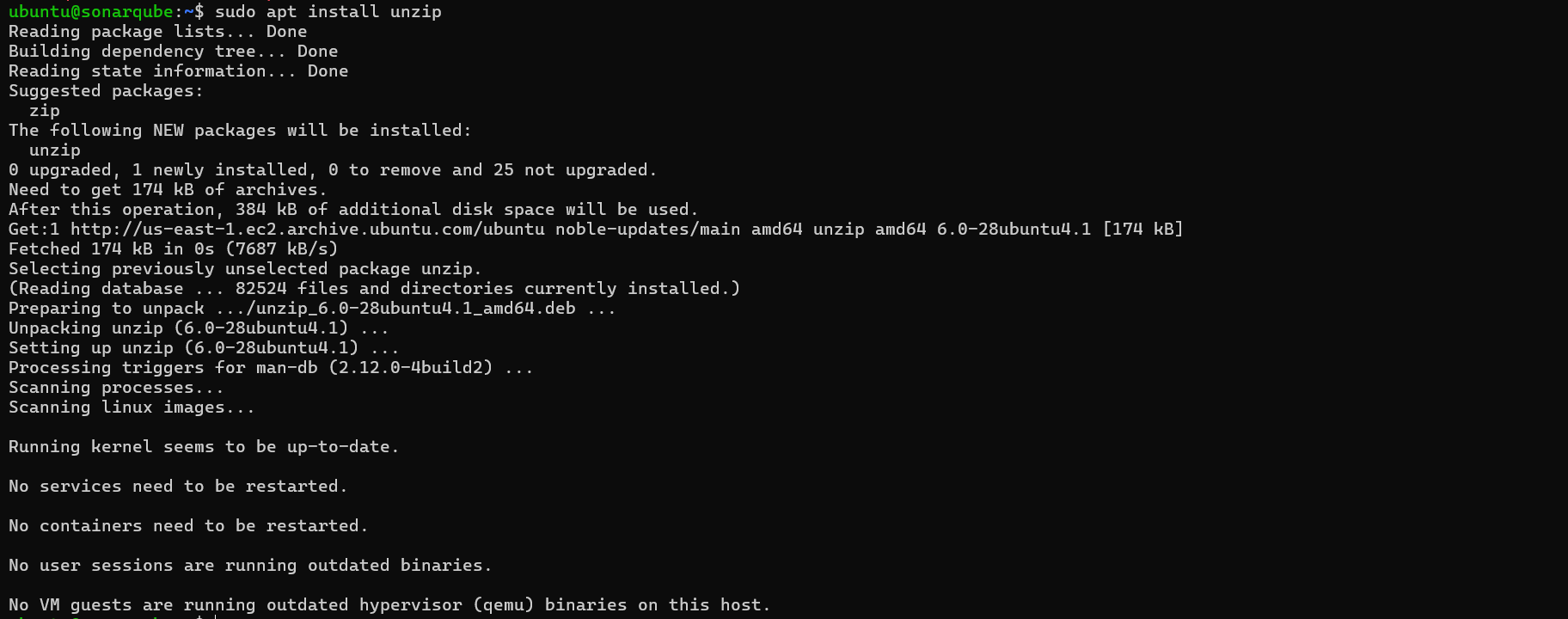


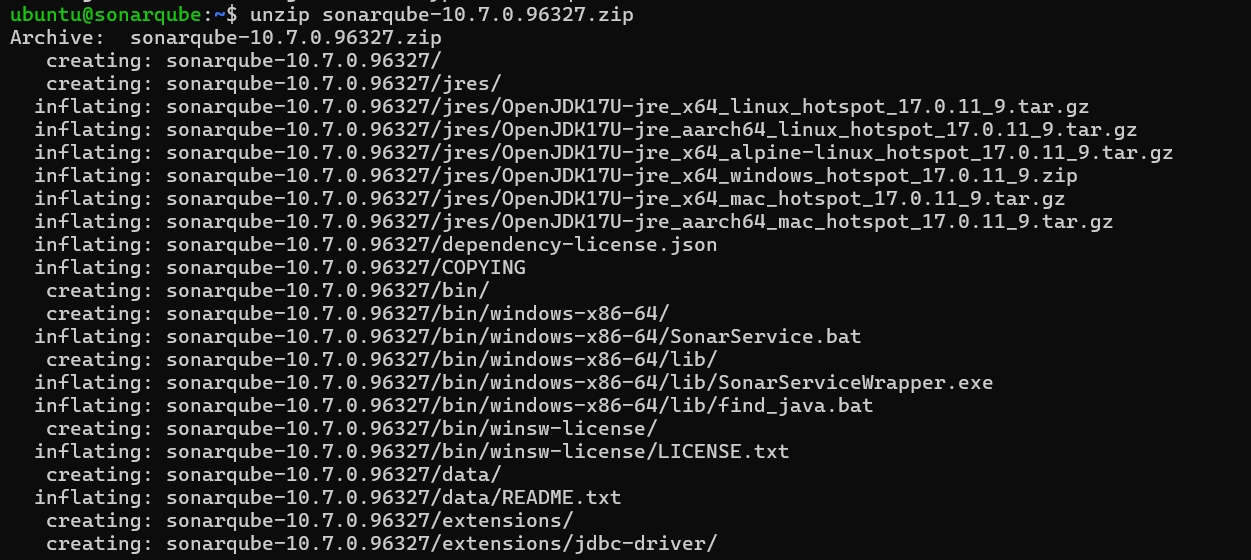
1. **Install SonarQube**:
   * Download and install SonarQube using the following commands:

**wget** [**https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.7.96285.zip**](https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.7.96285.zip)

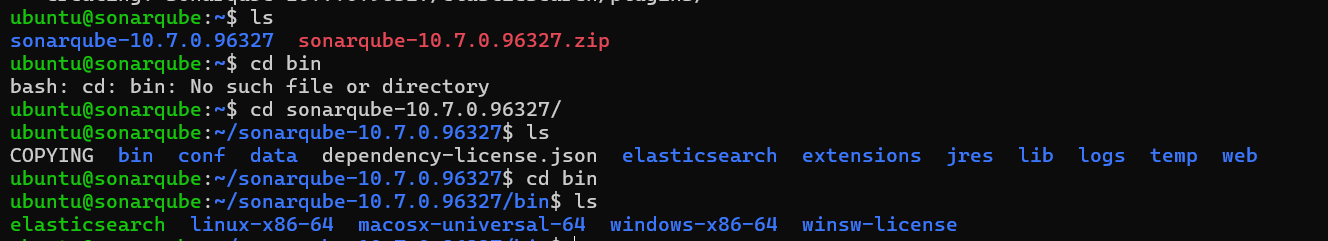


**unzip sonarqube-9.9.7.96285.zip**

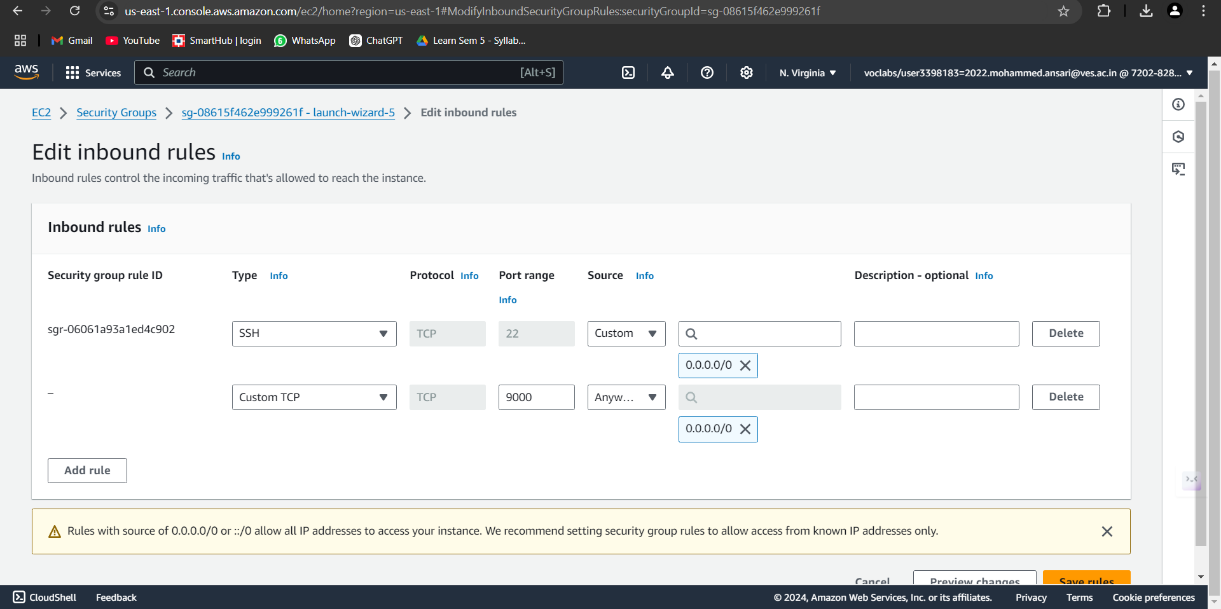




**cd sonarqube-9.9.7.96285**



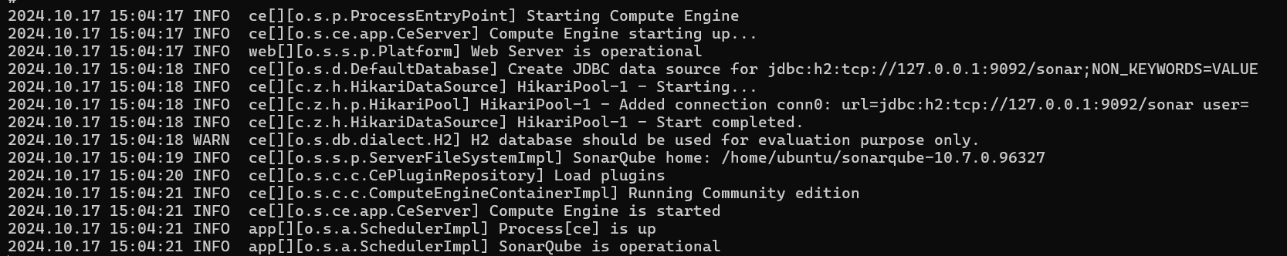
1. **Configure Inbound Rules**:
   * Edit the inbound rules in your AWS security group to allow traffic on port **9000** for accessing SonarQube.



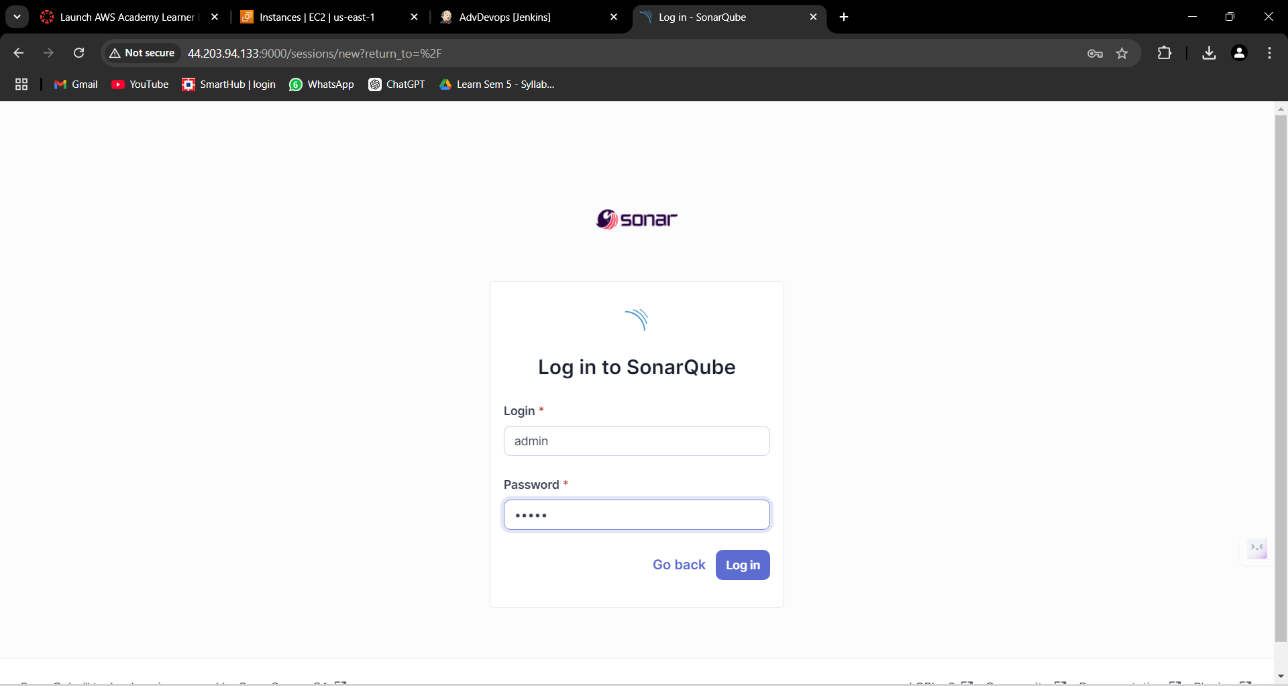
1. **Start SonarQube**:
   * Navigate to the SonarQube bin directory and start the SonarQube server:

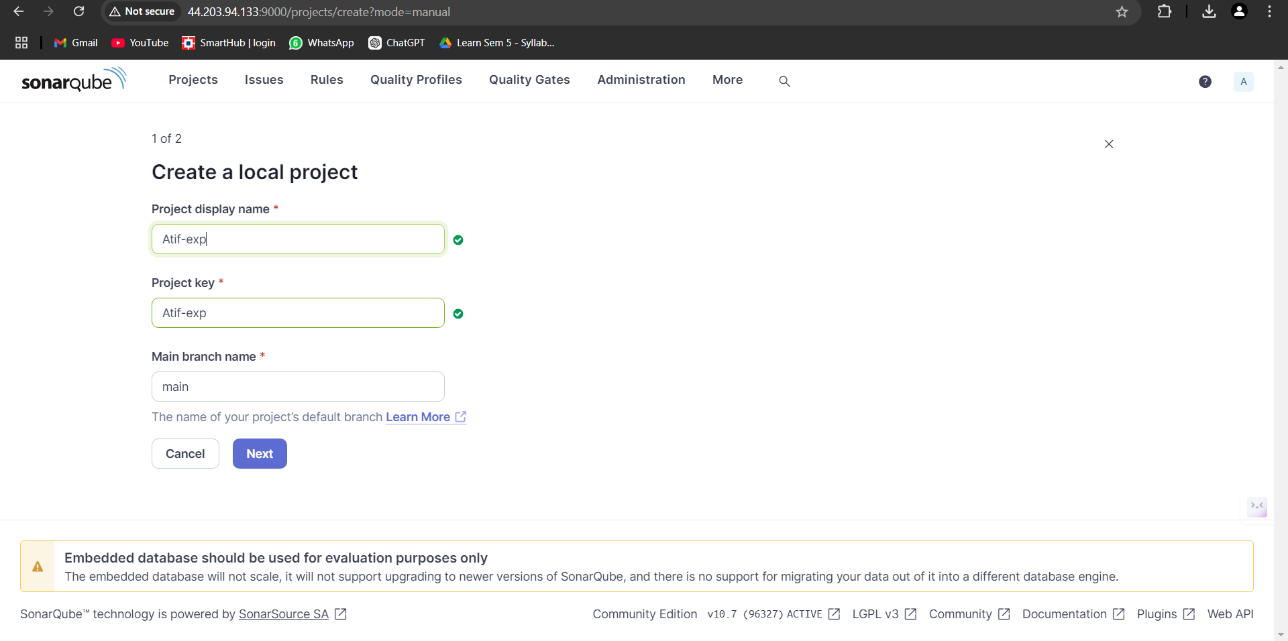
**cd sonarqube-9.9.7.96285/bin/linux-x86-64**

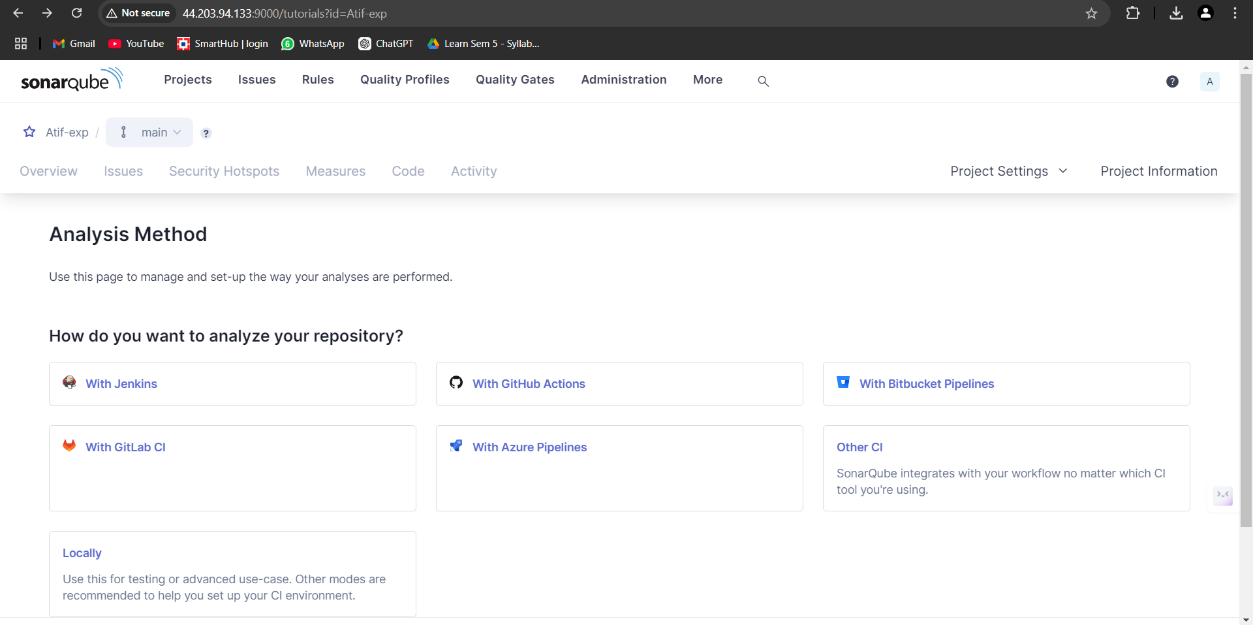
**./sonar.sh start**



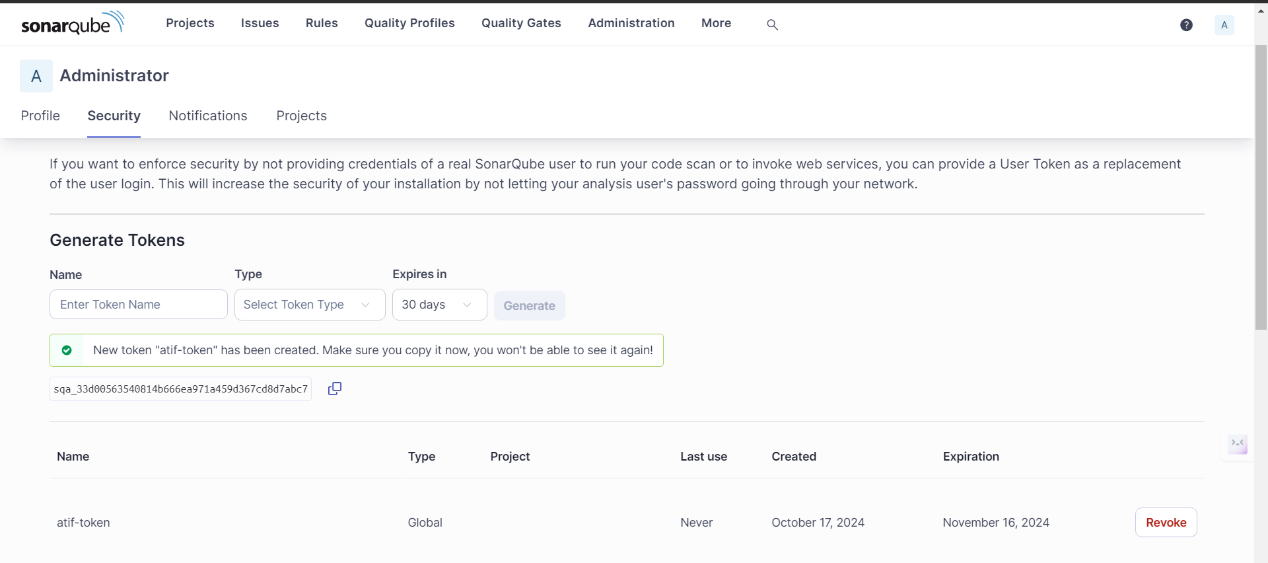
* + Access SonarQube in your browser at **http://your-sonarqube-public-ip:9000**





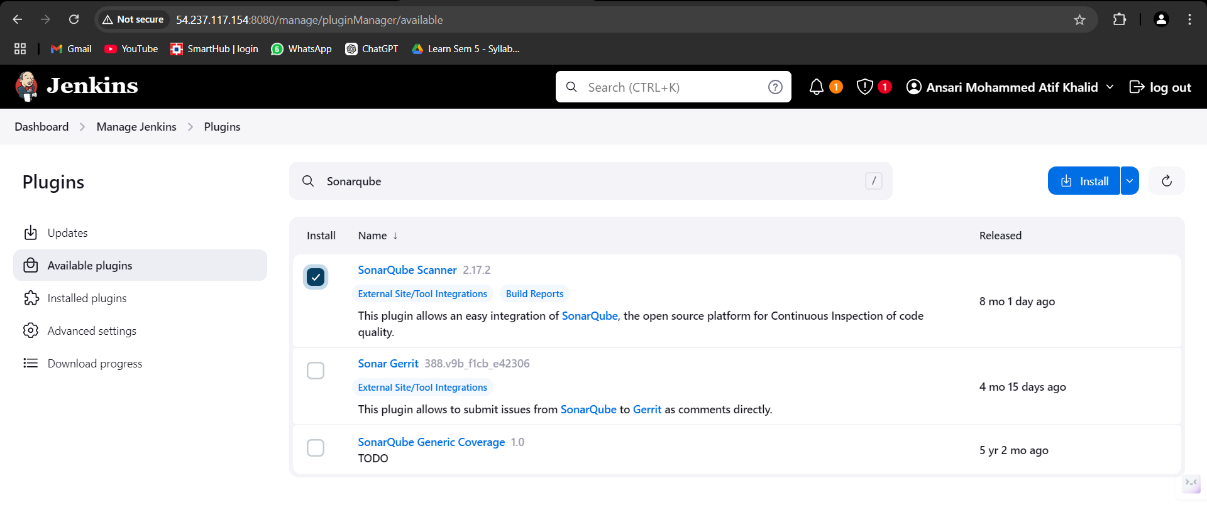


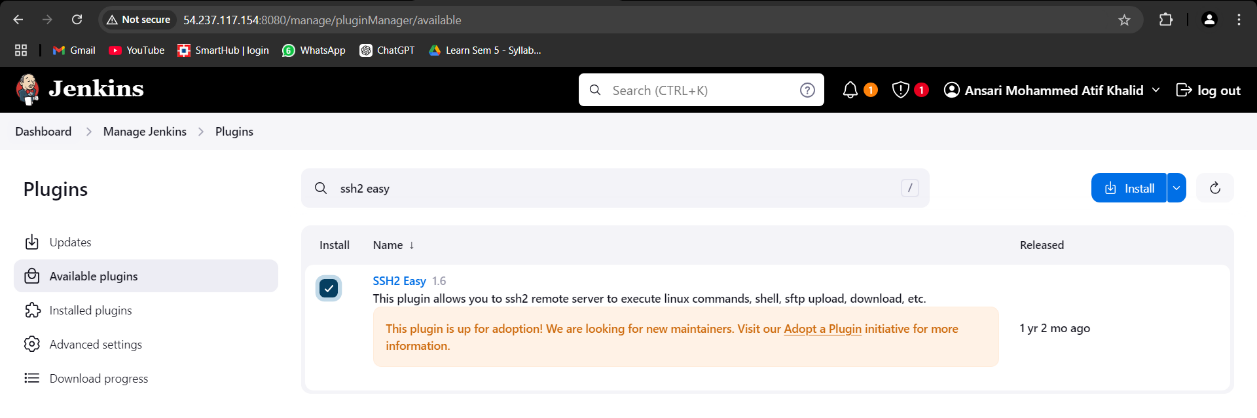
1. **Configure SonarQube**:
   * On your first access, create a new password for the admin account and generate a token for Jenkins integration. Name the token **atif-token**.



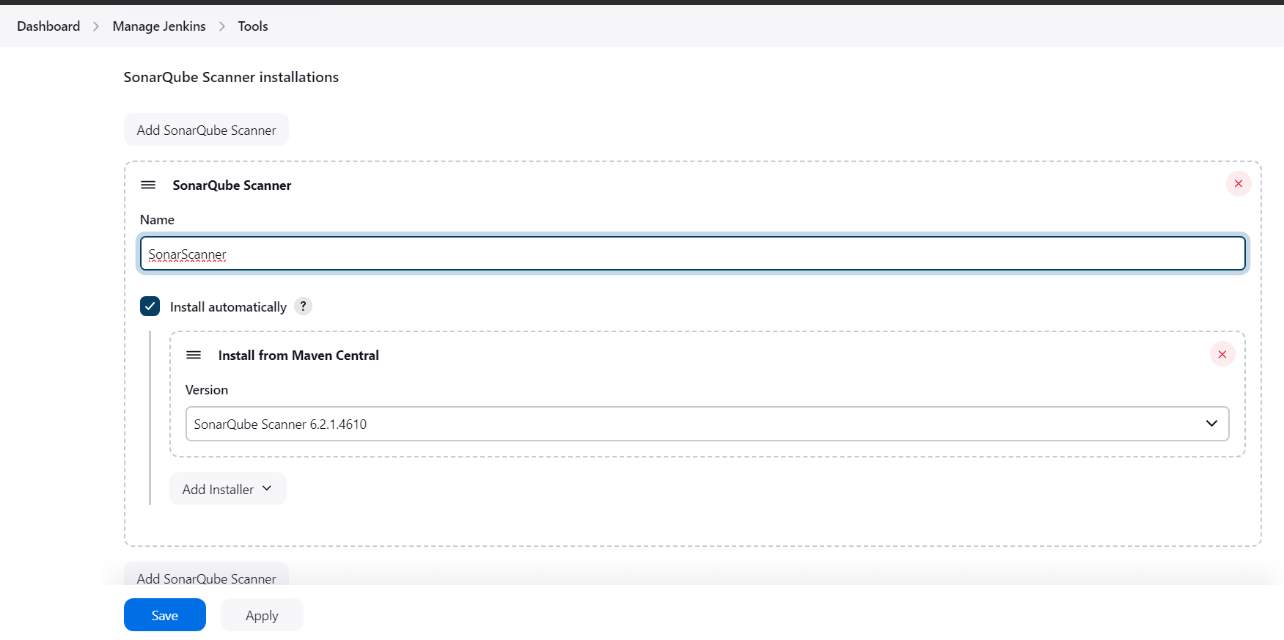
**5.3. Integrating Jenkins with SonarQube**

1. **Install Necessary Plugins in Jenkins**:
   * In the Jenkins dashboard, navigate to **Manage Jenkins > Manage Plugins** and install the **SonarQube Scanner** and **SSH2** plugins.

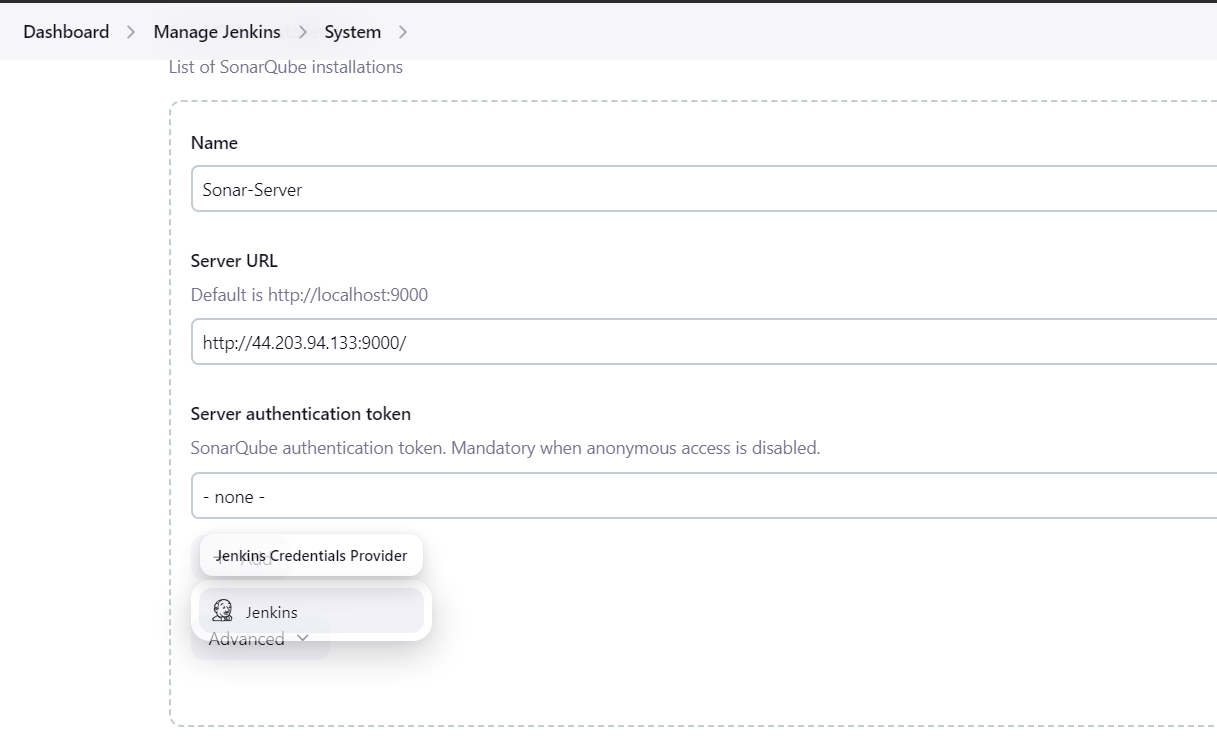


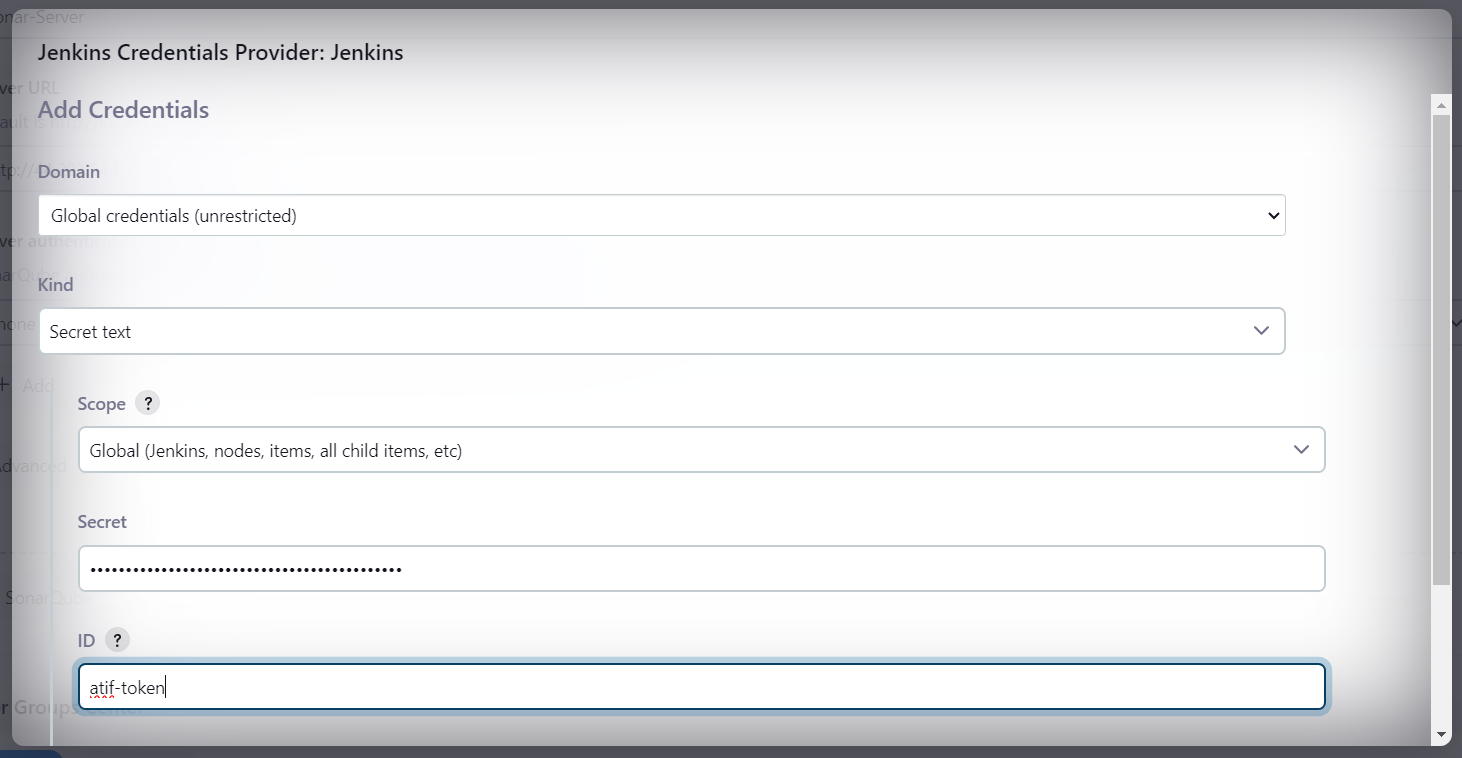


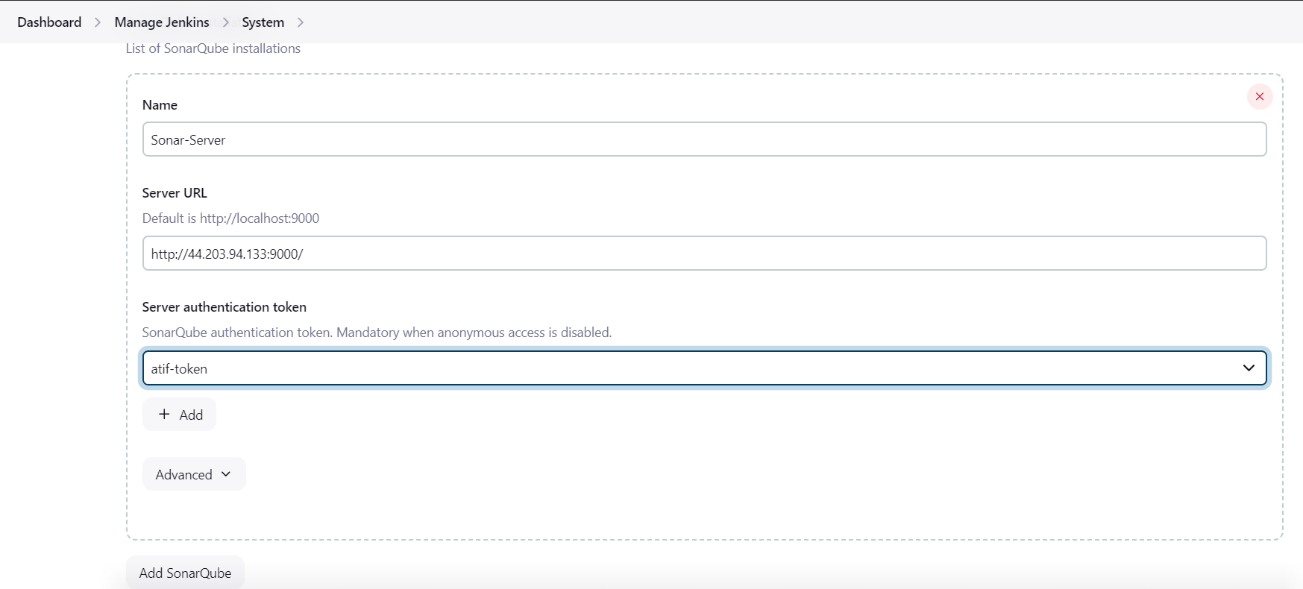
1. **Configure SonarQube Scanner in Jenkins**:
   * Under **Manage Jenkins > Global Tool Configuration**, add the SonarQube scanner. This allows Jenkins to use SonarQube for code analysis.



1. **Setting Up SonarQube Server in Jenkins**:
   * In **Manage Jenkins > Configure System**, add the SonarQube server by entering the URL (http://your-sonarqube-public-ip:9000).
   * Add the token generated earlier (Atif-token) as credentials.







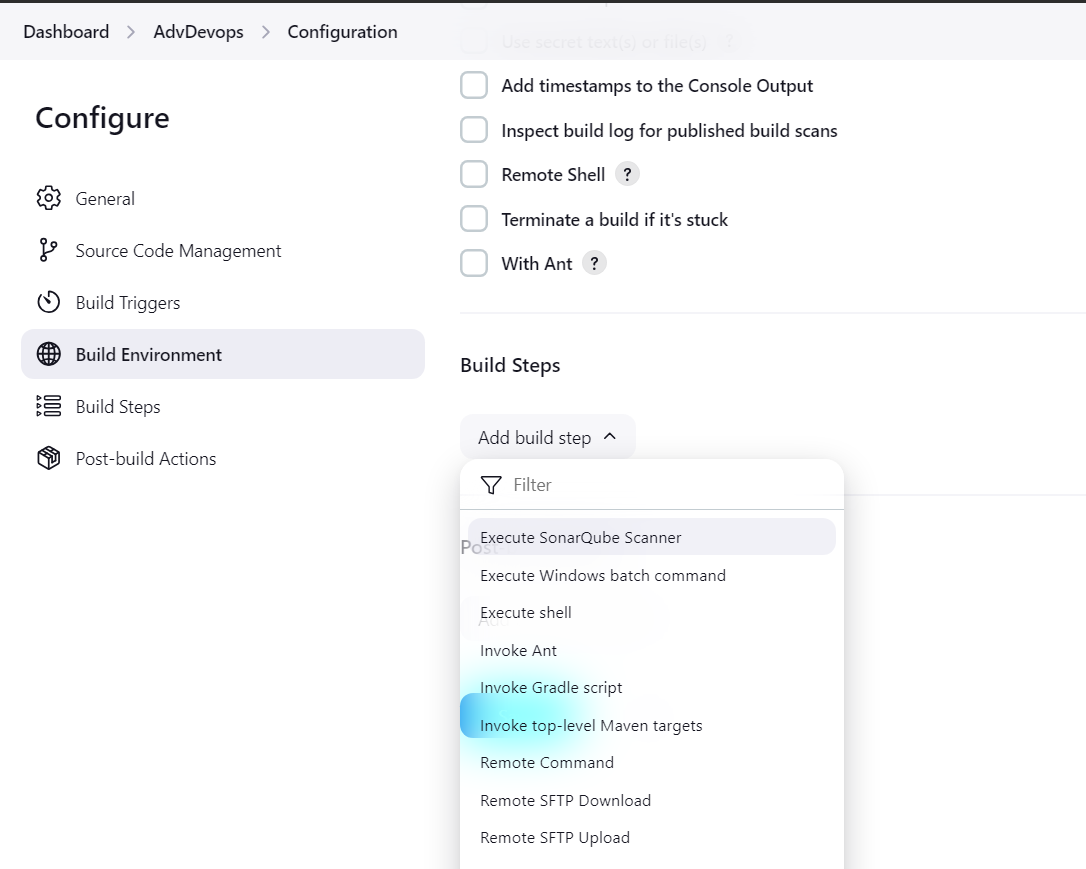
1. **Configuring Build Steps in Jenkins**:
   * Go back to the **AdvDevops** project configuration in Jenkins, scroll down to the **Build Steps** section, and select **Execute SonarQube Scanner**.
   * In the analysis properties, add the following:

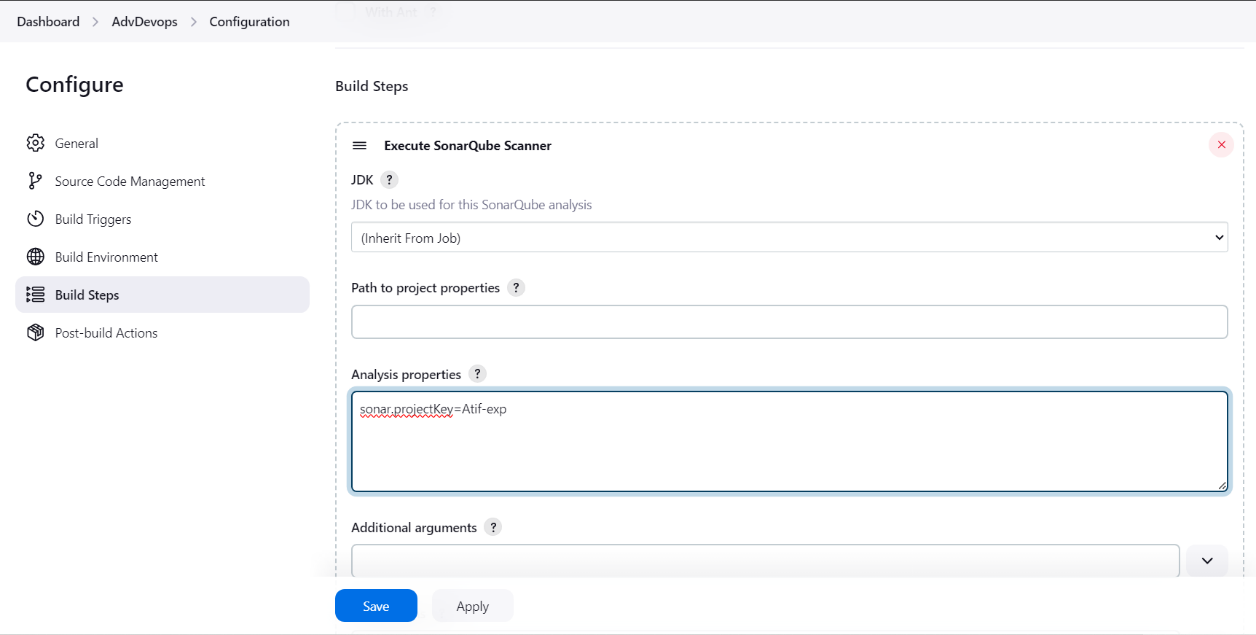
**sonar.projectKey=your-project-key**

**sonar.sources=.**

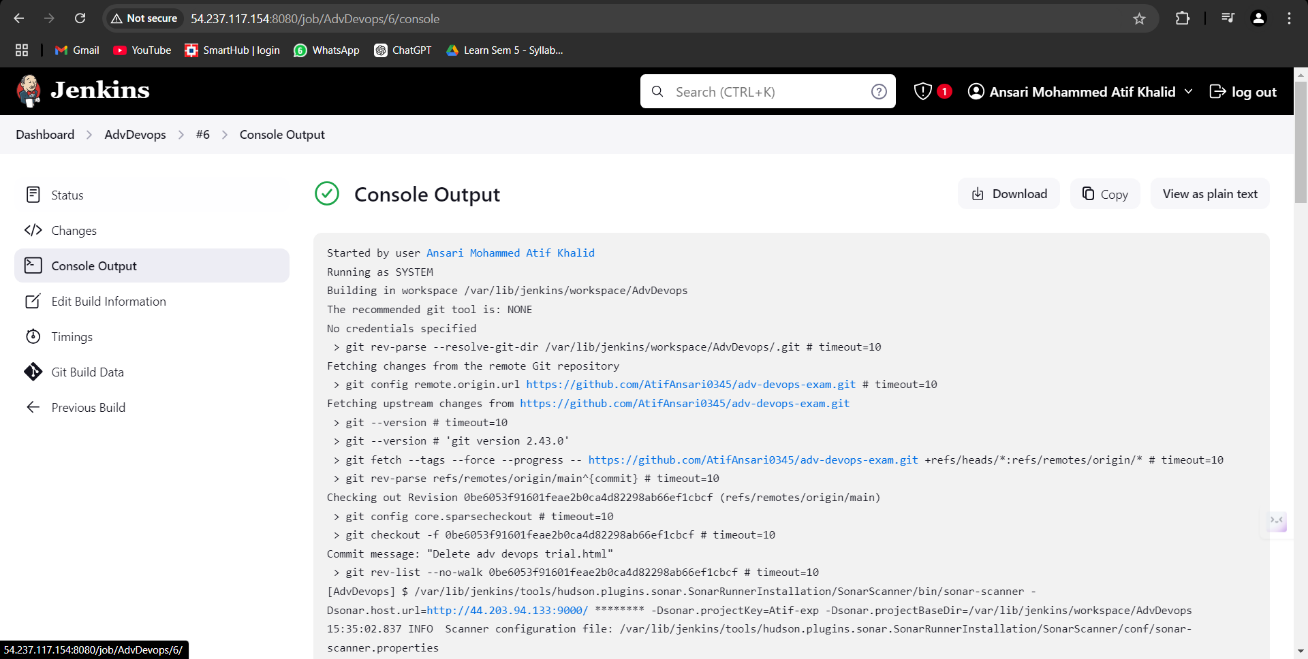
**sonar.host.url=http://your-sonarqube-public-ip:9000**

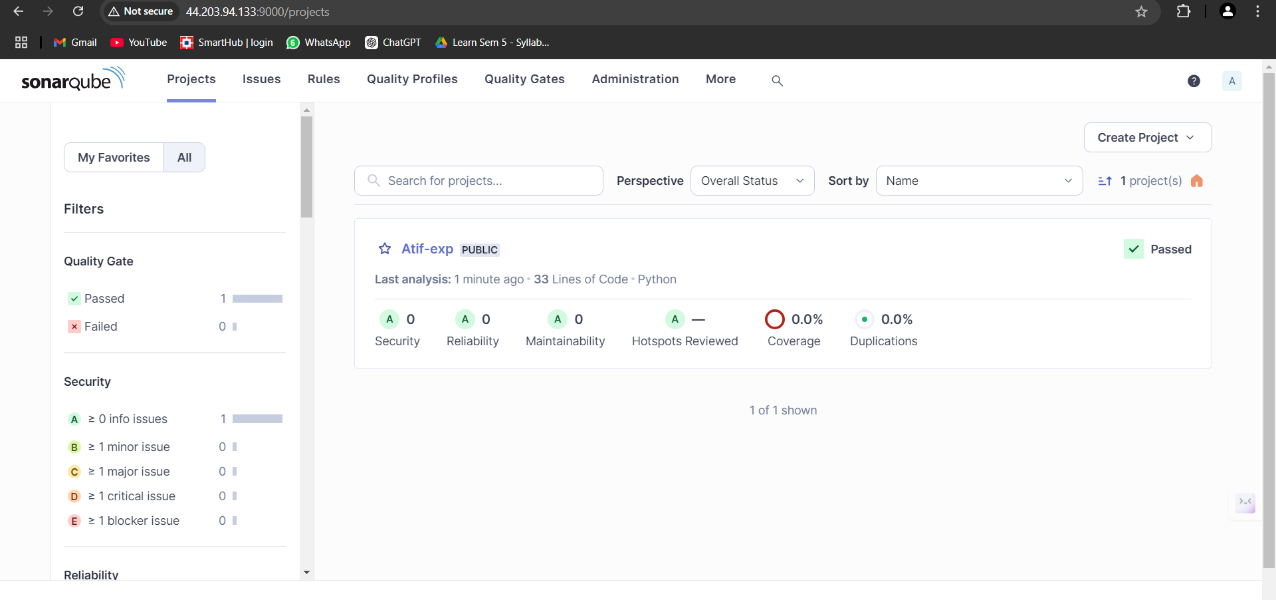
**sonar.login=Atif-token**





1. **Run the Pipeline**:
   * Trigger the build in Jenkins. Upon successful completion, review the build logs to verify there are no errors.
   * Visit SonarQube to view the results of the static code analysis, which will provide insights into code quality.





**6. Key Features & Applications**

* **Automated Builds**: Automatically builds and tests the code with every change pushed to the repository.
* **Static Code Analysis**: Analyzes the code for potential vulnerabilities, bugs, and code smells using SonarQube.
* **Real-time Feedback**: Developers receive immediate feedback on their code quality, facilitating timely corrections.
* **Integration with GitHub**: Utilizes GitHub webhooks for seamless integration between the repository and Jenkins.

**7. Conclusion**

This case study successfully demonstrated the setup of a Continuous Integration environment using Jenkins and SonarQube on AWS Cloud9. The integration enhances code quality management by automatically analyzing code during the build process. This approach not only streamlines development workflows but also ensures that code meets quality standards before deployment.