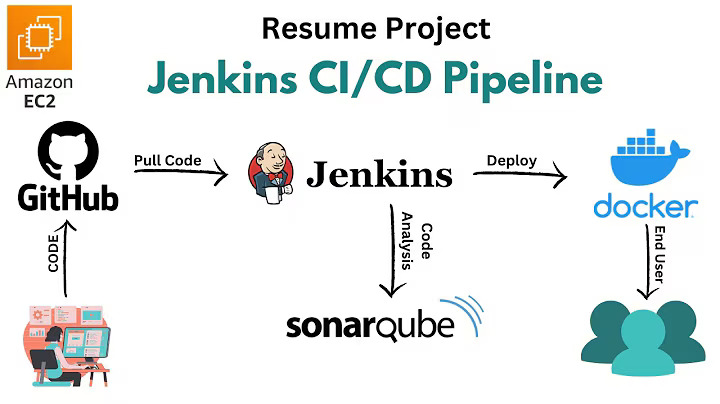
# **EC2 Deployment, CI/CD Pipeline, and Code Improvement Report**

## **1. Introduction**

This report demonstrates the use of cloud services and DevOps practices to develop a Continuous Integration, Development, and Deployment (CI/CD) pipeline. It outlines the setup and configuration of an AWS EC2 instance, the integration of GitHub and Jenkins for automated build, the implementation of static code analysis using SonarQube, and enhancements to a Python project for improved error handling and input validation. The process follows a structured, automation-driven approach to streamline code execution, maintain software quality, and enforce best practices through version control and continuous testing.



By implementing this pipeline, the project aims to streamline the software development lifecycle, ensuring efficient collaboration, continuous testing, and reliable deployment.

## **2. Task Breakdown and Implementation**

**The project is structured into six key tasks:**

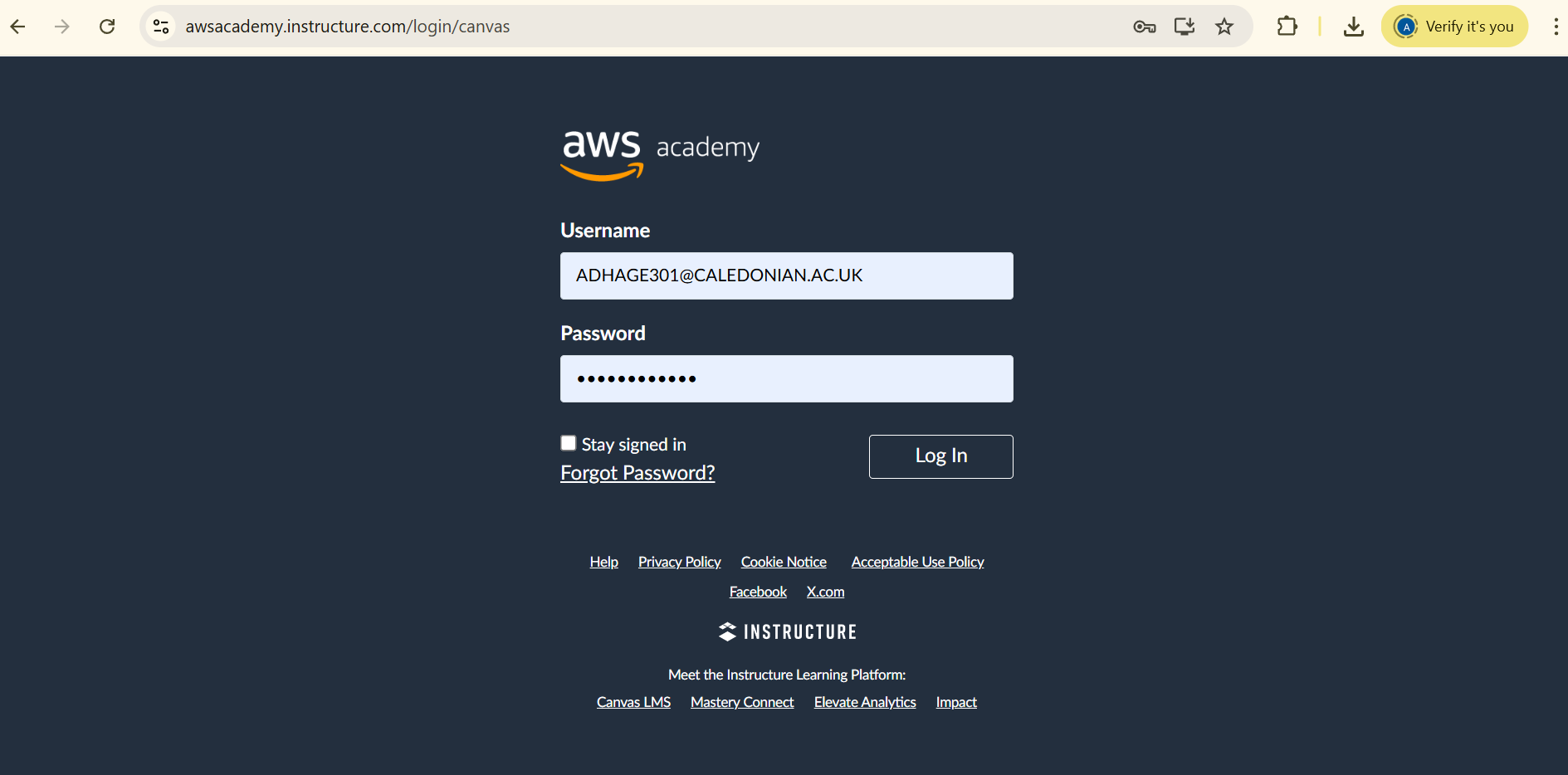
* **Task 1**: Configuring an Amazon EC2 instance.
* **Task 2**: Integrating Git with the EC2 machine.
* **Task 3**: Connecting Jenkins with the group’s Git repository.
* **Task 4**: Setting up SonarQube and SonarScanner for code quality analysis.
* **Task 5**: Conducting automated code testing.
* **Task 6**: Modifying the code and verifying the changes through testing.

### **Task 1: Deploying an AWS EC2 Instance and Installing Required Software**

This task involved mainly two steps, deploying an EC2 instance running Ubuntu operating software, of size **t2.large** with inbound configurations for **port 8080** and **port 9000 as well as Python, Git, and Jenkins** installations.

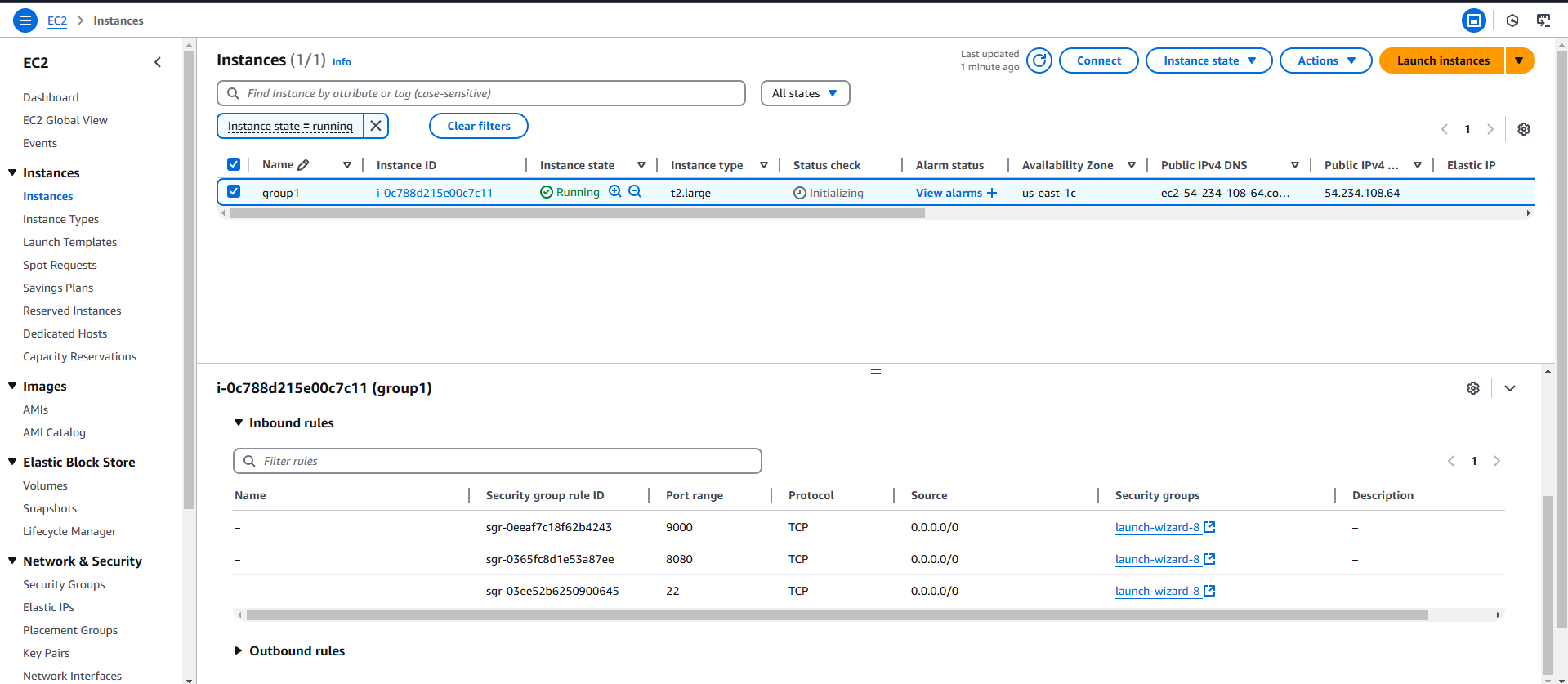
**Deploying EC2 Instance**

Creating an **EC2 instance** (Virtual Machine) requires an **AWS account** on the **Amazon Web Services (AWS) Cloud platform**. For this project, the AWS account was set up through **AWS Academy Learner Lab** using an **invitation link** provided by the module instructor. University credentials were used to create and access the accounts.



An Amazon EC2 instance was deployed with the following specifications:

* Instance Type: t2.large
* Storage: 8GB volume
* Operating System: Ubuntu (latest version)
* Open Ports: 8080, 9000 (to allow Jenkins and SonarQube access)



A **t2.large** instance was chosen because Jenkins and SonarQube require a minimum of **1 CPU and 1 GB of memory**, while SonarQube specifically needs **2 CPUs and 4 GB of memory** for optimal performance.

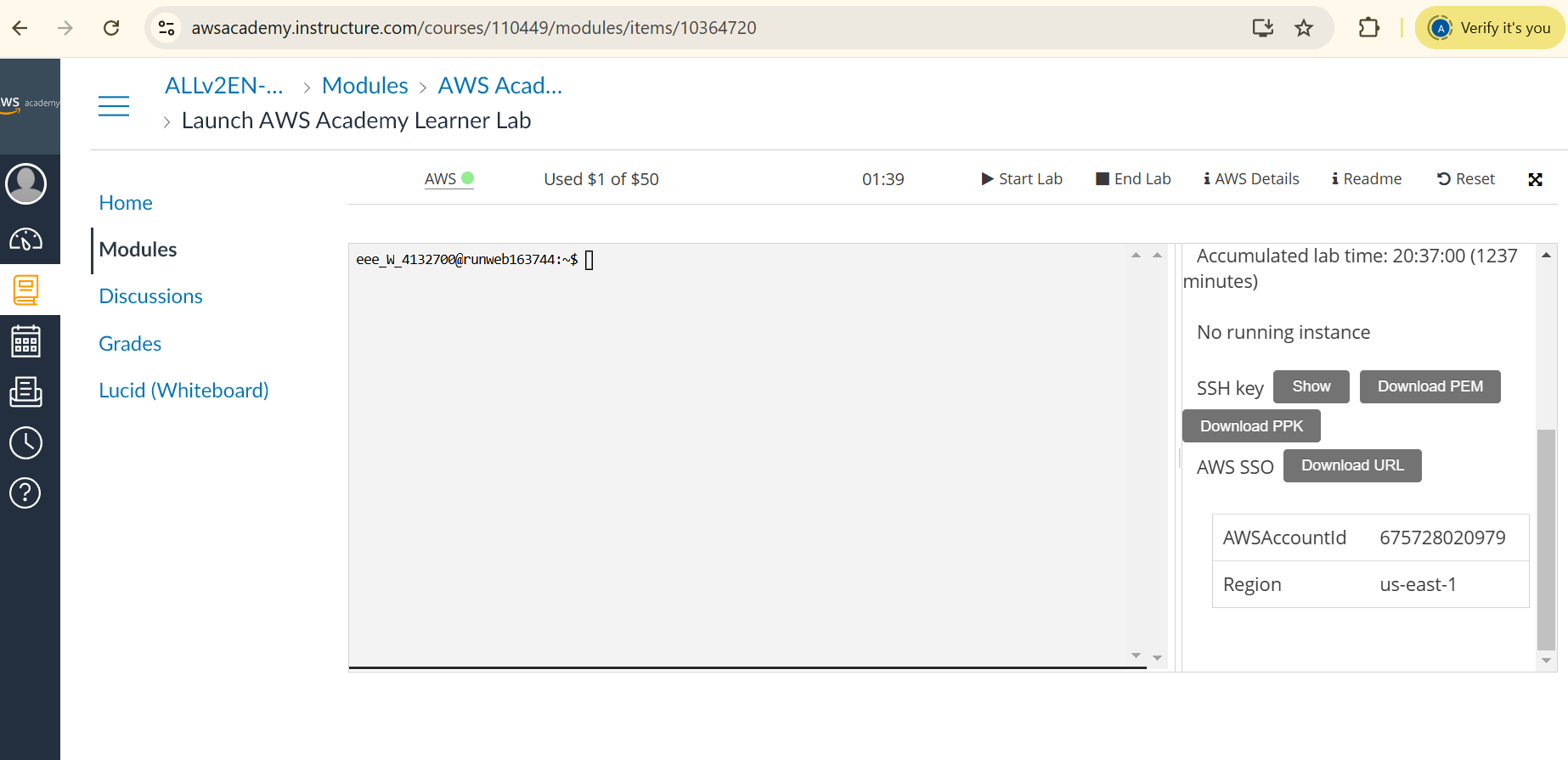
The inbound configurations for **port 8080** and **port 9000** were set to allow access to Jenkins and SonarQube on the **Group1 instance machine**, as these ports are their default server ports. **Port 8080** is used for the Jenkins web interface, while **port 9000** is required for SonarQube's dashboard and analysis services.

After setting up the instance, essential software packages were installed:

* Python: To support the Dex2Hex project execution
* Git: For version control and GitHub integration
* Jenkins: For CI/CD automation

To install **Git, Python, and Jenkins**, a secure connection to the **Group1 EC2 instance** was created using **Git Bash**, the **SSH key** of the AWS virtual machine, and the Public DNS or IP address of the EC2 instance.

Downloading the labsuser.pem file, which contains the SSH key for accessing the virtual machine.

Figure 1: figure 2 downloading labsuser.pem file

### **Steps for Connection:**

1. Open Git Bash: Launch Git Bash on your local machine. Git Bash is a command-line interface that allows you to interact with your machine and remote servers via SSH.
2. Navigate to the Directory Containing the labsuser.pem File: Use the cd command to navigate to the directory where the labsuser.pem file is stored:

**Command:**

cd "path/to/labsuser.pem"

Replace "path/to/labsuser.pem" with the actual path to the labsuser.pem file on your local machine.

1. Set Permissions for the labsuser.pem File: Set the correct file permissions for the labsuser.pem file to ensure it is secure and can be used for the SSH connection:

**Command:**

chmod 400 labsuser.pem

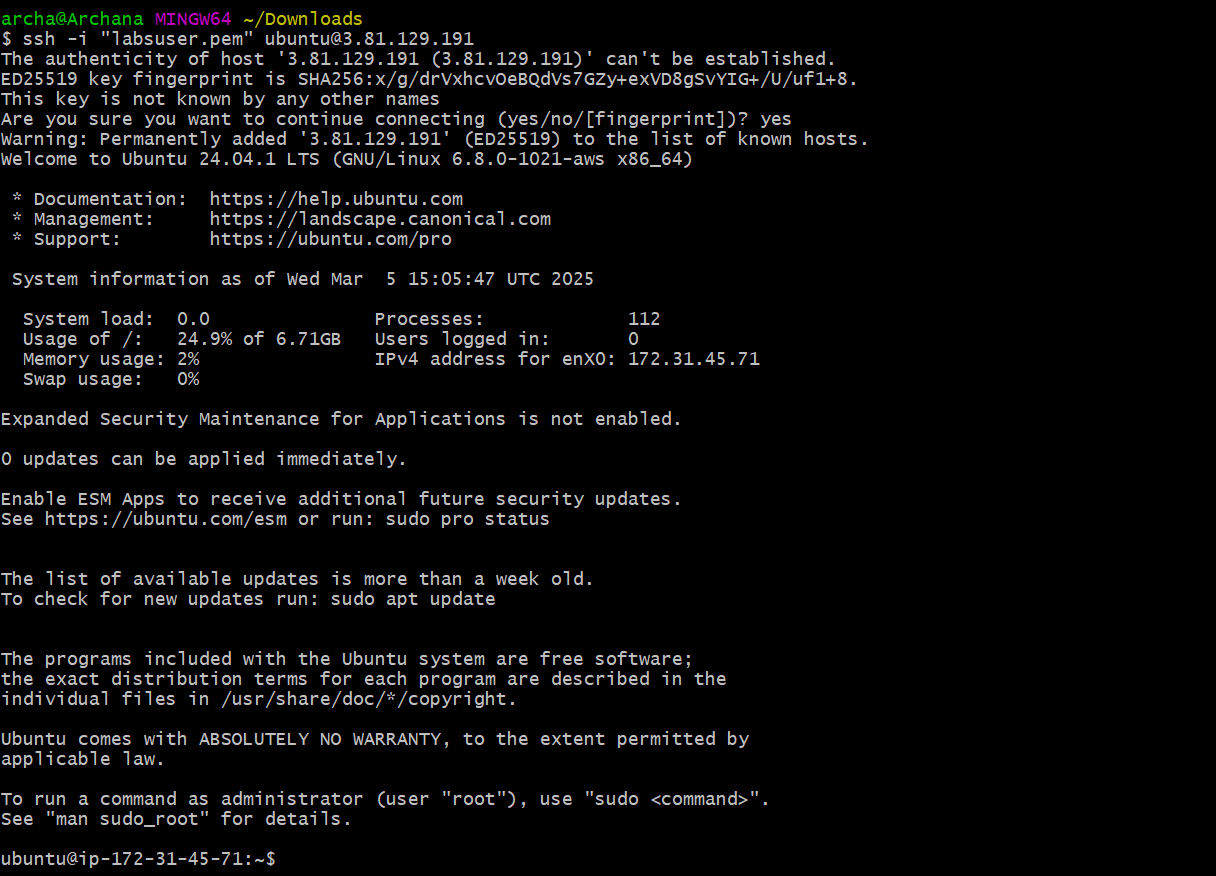
This command ensures that the private key file is readable only by the owner, which is required for secure SSH connections.

1. SSH into the EC2 Instance: Use the SSH command to establish a secure connection to the EC2 instance. The format for this command is:

**command:**

ssh -i "labsuser.pem" ubuntu@PUBLIC\_DNS

Replace PUBLIC\_DNS with the actual public DNS or IP address of the EC2 instance. For example: **ssh -i "labsuser.pem" ubuntu@3.81.129.191**



### **Jenkins Installing**

#### sudo apt update

sudo apt install fontconfig openjdk-17-jre

#### java -version

#### **Command:**

sudo apt update

Explanation:  
This command updates the system’s package index, ensuring that the latest versions of software are retrieved from the repositories. It does not install or upgrade any packages but refreshes the list of available software to ensure an up-to-date installation process.

#### **Command:**

sudo apt install fontconfig openjdk-17-jre

Explanation:  
This command installs the necessary packages for running Java applications:

* fontconfig: A library that manages fonts, ensuring Java applications can render text properly.
* openjdk-17-jre: The Java Runtime Environment (JRE) for OpenJDK 17, which allows users to execute Java applications but does not include development tools like the Java compiler (javac).

#### **Command:**

java -version

Explanation:  
This command verifies the installation of Java by displaying the installed Java version, runtime environment, and JVM details. If Java has been installed successfully, it will output the version information.

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 jenkins

#### **Command:**

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

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

Explanation:  
This command downloads the official Jenkins repository key from pkg.jenkins.io and saves it to /usr/share/keyrings/jenkins-keyring.asc. This key is required to verify the authenticity of the Jenkins packages before installation.

#### **Command:**

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

Explanation:  
This command adds the Jenkins repository to the system’s package sources list. The signed-by option ensures that only packages signed with the downloaded key are trusted.

#### **Command:**

sudo apt-get update

Explanation:  
This command refreshes the package index to include the newly added Jenkins repository, ensuring that the latest Jenkins packages can be installed.

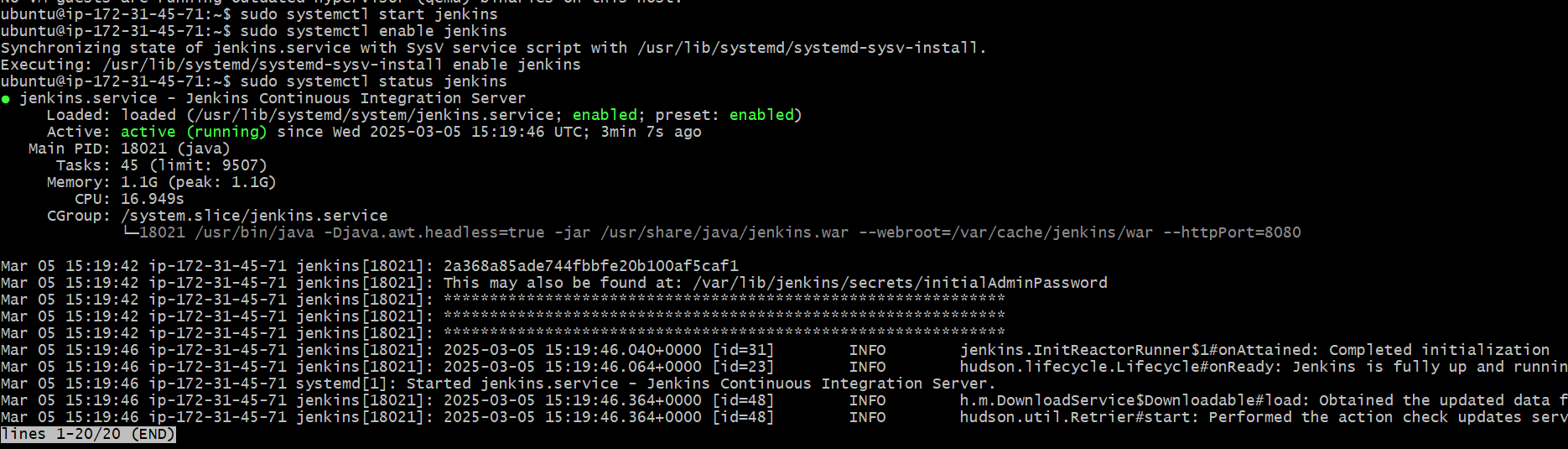
#### **Command:**

sudo apt-get install jenkins

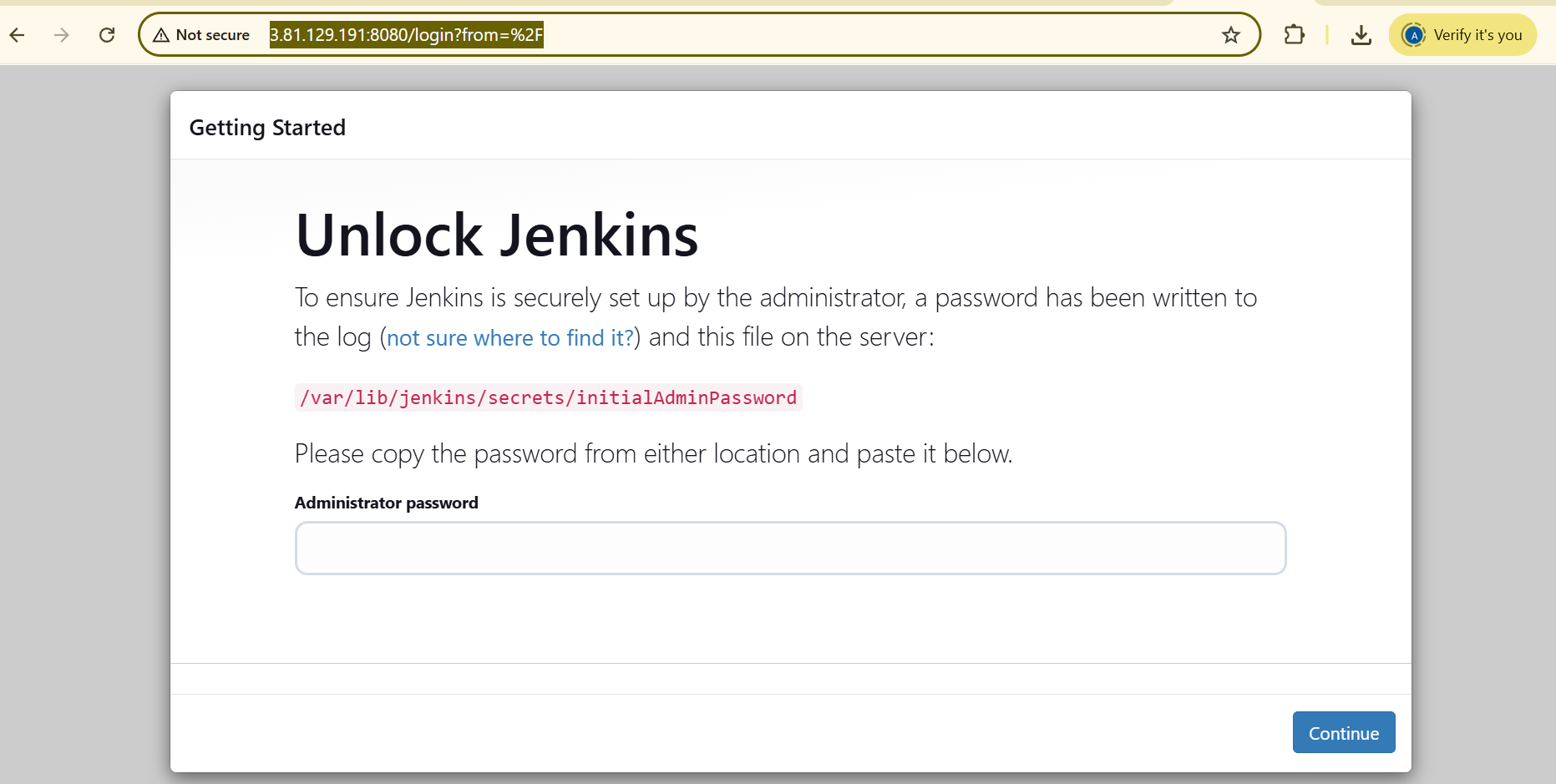
sudo systemctl start Jenkins

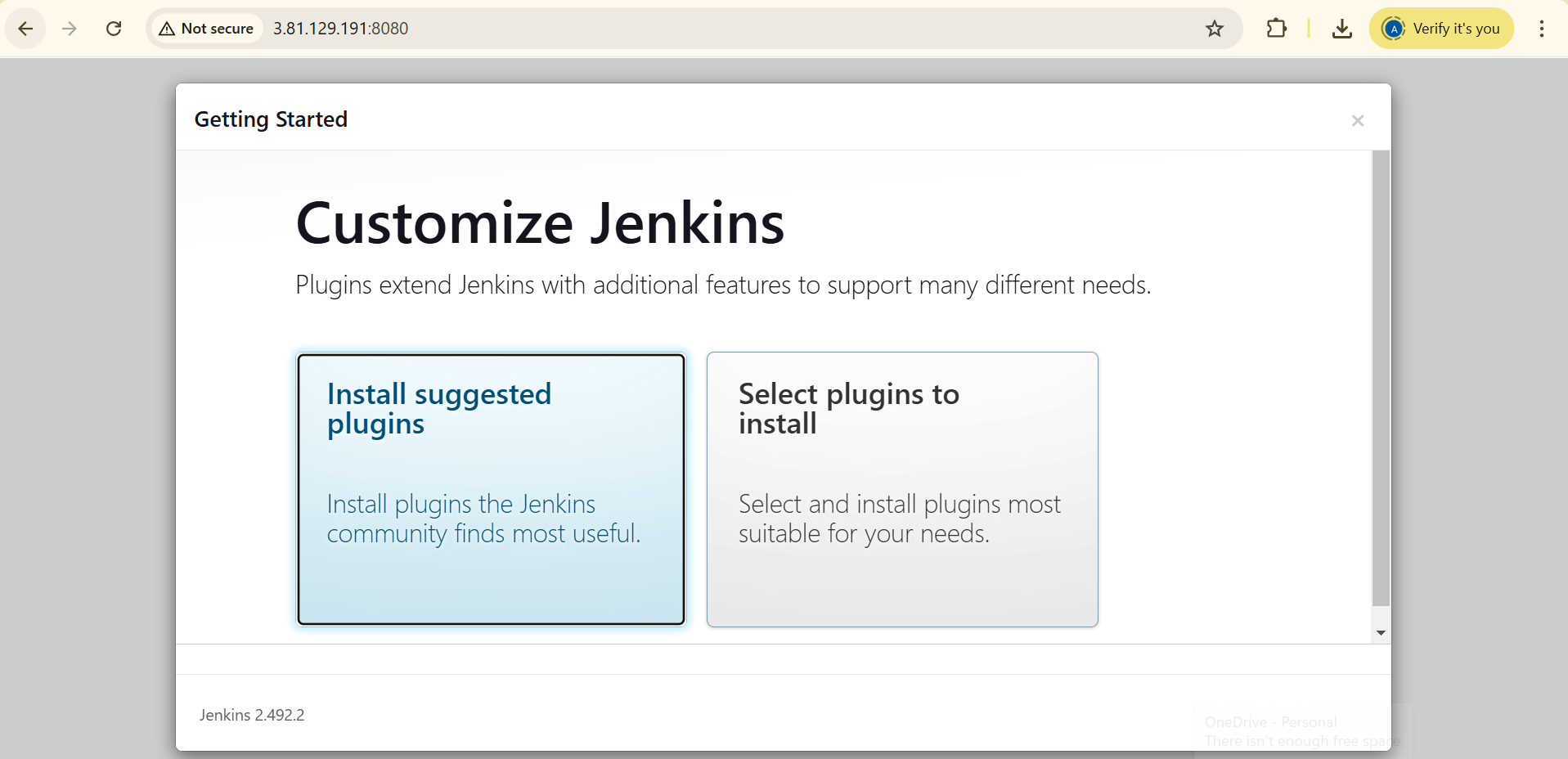
sudo systemctl enable Jenkins

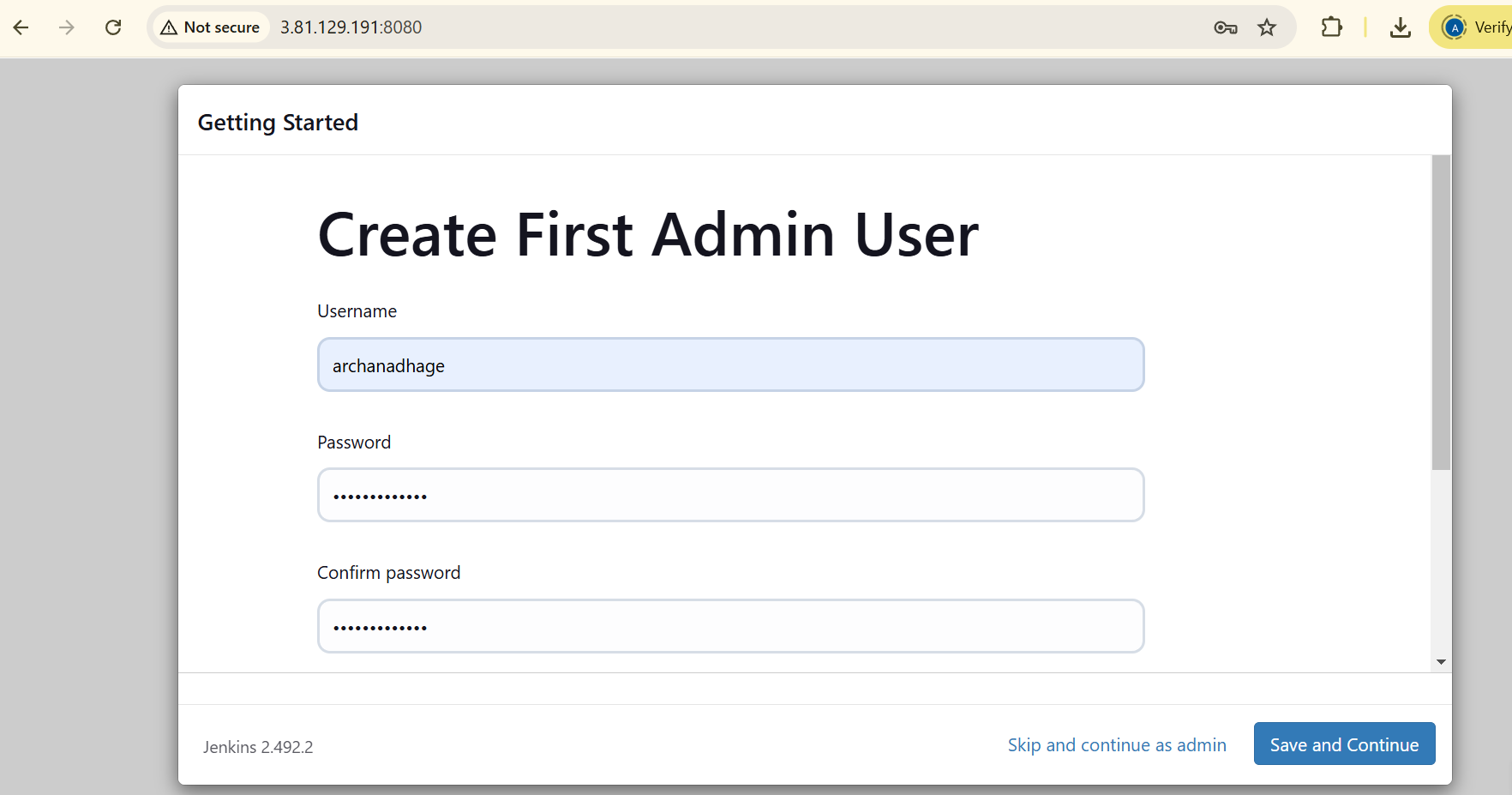
sudo systemctl status jenkins

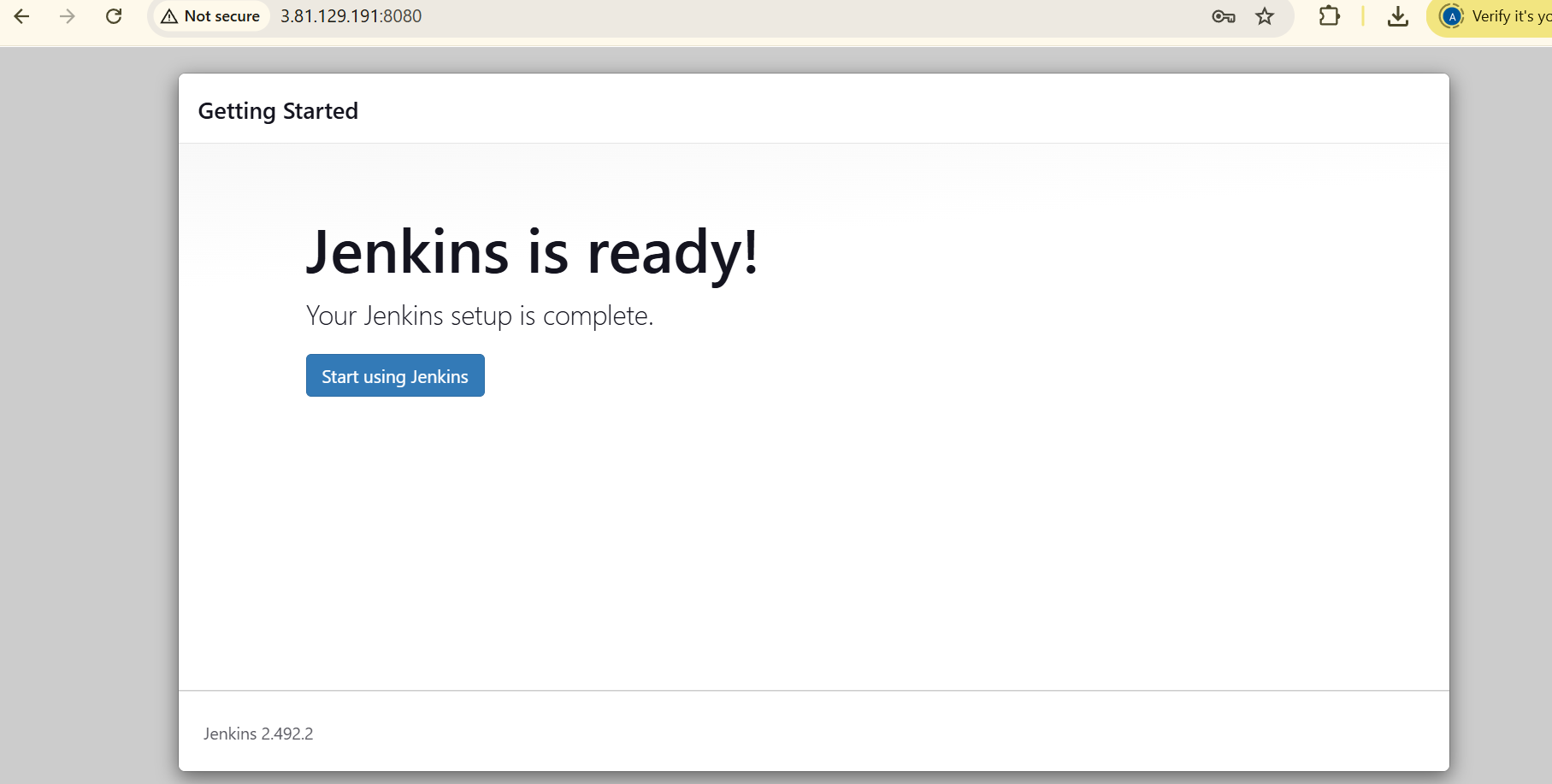


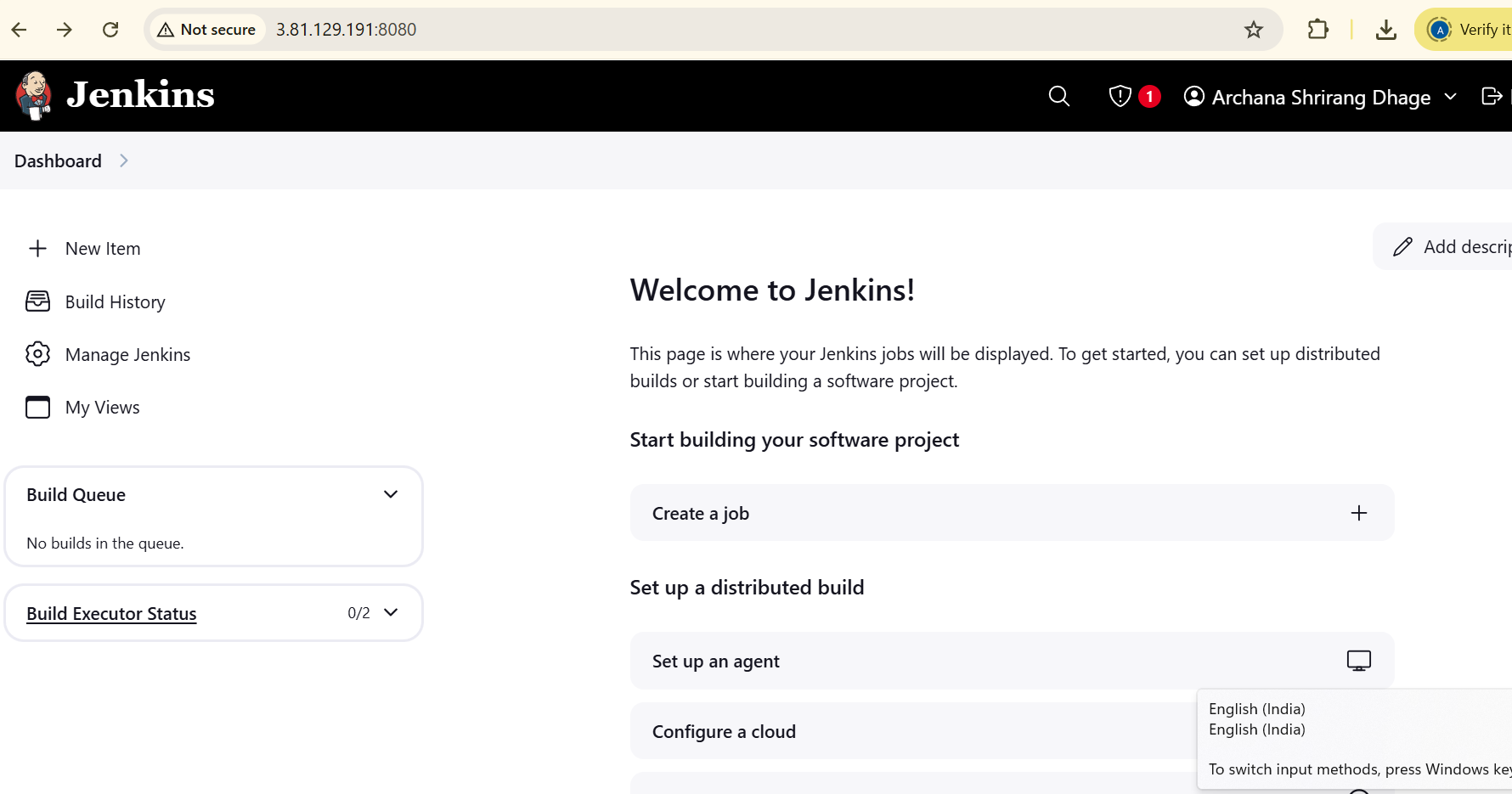
Explanation:  
This command installs Jenkins from the official repository, setting up the necessary dependencies and configurations. After installation, Jenkins runs as a service, typically accessible via http://localhost:8080 in a web browser.











### **Python Installing**

Command

sudo apt install -y python3

python3 –version

**explanation**

The command sudo apt install -y python3 installs Python 3 on an Ubuntu system. It uses apt, the package manager for Ubuntu, to fetch and install the latest available version of Python 3 from the official repositories. The -y flag automatically confirms the installation, bypassing the need for user input. Python 3 is the core version of Python used for running modern Python applications, and this command ensures that the system has the required Python interpreter installed to execute Python code and support various Python-based applications.

### **Git Installing**

Comand

sudo apt update sudo apt install -y git

git –version

**Explanation**

The commands sudo apt update and sudo apt install -y git are used to update the package list and install Git on an Ubuntu system. First, sudo apt update updates the local repository information, ensuring the system knows about the latest available software and package versions. Then, sudo apt install -y git installs Git, a version control system, on the system. The -y flag automatically confirms the installation, bypassing any prompts. The sudo ensures the commands are executed with administrative privileges, allowing changes to system software.

### **Task 2: Configuring Git and Setting Up a Remote Repository**

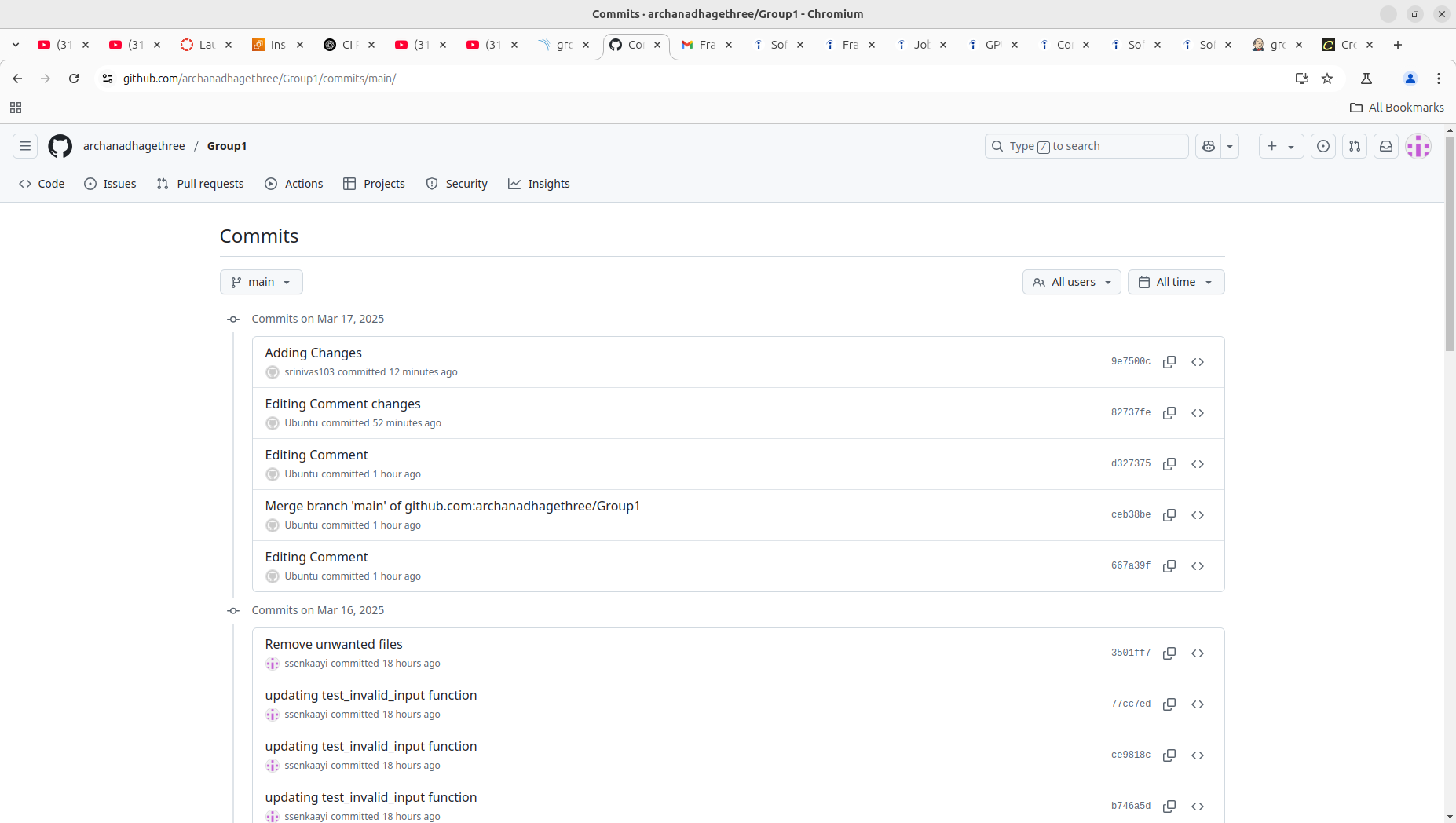
**Git configurations**

git config –global user.name ssenkaayi

git config –global user.email ssenkaaayi@gmail.com

git config –global -l

The commands git config --global user.name "ssenkaayi" and git config --global user.email "ssenkaaayi@gmail.com" are used to set the global username and email address for Git, which will be associated with all commits made by the user on the system. The --global flag ensures that these configurations apply to all repositories on the machine. After setting the name and email, the command git config --global -l lists all the global configuration settings for Git, allowing the user to verify their configurations, including the username and email. These settings help Git track the author's identity for each commit and can be important for collaborative version control.



Git was configured on the EC2 instance, and a remote GitHub repository called Group1 was created on Archana’s GitHub account. The Dec2Hex.py file was pushed to this repository, ensuring visibility on both GitHub and the EC2 instance. This step enables automated version control and integration with Jenkins.

Below are some of the extracted commits from Archana’s GitHub repository.

### 1a59f6c Changes in comment

1. 9e7500c Adding Changes
2. 82737fe Editing Comment changes
3. d327375 Editing Comment
4. ceb38be Merge branch 'main' of github.com:archanadhagethree/Group1
5. 667a39f Editing Comment
6. 3501ff7 Remove unwanted files
7. 77cc7ed updating test\_invalid\_input function
8. ce9818c updating test\_invalid\_input function
9. b746a5d updating test\_invalid\_input function

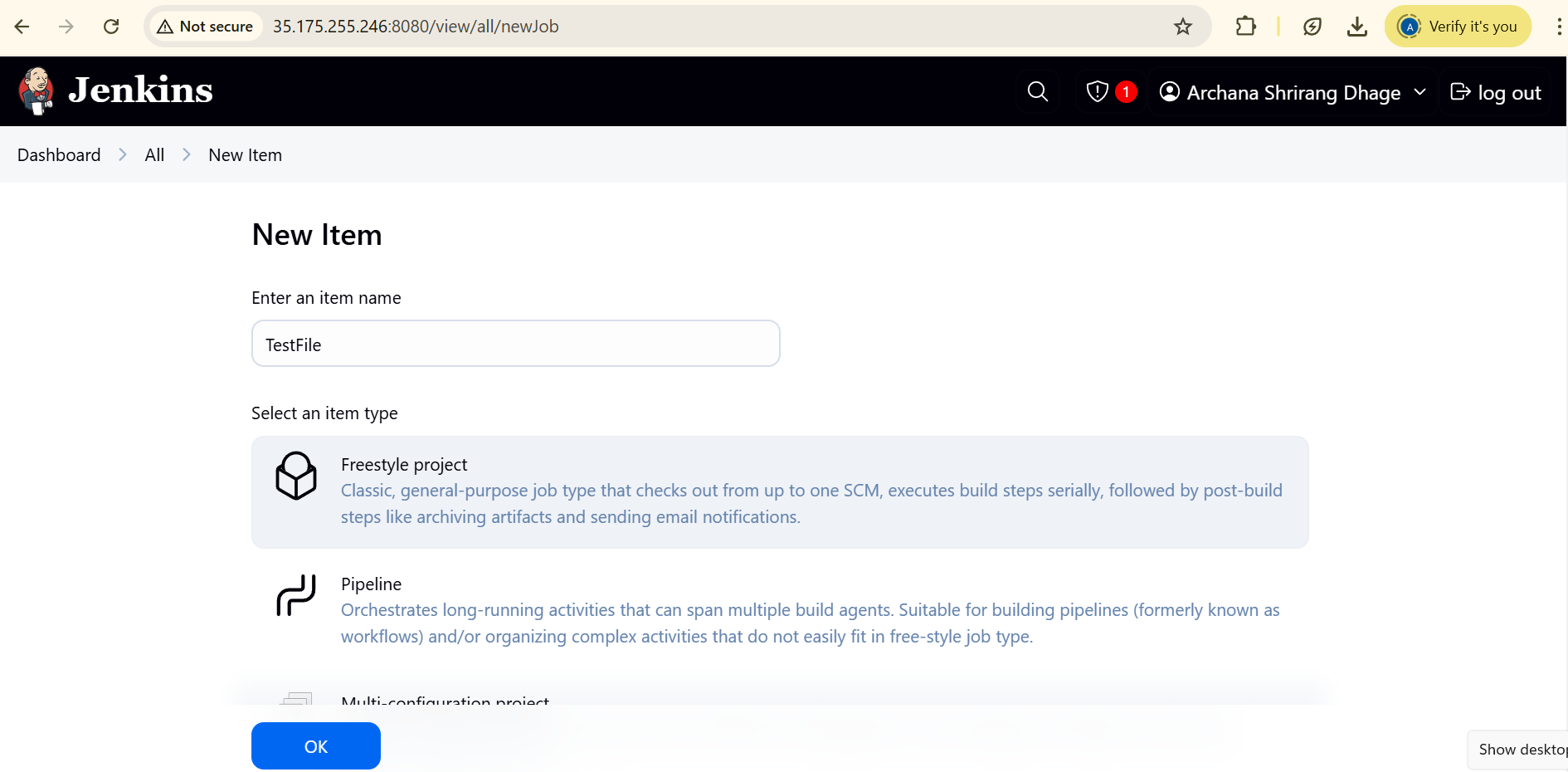
### **Task 3: Setting Up a Jenkins Freestyle Project**

A Jenkins Freestyle Project was configured to automate the following processes:

a) Detect changes in the GitHub repository – The Jenkins job was set to trigger automatically when changes were pushed.  
b) Compile the code – A build step was added to ensure the Python script executes correctly.  
c) Run the code – The project was executed with test inputs to validate the functionality.

### ****Step 1: Create a New Freestyle Project****

1. After Jenkins is set up, click on **New Item** in the Jenkins dashboard.
2. Enter a name for your project (e.g., "MyFreestyleProject").
3. Select **Freestyle project** and click **OK**.

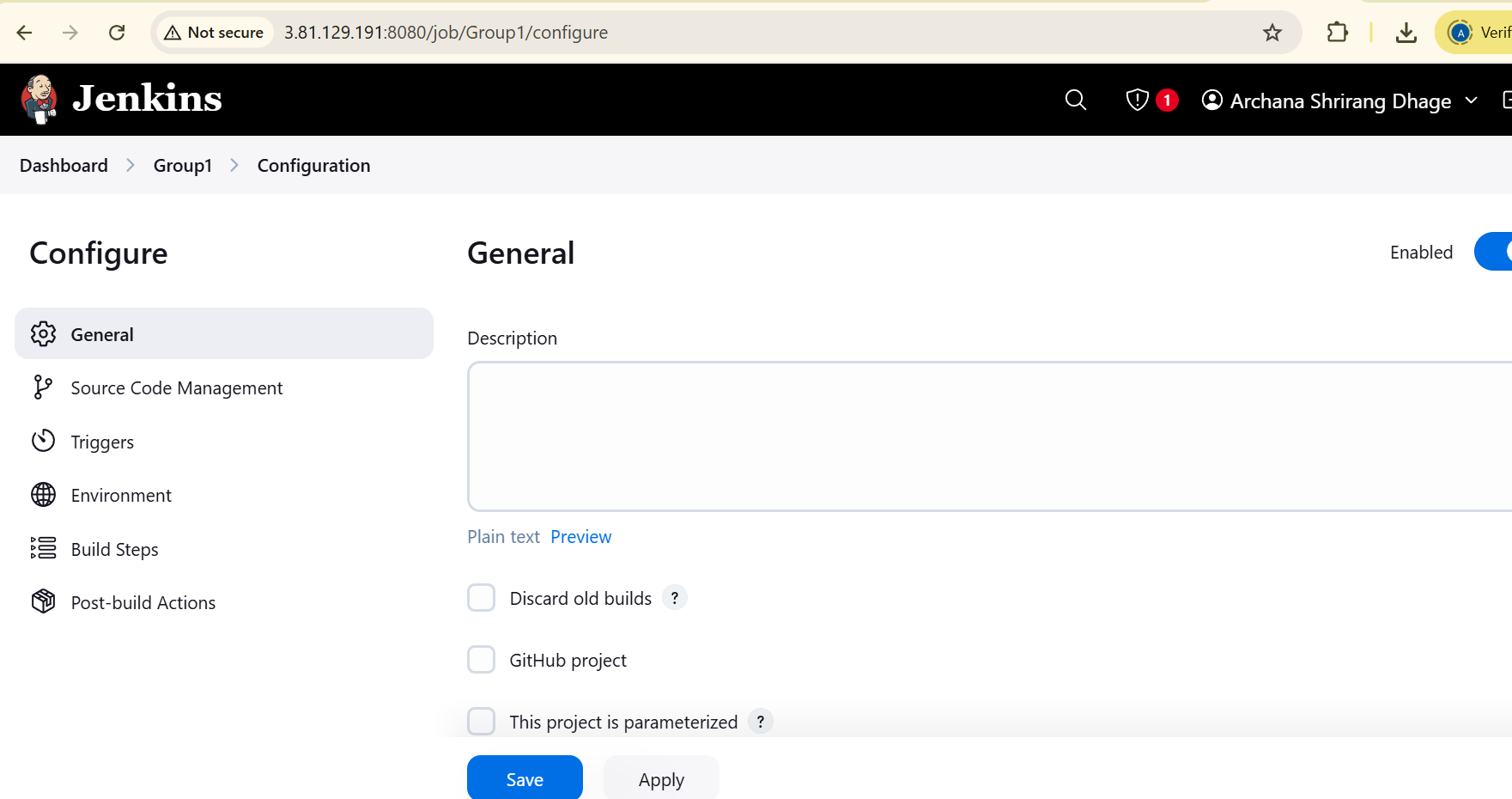


### ****Step 2: Configure the Freestyle Project****

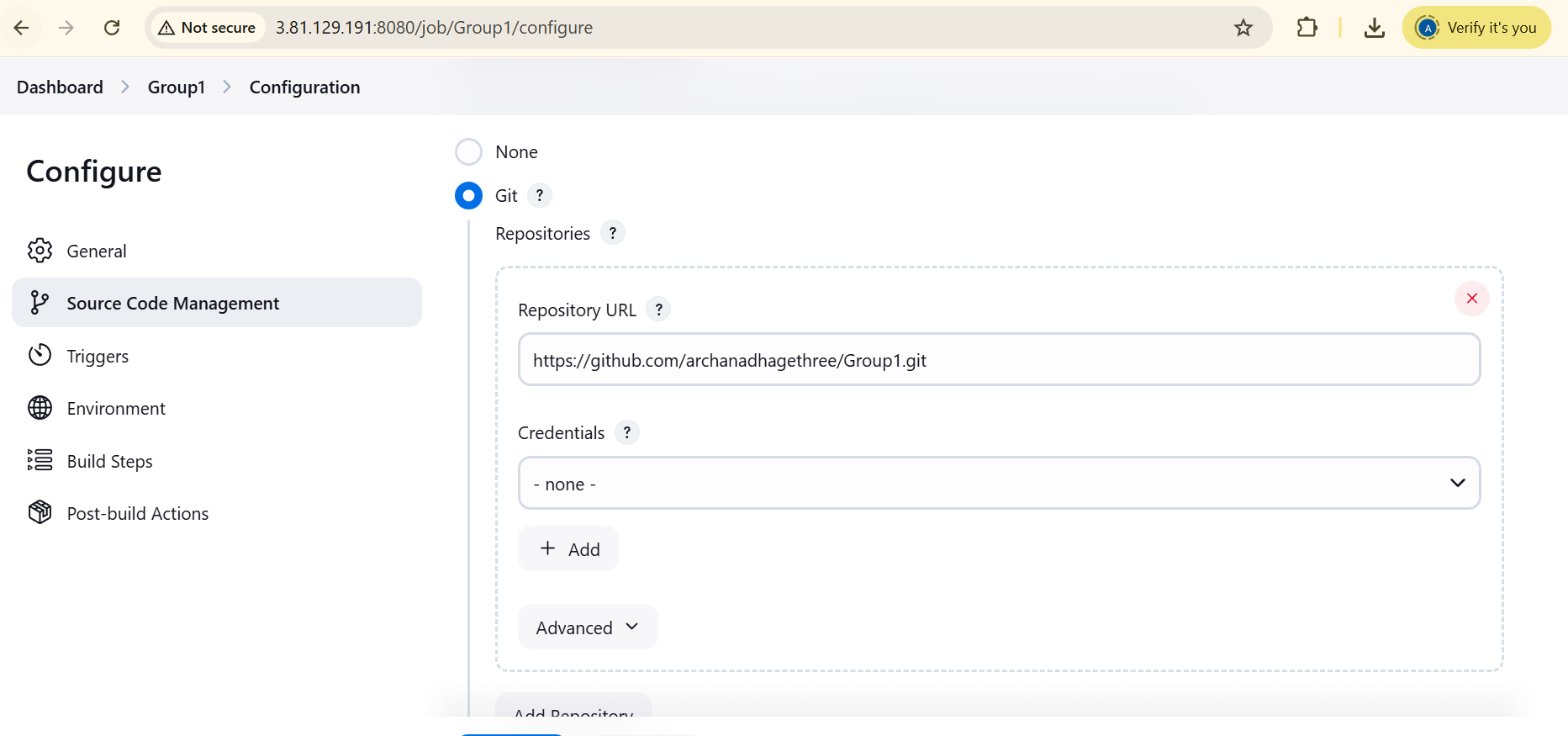
**General**:

Optionally, provide a description for your project.

**Source Code Management**: If your project involves source code, select a version control system (e.g., Git) and enter the repository URL.

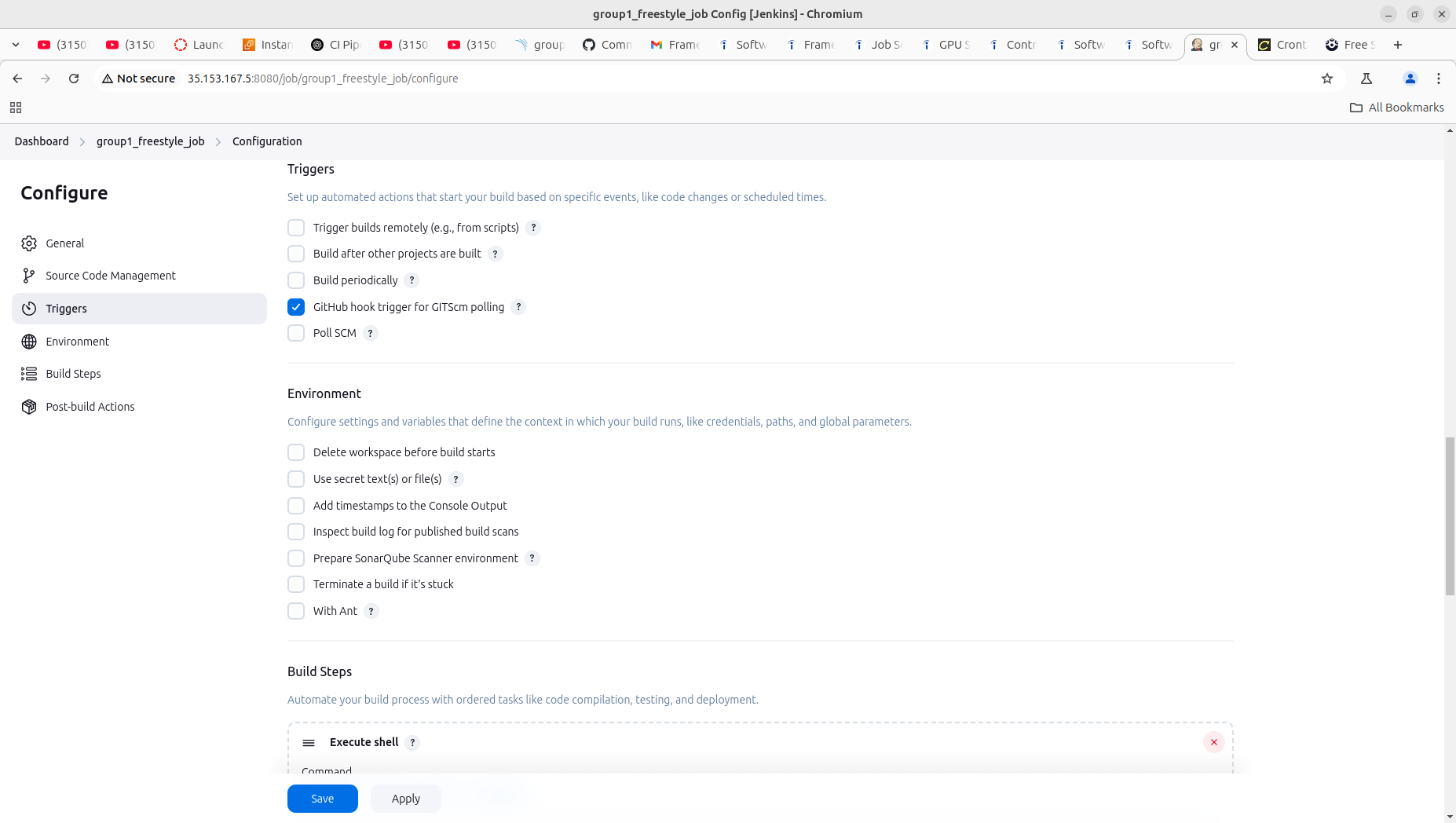


If using Git, you will need to specify the repository URL and credentials if necessary.



**Build Triggers**:

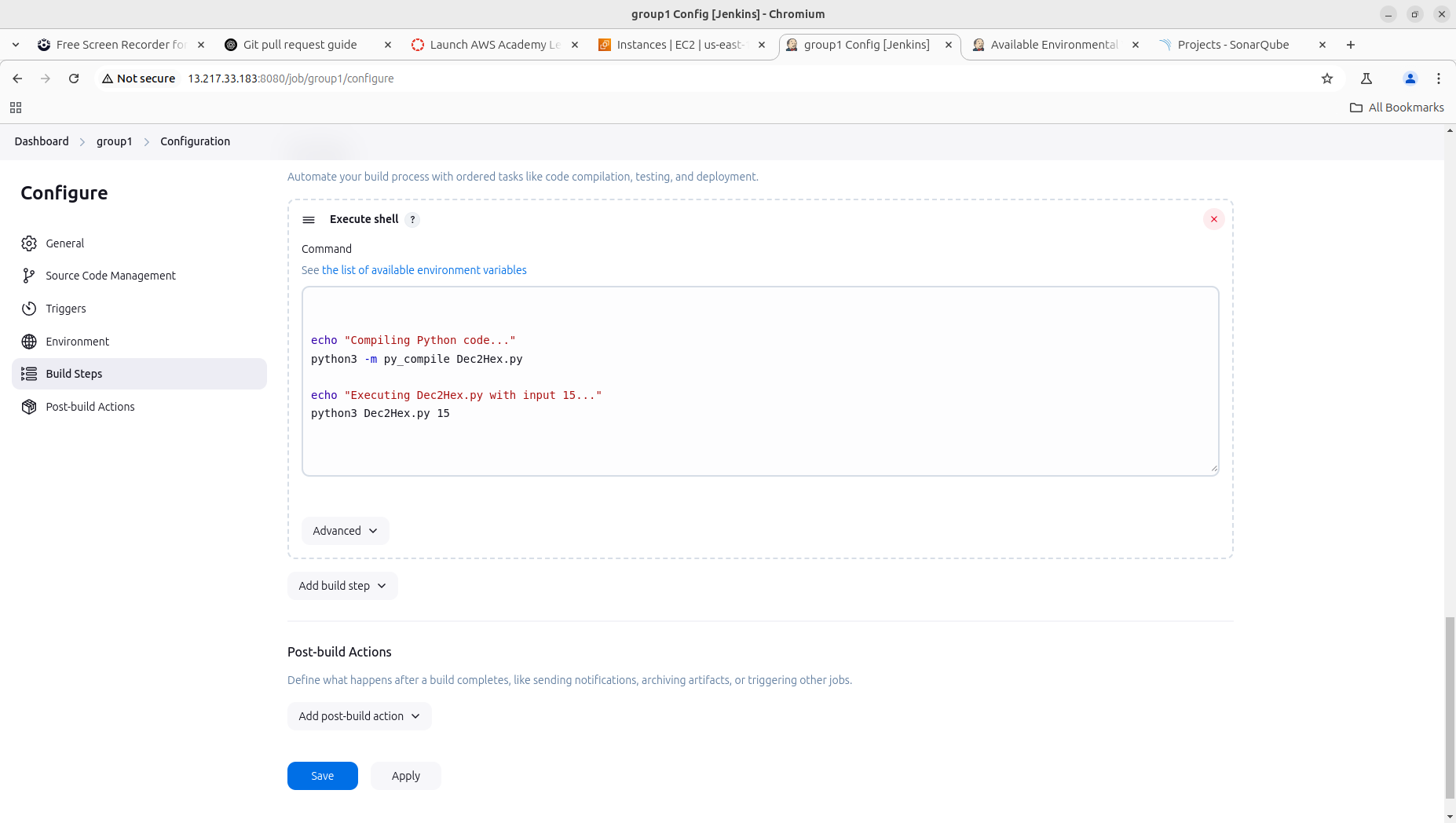
Set triggers to automatically start the build using **GitHub hook trigger for GITScm polling,** To trigger a build when a change is made in the repository.



**Build**:

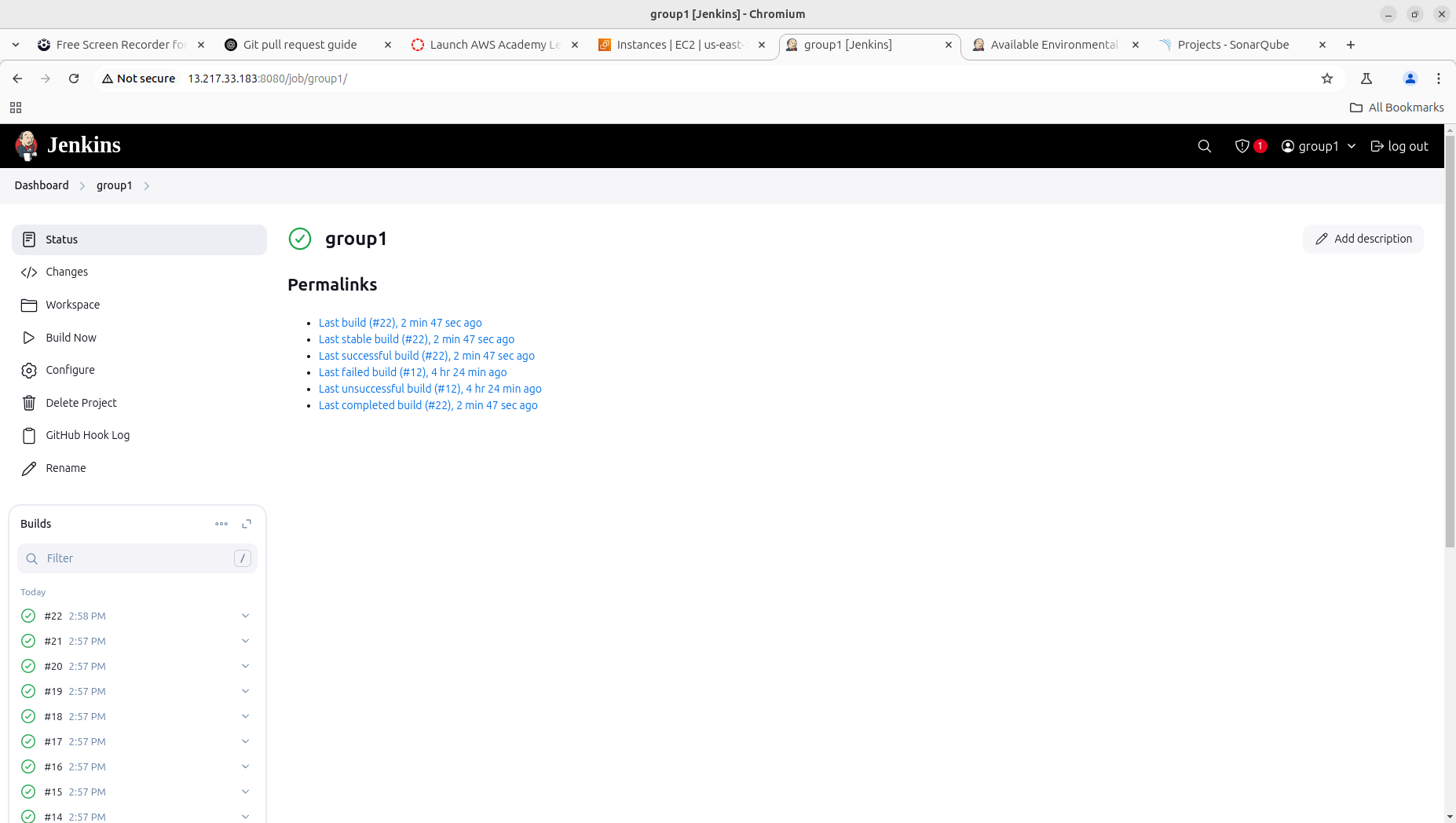
Define the build steps, such as:

* **Execute Shell**: Run shell commands or scripts.

****

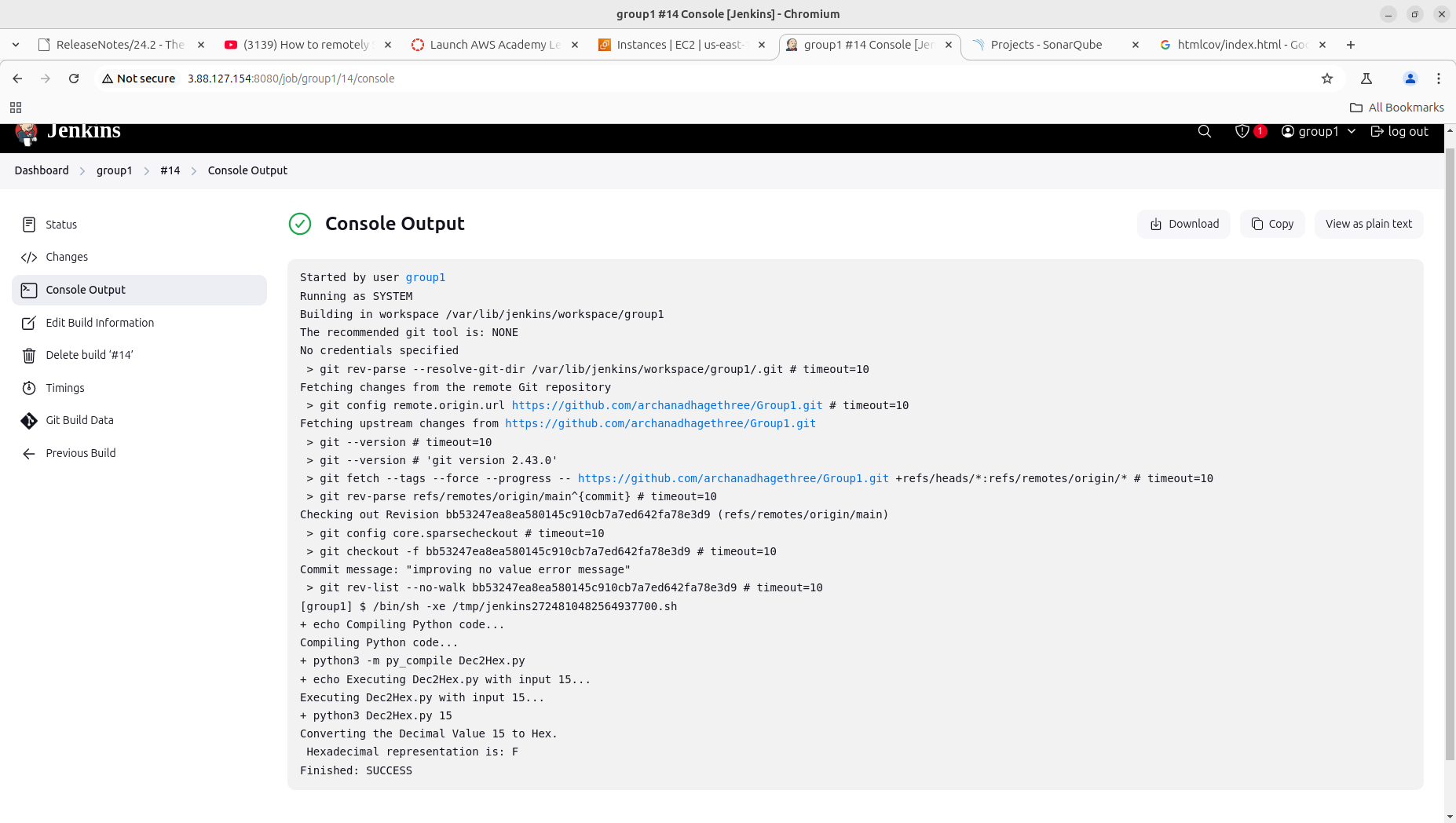
### ****Step 3: Save and Build****

* Once all the configurations are set, click **Save** to save the project.
* To run the project, click on **Build Now**.



### ****Step 4: Monitor the Build****

* Jenkins will start the build process, and you can monitor the progress on the project dashboard.
* After completion, you can view the build logs and check whether the build was successful or failed.



### **Task 4: Implementing Static Code Analysis with SonarQube**

To implement **Static Code Analysis with SonarQube** on **Dec2Hex.py file**, the following steps were taken:

### ****Step 1: Installing and Setting Up SonarQube****

**Install SonarQube**:

The following codes were run to install, sonarQube on group1 EC2 instance.

### 1.1 Update and Upgrade System Packages

1. To ensure all system packages are up to date, run the following commands:
2. sudo apt update -y && sudo apt upgrade -y

### 1.2Install Required Dependencies

1. Install OpenJDK 17, PostgreSQL, and other necessary packages:
2. sudo apt install openjdk-17-jdk wget unzip -y
3. java -version

### 1.3Create a SonarQube User

1. To enhance security, create a dedicated SonarQube user:
2. sudo adduser --system --no-create-home --group --disabled-login sonarqube

### 1.4: Install and Configure PostgreSQL

1. Install PostgreSQL and create a database for SonarQube:
2. sudo apt install postgresql postgresql-contrib -y
3. sudo -i -u postgres psql
4. Execute the following PostgreSQL commands:
5. CREATE DATABASE sonarqube;
6. CREATE USER sonar WITH ENCRYPTED PASSWORD 'sonar@123';
7. GRANT ALL PRIVILEGES ON DATABASE sonarqube TO sonar;
8. ALTER DATABASE sonarqube OWNER TO sonar;
9. \q

### 1.5: Download and Install SonarQube

1. Navigate to the /opt directory and download SonarQube:
2. cd /opt
3. sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.3.0.82913.zip
4. Extract and move the SonarQube files:
5. sudo apt update -y
6. sudo apt install unzip -y
7. sudo unzip sonarqube-10.3.0.82913.zip
8. sudo mv sonarqube-10.3.0.82913 sonarqube
9. Change ownership of the SonarQube directory:
10. sudo chown -R sonarqube:sonarqube /opt/sonarqube

### 1.6: Configure SonarQube

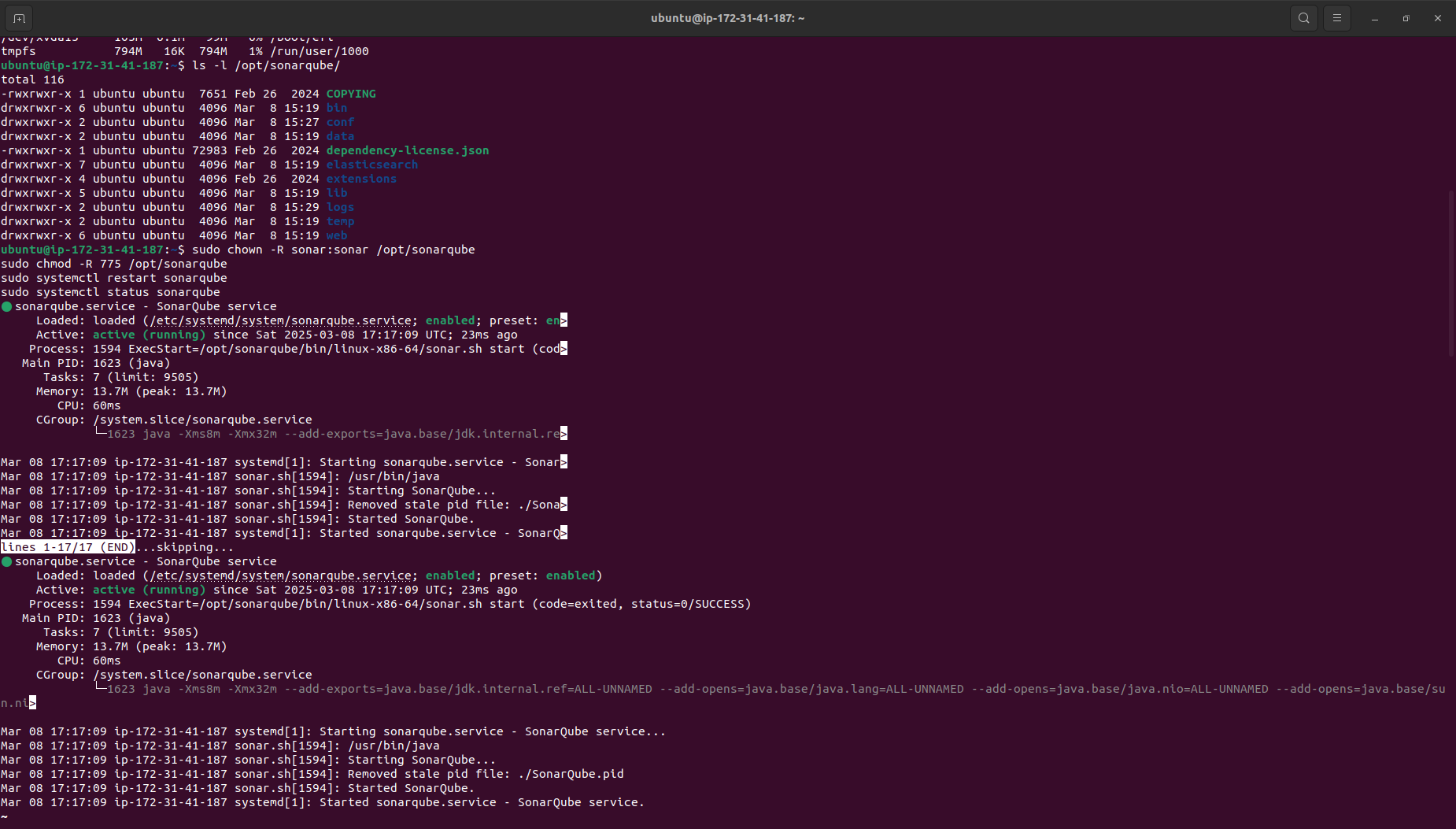
1. Edit the SonarQube configuration file:
2. sudo nano /opt/sonarqube/conf/sonar.properties
3. Update the file with the following database configurations:
4. sonar.jdbc.username=sonar
5. sonar.jdbc.password=sonar@123
6. sonar.jdbc.url=jdbc:postgresql://localhost:5432/sonarqube
7. sonar.web.host=0.0.0.0

### 1.7: Create a Systemd Service for SonarQube

1. Create a service file:
2. sudo nano /etc/systemd/system/sonarqube.service
3. Add the following content:
4. [Unit]
5. Description=SonarQube service
6. After=syslog.target network.target
7. [Service]
8. Type=forking
9. User=sonarqube
10. Group=sonarqube
11. ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
12. ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop
13. Restart=always
14. [Install]
15. WantedBy=multi-user.target1.8: Enable and Start SonarQube Service
16. Reload systemd, enable, and start the SonarQube service:
17. sudo systemctl daemon-reload
18. sudo systemctl enable sonarqube
19. sudo systemctl start sonarqube

### 1.9: Verify SonarQube Service

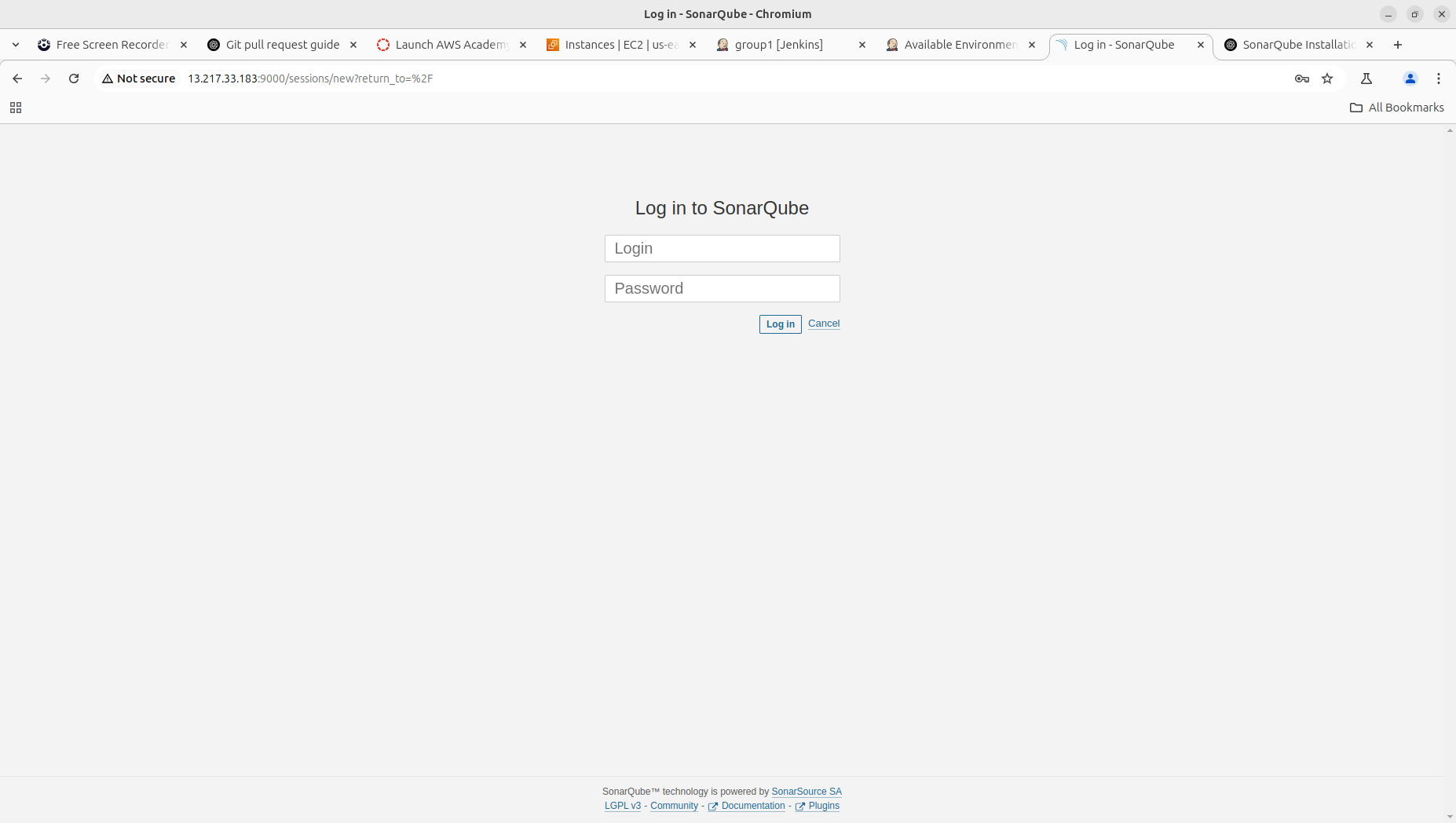
1. Check the status of the SonarQube service:
2. sudo systemctl status sonarqube



The above installation steps were derived from the official [SonarQube Documentation](https://docs.sonarqube.org/) and best practices for deploying SonarQube on Ubuntu.

**Access SonarQube**:

After installation, open your browser and visit http://localhost:9000 to access the SonarQube dashboard. Log in using the default credentials (admin for both username and password).



### ****Step 2: Manual Installation and Configuration of SonarQube Scanner****

SonarQube Scanner is used to analyze the source code in your project.

**2.1. Installation Steps**

1. **Navigate to the Home Directory**

cd ~ # Change to your home directory

1. **Download SonarScanner**

wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip

This command retrieves the SonarScanner binary from the official SonarSource repository.

1. **Extract the Downloaded File**

unzip sonar-scanner-cli-5.0.1.3006-linux.zip

This command extracts the contents of the downloaded ZIP file.

1. **Move SonarScanner to** /opt/

sudo mv sonar-scanner-5.0.1.3006-linux /opt/sonar-scanner

Moving the directory to /opt/ allows system-wide access to SonarScanner.

1. **Change Ownership of the Directory**

sudo chown -R $USER:$USER /opt/sonar-scanner

This command ensures that the current user has ownership over the SonarScanner directory, allowing necessary read/write permissions.

1. **Update System Path**

export PATH=$PATH:/opt/sonar-scanner/bin

source ~/.bashrc # or source ~/.zshrc

These commands add SonarScanner to the system’s PATH, making it accessible from any directory.

1. **Verify Installation**

sonar-scanner --version

The installation steps provided in this document are based on the official SonarSource documentation, available at: [SonarQube Scanner Installation Guide](https://docs.sonarsource.com/sonarqube/latest/analysis/scan/sonarscanner/).

### ****Step 3: Create a SonarQube Project for Python****

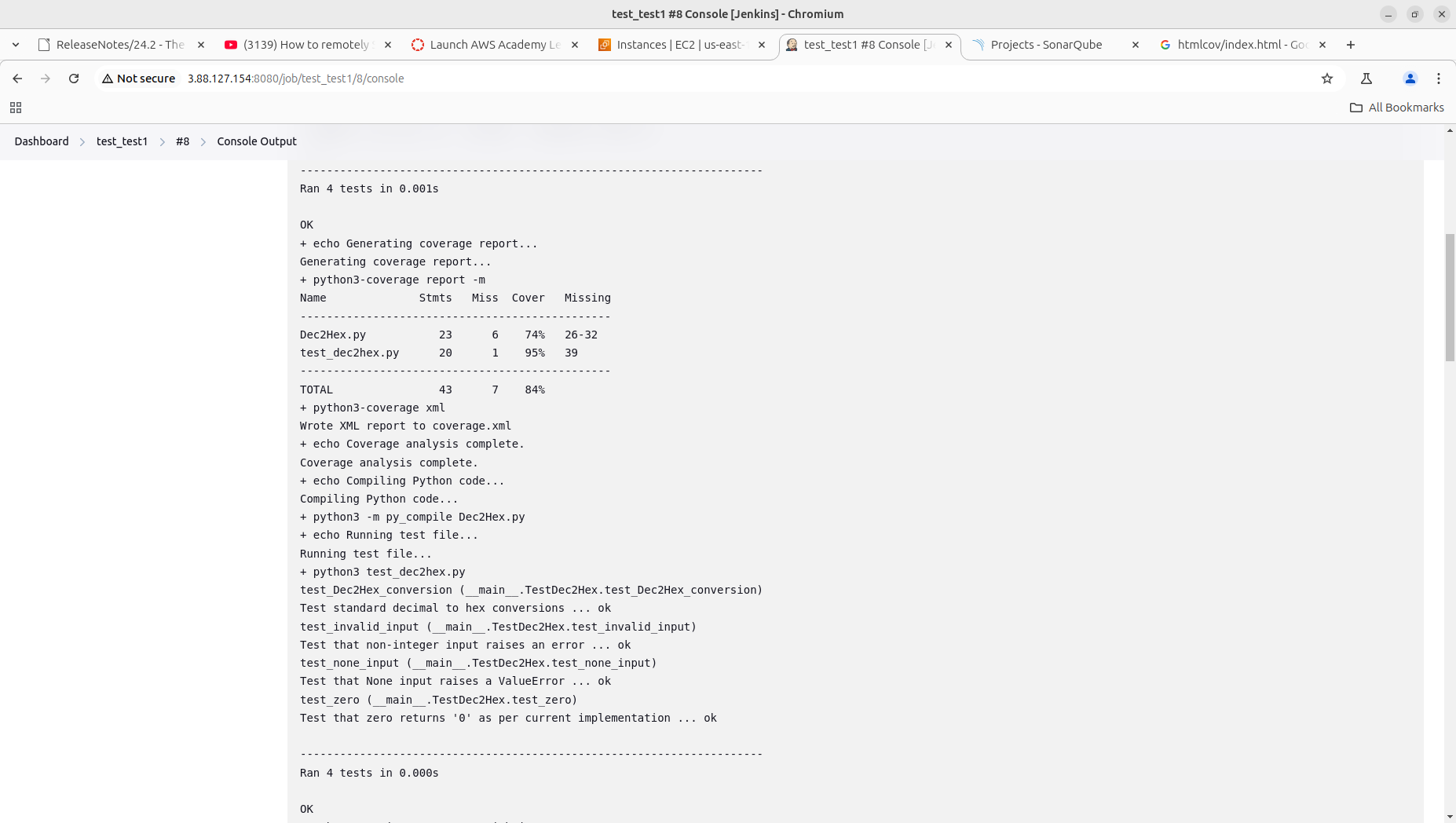
1. **Create a New Project**:
   * Navigate to **Projects** > **Create Project**.
   * Choose a unique project key and name for your Python project (e.g., python\_project).
2. **Generate an Authentication Token**:
   * Go to **My Account > Security** and generate a new authentication token to connect SonarQube with your Python project.

### ****Step 4: Integrate SonarQube with Jenkins CI/CD****

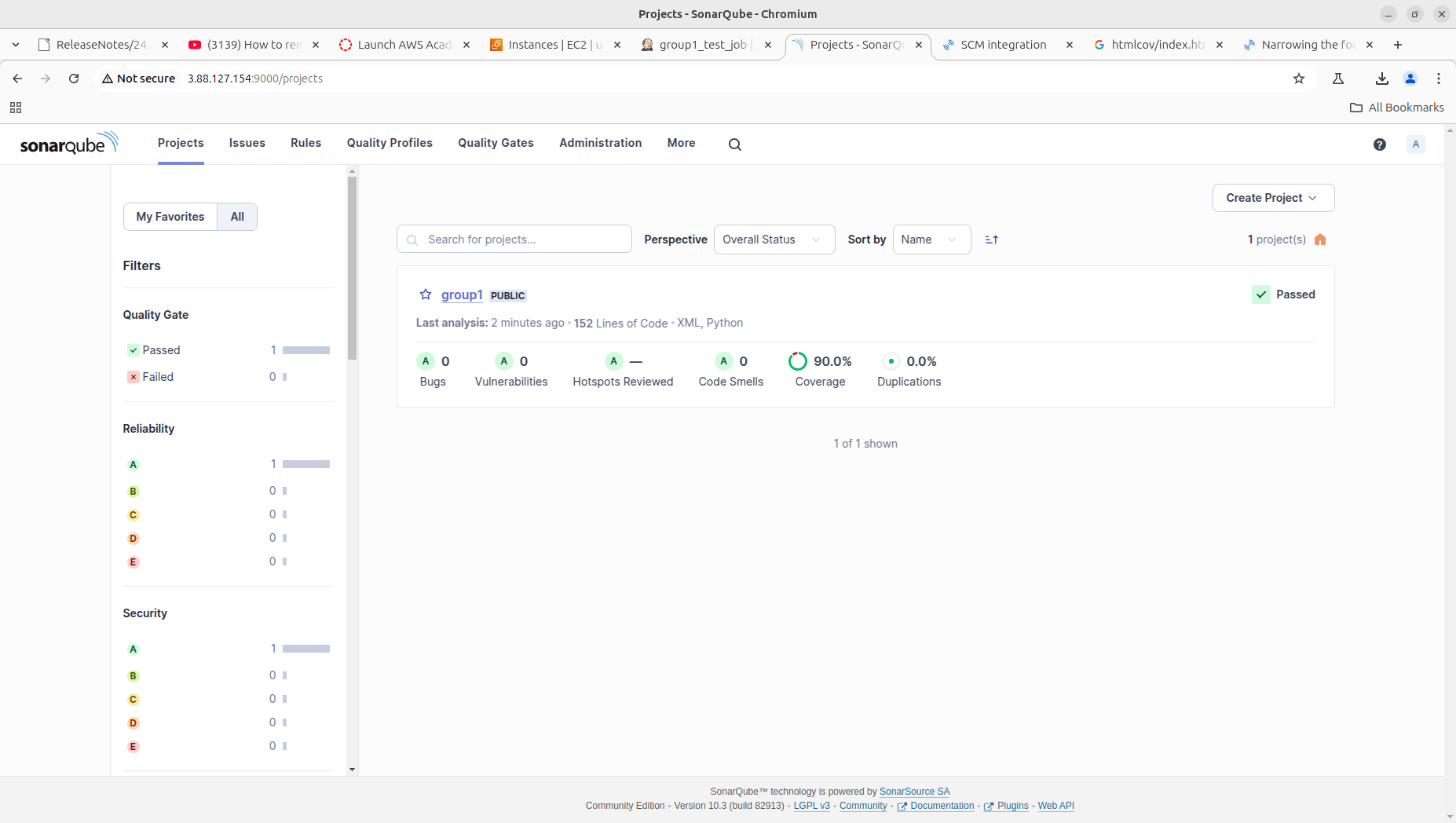
**To enhance code quality,** SonarQube **and** SonarScanner **were integrated into the Jenkins pipeline. This allowed the detection of potential bugs, security vulnerabilities, and code smells in the Python project. SonarQube provided actionable feedback on code improvements.**

To integrate **SonarQube** with **Jenkins** for static code analysis, first, install the **SonarQube Scanner** plugin in Jenkins. Then, configure **SonarQube Server** under **Manage Jenkins → Configure System**, providing the server URL and authentication token.

Next, In the **group1** job configuration, add a **build step** to execute sonar-scanner, ensuring that the **SonarQube authentication token** is set under **Build Environment** or as a Jenkins credential.



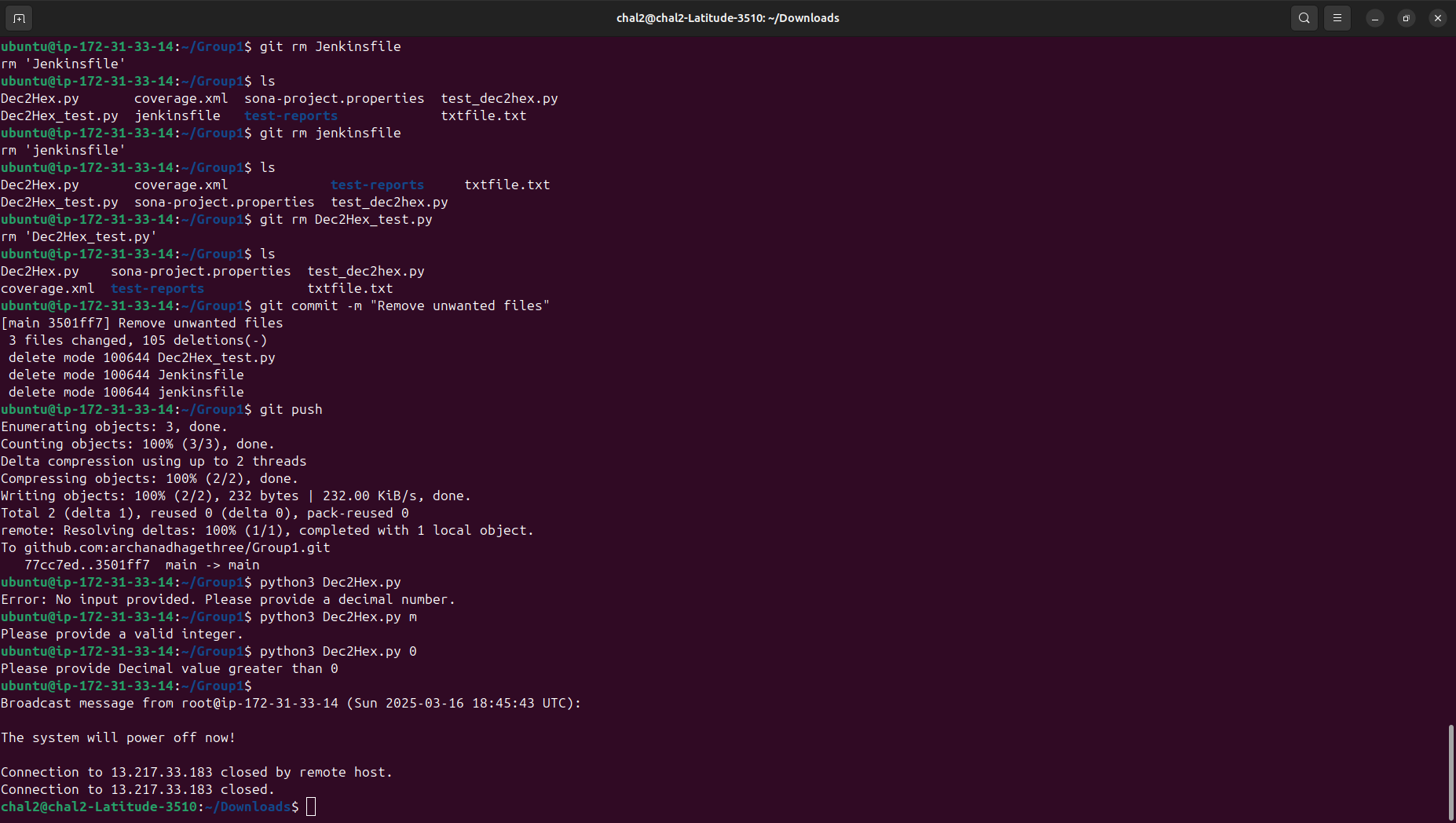
Finally, trigger a build to analyze code quality, and view results in **SonarQube Dashboard** for insights into issues, code smells, and vulnerabilities.

****

### **Task 5: Improving the Python Project**

The Dex2Hex.py script was enhanced to handle input validation errors:

* Error handling for missing arguments – The script now returns an appropriate error message if no input is provided.
* Validation of integer inputs – The script prevents the conversion of non-integer inputs, ensuring that it does not break execution while also handling invalid inputs gracefully.



### **Final code :**

1. import sys
2. def decimal\_to\_hex(decimal\_value):
3. if not isinstance(decimal\_value, int):
4. raise TypeError("Input must be an integer") # Check Input value only integer number.
5. if decimal\_value<1:
6. return print("Please provide Decimal value greater than 0") # Check Input value greater than 0
7. hex\_chars = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']
8. hexadecimal = ""
9. num = decimal\_value
10. print(f"Converting the Decimal Value {num} to Hex...")
11. while num != 0:
12. rem = num % 16
13. hexadecimal = hex\_chars[rem] + hexadecimal # Ensure concatenation works
14. num //= 16
15. print(f"Hexadecimal representation is: {hexadecimal}")
16. return hexadecimal
17. if \_\_name\_\_ == "\_\_main\_\_":
18. if len(sys.argv) > 1:
19. try:
20. decimal\_value = int(sys.argv[1])
21. decimal\_to\_hex(decimal\_value)
22. except ValueError:
23. print("Please provide a valid integer.")
24. else:
25. print("Error: No input provided. Please provide a decimal number.")
26. sys.exit(1)

**Unit Test File:**

To validate both the initial and modified versions of the **Dec2Hex.py** file, we implemented a **unit test file** designed to ensure the accuracy and robustness of the code. This test file comprises three key functions: **test\_invalid\_input**, **test\_input\_is\_not\_integer**, and **test\_decimal\_less\_than\_one**, which were essential in evaluating the modifications made in **Task 5**.

* **test\_invalid\_input** verifies that an input value is provided.
* **test\_input\_is\_not\_integer** ensures that the input is a valid integer.
* **test\_decimal\_less\_than\_one** checks whether the input is greater than one.

By integrating these unit tests, we reinforced the program’s reliability, ensuring it correctly handles different input scenarios while maintaining code integrity. This rigorous testing approach contributed to the overall quality assurance and stability of the **Dec2Hex.py** project.

Below is the unit test file code

1. import unittest
2. from io import StringIO
3. import sys
4. from Dec2Hex import decimal\_to\_hex
5. # Assuming the decimal\_to\_hex function is in the same file, if not, import it like this:
6. # from your\_module import decimal\_to\_hex
7. class TestDecimalToHex(unittest.TestCase):
8. def test\_valid\_decimal(self):
9. # Test with a valid decimal value
10. decimal\_value = 255
11. expected\_output = "FF"
12. # Capture the print output
13. captured\_output = StringIO()
14. sys.stdout = captured\_output
15. decimal\_to\_hex(decimal\_value)
16. sys.stdout = sys.\_\_stdout\_\_ # Reset redirect.
17. # Check if the output matches
18. self.assertIn(f"Hexadecimal representation is: {expected\_output}", captured\_output.getvalue())
19. def test\_input\_is\_not\_integer(self):
20. # Test if the function raises TypeError for non-integer input
21. with self.assertRaises(TypeError):
22. decimal\_to\_hex("string")
23. def test\_decimal\_less\_than\_one(self):
24. # Test if the function handles a decimal value less than 1
25. captured\_output = StringIO()
26. sys.stdout = captured\_output
27. decimal\_to\_hex(0)
28. sys.stdout = sys.\_\_stdout\_\_ # Reset redirect.
29. self.assertIn("Please provide Decimal value greater than 0", captured\_output.getvalue())
30. def test\_valid\_edge\_case(self):
31. # Test with another valid decimal value (e.g., 16)
32. decimal\_value = 16
33. expected\_output = "10"
34. captured\_output = StringIO()
35. sys.stdout = captured\_output
36. decimal\_to\_hex(decimal\_value)
37. sys.stdout = sys.\_\_stdout\_\_ # Reset redirect.
38. self.assertIn(f"Hexadecimal representation is: {expected\_output}", captured\_output.getvalue())
39. def test\_invalid\_input(self):
40. """Test when invalid input is provided (e.g., None or a string)."""
41. with self.assertRaises(TypeError):
42. decimal\_to\_hex(None)
43. with self.assertRaises(TypeError):
44. decimal\_to\_hex("string")
45. if \_\_name\_\_ == "\_\_main\_\_":
46. unittest.main()

### **Task 6: Version Control and CI Pipeline Execution**

The improved project was tested using the CI pipeline set up in Jenkins. The pipeline was updated to ensure that all test conditions were passed, and the build resulted in a successful execution.

Additionally, SonarQube feedback was addressed to improve the overall code quality. Throughout the process, Git version control was used to track changes, allowing a structured development workflow with clear commit history.

## 

## **3. Conclusion**

By leveraging DevOps practices, we effectively enhanced communication, collaboration, and automation in the **Dec2Hex.py** practical project. Utilizing **Git** for version control, **Jenkins** for CI/CD automation, and **SonarQube** for static code analysis, we streamlined the development workflow and ensured code quality.

To achieve comprehensive code coverage and quality assessment, external software tools were installed on the **Group1 EC2 instance**, as SonarQube primarily focuses on detecting bugs and maintaining code integrity. The successful deployment of the EC2 instance, along with the seamless integration of Jenkins, significantly improved the project's reliability and maintainability.

Moreover, enhancements to **input validation and error handling** strengthened the robustness of the Python script, while proper version control provided a structured development history. This setup establishes a **scalable foundation** for continuous integration and software quality assurance, ensuring long-term efficiency and maintainability of the project.