Advance Devops Case Study (Topic 9)

Problem Statement: Static Analysis Integration with Terraform

- Concepts Used: Jenkins, SonarQube, Terraform.
- Problem Statement: "Use Terraform to set up the infrastructure for a Jenkins server and a SonarQube instance. Then, configure a Jenkins pipeline to perform a static analysis of a Python application using SonarQube."
- Tasks:
  - Write a Terraform script to deploy Jenkins and SonarQube on AWS.
  - Set up a Jenkins pipeline to analyze a Python codebase using SonarQube.
  - Trigger the pipeline and review the analysis results in SonarQube.

1. Introduction

Case Study Overview:

This case study focuses on integrating static analysis tools into a CI/CD pipeline by leveraging Terraform to provision a Jenkins server and a SonarQube instance on AWS. The goal is to automate the deployment process and enable continuous integration of code analysis for a Python application.

Key Feature and Application:

The unique feature of this integration is the automated static analysis of Python code using SonarQube within a Jenkins pipeline. This allows developers to receive immediate feedback on code quality, identifying potential issues such as bugs, code smells, and security vulnerabilities before deployment. This proactive approach enhances code quality and reduces technical debt in software development.

Third-Year Project Integration:

This case study directly integrates a semester project developed in Python, enhancing the practical application of the concepts learned throughout the course. By incorporating static analysis into the CI/CD pipeline, students can demonstrate their understanding of DevOps principles while applying them to their Python application. The integration allows for immediate feedback on code quality through tools like SonarQube within a Jenkins pipeline, identifying potential issues such as bugs and vulnerabilities. Additionally, using Terraform to automate the deployment of the necessary infrastructure provides valuable experience in infrastructure as code (IaC), preparing students for real-world scenarios in software development and deployment practices.

2. Step by Step Explanation

Prerequisites:

1. Amazon Web Service account.
2. IAM user created in the aws account
3. Download terraform
4. Perform the following steps:
    - Open command prompt in your pc
    - Run the command aws configure
    - Fill in the details of the IAM user and other options asked.

This process configures the AWS CLI to connect to your AWS account using the provided IAM user credentials. Once configured, you will be able to create and manage AWS resources directly from the terminal.

Steps to perform the case study:

Step 1: Setup terraform infrastructure

1. Open cmd or powershell in your device. Make a new directory with an appropriate name. Cd into the directory and make a file named “main.tf”.

   mkdir <file_name>

   cd <file_name>

   code main.tf

```
PS C:\Users\raksh\Desktop> mkdir caseStudy-advDevops


    Directory: C:\Users\raksh\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        20-10-2024     22:00                caseStudy-advDevops


PS C:\Users\raksh\Desktop> cd .\caseStudy-advDevops\
PS C:\Users\raksh\Desktop\caseStudy-advDevops> code main.tf
```

2. Here we will write a terraform script. This script should create an appropriate security group with inbound rules set for SSH on port 22, Custom tcp for both port 8080 and 9000. Setting up security group ensures that we are able to ssh into the instance, and open jenkins and sonarqube on respective ports once they are installed.

Script:

```
provider "aws" {
  region = "us-east-1" # Change to your preferred region
}

resource "aws_security_group" "jenkins_sonarqube_sg" {
  name        = "jenkins_sonarqube_sg_new"
  description = "Allow HTTP, HTTPS, and SSH traffic"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Change to your IP for security
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Change to your IP for security
  }

  ingress {
    from_port   = 9000
    to_port     = 9000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Change to your IP for security
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1" # Allow all outbound traffic
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "jenkins" {
   ami             = "ami-06b21ccaeff8cd686" # Update with the latest Amazon Linux 2023
AMI ID
  instance_type = "t3.medium"
  key_name      = "sharma" # Change to your key pair name
  security_groups = [aws_security_group.jenkins_sonarqube_sg.name]
```

```
  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo amazon-linux-extras install java-openjdk11 -y
              wget -O jenkins.war http://mirrors.jenkins.io/war-stable/latest/jenkins.war
              nohup java -jar jenkins.war --httpPort=8080 > jenkins.log 2>&1 &
              EOF

  tags = {
    Name = "Jenkins_server"
  }
}

resource "aws_instance" "sonarqube" {
  ami             = "ami-06b21ccaeff8cd686" # Update with the latest Amazon Linux 2023 AMI ID
  instance_type = "t3.medium"
  key_name      = "sharma" # Change to your key pair name
  security_groups = [aws_security_group.jenkins_sonarqube_sg.name]

  user_data = <<-EOF
              #!/bin/bash
              sudo yum update -y
              sudo amazon-linux-extras install java-openjdk11 -y
                                                                                wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.5.zip
              sudo yum install -y unzip
              unzip sonarqube-9.9.5.zip
              cd sonarqube-9.9.5/bin/linux-x86-64
              ./sonar.sh start
              EOF

  tags = {
    Name = "SonarQube_server"
  }
}
```

We have set the instance type as “t3.medium”. This is because both jenkins and sonrqube have some minimum cpu and ram requirements. These requirements are met when we create instances of type t3.medium or higher.

main.tf

```
provider "aws" {
  region = "us-east-1" # Change to your preferred region
}

resource "aws_security_group" "jenkins_sonarqube_sg" {
  name        = "jenkins_sonarqube_sg"
  description = "Allow HTTP, HTTPS, and SSH traffic"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Change to your IP for security
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Change to your IP for security
  }
```

3. Open the newly created directory(with main.tf) in command prompt. Run the commands:

   terraform init
   terraform plan
   terraform apply

   These commands will firstly initialize the terraform repository in the folder. Terraform plan will show us the steps and the execution plan which will be carried out once we apply. Terraform apply will execute the main.tf file and create two instances.

```
PS C:\Users\raksh\Desktop\caseStudy-advDevops> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
PS C:\Users\raksh\Desktop\caseStudy-advDevops> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.jenkins will be created
  + resource "aws_instance" "jenkins" {
      + ami                                  = "ami-06b21ccaeff8cd686"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
```

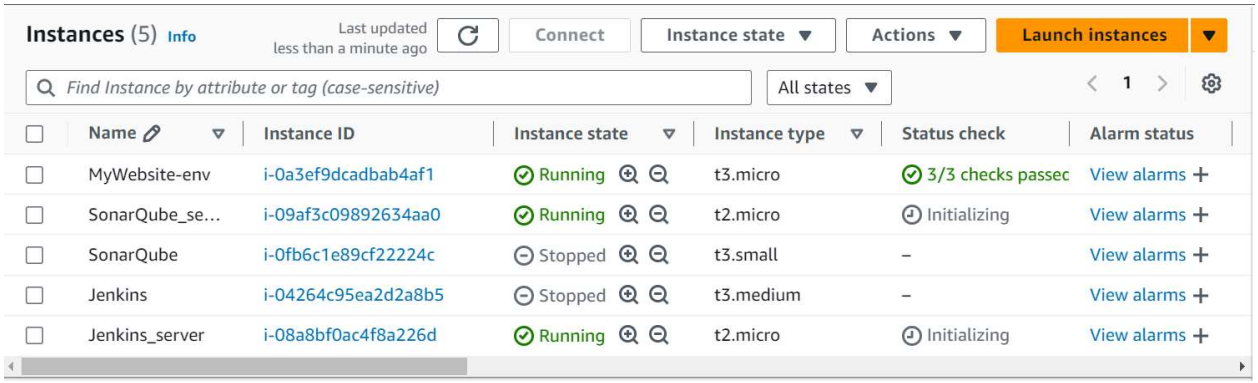
```
PS C:\Users\raksh\Desktop\caseStudy-advDevops> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.jenkins will be created
  + resource "aws_instance" "jenkins" {
      + ami                                  = "ami-06b21ccaeff8cd686"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
```

4. Login into your aws account. Go to the EC2 dashboard and open instances. You can see the instances (Jenkins_server and SonarQube_server) created and running.



5. SSH into both the instances:
   Click on instance id > Connect > SSH > copy the command given in example and paste it in command prompt.
   Make sure to run the command in the directory where you have your key.pem file, this key.pem file we mentioned in our terraform script.

Jenkins:

```
PS C:\Users\raksh\Downloads> ssh -i "sharma.pem" ec2-user@ec2-52-91-238-114.compute-1.amazonaws.com
The authenticity of host 'ec2-52-91-238-114.compute-1.amazonaws.com (52.91.238.114)' can't be established.
ED25519 key fingerprint is SHA256:1BAo3jeB+1OoLDoukqUjsUBrdRn+8NKr5A6iqsq9RAA.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-91-238-114.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-33-60 ~]$
```

SonarQube

```
PS C:\Users\raksh\Downloads> ssh -i "sharma.pem" ec2-user@ec2-54-152-168-139.compute-1.amazonaws.com
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-35-144 ~]$
```

Step 2: Install Jenkins and other softwares in Jenkins instance

1. Run the following commands in the jenkins instance to install java 17 and jenkins on the instance.

   sudo yum update -y
   sudo yum install java-17-amazon-corretto -y
   sudo wget -O /etc/yum.repos.d/jenkins.repohttps://pkg.jenkins.io/redhat-stable/jenkins.repo
   sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
   sudo yum install jenkins -y --nogpgcheck

```
Installing:
 jenkins                  noarch                  2.462.3-1.1                  jenkins                   89 M

Transaction Summary
================================================================================================================================
Install  1 Package

Total download size: 89 M
Installed size: 89 M
Downloading Packages:
jenkins-2.462.3-1.1.noarch.rpm                                                  12 MB/s |  89 MB     00:07
--------------------------------------------------------------------------------------------------------------------------------
Total                                                                           12 MB/s |  89 MB     00:07
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                                        1/1
  Running scriptlet: jenkins-2.462.3-1.1.noarch                                                             1/1
  Installing       : jenkins-2.462.3-1.1.noarch                                                             1/1
  Running scriptlet: jenkins-2.462.3-1.1.noarch                                                             1/1
  Verifying        : jenkins-2.462.3-1.1.noarch                                                             1/1

Installed:
  jenkins-2.462.3-1.1.noarch

Complete!
```

After successful installation you will see the above screen. To verify installation you can run the “java -version” and “jenkins -version” command. If the installation is successful, it will return the versions of both software installed. Java 17 is used because previous version of Java is not compatible with jenkins.

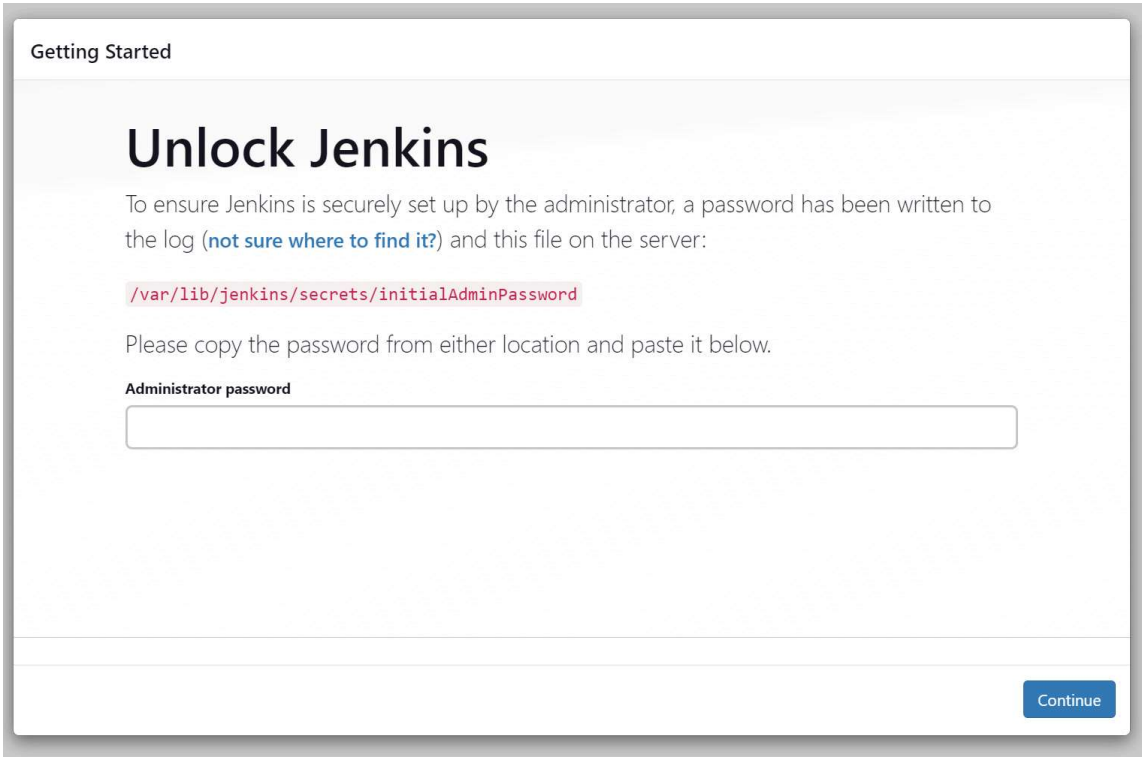
2. Start the jenkins server

   sudo systemctl start jenkins
   sudo systemctl enable jenkins
   sudo systemctl status jenkins

```
[ec2-user@ip-172-31-33-60 ~]$ sudo systemctl start jenkins
[ec2-user@ip-172-31-33-60 ~]$ sudo systemctl enable jenkins
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /usr/lib/systemd/system/jenkins.service.
[ec2-user@ip-172-31-33-60 ~]$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: disabled)
     Active: active (running) since Sun 2024-10-20 16:51:40 UTC; 18s ago
   Main PID: 25535 (java)
      Tasks: 51 (limit: 4566)
     Memory: 582.1M
        CPU: 21.305s
     CGroup: /system.slice/jenkins.service
             └─25535 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenki>

Oct 20 16:51:35 ip-172-31-33-60.ec2.internal jenkins[25535]: 85b58b3be8bc4ac18945d281995cf5da
Oct 20 16:51:35 ip-172-31-33-60.ec2.internal jenkins[25535]: This may also be found at: /var/lib/jenkins/secrets/initia>
Oct 20 16:51:35 ip-172-31-33-60.ec2.internal jenkins[25535]: *************************************************************>
Oct 20 16:51:35 ip-172-31-33-60.ec2.internal jenkins[25535]: *************************************************************>
Oct 20 16:51:35 ip-172-31-33-60.ec2.internal jenkins[25535]: *************************************************************>
Oct 20 16:51:40 ip-172-31-33-60.ec2.internal jenkins[25535]: 2024-10-20 16:51:40.132+0000 [id=32]        INFO        je>
Oct 20 16:51:40 ip-172-31-33-60.ec2.internal jenkins[25535]: 2024-10-20 16:51:40.197+0000 [id=24]        INFO        hu>
Oct 20 16:51:40 ip-172-31-33-60.ec2.internal systemd[1]: Started jenkins.service - Jenkins Continuous Integration Serve>
Oct 20 16:51:40 ip-172-31-33-60.ec2.internal jenkins[25535]: 2024-10-20 16:51:40.283+0000 [id=49]        INFO        h.>
Oct 20 16:51:40 ip-172-31-33-60.ec2.internal jenkins[25535]: 2024-10-20 16:51:40.284+0000 [id=49]        INFO        hu>
lines 1-20/20 (END)
```

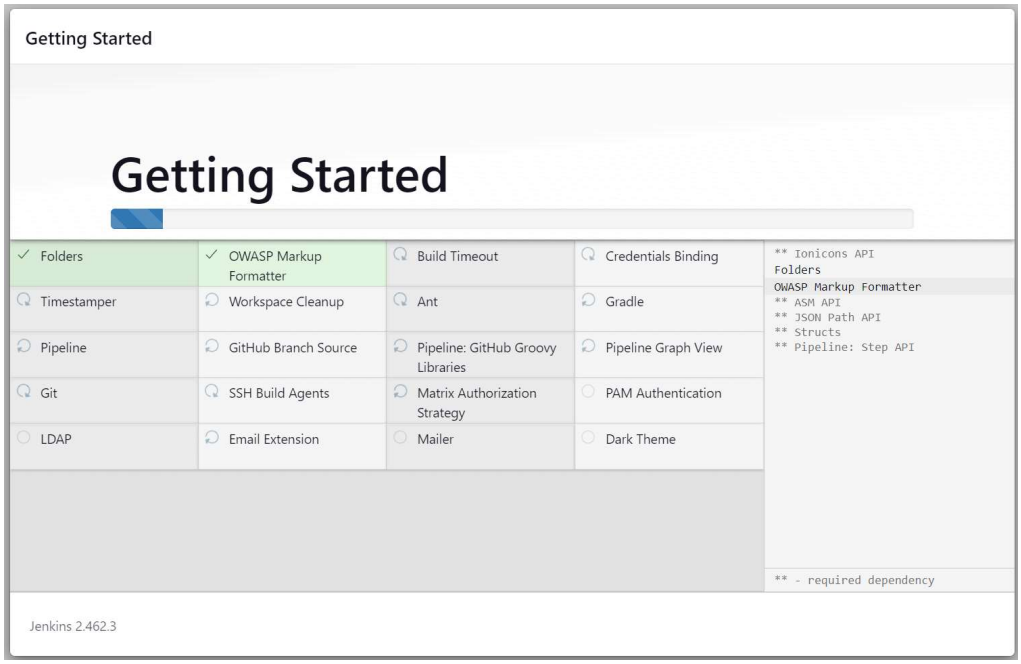
3. As we can see, status is showing active. Now to test it, go to the following url: http://<public_ip_address_of_instance>:8080

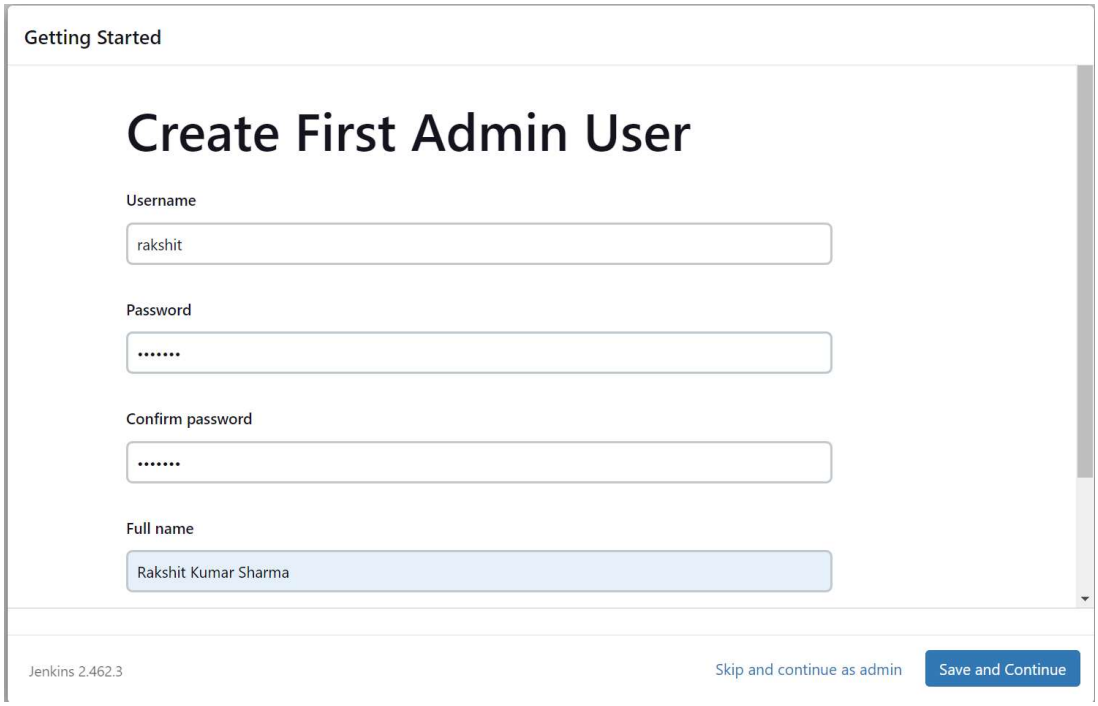


Copy the path given and run the following command in terminal of jenkins instance:

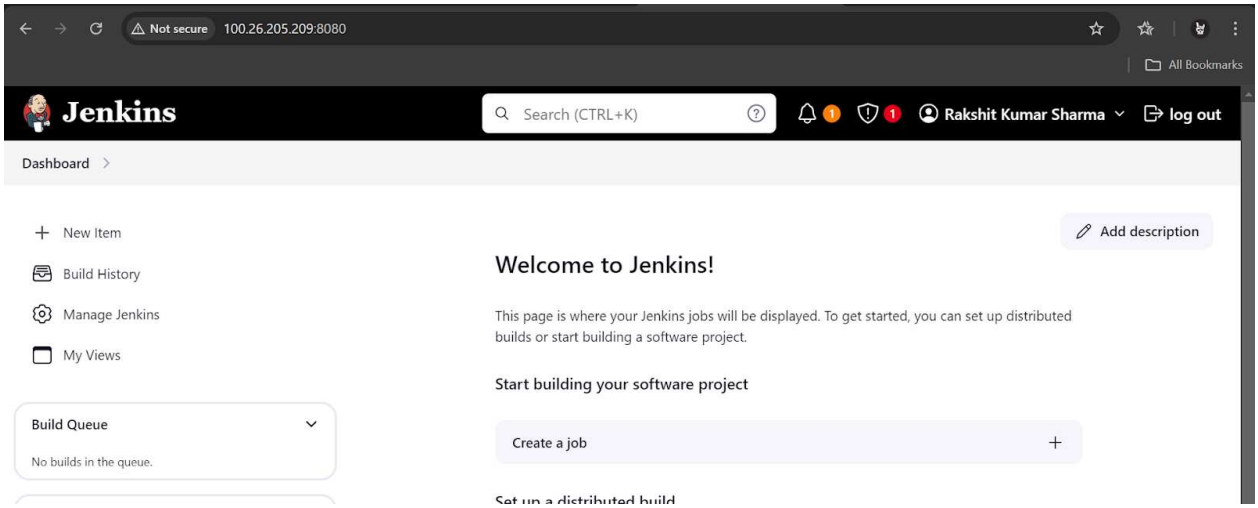
sudo cat <file_path>

It will return a alpha numeric value, copy it and paste it in the password.





Complete the steps to create jenkins account and login to your account.



We have successfully installed Jenkins and it is running on port 8080.

4. Now we need to install some other software that will be required in the experiment.
5. First we will install Git. Git is needed as we will be using github to access our project.

   sudo yum install git

```
[ec2-user@ip-172-31-33-60 ~]$ sudo yum install git
Last metadata expiration check: 0:03:14 ago on Sun Oct 20 16:49:39 2024.
Dependencies resolved.
================================================================================
 Package                 Architecture        Version                    Repository                Size
================================================================================
Installing:
 git                     x86_64              2.40.1-1.amzn2023.0.3      amazonlinux               54 k
Installing dependencies:
 git-core                x86_64              2.40.1-1.amzn2023.0.3      amazonlinux              4.3 M
 git-core-doc            noarch              2.40.1-1.amzn2023.0.3      amazonlinux              2.6 M
 perl-Error              noarch              1:0.17029-5.amzn2023.0.2   amazonlinux               41 k
 perl-File-Find          noarch              1.37-477.amzn2023.0.6      amazonlinux               26 k
 perl-Git                noarch              2.40.1-1.amzn2023.0.3      amazonlinux               42 k
 perl-TermReadKey        x86_64              2.38-9.amzn2023.0.2        amazonlinux               36 k
 perl-lib                x86_64              0.65-477.amzn2023.0.6      amazonlinux               15 k

Transaction Summary
================================================================================
Install  8 Packages

Total download size: 7.1 M
Installed size: 34 M
Is this ok [y/N]: y
Downloading Packages:
(1/8): git-2.40.1-1.amzn2023.0.3.x86_64.rpm                 1.0 MB/s |  54 kB     00:00
```

6. Next we need to install Sonar Scanner.
   First we will download a sonar scanner package in our device

Download Linux x64 version as our instance is Amazon Linux.

Next we will transfer this zip file to our instance:
scp -i <path_to_key.pem"> <path_to_downloaded_zip_file>
ec2-user@<public_ip_address_of_instance>:/home/ec2-user/

```
PS C:\Users\raksh> scp -i "C:\Users\raksh\Downloads\sharma.pem" "C:\Users\raksh\Downloads\sonar-scanner-cli-6.2.1.4610-linux-x64.zip" ec2-user@100.26.205.209:/home/ec2-user/
The authenticity of host '100.26.205.209 (100.26.205.209)' can't be established.
ED25519 key fingerprint is SHA256:1BAo3jeB+1OoLDoukqUjsUBrdRn+8NKr5A6iqsq9RAA.
This host key is known by the following other names/addresses:
    C:\Users\raksh/.ssh/known_hosts:107: ec2-52-91-238-114.compute-1.amazonaws.com
    C:\Users\raksh/.ssh/known_hosts:111: ec2-100-26-205-209.compute-1.amazonaws.com
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Warning: Permanently added '100.26.205.209' (ED25519) to the list of known hosts.
sonar-scanner-cli-6.2.1.4610-linux-x64.zip                                    100%   55MB   3.6MB/s   00:15
```

Run this command in a different terminal.

7. Verify the transfer with ls commands in jenkins terminal
   Unzip the file
   unzip <file_name>

```
[ec2-user@ip-172-31-33-60 ~]$ ls
sonar-scanner-cli-6.2.1.4610-linux-x64.zip
[ec2-user@ip-172-31-33-60 ~]$ unzip sonar-scanner-cli-6.2.1.4610-linux-x64.zip
Archive:  sonar-scanner-cli-6.2.1.4610-linux-x64.zip
   creating: sonar-scanner-6.2.1.4610-linux-x64/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/security/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/security/policy/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/security/policy/unlimited/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/security/policy/limited/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/sdp/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/management/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/lib/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/lib/server/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/lib/security/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/lib/jfr/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/jdk.incubator.vector/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/jdk.charsets/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.scripting/
```

8. Run the following commands to configure sonar-scanner:

   sudo mv sonar-scanner-6.2.1.4610-linux-x64 /opt/sonar-scanner
   echo 'export PATH=$PATH:/opt/sonar-scanner/bin' >> ~/.bashrc
   source ~/.bashrc
   sonar-scanner -v

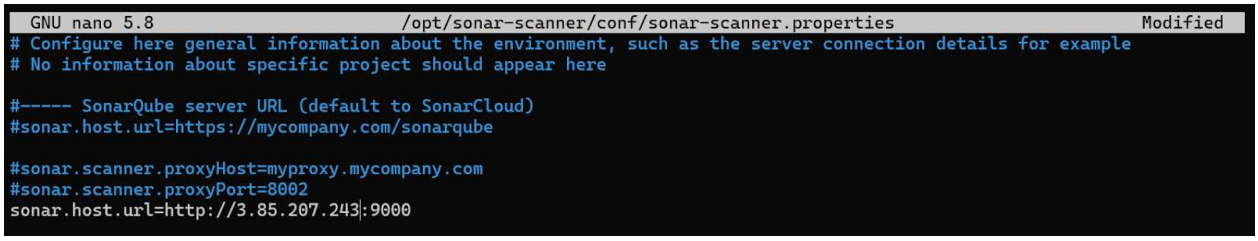
```
[ec2-user@ip-172-31-33-60 ~]$ sonar-scanner -v
16:57:42.634 INFO  Scanner configuration file: /opt/sonar-scanner/conf/sonar-scanner.properties
16:57:42.638 INFO  Project root configuration file: NONE
16:57:42.657 INFO  SonarScanner CLI 6.2.1.4610
16:57:42.659 INFO  Java 17.0.12 Eclipse Adoptium (64-bit)
16:57:42.659 INFO  Linux 6.1.112-122.189.amzn2023.x86_64 amd64
```

This verifies installation

9. Open the sonar-scanner.properties file in nano editor and add the following line

   nano
   /opt/sonar-scanner/sonar-scanner-6.2.1.4610-linux-x64/conf/sonar-scanner.properties

add: sonar.host.url=http://<your-sonarqube-ip>:9000



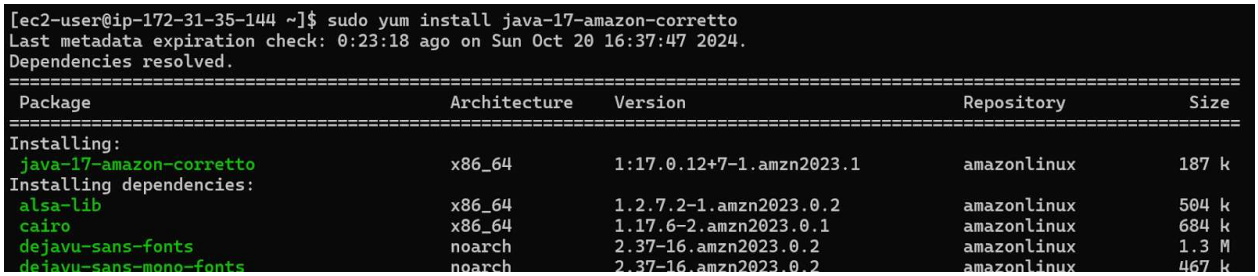
Ctrl O >Enter > Ctrl X to save changes.

All installations for the Jenkins instance are now complete. Proceed with installation of sonarqube on respective instance.

Step 3: Install SonarQube on the Sonar instance

1. Here also install the java 17 version.

   sudo yum install java-17-amazon-corretto



2. Download SonarQube 9.9.2 LTS and place it in the /opt directory. You can use the wget command to download the package:

   sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.2.77730.zip -P /opt

3. Run the following commands to create a new user. Unzip the zip file and rename the extracted directory.

   sudo useradd sonar
   sudo apt install unzip -y
   sudo unzip /opt/sonarqube-9.9.2.77730.zip -d /opt
   sudo mv /opt/sonarqube-9.9.2.77730 /opt/sonar
   sudo chown -R sonar:sonar /opt/sonar

```
[ec2-user@ip-172-31-35-144 ~]$ sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.2.77730.z
ip -P /opt
--2024-10-20 17:01:47--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.2.77730.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 99.84.191.87, 99.84.191.75, 99.84.191.23, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|99.84.191.87|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 294180180 (281M) [binary/octet-stream]
Saving to: '/opt/sonarqube-9.9.2.77730.zip'

sonarqube-9.9.2.77730.zip     100%[==============================================>] 280.55M  85.3MB/s    in 3.3s

2024-10-20 17:01:51 (85.2 MB/s) - '/opt/sonarqube-9.9.2.77730.zip' saved [294180180/294180180]
```

```
[ec2-user@ip-172-31-35-144 ~]$ sudo useradd sonar
[ec2-user@ip-172-31-35-144 ~]$ sudo unzip /opt/sonarqube-9.9.2.77730.zip -d /opt
sudo mv /opt/sonarqube-9.9.2.77730 /opt/sonar
Archive:  /opt/sonarqube-9.9.2.77730.zip
   creating: /opt/sonarqube-9.9.2.77730/
  inflating: /opt/sonarqube-9.9.2.77730/dependency-license.json
  inflating: /opt/sonarqube-9.9.2.77730/COPYING
   creating: /opt/sonarqube-9.9.2.77730/bin/
   creating: /opt/sonarqube-9.9.2.77730/bin/windows-x86-64/
  inflating: /opt/sonarqube-9.9.2.77730/bin/windows-x86-64/SonarService.bat
   creating: /opt/sonarqube-9.9.2.77730/bin/windows-x86-64/lib/
  inflating: /opt/sonarqube-9.9.2.77730/bin/windows-x86-64/lib/SonarServiceWrapper.exe
  inflating: /opt/sonarqube-9.9.2.77730/bin/windows-x86-64/lib/find_java.bat
   creating: /opt/sonarqube-9.9.2.77730/bin/winsw-license/
```

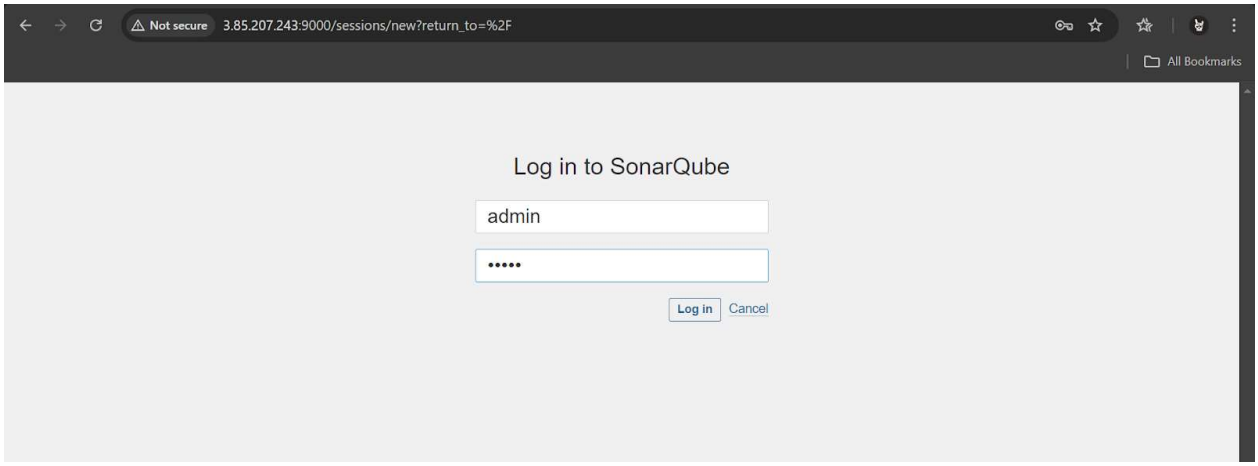
4. SonarQube is now installed successfully. We will start the sonarqube server with the following commands:

   sudo su sonar
   /op/sonar/bin/linux-x86-64/sonar.sh start
   /op/sonar/bin/linux-x86-64/sonar.sh status

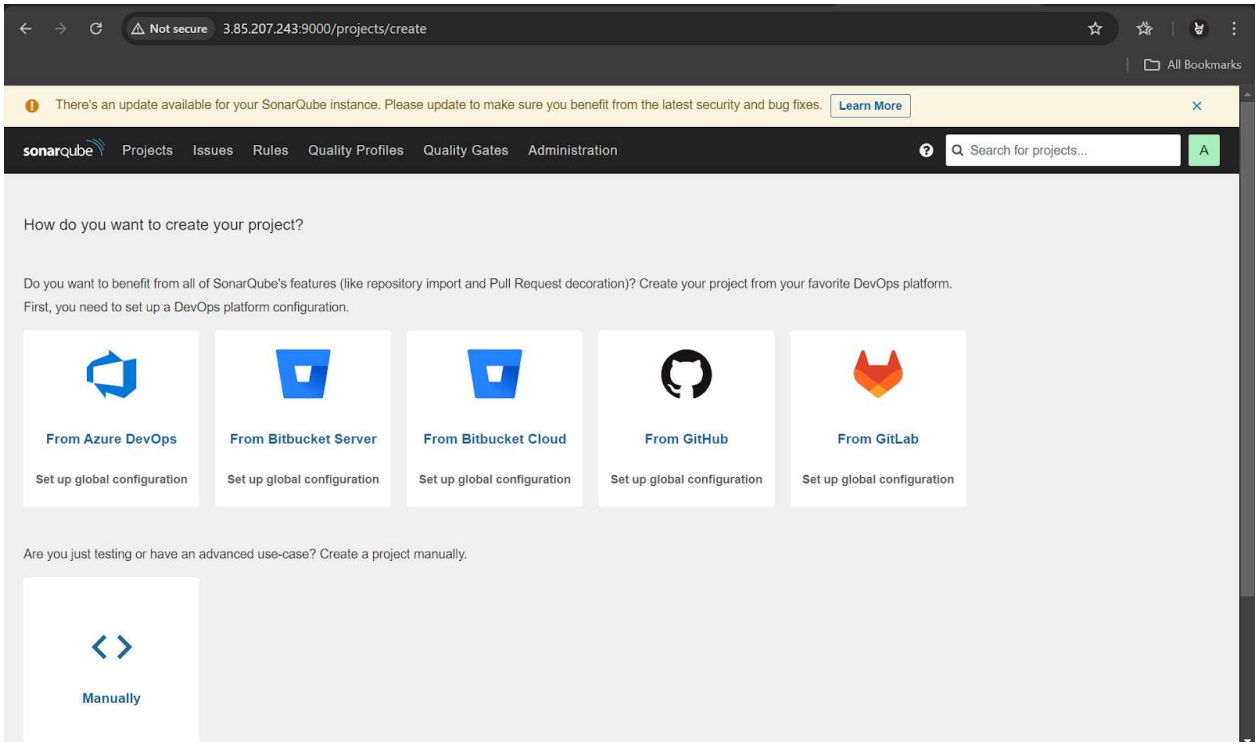
```
[ec2-user@ip-172-31-35-144 ~]$ sudo su sonar
[sonar@ip-172-31-35-144 ec2-user]$ /opt/sonar/bin/linux-x86-64/sonar.sh start
/usr/bin/java
Starting SonarQube...
Started SonarQube.
[sonar@ip-172-31-35-144 ec2-user]$ /opt/sonar/bin/linux-x86-64/sonar.sh status
/usr/bin/java
SonarQube is running (26089).
[sonar@ip-172-31-35-144 ec2-user]$
```

   This shows sonar server is setup and running successfully

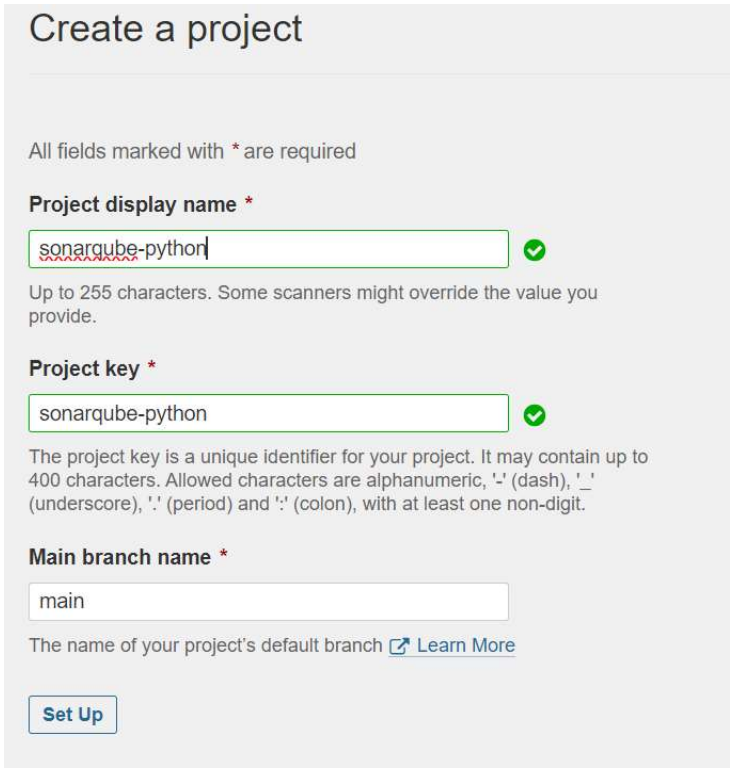
5. Go to the url http://<public_ip_address_of_instance>:9000



Enter username and password as admin. It will next ask you to change the password. After completion of steps, you will be redirected to the sonarqube dashboard.

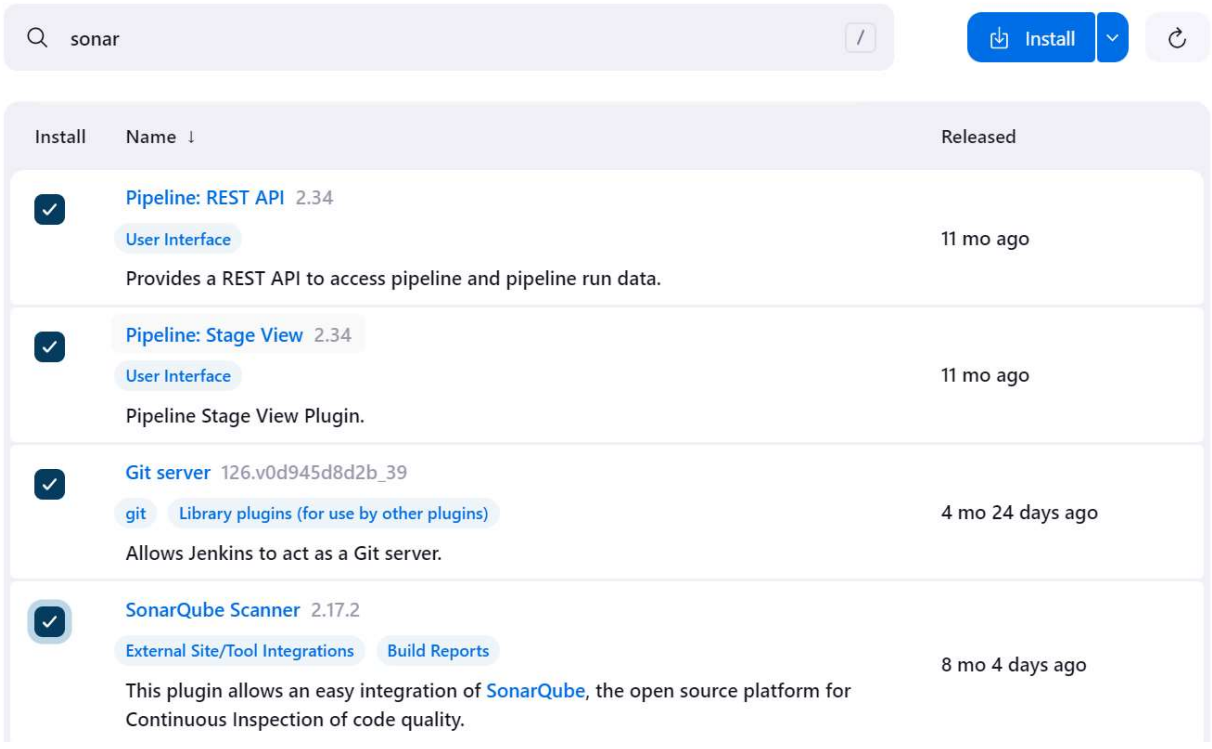


6. Create a manual project and give it an appropriate name.

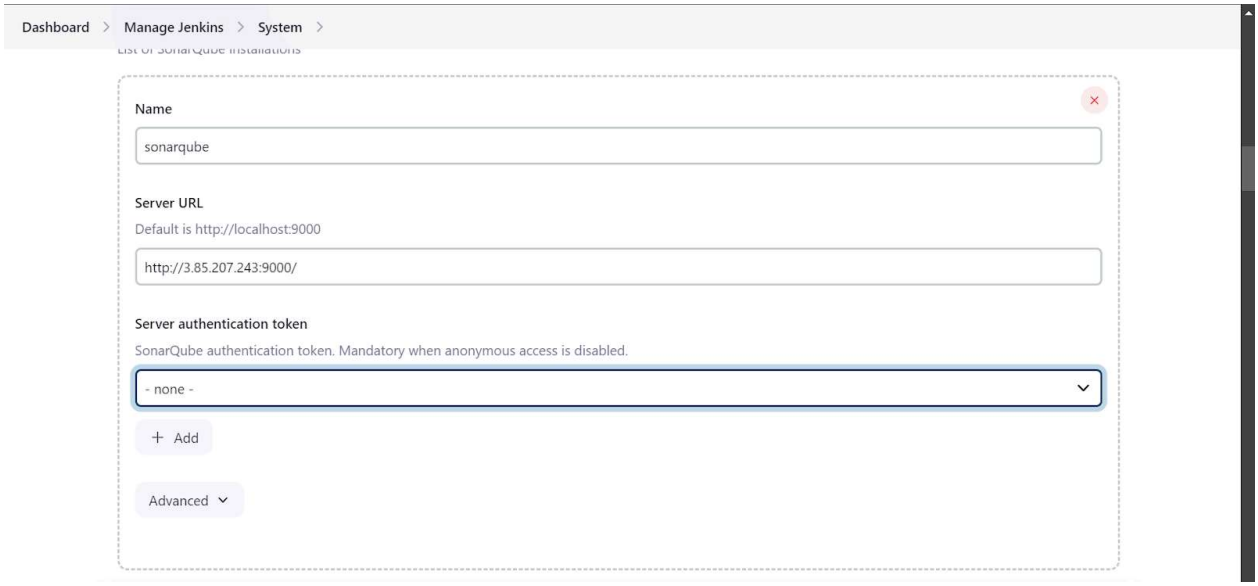


Step 4: Configure Jenkins

1. Go to Manage Jenkins > Plugins, and install the following plugins



2. Go to Manage Jenkins > System.
   First add a sonarqube server



Give the name and url as http://<public_ip_address_of_instance>:9000

Also add a github server since our source code will be taken from github.

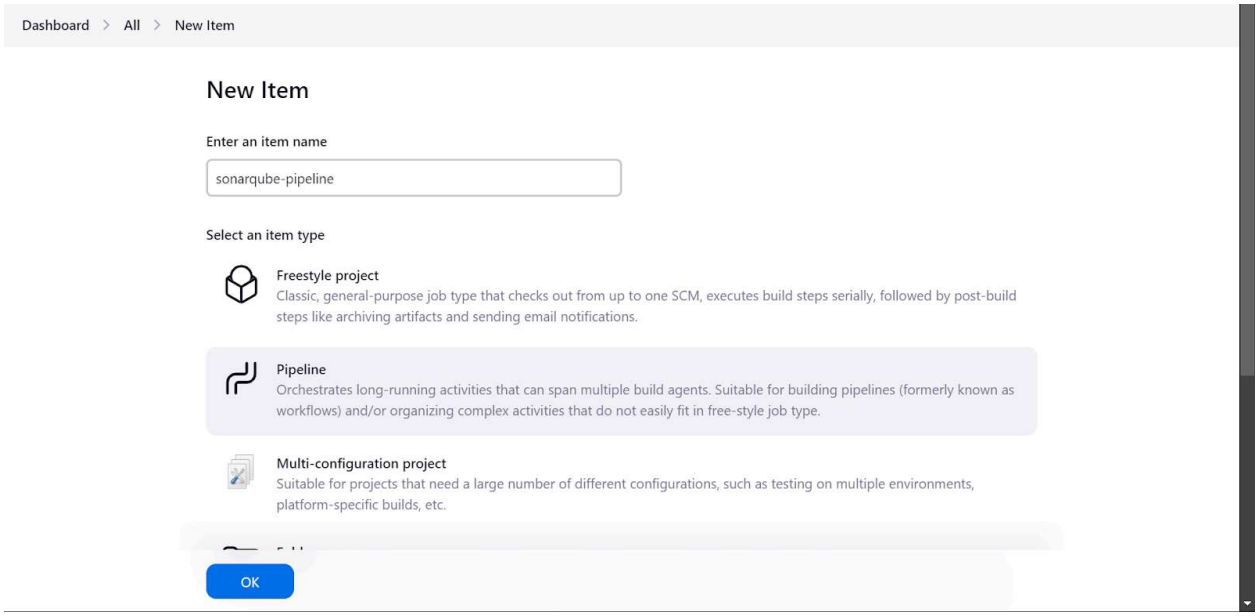
3. Now go to Manage Jenkins > Tools
   Give details of git installation

Path to Git executable can be found by running the command “which git” in the jenkins terminal.

Also add a sonar scanner.

Step 5: Create a jenkins pipeline

1. Click on New item in left panel of jenkins dashboard.



Give it an appropriate name and select type as “pipeline”.

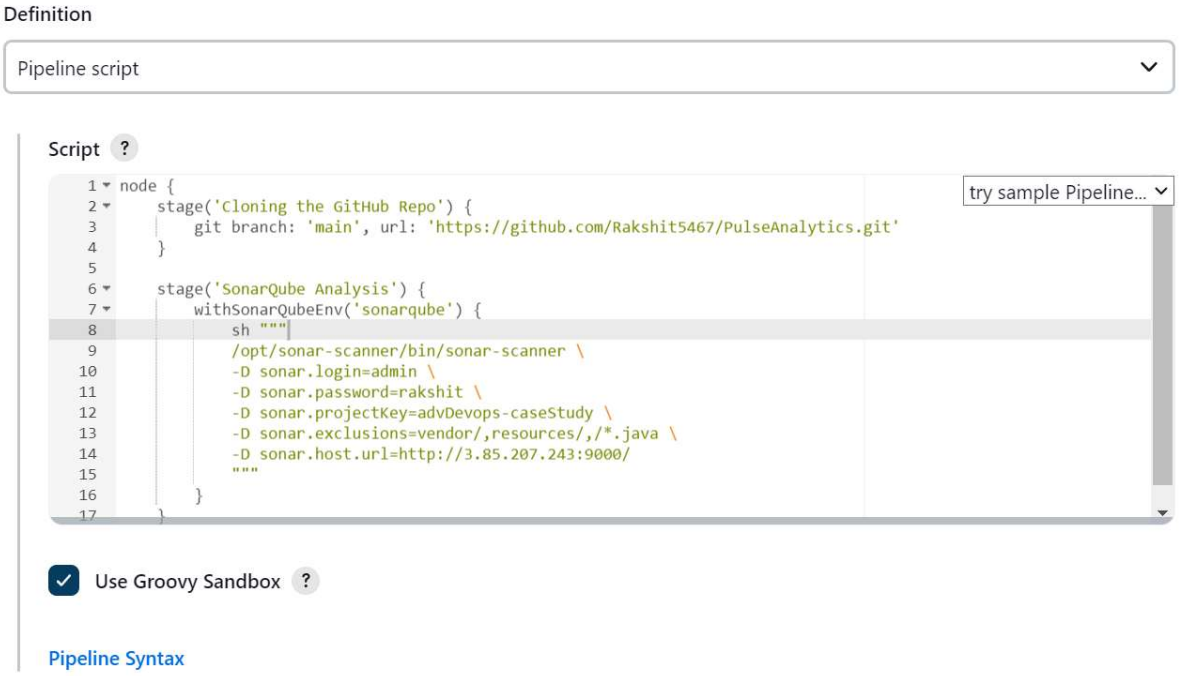
2. Check the “Github project” box. Enter the url of your github project repository in url. Make sure that the repository is public.

3. Now in the pipeline script enter the following script. This will clone the particular github repo and will analyze the build in sonarqube.

```
    node {
stage('Cloning the GitHub Repo') {
   git branch: 'main', url: 'https://github.com/Rakshit5467/PulseAnalytics.git'
}

stage('SonarQube Analysis') {
   withSonarQubeEnv('sonarqube') {
      sh """
      <path_to_sonar_scanner> \
      -D sonar.login=<username> \
      -D sonar.password=<password> \
      -D sonar.projectKey=<sonarqube-project-key> \
      -D sonar.exclusions=vendor/,resources/,/*.java \
```

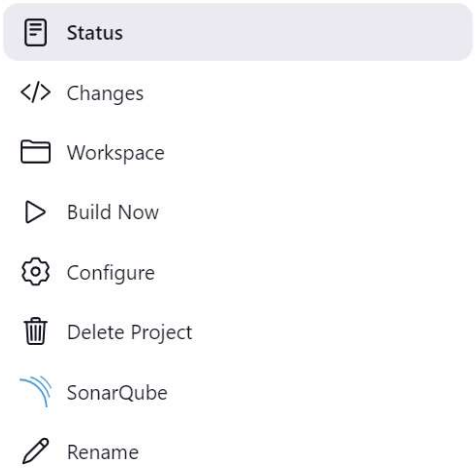
```
        -D sonar.host.url=http://<public_ip_address_of_sonar_instance>:9000
        """
    }
  }
}
```

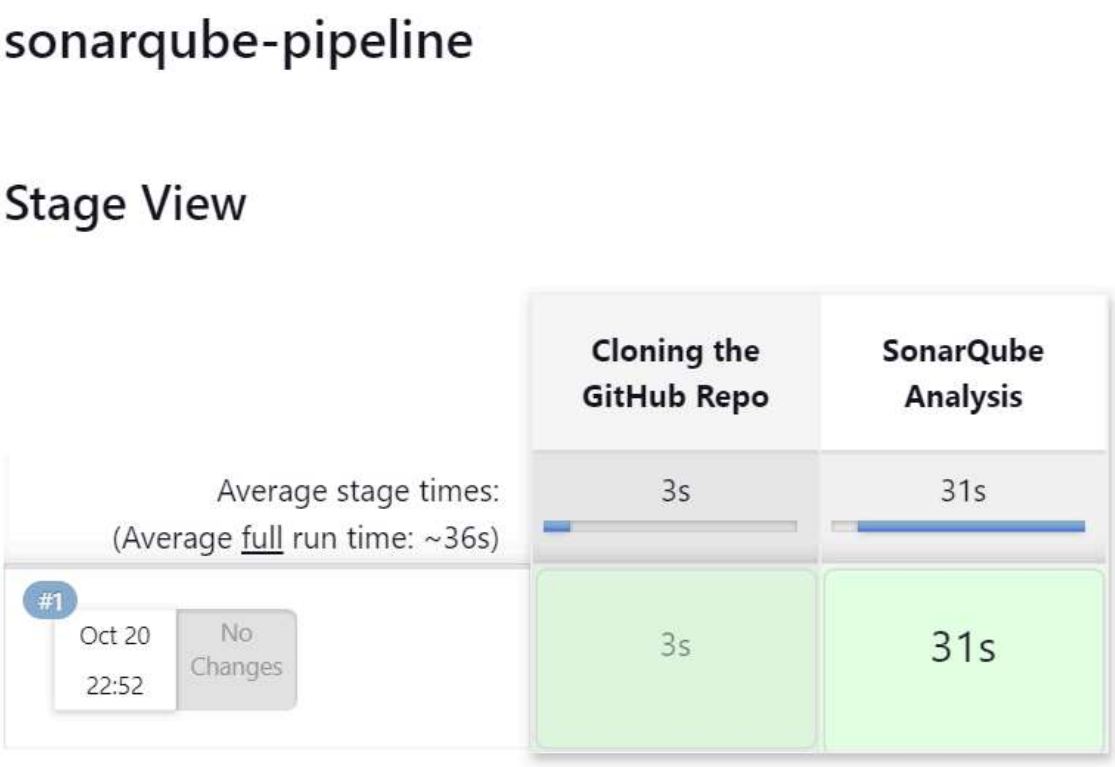


Path to sonar-scanner can be found using the command “which sonar-scanner”

Save all the configuration

4. Open the pipeline and click on build now in the left panel.





Stage view will show successful build of the pipeline in both stages.

5. Open the build and click on console output.

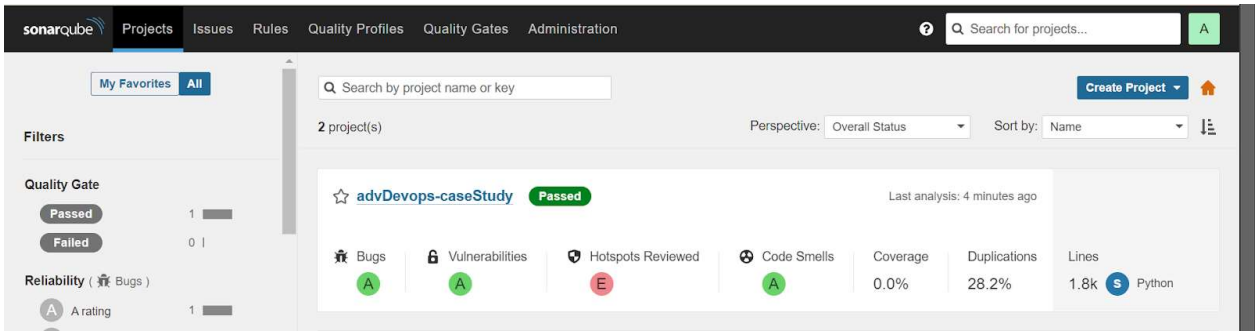
Console Output

Download Copy View as plain text

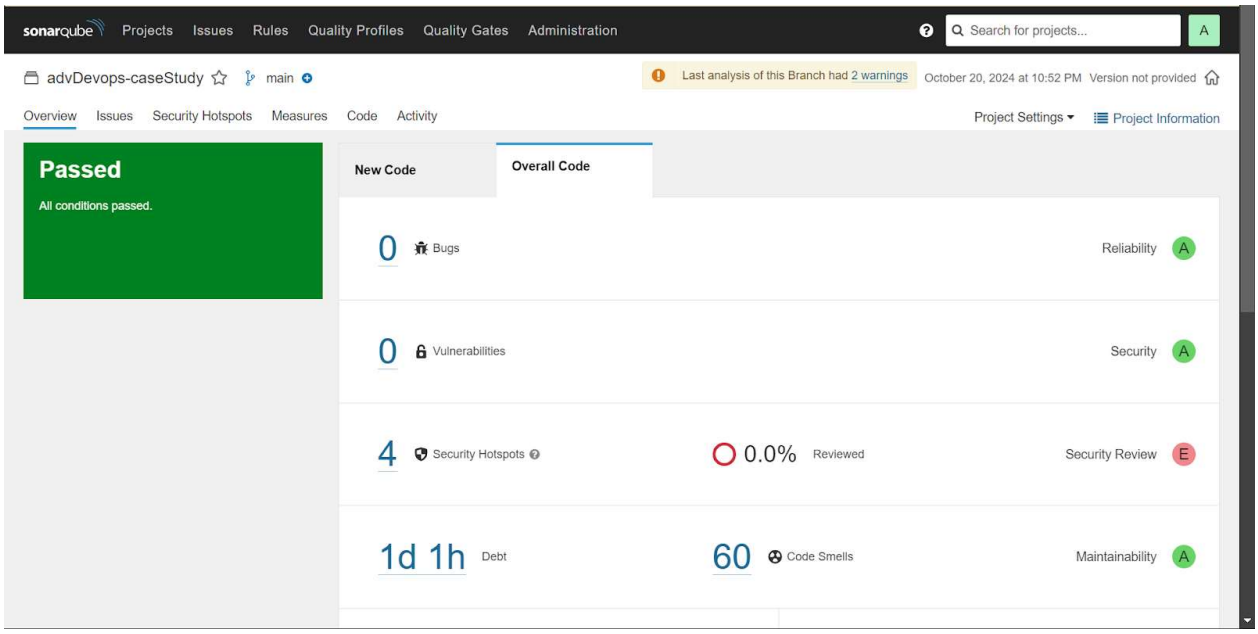
```
Started by user Rakshit Kumar Sharma
[Pipeline] Start of Pipeline (hide)
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/sonarqube-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Cloning the GitHub Repo)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/Rakshit5467/PulseAnalytics.git
 > /usr/bin/git init /var/lib/jenkins/workspace/sonarqube-pipeline # timeout=10
Fetching upstream changes from https://github.com/Rakshit5467/PulseAnalytics.git
 > /usr/bin/git --version # timeout=10
 > git --version # 'git version 2.40.1'
 > /usr/bin/git fetch --tags --force --progress -- https://github.com/Rakshit5467/PulseAnalytics.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

```
17:22:42.112 INFO  More about the report processing at http://3.85.207.243:9000/api/ce/task?
id=AZKq898gB1xtQ2llypk0
17:22:42.129 INFO  Analysis total time: 22.489 s
17:22:42.130 INFO  EXECUTION SUCCESS
17:22:42.131 INFO  Total time: 30.659s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

6. Go to the sonarqube project. While performing the experiment, I entered a different name in the pipeline script for the sonarqube key. Since it could not find the particular project in sonarqube, it automatically created a new project.



7. Open the project to see the details of the analysis.



Conclusion:

This case study showcases the seamless integration of static analysis into a CI/CD pipeline by automating the deployment of Jenkins and SonarQube on AWS using Terraform. By configuring Jenkins to trigger SonarQube analysis for a Python codebase, developers receive immediate feedback on code quality, helping to identify issues such as bugs, code smells, and security vulnerabilities. This proactive approach not only improves code quality but also streamlines the development process, reducing technical debt and ensuring that issues are caught early, before deployment. The use of Terraform for infrastructure provisioning also reinforces best practices in Infrastructure as Code (IaC).