Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Advance DevOps Case Study (Topic 9)

Aim: 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.

Introduction:

- **Case Study Overview:**

This case study illustrates how to integrate static code analysis into a DevOps pipeline using Terraform, Jenkins, and SonarQube. By utilizing Terraform, the deployment of Jenkins and SonarQube servers on AWS is automated, ensuring a consistent and scalable infrastructure.

Jenkins serves as the CI/CD server, automating tasks like building and testing code, while SonarQube analyzes the code for bugs, vulnerabilities, and code smells. The setup provides immediate feedback to developers, enabling early detection of issues.

With Terraform's Infrastructure-as-Code approach, the entire process is streamlined, reducing manual setup and ensuring a reliable environment. This case study will demonstrate how these tools work together, from deploying infrastructure to configuring the Jenkins pipeline and reviewing SonarQube analysis results, promoting better code quality and efficient development practices.

- **Key Feature and Application:**

The unique feature of this integration is the automated deployment and configuration of the CI/CD pipeline and code analysis tools using Terraform, making the setup reproducible and scalable. This infrastructure-as-code (IaC) approach ensures that every part of the setup—from servers to configuration—can be managed and version-controlled, leading to a more reliable and consistent environment.

Practical Use:

- **Enhanced Code Quality:** Developers can receive immediate feedback on code issues, such as bugs, code smells, and vulnerabilities, during the CI/CD process.
- **Automated Deployment:** Utilizing Terraform allows for quick and consistent deployment of the Jenkins and SonarQube infrastructure, reducing the manual effort required.
- **Scalability and Efficiency:** Since all components are managed as code, the infrastructure can be easily scaled or replicated, making it suitable for dynamic environments.

Prerequisites:

1. An AWS account with an IAM user set up on it.
2. Terraform downloaded on your system.

Step-by-Step Explanation:

Step 1: Create a folder for your project and create the following files inside of it:-

a. main.tf:-

```
resource "aws_instance" "jenkins" {
  ami           = var.jenkins_ami_id
  instance_type = var.instance_type
  key_name      = var.key_pair_name
  security_groups = [aws_security_group.jenkins_sg.name]

  tags = {
    Name = "Jenkins Server"
  }
}
```

```
resource "aws_instance" "sonarqube" {
  ami           = var.sonarqube_ami_id
  instance_type = var.instance_type
  key_name      = var.key_pair_name
  security_groups = [aws_security_group.sonarqube_sg.name]

  tags = {
    Name = "SonarQube Server"
  }
}
```

b. outputs.tf:-

```
output "jenkins_public_ip" {
  description = "The public IP of the Jenkins server"
  value       = aws_instance.jenkins.public_ip
}

output "sonarqube_public_ip" {
  description = "The public IP of the SonarQube server"
  value       = aws_instance.sonarqube.public_ip
}
```

c. provider.tf:-

```
provider "aws" {
  region = var.aws_region
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}
```

d. security_groups.tf:-

```
resource "aws_security_group" "jenkins_sg" {
  name        = "jenkins_security_group"
  description = "Allow SSH and HTTP access for Jenkins"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_security_group" "sonarqube_sg" {
  name        = "sonarqube_security_group"
  description = "Allow SSH and HTTP access for SonarQube"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 9000
    to_port     = 9000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

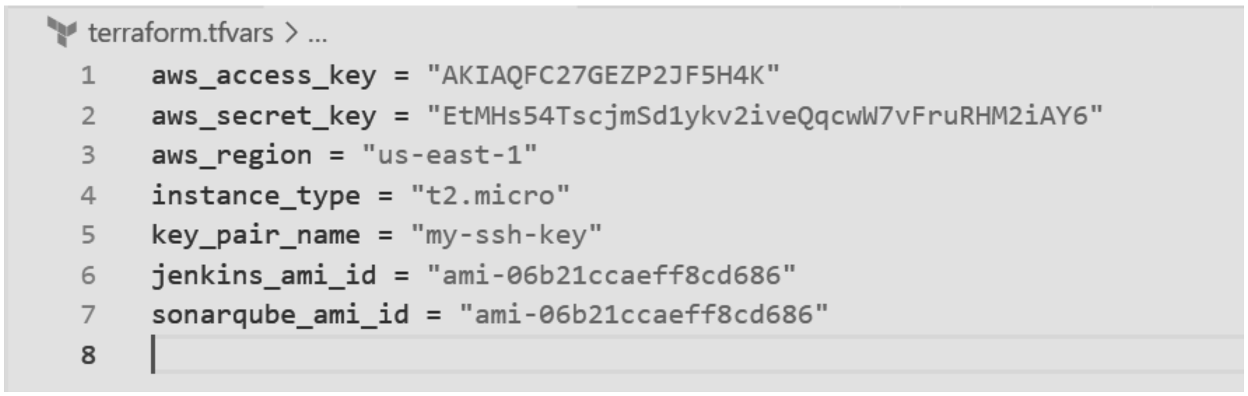
e. variables.tf:-

```
variable "aws_region" {
  description = "The AWS region to deploy resources in"
  default     = "us-east-1"
}
variable "aws_access_key" {
  description = "Your AWS access key"
}
variable "aws_secret_key" {
  description = "Your AWS secret key"
}
variable "instance_type" {
  description = "The EC2 instance type"
  default     = "t2.micro"
}
variable "key_pair_name" {
  description = "The name of the key pair for SSH access"
}
variable "jenkins_ami_id" {
  description = "The AMI ID for the Jenkins server"
}
variable "sonarqube_ami_id" {
  description = "The AMI ID for the SonarQube server"
}
```

f. terraform.tfvars:-

```
aws_access_key = "YOUR_ACCESS_KEY_HERE"
aws_secret_key = "YOUR_SECRET_KEY_HERE"
aws_region = "us-east-1"
instance_type = "t2.micro"
key_pair_name = "my-ssh-key"
jenkins_ami_id = "ami-0abcdef1234567890"
sonarqube_ami_id = "ami-0abcdef1234567890"
```

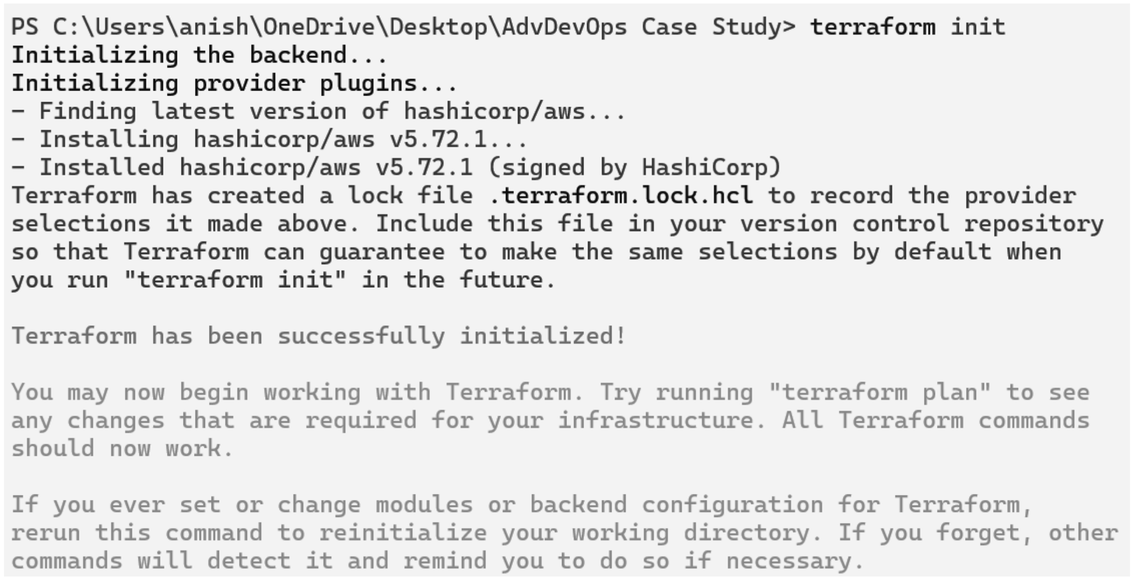
In the above file, replace the placeholder text with your values as such:-



```
aws_access_key = "AKIAQFC27GEZP2JF5H4K"
aws_secret_key = "EtMHs54TscjmSd1ykv2iveQqcwW7vFruRHM2iAY6"
aws_region = "us-east-1"
instance_type = "t2.micro"
key_pair_name = "my-ssh-key"
jenkins_ami_id = "ami-06b21ccaeff8cd686"
sonarqube_ami_id = "ami-06b21ccaeff8cd686"
```

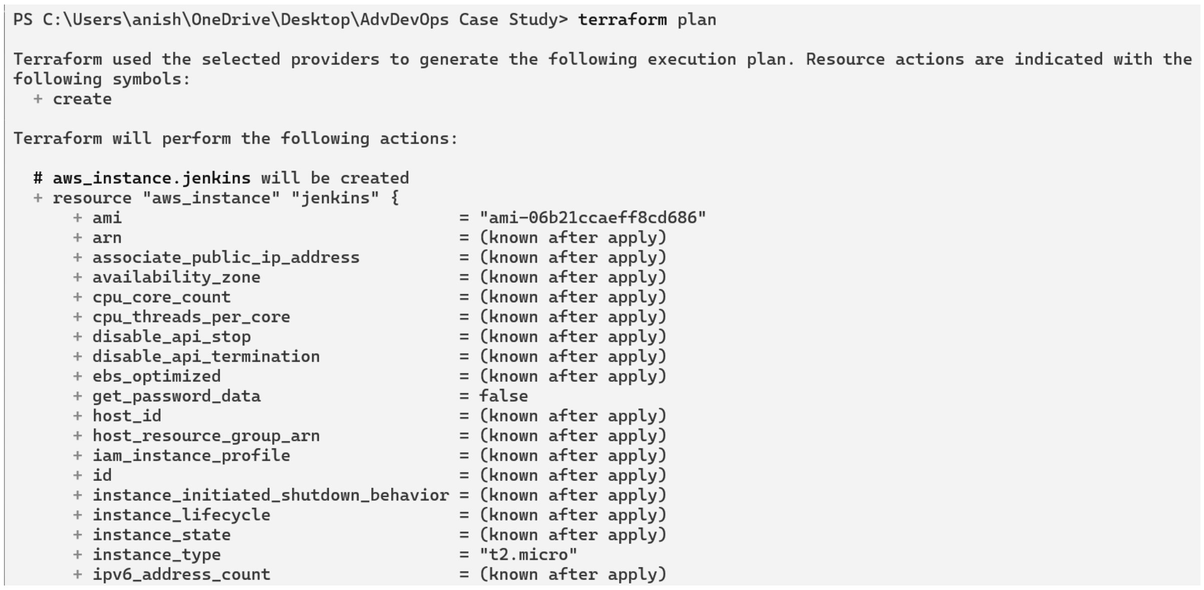
Step 2: Open a terminal inside of the folder that was created for your project and run the following commands in the terminal:-

1. terraform init

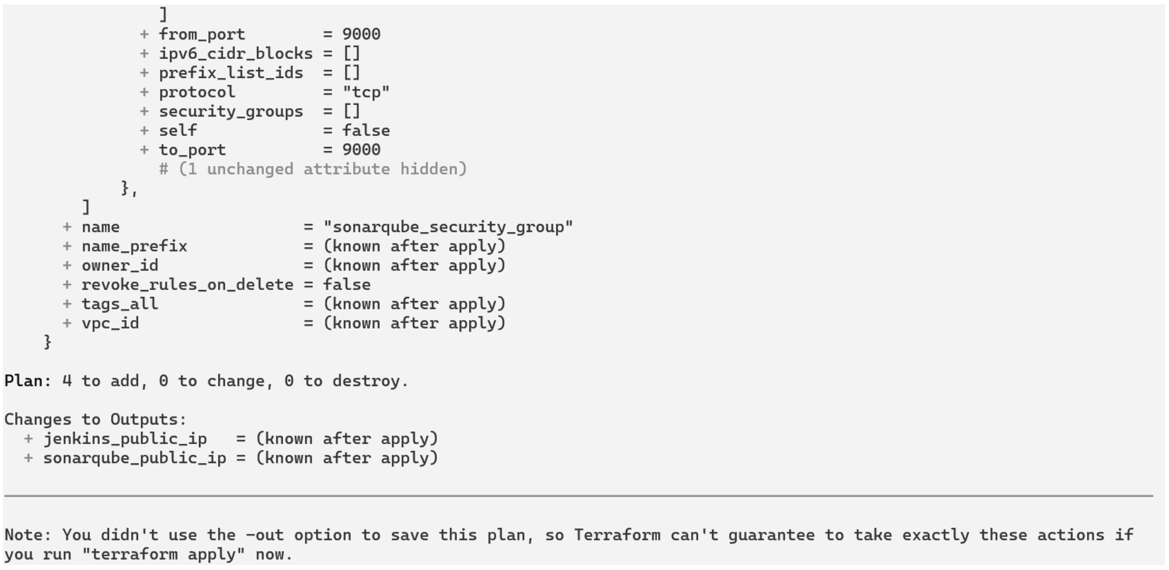


```
PS C:\Users\anish\OneDrive\Desktop\AdvDevOps Case Study> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock 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.
```

2. terraform plan

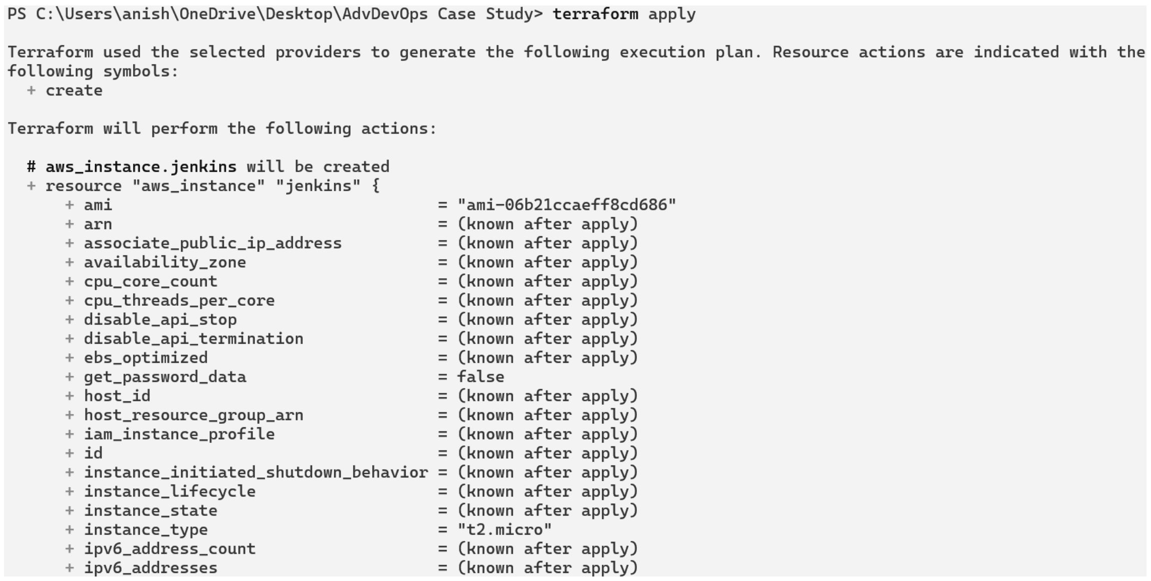


```
PS C:\Users\anish\OneDrive\Desktop\AdvDevOps Case Study> terraform plan

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)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
```

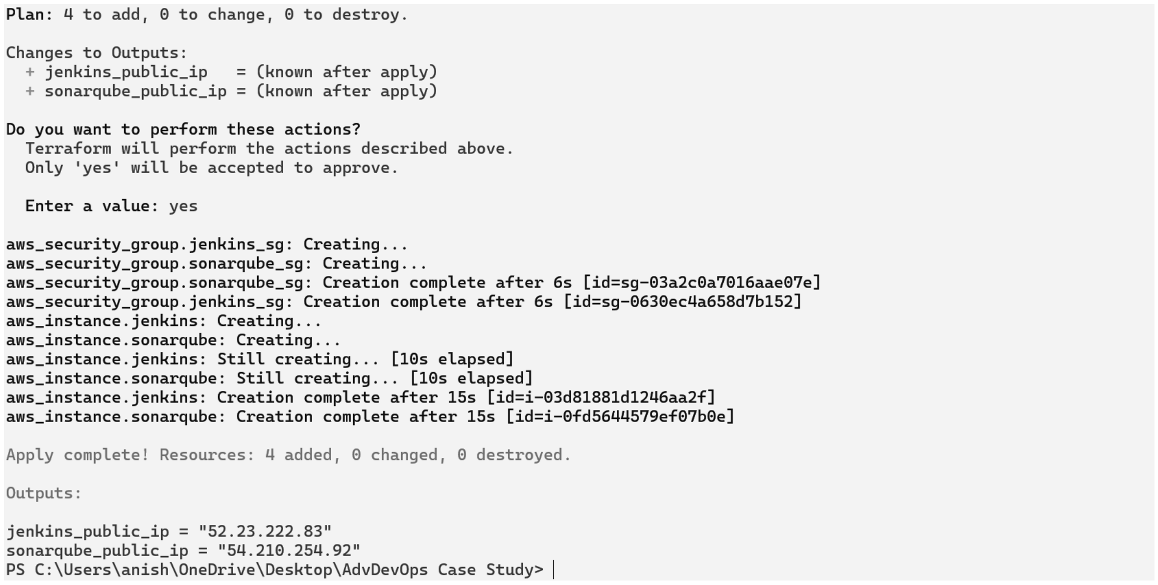


```
                ]
              + from_port        = 9000
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "tcp"
              + security_groups  = []
              + self             = false
              + to_port          = 9000
                # (1 unchanged attribute hidden)
            },
        ]
      + name                   = "sonarqube_security_group"
      + name_prefix            = (known after apply)
      + owner_id               = (known after apply)
      + revoke_rules_on_delete = false
      + tags_all               = (known after apply)
      + vpc_id                 = (known after apply)
    }

Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + jenkins_public_ip   = (known after apply)
  + sonarqube_public_ip = (known after apply)

────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if
you run "terraform apply" now.
```

3. terraform apply:-



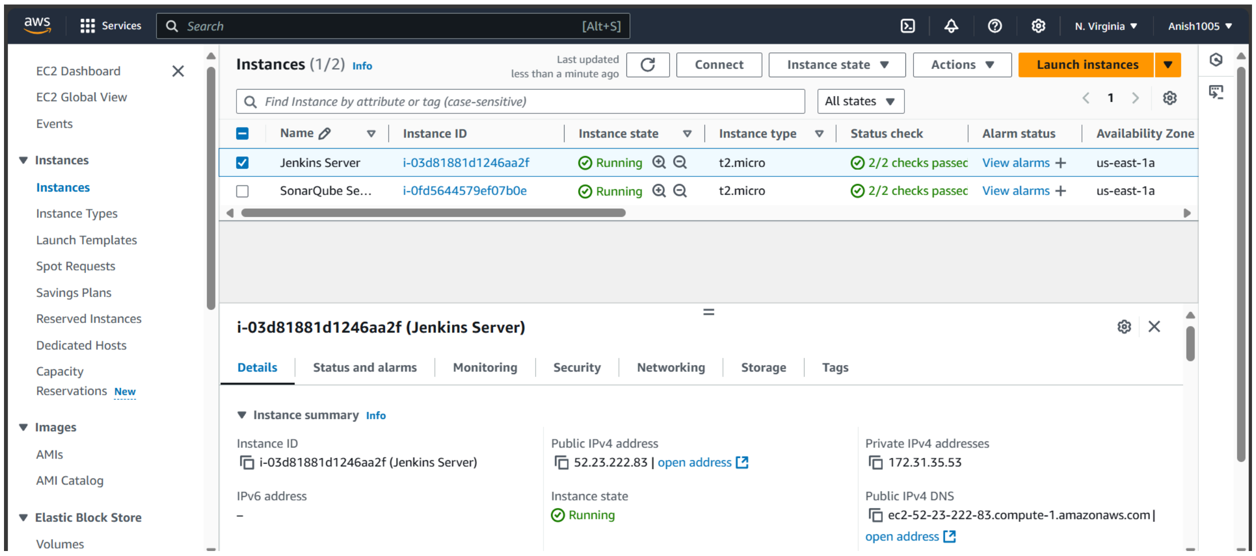
```
PS C:\Users\anish\OneDrive\Desktop\AdvDevOps Case Study> terraform apply

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)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
```



```
Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + jenkins_public_ip   = (known after apply)
  + sonarqube_public_ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_security_group.jenkins_sg: Creating...
aws_security_group.sonarqube_sg: Creating...
aws_security_group.sonarqube_sg: Creation complete after 6s [id=sg-03a2c0a7016aae07e]
aws_security_group.jenkins_sg: Creation complete after 6s [id=sg-0630ec4a658d7b152]
aws_instance.jenkins: Creating...
aws_instance.sonarqube: Creating...
aws_instance.jenkins: Still creating... [10s elapsed]
aws_instance.sonarqube: Still creating... [10s elapsed]
aws_instance.jenkins: Creation complete after 15s [id=i-03d81881d1246aa2f]
aws_instance.sonarqube: Creation complete after 15s [id=i-0fd5644579ef07b0e]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

jenkins_public_ip = "52.23.222.83"
sonarqube_public_ip = "54.210.254.92"
PS C:\Users\anish\OneDrive\Desktop\AdvDevOps Case Study>
```

‘terraform init’ initializes the working directory, ‘terraform plan’ previews changes without applying them, and ‘terraform apply’ executes the planned changes to create or modify infrastructure.
As a result of running the above, two instances (one for Jenkins server and the other for the SonarQube server) are created on our AWS account.

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25



Instances (1/2) Info

Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Jenkins Server	i-03d81881d1246aa2f	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a
SonarQube Se...	i-0fd5644579ef07b0e	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a

i-03d81881d1246aa2f (Jenkins Server)

Instance summary Info

Instance ID i-03d81881d1246aa2f (Jenkins Server)	Public IPv4 address 52.23.222.83 | open address	Private IPv4 addresses 172.31.35.53
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-52-23-222-83.compute-1.amazonaws.com | open address

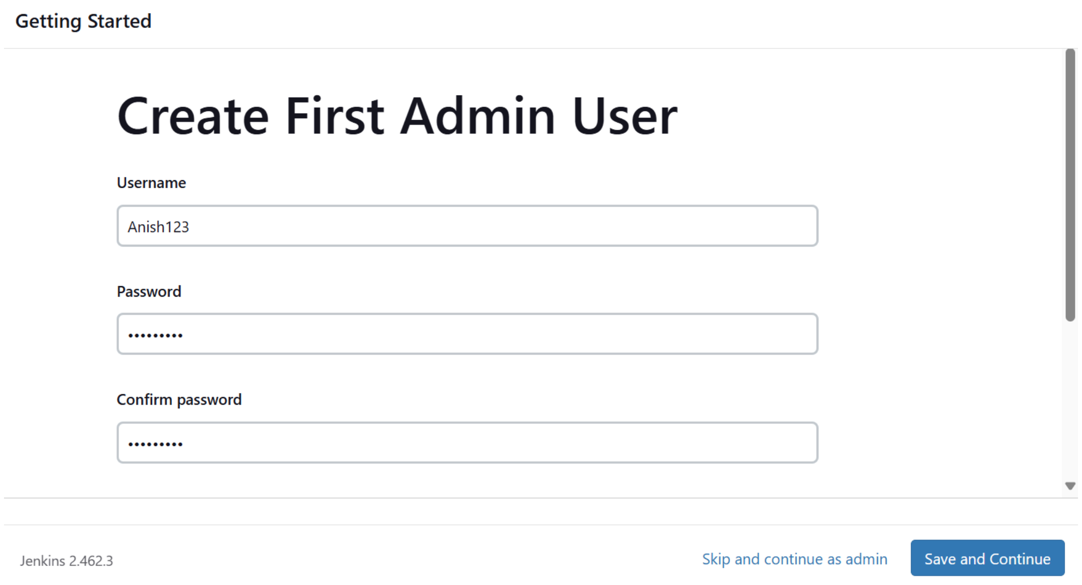
Step 3: Connect both the instances to your local terminal using SSH.

Step 4: Go onto your AWS console, change the instance state of both the instances to ‘Stopped’ and change the instance type of ‘Jenkins Server’ from t2.micro to t3.medium and change the instance type of ‘SonarQube Server’ from t2.micro to t3.small. This must be done because both Jenkins and SonarQube require a lot of space (in GBs) and t2.micro doesnt have enough memory space to accommodate either Jenkins or SonarQube. Once this is done, change the instance type of both the instances to ‘Running’.
Make sure to stop your instances when they aren’t in use because t3.micro is significantly more expensive than t2.micro and keeping them running can add to your bill significantly.

Step 5: Open the terminal for your Jenkins server and run the following commands to install Jenkins onto your instance:-

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

Running the above commands should install Jenkins onto your instance. Now, to open Jenkins, paste the following into your browser:-
http://<public_ip_address_of_your_Jenkins_instance>:8080
The Jenkins Sign Up page should appear. Complete the configuration:



Getting Started

Create First Admin User

Username

Anish123

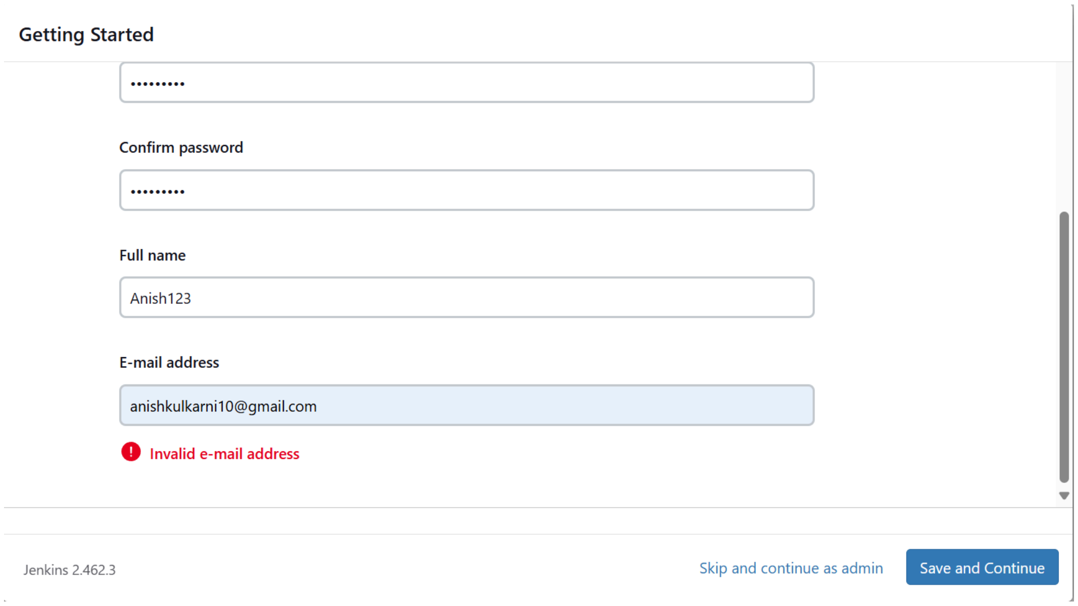
Password

•••••••••

Confirm password

•••••••••

Jenkins 2.462.3



Getting Started

•••••••••

Confirm password

•••••••••

Full name

Anish123

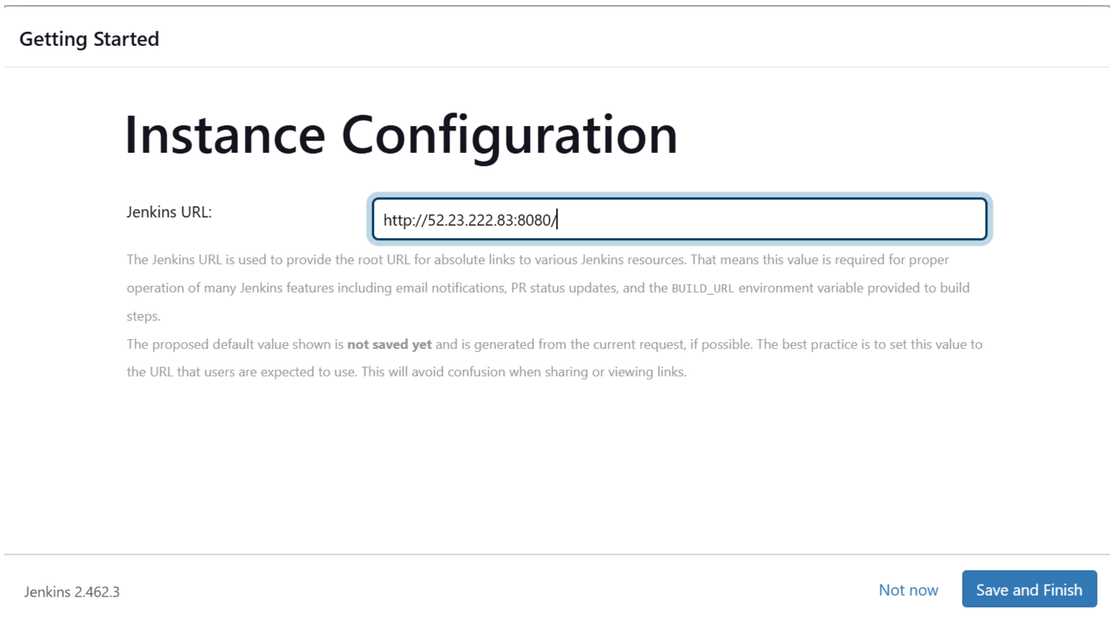
E-mail address

anishkulkarni10@gmail.com

Invalid e-mail address

Jenkins 2.462.3

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25



Getting Started

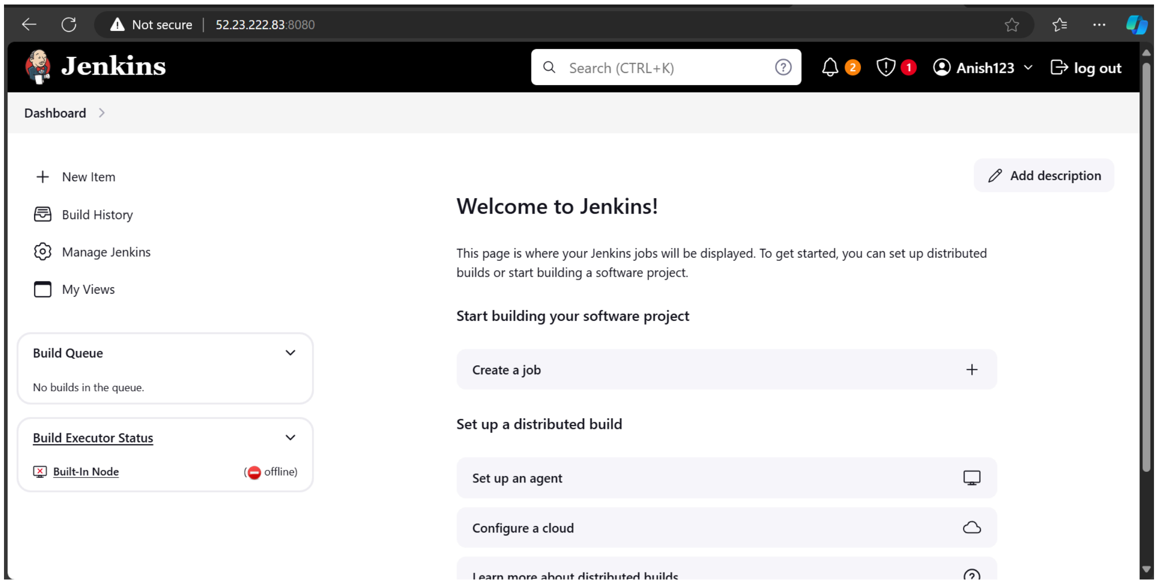
Instance Configuration

Jenkins URL: http://52.23.222.83:8080/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.462.3



Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds

Build Queue

No builds in the queue.

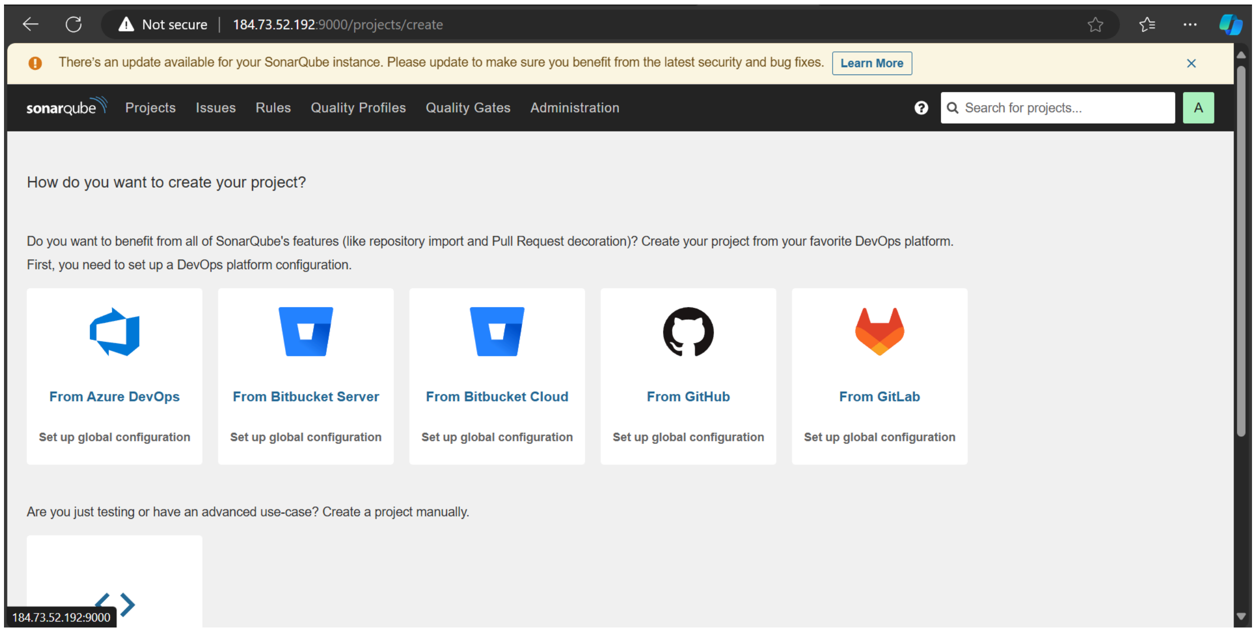
Build Executor Status

Built-In Node (offline)

The Jenkins dashboard should open.

Step 6: Open the terminal for your SonarQube server and run the following commands to install SonarQube onto your instance:-
sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.2.77730.zip -P /opt
sudo unzip /opt/sonarqube-9.9.2.77730.zip -d /opt
sudo mv /opt/sonarqube-9.9.2.77730 /opt/sonarqube
sudo chown -R sonar:sonar /opt/sonarqube
cd /opt/sonarqube/bin/linux-x86-64
./sonar.sh start

Running the above commands should install SonarQube onto your instance. Now, to open SonarQube, paste the following into your browser:-
http://<public_ip_address_of_your_SonarQube_instance>:9000
The SonarQube Sign Up page should appear. The default credentials are admin/admin. Then, change the password to the password of your choice and complete the configuration.
After completing all of the above, the SonarQube dashboard should appear:-

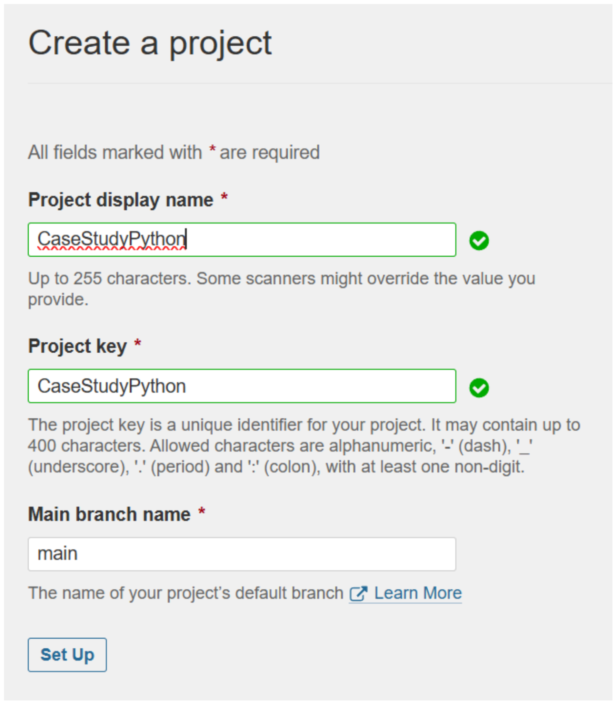

Not secure | 184.73.52.192:9000/projects/create
There's an update available for your SonarQube instance. Please update to make sure you benefit from the latest security and bug fixes.
Learn More
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Search for projects...
A
From Azure DevOps
Set up global configuration
From Bitbucket Server
Set up global configuration
From Bitbucket Cloud
Set up global configuration
From GitHub
Set up global configuration
From GitLab
Set up global configuration
184.73.52.192:9000

How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform.
First, you need to set up a DevOps platform configuration.

Are you just testing or have an advanced use-case? Create a project manually.

Step 7: Create a SonarQube project by clicking on 'Manually' on the dashboard and giving it a name and completing its configuration:-



Create a project

All fields marked with * are required

Project display name *

CaseStudyPython

Up to 255 characters. Some scanners might override the value you provide.

Project key *

CaseStudyPython

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Main branch name *

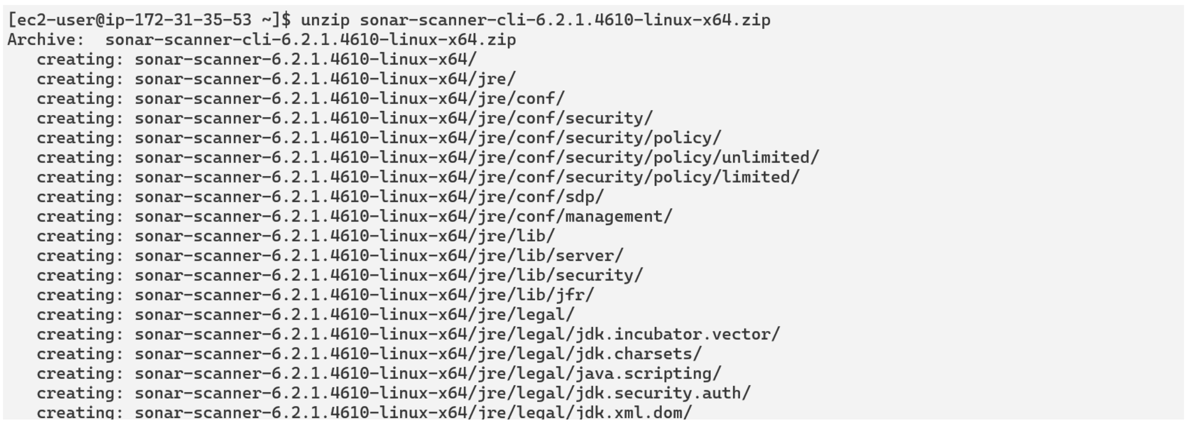
main

The name of your project's default branch

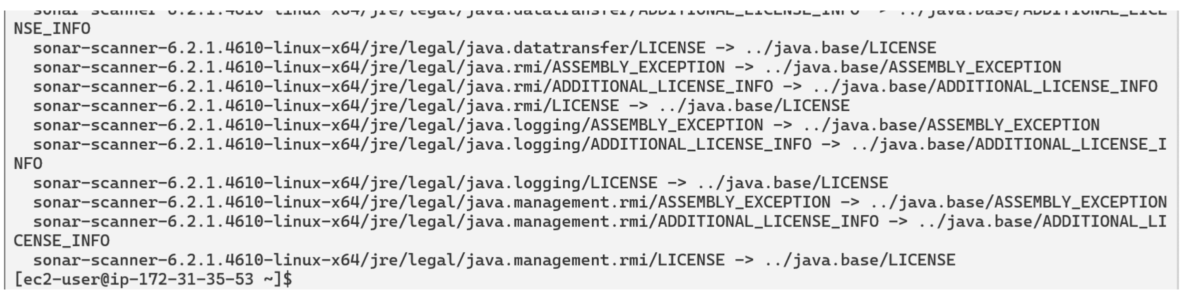
Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Step 8: Install SonarScanner CLI onto your system.

Step 9: Go onto the terminal of your Jenkins server and run the following commands:-
scp -i "<location_of_your_.pem_file" "location_of_sonar-scanner_cli" ec2-user@<ip_address_of_SonarQube_server>:/home/ec2-user/
unzip sonar-scanner-cli-6.2.0.4584-windows-x64.zip

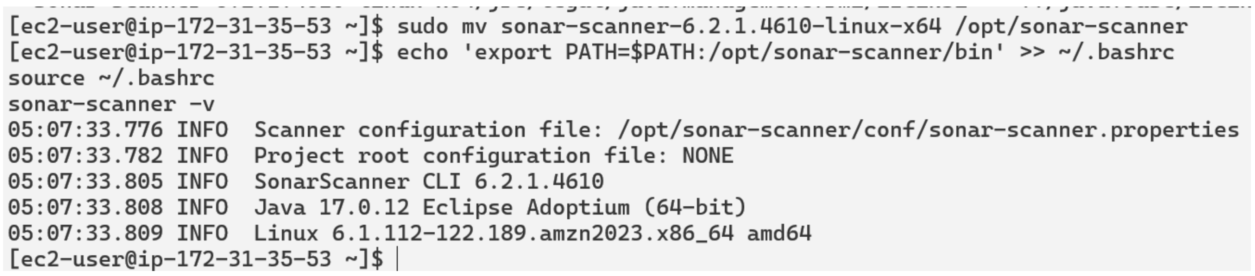


```
[ec2-user@ip-172-31-35-53 ~]$ unzip sonar-scanner-cli-6.2.1.4610-linux-x64.zip
Archive:  sonar-scanner-cli-6.2.1.4610-linux-x64.zip
   creating: sonar-scanner-6.2.1.4610-linux-x64/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/conf/
   creating: 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/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/jdk.security.auth/
   creating: sonar-scanner-6.2.1.4610-linux-x64/jre/legal/jdk.xml.dom/
```



```
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.datatransfer/ADDITIONAL_LICENSE_INFO -> ../java.base/ADDITIONAL_LICE
NSE_INFO
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.datatransfer/LICENSE -> ../java.base/LICENSE
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.rmi/ASSEMBLY_EXCEPTION -> ../java.base/ASSEMBLY_EXCEPTION
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.rmi/ADDITIONAL_LICENSE_INFO -> ../java.base/ADDITIONAL_LICENSE_INFO
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.rmi/LICENSE -> ../java.base/LICENSE
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.logging/ASSEMBLY_EXCEPTION -> ../java.base/ASSEMBLY_EXCEPTION
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.logging/ADDITIONAL_LICENSE_INFO -> ../java.base/ADDITIONAL_LICENSE_I
NFO
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.logging/LICENSE -> ../java.base/LICENSE
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.management.rmi/ASSEMBLY_EXCEPTION -> ../java.base/ASSEMBLY_EXCEPTION
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.management.rmi/ADDITIONAL_LICENSE_INFO -> ../java.base/ADDITIONAL_LI
CENSE_INFO
  sonar-scanner-6.2.1.4610-linux-x64/jre/legal/java.management.rmi/LICENSE -> ../java.base/LICENSE
[ec2-user@ip-172-31-35-53 ~]$
```

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



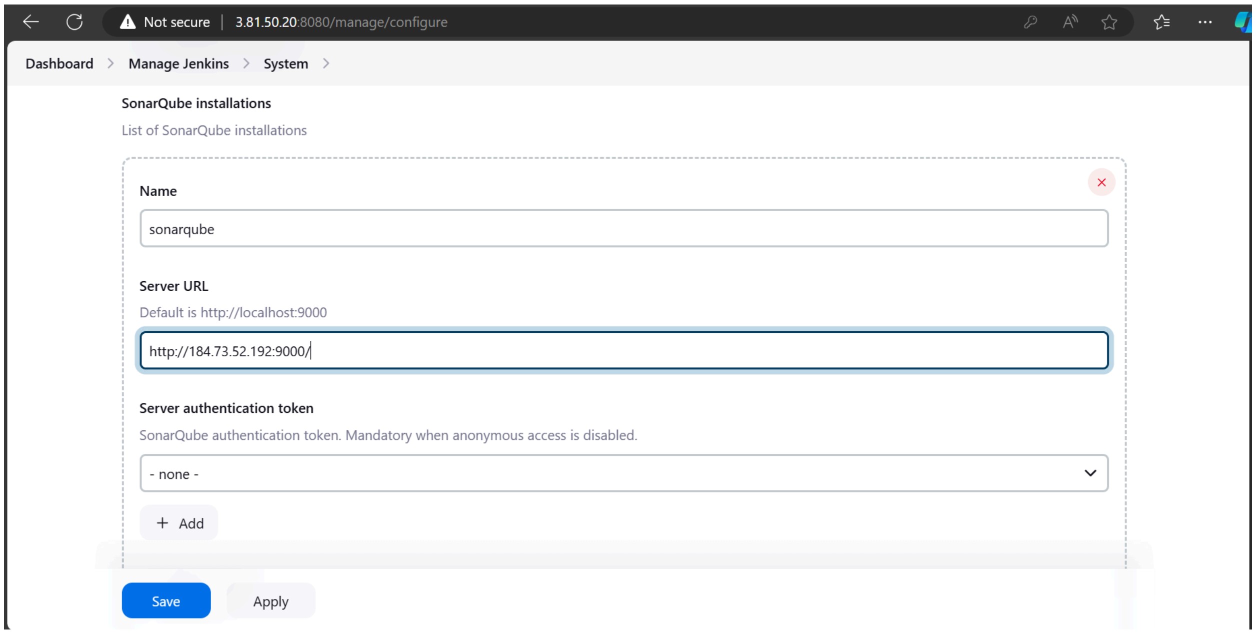
```
[ec2-user@ip-172-31-35-53 ~]$ sudo mv sonar-scanner-6.2.1.4610-linux-x64 /opt/sonar-scanner
[ec2-user@ip-172-31-35-53 ~]$ echo 'export PATH=$PATH:/opt/sonar-scanner/bin' >> ~/.bashrc
source ~/.bashrc
sonar-scanner -v
05:07:33.776 INFO  Scanner configuration file: /opt/sonar-scanner/conf/sonar-scanner.properties
05:07:33.782 INFO  Project root configuration file: NONE
05:07:33.805 INFO  SonarScanner CLI 6.2.1.4610
05:07:33.808 INFO  Java 17.0.12 Eclipse Adoptium (64-bit)
05:07:33.809 INFO  Linux 6.1.112-122.189.amzn2023.x86_64 amd64
[ec2-user@ip-172-31-35-53 ~]$ 
```

nano /opt/sonar-scanner/sonar-scanner-6.2.1.4610-linux-x64/conf/sonar-scanner.properties
When the sonar.properties file opens, add the following to the file:-
sonar.host.url=http://<your-sonarqube-ip>:9000

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Step 10: Go to Manage Jenkins > Plugins from your Jenkins dashboard and install the following plugins:-
Pipeline: Stage View
Pipeline: Rest API
SonarQube Scanner
Git server

Step 11: Go to Manage Jenkins > System from your Jenkins dashboard, scroll down to 'SonarQube installations' and click on 'Add SonarQube'. Then give it a name and add the IP address of your SonarQube.


Not secure | 3.81.50.20:8080/manage/configure
Dashboard > Manage Jenkins > System >
Save
Apply
Add

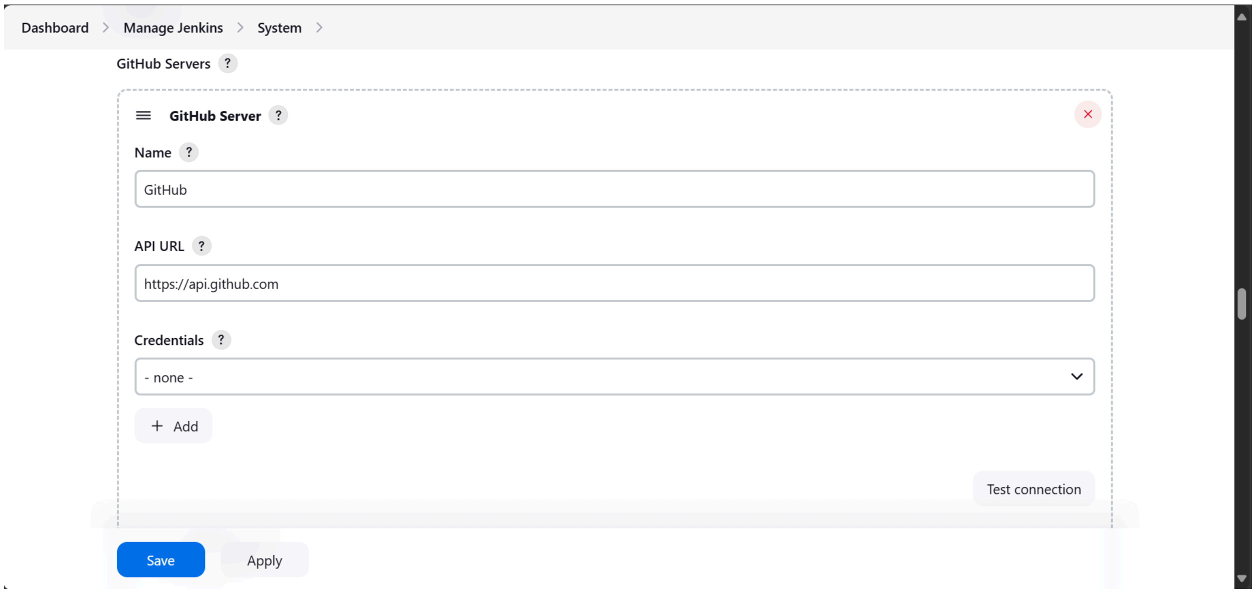
SonarQube installations
List of SonarQube installations

Name: sonarqube

Server URL
Default is http://localhost:9000
http://184.73.52.192:9000/

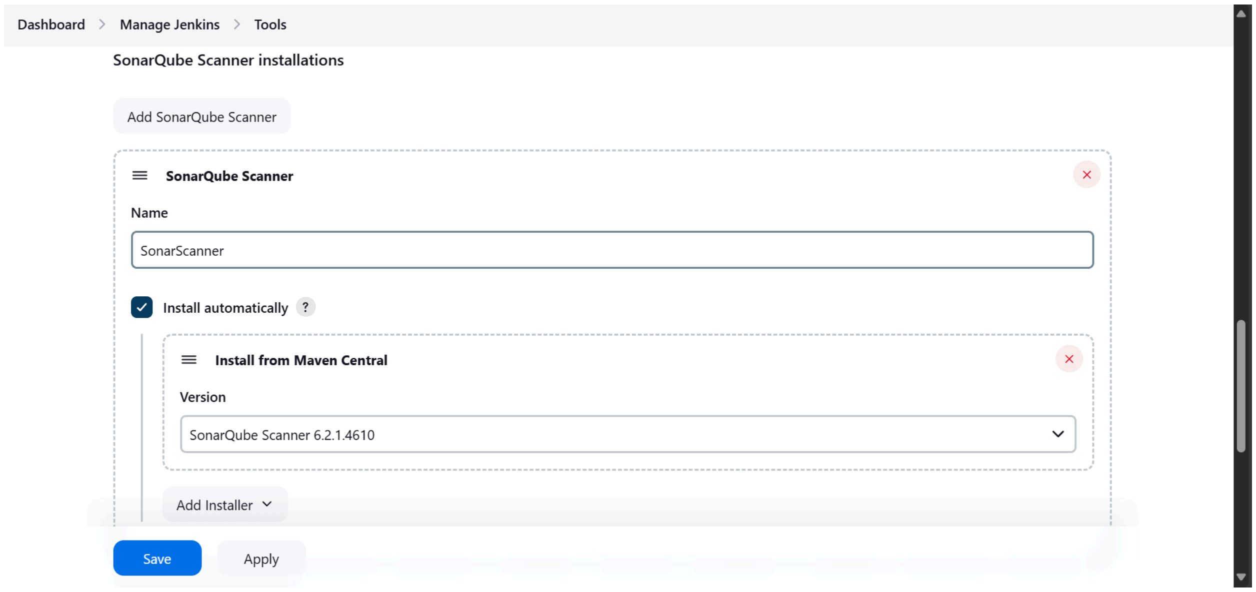
Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.
- none -

Then, navigate to 'GitHub Servers' and write the name as 'GitHub'.



GitHub Servers
GitHub Server
Name: GitHub
API URL: https://api.github.com
Credentials: - none -

Step 12: Go to Manage Jenkins > Tools from your Jenkins dashboard and scroll down to ‘SonarQube Scanner installations’ and click on ‘Add SonarQube Scanner’. Then, give it a name and choose the default version and click on ‘Save’.



Dashboard > Manage Jenkins > Tools

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name: SonarScanner

Install automatically: ✓

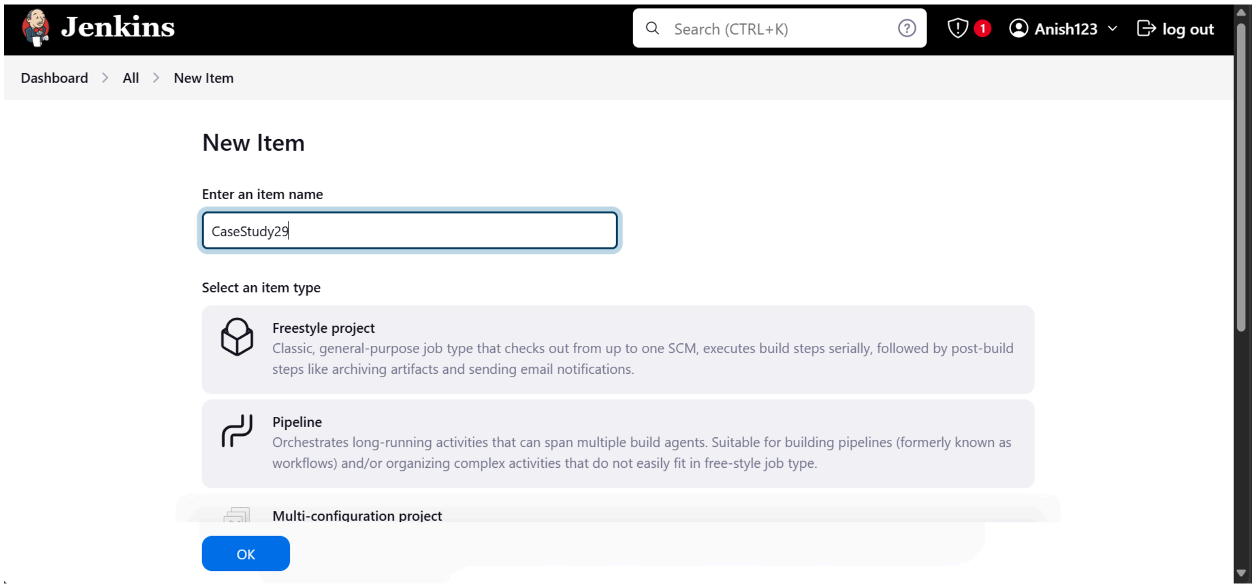
Install from Maven Central

Version: SonarQube Scanner 6.2.1.4610

Add Installer

Save Apply

Step 13: On your Jenkins dashboard, click on ‘New Item’. Then, give your project a name and click on ‘Pipeline’ as Item type.



Jenkins

Dashboard > All > New Item

New Item

Enter an item name: CaseStudy29

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

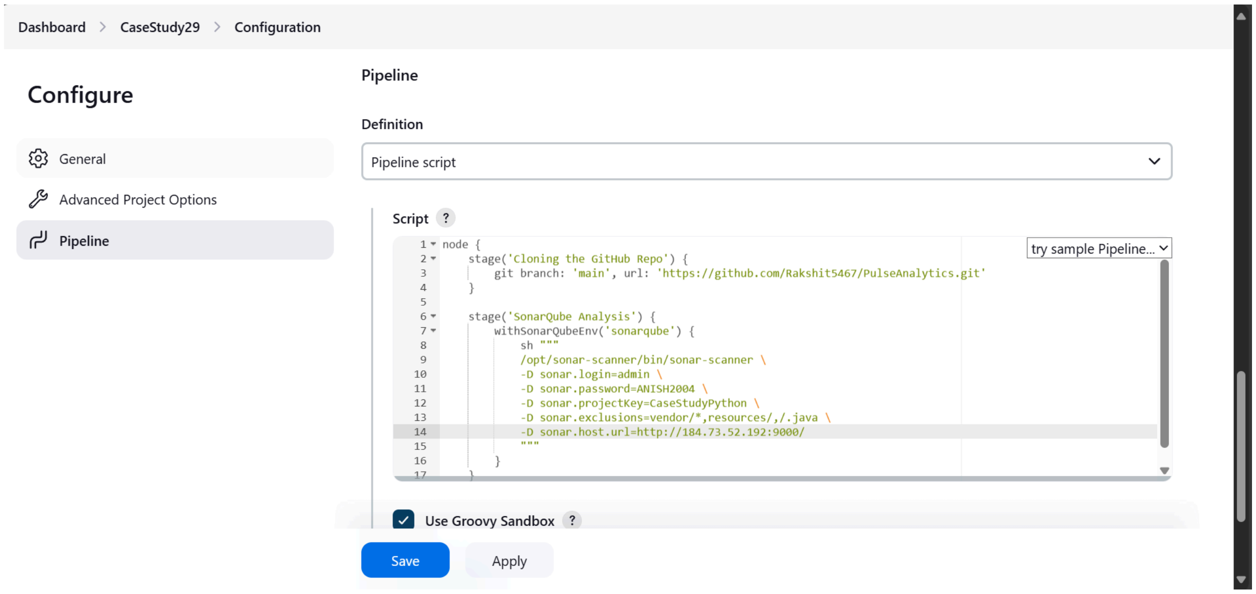
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project

OK

Then, in the Git section, add the link to the repository which contains the python project that you are going to test SonarQube on.
Next, scroll down to the ‘Pipeline script’ section and enter the following:-

```
node {
    stage('Cloning the GitHub Repo') {
        git branch: 'main', url: '<link_to_repository_of_python_project’
    }

    stage('SonarQube Analysis') {
        withSonarQubeEnv('sonarqube') {
            sh """
            <path_to_your_sonar-scanner>\
            -D sonar.login=admin \
            -D sonar.password=<your_password> \
            -D sonar.projectKey=<your_project_key> \
            -D sonar.exclusions=vendor/*,resources/,/.java \
            -D sonar.host.url=<url_of_your_sonarqube>
            """
        }
    }
}
```



Dashboard > CaseStudy29 > Configuration

Configure

General
Advanced Project Options
Pipeline

Pipeline

Definition: Pipeline script

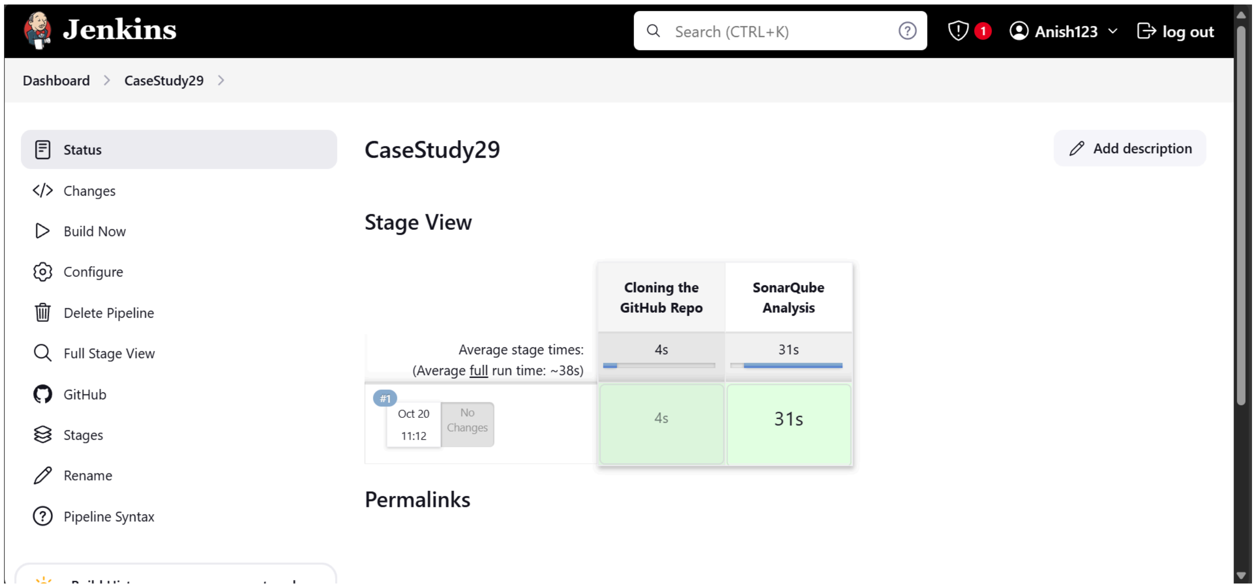
Script

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

    stage('SonarQube Analysis') {
        withSonarQubeEnv('sonarqube') {
            sh """
            /opt/sonar-scanner/bin/sonar-scanner \
            -D sonar.login=admin \
            -D sonar.password=ANISH2004 \
            -D sonar.projectKey=CaseStudyPython \
            -D sonar.exclusions=vendor/*,resources/,/.java \
            -D sonar.host.url=http://184.73.52.192:9000/
            """
        }
    }
```

Use Groovy Sandbox [✓]

Then, click on Save.

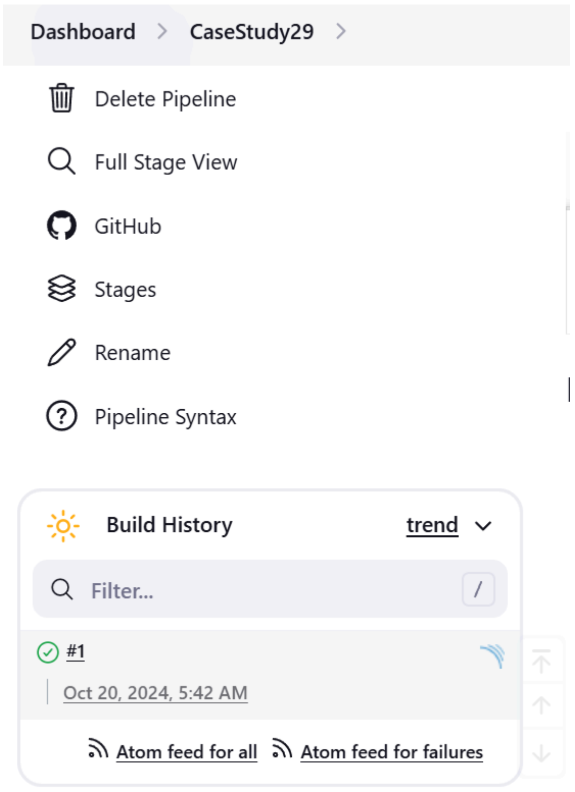
Step 14: Go to the Jenkins dashboard and click on 'Build now' from the left sidebar.



CaseStudy29

Stage View

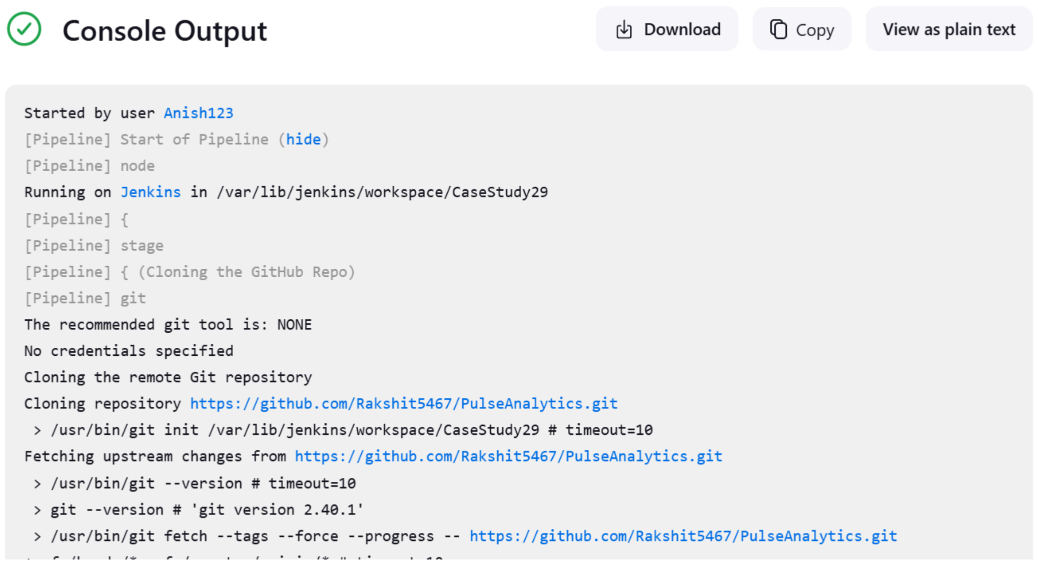
	Cloning the GitHub Repo	SonarQube Analysis
Average stage times: (Average full run time: ~38s)	4s	31s
#1 Oct 20 11:12 No Changes	4s	31s



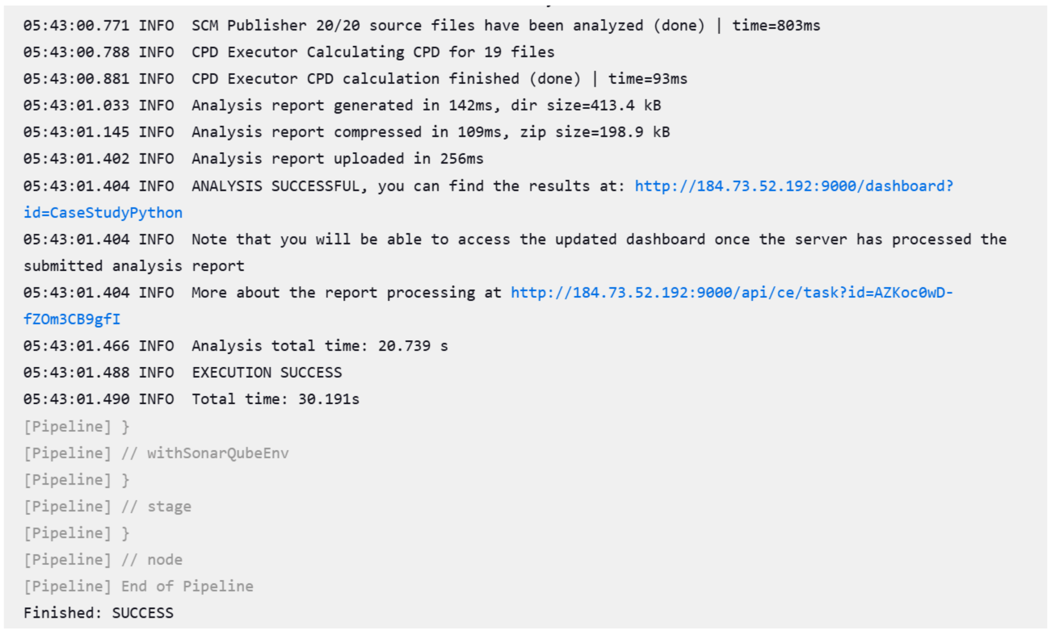
This builds your project.

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

You can also check your console output:-

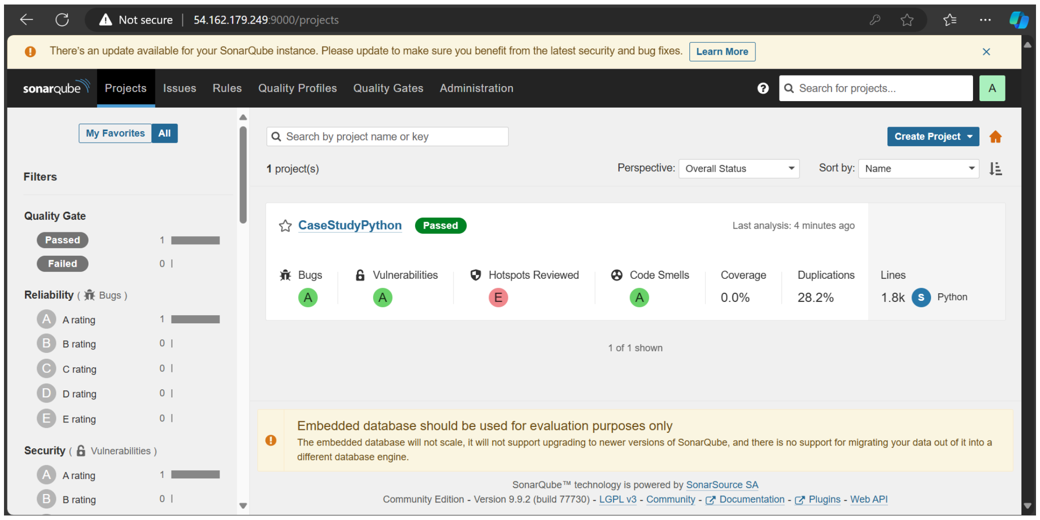


```
Started by user Anish123
[Pipeline] Start of Pipeline (hide)
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/CaseStudy29
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Cloning the GitHub Repo)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/Rakshit5467/PulseAnalytics.git
 > /usr/bin/git init /var/lib/jenkins/workspace/CaseStudy29 # 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
```



```
05:43:00.771 INFO  SCM Publisher 20/20 source files have been analyzed (done) | time=803ms
05:43:00.788 INFO  CPD Executor Calculating CPD for 19 files
05:43:00.881 INFO  CPD Executor CPD calculation finished (done) | time=93ms
05:43:01.033 INFO  Analysis report generated in 142ms, dir size=413.4 kB
05:43:01.145 INFO  Analysis report compressed in 109ms, zip size=198.9 kB
05:43:01.402 INFO  Analysis report uploaded in 256ms
05:43:01.404 INFO  ANALYSIS SUCCESSFUL, you can find the results at: http://184.73.52.192:9000/dashboard?
id=CaseStudyPython
05:43:01.404 INFO  Note that you will be able to access the updated dashboard once the server has processed the
submitted analysis report
05:43:01.404 INFO  More about the report processing at http://184.73.52.192:9000/api/ce/task?id=AZKoc0wD-
fZOm3CB9gfI
05:43:01.466 INFO  Analysis total time: 20.739 s
05:43:01.488 INFO  EXECUTION SUCCESS
05:43:01.490 INFO  Total time: 30.191s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Step 15: You can also go onto your SonarQube dashboard to check the output on your project.



Not secure | 54.162.179.249:9000/projects

There's an update available for your SonarQube instance. Please update to make sure you benefit from the latest security and bug fixes. Learn More

Filters

Quality Gate

Passed	1
Failed	0

Reliability (Bugs)

A rating	1
B rating	0
C rating	0
D rating	0
E rating	0

Security (Vulnerabilities)

A rating	1
B rating	0

1 project(s) Perspective: Overall Status Sort by: Name

CaseStudyPython Passed Last analysis: 4 minutes ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
A	A	E	A	0.0%	28.2%	1.8k Python

1 of 1 shown

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Community Edition - Version 9.9.2 (build 77730) - LGPL v3 - Community - Documentation - Plugins - Web API

Conclusion: This case study demonstrates how integrating Terraform, Jenkins, and SonarQube can effectively automate infrastructure setup and continuous code analysis. By using Terraform's Infrastructure-as-Code approach, the process of deploying Jenkins and SonarQube on AWS becomes scalable, consistent, and easy to manage, reducing manual effort. Jenkins automates tasks like building, testing, and deploying code, while SonarQube scans for issues such as bugs, code smells, and vulnerabilities, providing developers with early feedback. Together, these tools streamline the CI/CD pipeline, improving code quality and reducing the time spent on fixing problems.