

Static Analysis Integration with Terraform

1. Introduction

Case Study Overview:

This report outlines the deployment of Jenkins and SonarQube on Amazon Web Services (AWS) using Terraform. Jenkins serves as a continuous integration and continuous delivery (CI/CD) tool, automating the software development process, while SonarQube provides an essential code quality management platform. Together, they enhance the software development lifecycle by ensuring code quality and facilitating faster delivery of applications.

Background:

In the rapidly evolving landscape of software development, the integration of automation tools is essential for improving efficiency and collaboration among development teams. Jenkins is known for its flexibility and extensibility through various plugins, allowing for automated builds, tests, and deployments. SonarQube complements Jenkins by offering real-time code analysis, identifying potential vulnerabilities and code smells, which ultimately contributes to more maintainable codebases.

Importance of Automation in DevOps:

Automation is a cornerstone of DevOps practices, promoting collaboration between development and operations teams. By utilizing Jenkins for CI/CD processes and SonarQube for code quality checks, organizations can streamline their workflows and improve code quality. Automation not only reduces manual effort but also accelerates feedback loops, enabling teams to respond quickly to changes and improve overall software quality.

Objectives of the Deployment:

The objectives of this deployment include:

- **Automated Infrastructure Provisioning:** Use Terraform to provision and configure EC2 instances for Jenkins and SonarQube automatically.
- **CI/CD Pipeline Establishment:** Create a CI/CD pipeline in Jenkins to deploy a Python project from a GitHub repository.
- **Code Quality Analysis:** Implement SonarQube to analyze the deployed Python project, identifying code smells and potential issues.

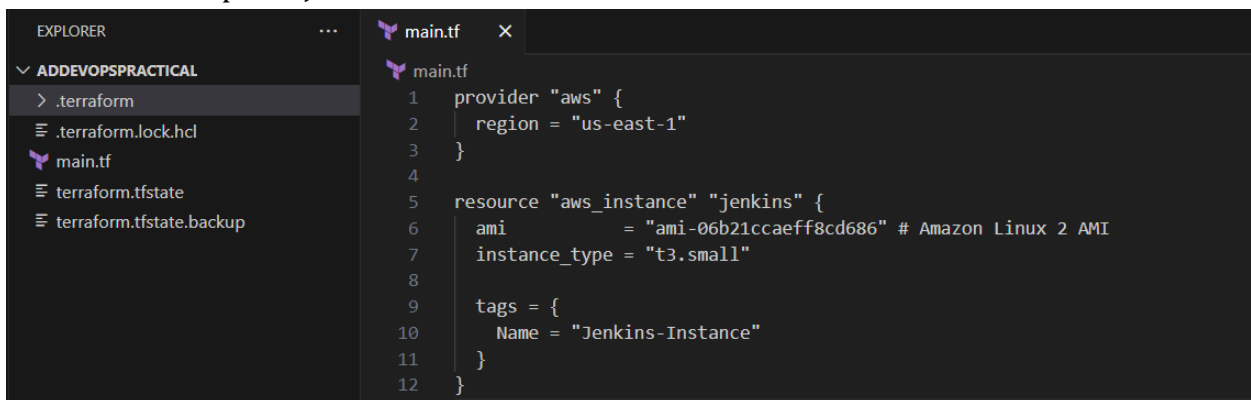
2. Step-by-Step Explanation

Step 1: Setting Up the EC2 Instances with Terraform

To set up the infrastructure, Terraform scripts were written to create two EC2 instances—one for Jenkins and one for SonarQube. The following steps outline the Terraform process:

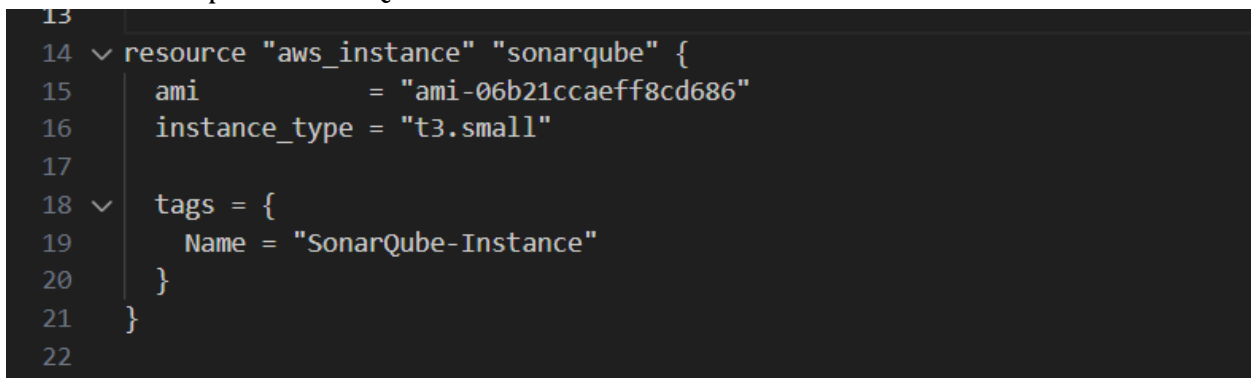
1. Install Terraform: Ensure that Terraform is installed on your local machine.
2. Create Terraform Configuration: Define the resources in a .tf file. The configuration specifies instance types, AMIs, security groups, and other necessary parameters.

Terraform Script for Jenkins Instance



```
1 provider "aws" {
2   region = "us-east-1"
3 }
4
5 resource "aws_instance" "jenkins" {
6   ami           = "ami-06b21ccaeff8cd686" # Amazon Linux 2 AMI
7   instance_type = "t3.small"
8
9   tags = {
10     Name = "Jenkins-Instance"
11   }
12 }
```

Terraform Script for SonarQube Instance



```
13
14 resource "aws_instance" "sonarqube" {
15   ami           = "ami-06b21ccaeff8cd686"
16   instance_type = "t3.small"
17
18   tags = {
19     Name = "SonarQube-Instance"
20   }
21 }
22
```

3. Apply Terraform Configuration: Run the command terraform init followed by terraform apply to provision the instances.

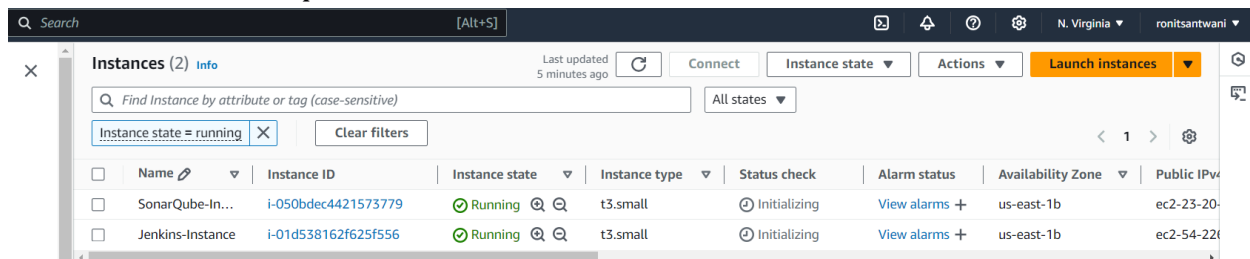
```
PS C:\AdDevopsPractical> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.72.1

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.
```

4. After above steps ,Go to AWS Instances and we can see our instances.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
SonarQube-In...	i-050bdec4421573779	Running	t3.small	Initializing	View alarms +	us-east-1b	ec2-23-20...
Jenkins-Instance	i-01d538162f625f556	Running	t3.small	Initializing	View alarms +	us-east-1b	ec2-54-22...

Step 2: Installing Jenkins and SonarQube

After provisioning the instances, Jenkins and SonarQube were installed as follows:

For Jenkins Installation:

- SSH into the Jenkins EC2 instance.



- Install Java and Jenkins using package managers.

Follow these commands and run this in your SSH in Jenkins Instance.

```
sudo yum install -y java-1.8.0-openjdk sudo wget -O /etc/yum.repos.d/jenkins.repo
```

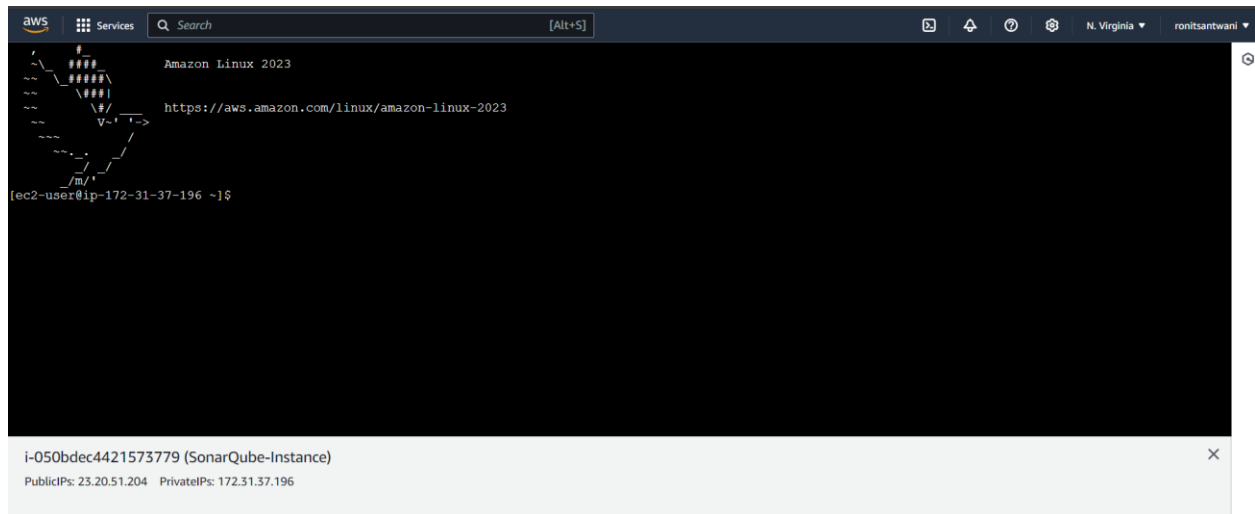
```
https://pkg.jenkins.io/redhat/jenkins.repo sudo rpm --import
```

```
https://pkg.jenkins.io/redhat/jenkins.io.key sudo yum install -y jenkins sudo systemctl
```

```
start jenkins
```

For SonarQube Installation:

- SSH into the SonarQube EC2 instance.



- Install Java and download SonarQube.

Follow these commands and run this in your SSH on SonarQube Instance.

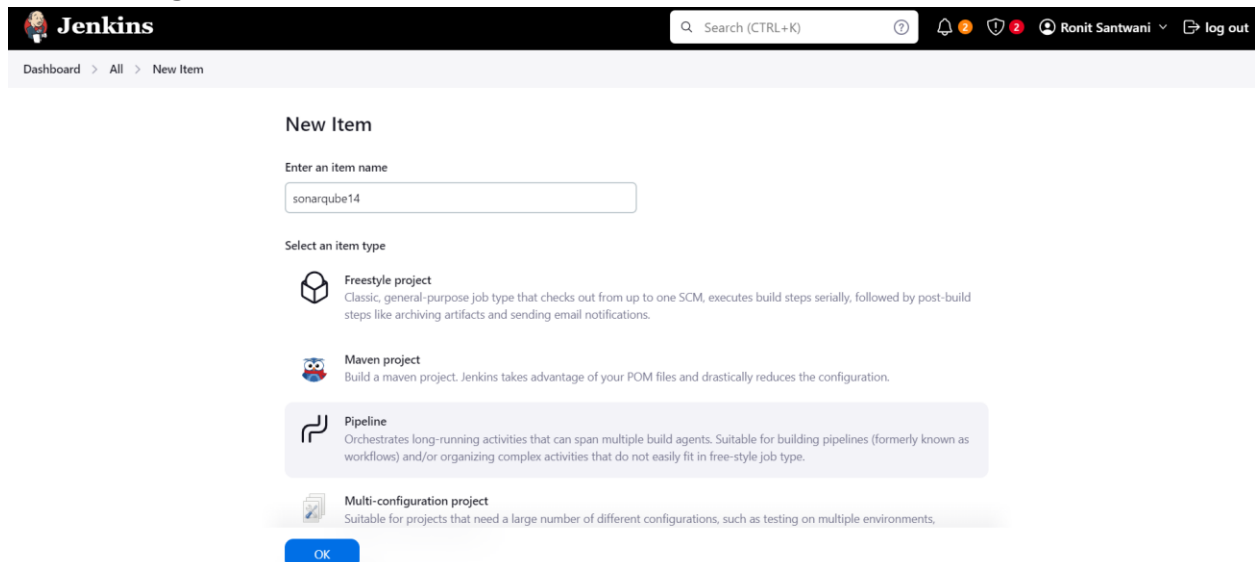
```
sudo yum install -y java-1.8.0-openjdk wget
```

```
https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.2.0.77676.zip sudo unzip sonarqube-10.2.0.77676.zip -d /opt/ cd /opt/sonarqube-10.2.0.77676/bin/linux-x86-64/ sudo ./sonar.sh start
```

Step 3: Deploying the Python Project

With Jenkins and SonarQube running, the next step was to deploy a Python project:

- Configure Jenkins Job:



- Create a new pipeline job in Jenkins.

Use the following Jenkinsfile to clone the GitHub repository and trigger SonarQube analysis.

```
pipeline {
  agent any

  stages {
    stage('Clone Repository') {
      steps {
        git branch: 'main', url: 'https://github.com/Rakshit5467/PulseAnalytics.git'
      }
    } stage('SonarQube Analysis')
    {
      steps {
        withSonarQubeEnv('sonarqube') {
          sh ''' /opt/sonar-
            scanner/bin/sonar-scanner \
              -D sonar.projectKey=YourProjectKey \
              -D sonar.login=admin \
              -D sonar.password=YourPassword \
              -D sonar.host.url=http://<SonarQube-Instance-IP>:9000
            '''
        }
      }
    }
  }
}
```

Dashboard > jenkinsonar > Configuration

Configure

- General
- Advanced Project Options**
- Pipeline

Pipeline

Definition

Pipeline script

```

1 node {
2   stage('Cloning the GitHub Repo') {
3     git branch: 'main', url: 'https://github.com/Rakshit5467/PulseAnalytics.git'
4   }
5
6   stage('SonarQube Analysis') {
7     withSonarQubeEnv('sonarqube') {
8       sh """
9         /opt/sonar-scanner/bin/sonar-scanner \
10        -D sonar.login=admin \
11        -D sonar.password=P@ssw0rd3108 \
12        -D sonar.projectKey=AdvDevOps-Casestudy \
13        -D sonar.exclusions=vendor/,resources/,/*.java \
14        -D sonar.host.url=http://34.224.85.138:9000
15      """
16    }
17  }

```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

Save Apply

- Run the Jenkins Job: Trigger the job to deploy the project, which automatically builds and initiates the SonarQube analysis.

Dashboard > jenkinsonar >

jenkinsonar

Add description

Stage View

Average stage times: 1s 32s
(Average full run time - 36s)

Stage	Cloning the GitHub Repo	SonarQube Analysis
#1 Oct 20, 14:31 No Changes	3s	32s
#2 Oct 20, 14:26 No Changes	75ms	failed
#3 Oct 20, 14:27 No Changes	244ms	failed

Permalinks

- Last build (#2), 2 min 11 sec ago
- Last failed build (#2), 2 min 11 sec ago
- Last unsuccessful build (#2), 2 min 11 sec ago
- Last completed build (#2), 2 min 11 sec ago

Build History

Filter...

Oct 20, 2024, 9:01 AM

Oct 20, 2024, 9:59 AM

Oct 20, 2024, 9:59 AM

https://github.com/Rakshit5467/PulseAnalytics.git

- See the Console Output.

The screenshot displays the Jenkins console output for a build named 'jenkinsonar #3'. The output shows the pipeline starting with a 'node' stage, cloning a repository from GitHub, and performing a 'git' checkout. The build is successful, and the console output includes the following log entries:

```

09:01:50.165 INFO 0/0 source files have been analyzed
09:01:50.165 INFO Sensor IaC Docker Sensor [iac] (done) | time=102ms
09:01:50.171 INFO ----- Run sensors on project
09:01:50.291 INFO Sensor Analysis Warnings import [csharp]
09:01:50.292 INFO Sensor Analysis Warnings import [csharp] (done) | time=1ms
09:01:50.292 INFO Sensor Zero Coverage Sensor
09:01:50.334 INFO Sensor Zero Coverage Sensor (done) | time=42ms
09:01:50.346 INFO SCM Publisher SCM provider for this project is: git
09:01:50.348 INFO SCM Publisher 20 source files to be analyzed
09:01:51.065 INFO SCM Publisher 20/20 source files have been analyzed (done) | time=716ms
09:01:51.089 INFO CPD Executor Calculating CPD for 19 files
09:01:51.278 INFO CPD Executor CPD calculation finished (done) | time=188ms
09:01:51.611 INFO Analysis report generated in 308ms, dir size=413.4 kB
09:01:51.765 INFO Analysis report compressed in 152ms, zip size=199.0 kB
09:01:52.067 INFO Analysis report uploaded in 302ms
09:01:52.073 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://34.224.85.130:9000/dashboard?id=AdvDevOps-Casestudy
09:01:52.073 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:01:52.111 INFO More about the report processing at http://34.224.85.130:9000/api/ce/task?id=A2KpKvgyw9So07ucOxw
09:01:52.111 INFO Analysis total time: 21.749 s
09:01:52.114 INFO EXECUTION SUCCESS
09:01:52.115 INFO Total time: 30.411s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

The bottom part of the screenshot shows the SonarQube dashboard with the following log entries:

```

09:01:50.165 INFO 0/0 source files have been analyzed
09:01:50.165 INFO Sensor IaC Docker Sensor [iac] (done) | time=102ms
09:01:50.171 INFO ----- Run sensors on project
09:01:50.291 INFO Sensor Analysis Warnings import [csharp]
09:01:50.292 INFO Sensor Analysis Warnings import [csharp] (done) | time=1ms
09:01:50.292 INFO Sensor Zero Coverage Sensor
09:01:50.334 INFO Sensor Zero Coverage Sensor (done) | time=42ms
09:01:50.346 INFO SCM Publisher SCM provider for this project is: git
09:01:50.348 INFO SCM Publisher 20 source files to be analyzed
09:01:51.065 INFO SCM Publisher 20/20 source files have been analyzed (done) | time=716ms
09:01:51.089 INFO CPD Executor Calculating CPD for 19 files
09:01:51.278 INFO CPD Executor CPD calculation finished (done) | time=188ms
09:01:51.611 INFO Analysis report generated in 308ms, dir size=413.4 kB
09:01:51.765 INFO Analysis report compressed in 152ms, zip size=199.0 kB
09:01:52.067 INFO Analysis report uploaded in 302ms
09:01:52.073 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://34.224.85.130:9000/dashboard?id=AdvDevOps-Casestudy
09:01:52.073 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:01:52.111 INFO More about the report processing at http://34.224.85.130:9000/api/ce/task?id=A2KpKvgyw9So07ucOxw
09:01:52.111 INFO Analysis total time: 21.749 s
09:01:52.114 INFO EXECUTION SUCCESS
09:01:52.115 INFO Total time: 30.411s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

The bottom part of the screenshot shows the SonarQube dashboard with the following log entries:

```

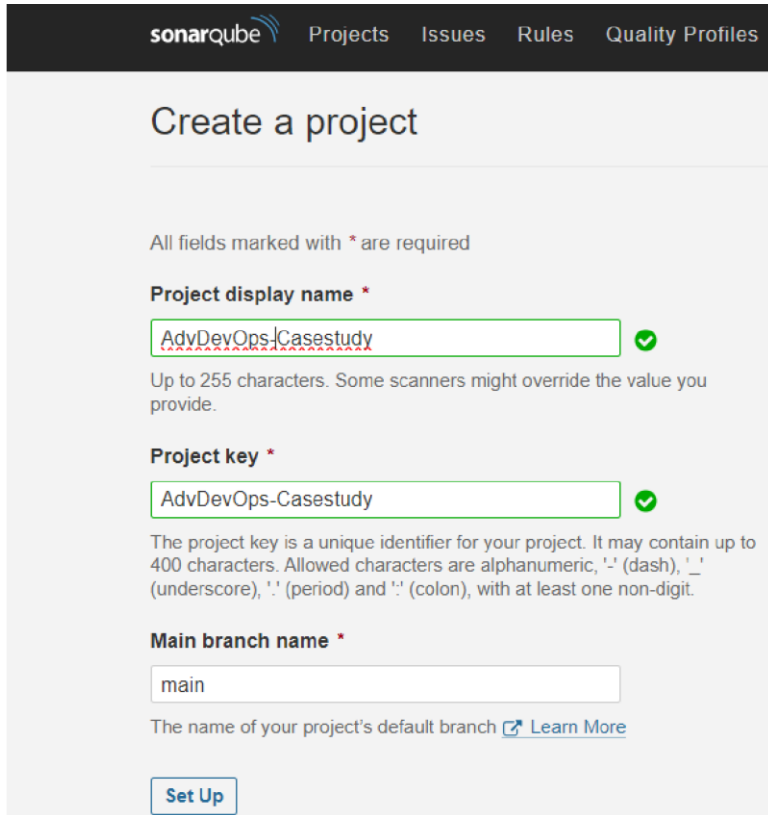
09:01:50.165 INFO 0/0 source files have been analyzed
09:01:50.165 INFO Sensor IaC Docker Sensor [iac] (done) | time=102ms
09:01:50.171 INFO ----- Run sensors on project
09:01:50.291 INFO Sensor Analysis Warnings import [csharp]
09:01:50.292 INFO Sensor Analysis Warnings import [csharp] (done) | time=1ms
09:01:50.292 INFO Sensor Zero Coverage Sensor
09:01:50.334 INFO Sensor Zero Coverage Sensor (done) | time=42ms
09:01:50.346 INFO SCM Publisher SCM provider for this project is: git
09:01:50.348 INFO SCM Publisher 20 source files to be analyzed
09:01:51.065 INFO SCM Publisher 20/20 source files have been analyzed (done) | time=716ms
09:01:51.089 INFO CPD Executor Calculating CPD for 19 files
09:01:51.278 INFO CPD Executor CPD calculation finished (done) | time=188ms
09:01:51.611 INFO Analysis report generated in 308ms, dir size=413.4 kB
09:01:51.765 INFO Analysis report compressed in 152ms, zip size=199.0 kB
09:01:52.067 INFO Analysis report uploaded in 302ms
09:01:52.073 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://34.224.85.130:9000/dashboard?id=AdvDevOps-Casestudy
09:01:52.073 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:01:52.111 INFO More about the report processing at http://34.224.85.130:9000/api/ce/task?id=A2KpKvgyw9So07ucOxw
09:01:52.111 INFO Analysis total time: 21.749 s
09:01:52.114 INFO EXECUTION SUCCESS
09:01:52.115 INFO Total time: 30.411s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

Step 4: Analyzing Code Quality in SonarQube

Once the project is deployed, SonarQube provides insights into the code quality:

- **Create a Manual Project:** In the SonarQube dashboard, create a new project and link it to the analysis performed by Jenkins.



The image shows the 'Create a project' form in SonarQube. It has a dark header with the SonarQube logo and navigation links: Projects, Issues, Rules, and Quality Profiles. The main heading is 'Create a project'. Below it, a note states 'All fields marked with * are required'. There are three required fields: 'Project display name *' with the value 'AdvDevOps.Casestudy', 'Project key *' with the value 'AdvDevOps-Casestudy', and 'Main branch name *' with the value 'main'. Each field has a green checkmark icon to its right. Below the 'Project key' field, there is explanatory text: '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.' At the bottom of the form is a 'Set Up' button.

sonarqube Projects Issues Rules Quality Profiles

Create a project

All fields marked with * are required

Project display name *

AdvDevOps.Casestudy ✓

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

Project key *

AdvDevOps-Casestudy ✓

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.

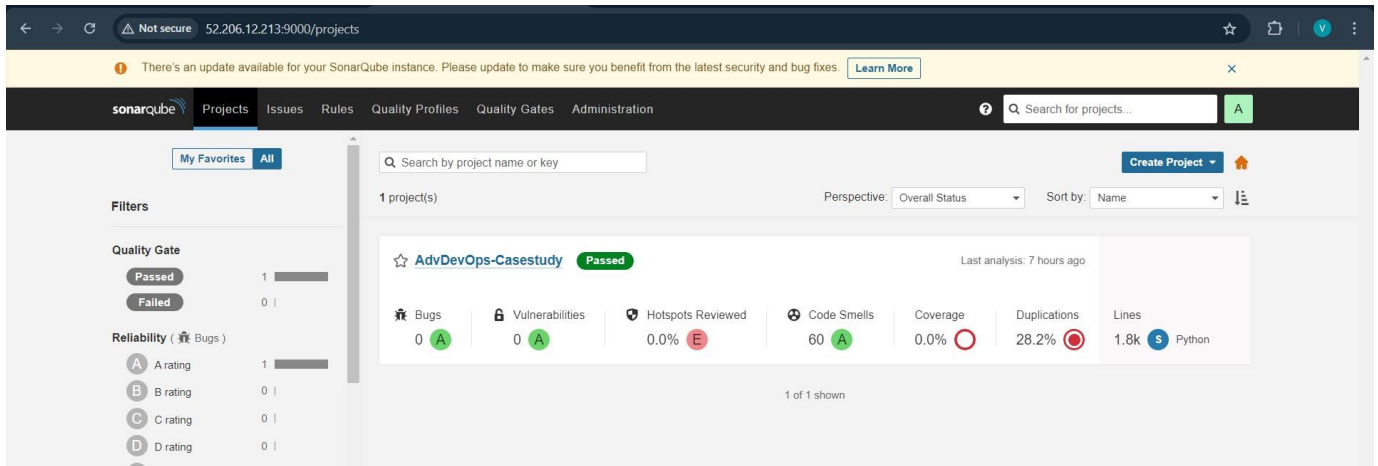
Main branch name *

main

The name of your project's default branch [Learn More](#)

[Set Up](#)

- Once the build is successfully completed, SonarQube automatically loads the project and displays the results, indicating a "Passed" status if no critical issues are found



The image shows the SonarQube project overview page for 'AdvDevOps-Casestudy'. The browser address bar shows '52.206.12.213:9000/projects'. A yellow banner at the top indicates an update is available. The navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar is present. The main content area shows the project name 'AdvDevOps-Casestudy' with a 'Passed' status and a green checkmark. Below this, a row of metrics is displayed: Bugs (0, A), Vulnerabilities (0, A), Hotspots Reviewed (0.0%, E), Code Smells (60, A), Coverage (0.0%, E), Duplications (28.2%, E), and Lines (1.8k, S Python). The left sidebar contains filters for Quality Gate (Passed: 1, Failed: 0) and Reliability (A rating: 1, B rating: 0, C rating: 0, D rating: 0). The bottom of the page indicates '1 of 1 shown'.

← → ↻ Not secure 52.206.12.213:9000/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](#)

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects...

My Favorites All

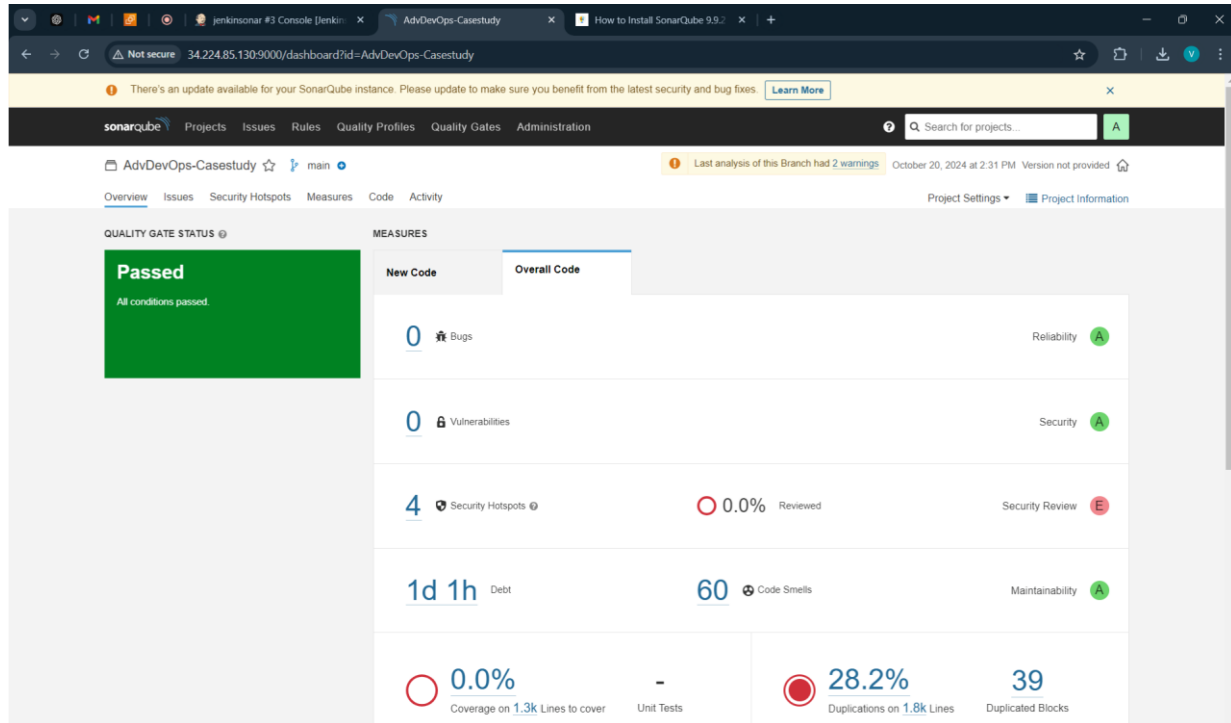
Search by project name or key

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

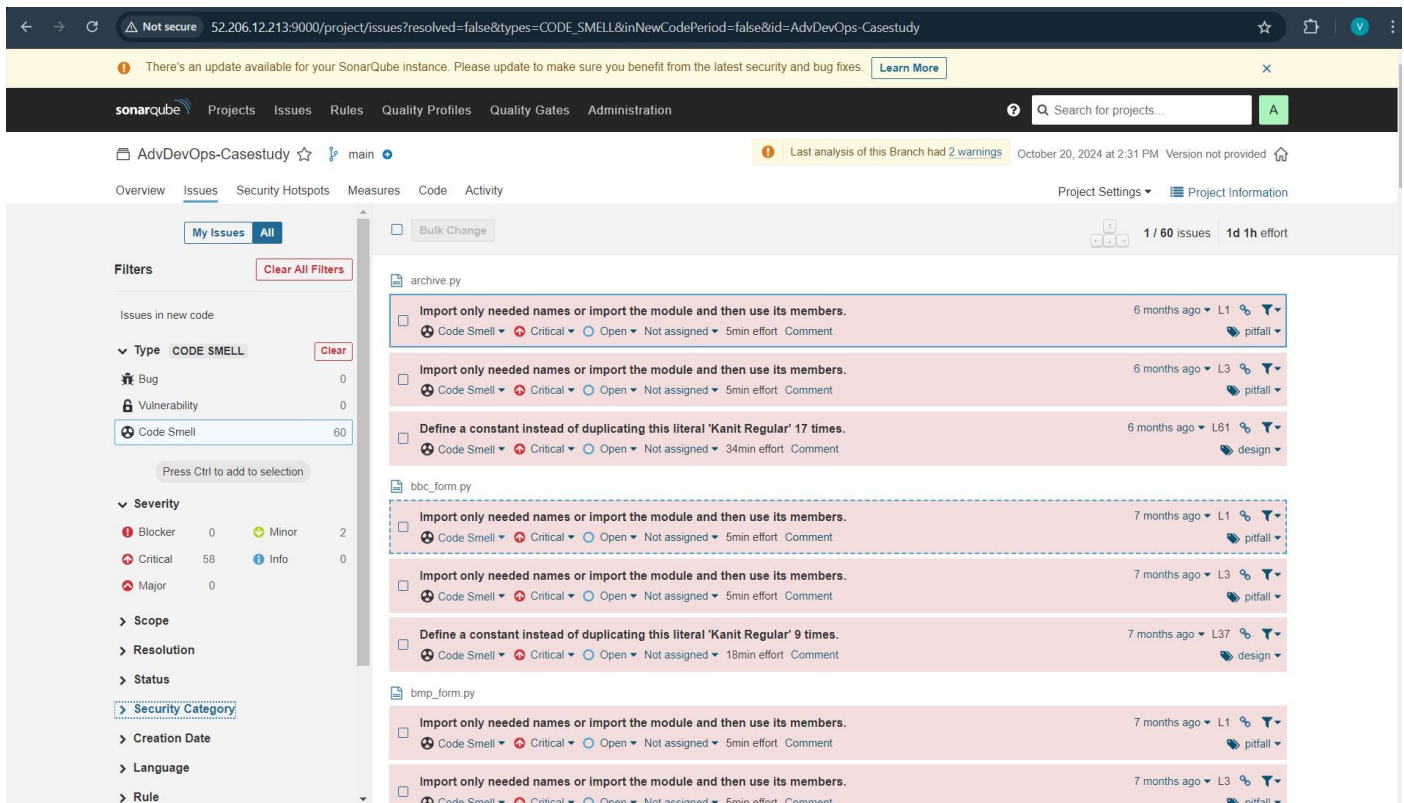
☆ AdvDevOps-Casestudy **Passed** Last analysis: 7 hours ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	0.0% E	60 A	0.0% E	28.2% E	1.8k S Python

1 of 1 shown



- **Identify Code Smells:** SonarQube identifies potential issues in the code, allowing developers to rectify them and maintain high coding standards.



3. Guidelines and Best Practices

1. Version Control Infrastructure

- Store Terraform scripts in a version control system like GitHub, GitLab, or Bitbucket to maintain a history of changes and ensure smooth collaboration among team members.

2. Secure Instances

- Use key pairs for SSH login instead of passwords to enhance security. Restrict access by only allowing trusted IP addresses to connect via SSH.
- Security Groups Configuration like
Open **port 8080 (TCP)** for Jenkins to enable access to the web interface.
Open **port 9000 (TCP)** for SonarQube to allow web-based project analysis and management.

3. Instance Type Recommendation

- Use **t3.small** or **t3.medium** instance types for Jenkins and SonarQube to ensure adequate CPU and memory, as both tools are resource-intensive. This will improve performance during large builds or code analysis operations.

4. Plugin Management

- **Jenkins:** Install essential plugins like:
Git Plugin for seamless integration with version control repositories.
Pipeline Plugin to manage CI/CD workflows effectively.
SonarQube Scanner Plugin for connecting Jenkins jobs with SonarQube for automated code analysis.
- **SonarQube:** Ensure the installation of relevant language analyzers or custom rules plugins to support the technologies used in your project. Keep plugins updated to avoid compatibility issues

3. Demonstration Preparation

1. Key Points

- **Overview of Deployment Process:**

- a. Begin with a concise overview of the entire deployment process. Discuss the Terraform scripts used to create the EC2 instances for Jenkins and SonarQube, emphasizing how Infrastructure as Code (IaC) simplifies and automates the deployment process.
- b. Highlight the specific configuration details, such as selecting appropriate instance types (e.g., t3.small or t3.medium) based on the resource requirements of Jenkins and SonarQube. Discuss the significance of setting up security groups to allow necessary traffic (e.g., TCP on ports 8080 for Jenkins and 9000 for SonarQube).

- **Integration of CI/CD and Code Quality Analysis:**

- a. Elaborate on how Jenkins facilitates Continuous Integration and Continuous Deployment (CI/CD) practices by automating the build, test, and deployment processes for software projects.
- b. Explain the role of SonarQube in this workflow, highlighting its ability to perform static code analysis, detect vulnerabilities, and identify code smells. Discuss how integrating these tools improves software quality and reduces time to market.

- **Demonstration of Functionality:**

- a. Prepare to showcase a live demonstration of the end-to-end workflow, including deploying a sample Python project from a GitHub repository via Jenkins.

After the deployment, navigate to SonarQube to demonstrate how the project is analyzed, presenting the resulting metrics and any issues identified. Emphasize the insights provided by SonarQube, such as code coverage, technical debt, and maintainability ratings.

2. Practice:

- **Rehearse the Demonstration:**

- a. Conduct multiple practice runs of the demonstration to ensure fluency and confidence. Familiarize yourself with all commands, configurations, and expected outcomes to address potential technical issues effectively
- b. Focus on explaining each step clearly, so your audience understands the significance of the actions being taken.

- **Technical Setup Check:**

- a. Prior to the demonstration, perform a thorough check of the technical setup. Ensure that the AWS EC2 instances for Jenkins and SonarQube are operational and that all necessary services are running.
- b. Verify that you can access the Jenkins UI and SonarQube dashboard seamlessly. Test the connectivity to your GitHub repository to ensure a smooth deployment process during the demonstration.

3. Questions:

- **Clarify Technical Aspects:**

- a. The specific configurations made in the Terraform scripts, including instance types, security groups, and IAM roles.

Ans: In the Terraform scripts, specific configurations were made to ensure the proper setup of the infrastructure. Appropriate EC2 instance types, such as **t2.micro**, were selected for both Jenkins and SonarQube based on resource requirements and cost considerations. Security groups were configured to allow inbound traffic, including port 8080 for Jenkins, port 9090 for SonarQube, and port 22 for SSH access, ensuring controlled connectivity. Additionally, IAM roles were assigned to the instances with the necessary permissions, enabling access to AWS services such as S3 and CloudWatch for logging or other operational needs. These configurations ensured a secure and functional deployment environment.

- b. The reasoning behind the choice of plugins installed in Jenkins (e.g., Git server, Pipeline, SonarQube Scanner) and how they enhance the CI/CD process.

Ans: The choice of plugins installed in Jenkins was driven by the need to streamline and enhance the CI/CD process. The **Git plugin** was installed to enable seamless integration with Git repositories, allowing Jenkins to pull code automatically whenever changes are committed. This ensures version control and facilitates automated builds. The **Pipeline plugin** was chosen to define and orchestrate CI/CD workflows as code, providing flexibility to create complex, multi-step pipelines for building, testing, and deploying applications. Additionally, the **SonarQube Scanner plugin** was installed to integrate Jenkins with SonarQube, automating the process of code quality analysis. This plugin ensures that code is continuously analyzed for bugs, vulnerabilities, and code smells during the build process, improving software quality.

- c. The steps taken to analyze the Python project in SonarQube, including any configurations made to the SonarQube project settings. Ans: To analyze the Python project in SonarQube, a new project was created with a unique key, and Python was set as the primary language. An authentication token was generated for secure integration with Jenkins. In Jenkins, the SonarQube Scanner plugin was configured, and analysis parameters like the project key and source directory were added to the pipeline. After triggering the build, the code was scanned, and the results, including code smells and vulnerabilities, were displayed on the SonarQube dashboard.

4.Conclusion

The deployment of Jenkins and SonarQube on AWS EC2 instances using Terraform exemplifies the power of automation in modern software development and DevOps practices. Automating infrastructure setup with Terraform ensures consistency, reduces the chances of manual errors, and accelerates deployment processes. By combining Jenkins for Continuous Integration/Continuous Delivery (CI/CD) with SonarQube for continuous code quality analysis, teams can streamline their development pipelines, ensuring that every code change is thoroughly tested and evaluated before release. This seamless integration enables early detection of code smells, bugs, and vulnerabilities, contributing to higher software quality and more maintainable projects over time.

This case study demonstrates how leveraging Infrastructure as Code (IaC) tools like Terraform, in conjunction with modern development tools, offers a robust solution for managing complex software lifecycles. It highlights the importance of continuous feedback loops in improving software quality while accelerating the release cycle, ultimately helping organizations achieve faster time to market, greater agility, and sustainable growth. Adopting such practices is essential for organizations aiming to remain competitive in today's dynamic software development landscape.