

AdvanceDevOps Practical Case_Study:

NITISH BHOSLE

D15A – 04

4. Continuous Integration with Static Code Analysis

Concepts Used: Jenkins, SonarQube, and AWS Cloud9(EC2 Instance).

Problem Statement: "Set up a Jenkins pipeline using AWS Cloud9 IDE to perform a static analysis of a Java/Python application. Integrate SonarQube for code quality checks."

Tasks:

- Install Jenkins and set up a basic pipeline.
- Configure SonarQube as part of the pipeline for static code analysis.
- Run the pipeline and generate a report for code quality issues.

1. Introduction

Case Study Overview

This case study demonstrates the setup of a Continuous Integration (CI) pipeline using Jenkins, integrated with SonarQube for static code analysis on AWS EC2 instance. The objective is to automate the quality checks of a Java/Python application and ensure that code quality issues are identified early in the development process. Jenkins, a popular CI/CD tool, automates code builds, testing, and deployment, while SonarQube provides comprehensive code quality and security analysis.

Key Feature and Application

The key feature of this project is the integration of static code analysis into the CI pipeline. This allows developers to ensure code quality and adherence to best practices without manual intervention. It ensures early detection of bugs, code smells, and potential security vulnerabilities, making the development process smoother and more reliable.

2. Step-by-Step Explanation

Step 1: Create EC2 Instances for Jenkins and SonarQube

1. Login to AWS Console:

- Navigate to the AWS Management Console and log in.

2. Launch EC2 Instances:

- Go to the EC2 Dashboard and click Launch Instance.

- Choose Ubuntu Server 22.04 LTS as your AMI.

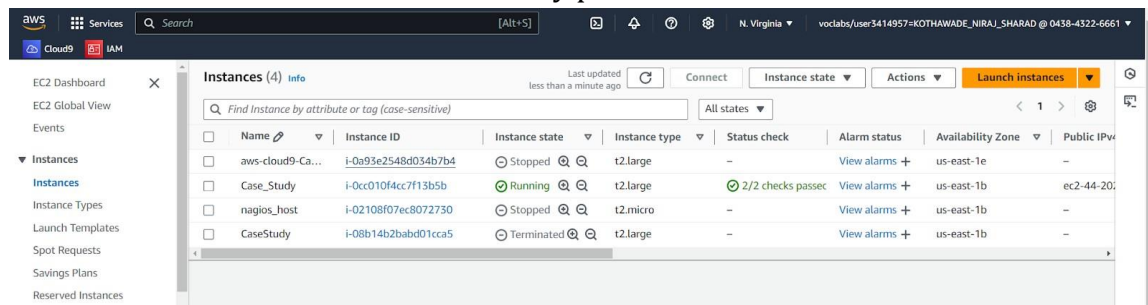
- Select an instance type (t2.large for Jenkins and SonarQube).

- Configure instance details (ensure Auto-assign Public IP).

- Add storage (default 8 GB, adjust for larger projects).

- Configure security group (allow ports 22, 8080 for Jenkins, 9000 for SonarQube).

- Launch instances and download the key pair for SSH access.



Step 2: Install Jenkins on EC2

1. Connect to EC2 via SSH:

- Use the key pair to SSH into the Jenkins EC2 instance.

2. Update the system:

```
sudo apt update
sudo apt upgrade
```

3. Install Java 11:

```
sudo apt install openjdk-11-jdk
```

4. Add Jenkins Repository:

- Add Jenkins GPG key and repository.

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources
```

5. Install Jenkins:

```
sudo apt update
sudo apt install jenkins
```

6. Start Jenkins:

```
sudo systemctl start jenkins
sudo systemctl status jenkins
```

```
root@ip-172-31-89-83:~# systemctl status jenkins
jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Wed 2024-10-23 08:55:07 UTC; 2min 21s ago
     Main PID: 10787 (java)
       Tasks: 44 (limit: 9507)
      Memory: 549.1M (peak: 557.4M)
         CPU: 13.588s
    CGroup: /system.slice/jenkins.service
            └─10787 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Oct 23 08:55:05 ip-172-31-89-83 jenkins[10787]: 8a833890ecbc406382bac565582d5842
Oct 23 08:55:05 ip-172-31-89-83 jenkins[10787]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Oct 23 08:55:05 ip-172-31-89-83 jenkins[10787]: *****
Oct 23 08:55:05 ip-172-31-89-83 jenkins[10787]: *****
Oct 23 08:55:05 ip-172-31-89-83 jenkins[10787]: *****
Oct 23 08:55:07 ip-172-31-89-83 jenkins[10787]: 2024-10-23 08:55:07.714+0000 [id=40] INFO jenkins.InitReactorRunner$1:onAttained: Completed initial
Oct 23 08:55:07 ip-172-31-89-83 jenkins[10787]: 2024-10-23 08:55:07.743+0000 [id=31] INFO hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up a
Oct 23 08:55:07 ip-172-31-89-83 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Oct 23 08:55:07 ip-172-31-89-83 jenkins[10787]: 2024-10-23 08:55:07.815+0000 [id=56] INFO h.m.DownloadService$Downloadable#load: Obtained the updat
Oct 23 08:55:07 ip-172-31-89-83 jenkins[10787]: 2024-10-23 08:55:07.815+0000 [id=56] INFO hudson.util.Retrier#start: Performed the action check upd
lines 1-20/20 (END)
```

7. Access Jenkins:

- Navigate to in your browser :

```
http://<EC2-Jenkins-Public-IP>:8080
```

- Follow on-screen instructions and use the initialAdminPassword to unlock Jenkins.

```
root@ip-172-31-89-83:~# cat /var/lib/jenkins/secrets/initialAdminPassword
8a833890ecbc406382bac565582d5842
```

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

Step 3: Install SonarQube on EC2

1. Connect to EC2 via SSH:

- Use the key pair to SSH into the SonarQube EC2 instance.

```
ssh -i /path/to/key.pem ubuntu@<EC2-SonarQube-Public-IP>
```

2. Update the system:

```
sudo apt update
sudo apt upgrade
```

3. Install prerequisites (unzip, wget):

```
sudo apt install unzip wget
```

4. Download and install SonarQube:

```
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
unzip sonarqube-9.9.0.65466.zip
sudo mv sonarqube-9.9.0.65466 /opt/sonarqube
```

5. Start SonarQube:

```
cd /opt/sonarqube/bin/linux-x86-64
./sonar.sh start
```

6. Access SonarQube:

- Navigate to in your browser :

```
http://<EC2-SonarQube-Public-IP>:9000
```

- Login using default credentials (admin/admin) and complete the setup.

7. Generate a token for Jenkins:

The screenshot shows the SonarQube Administration interface. At the top, there's a navigation bar with links to Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. Below this, the 'Administration' section is active, displaying 'Tokens of Administrator'. This section has a 'Generate Tokens' form with fields for 'Name' (with a placeholder 'Enter Token Name') and 'Expires in' (set to '30 days'), and a 'Generate' button. Below the form is a table of existing tokens:

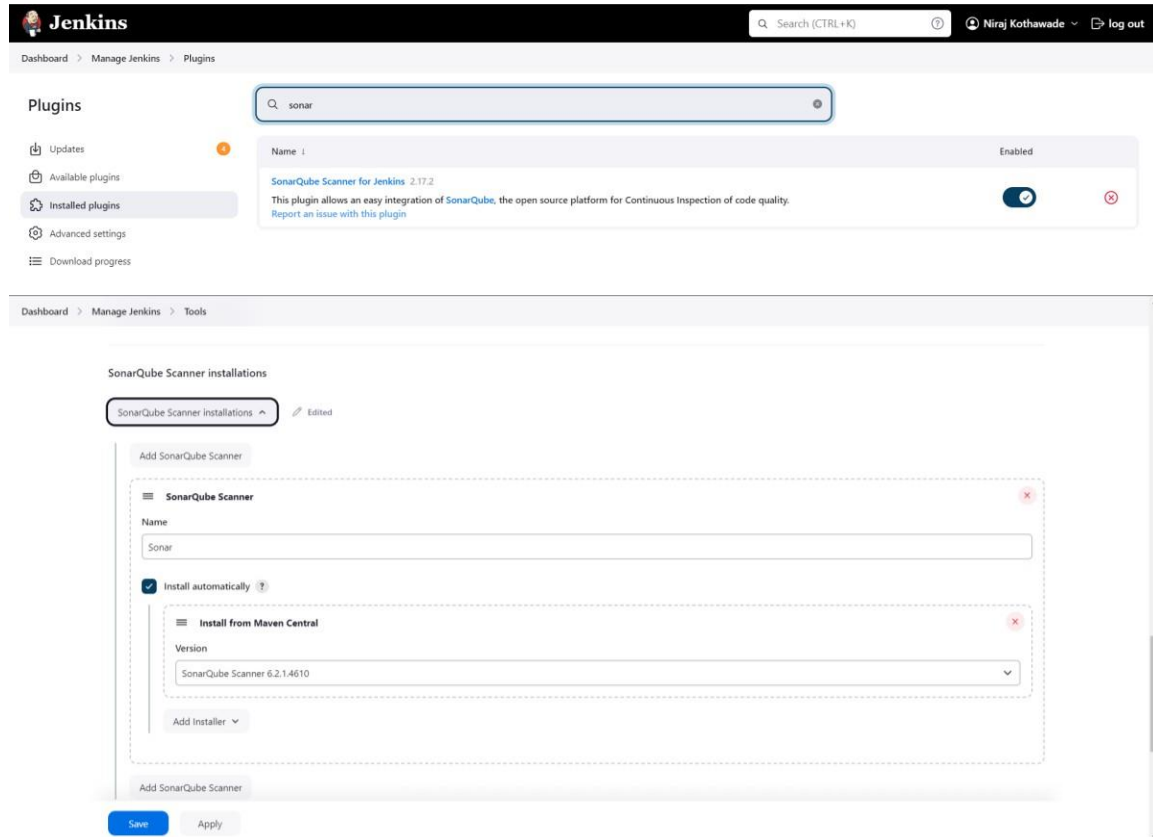
Name	Type	Project	Last use	Created	Expiration	
test	User		Never	October 23, 2024	—	Revoke

At the bottom right of the table is a 'Done' link. The footer of the page contains a warning: 'Embedded database should be used for evaluation purposes only' and links to documentation, plugins, and the web API.

Step 4: Configure Jenkins to Integrate with SonarQube

1. Install SonarQube Plugin in Jenkins:

- Go to Manage Jenkins → Manage Plugins.
- Install SonarQube Scanner plugin.



2. Configure SonarQube in Jenkins:

- Go to Manage Jenkins → Configure System.
- Add SonarQube server details (IP, token) and configure SonarQube Scanner under Global Tool Configuration.

Dashboard > Manage Jenkins > System >

if checked, job administrators will be able to inject a sonarQube server configuration as environment variables in the build.

☒ Environment variables

SonarQube installations

List of SonarQube installations

Name

sonar

Server URL

Default is http://localhost:9000

http://34.230.74.182:9000/

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

sonar-token

+ Add

Advanced ▾

Add SonarQube

Save Apply

Step 5: Set Up a Jenkins Pipeline for Static Code Analysis

1. Create a new Jenkins Pipeline job.

Jenkins

Search (CTRL+K)

Niraj Kothawade log out

Dashboard >

+ New Item

Build History

Manage Jenkins

My Views

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

All +

S	W	Name	Last Success	Last Failure	Last Duration
🔄	☀️	Case_Study	N/A	N/A	N/A

Icon: S M L

REST API Jenkins 2.462.3

2. In the pipeline script, add the following basic pipeline code:

```

pipeline {
    agent any

    stages {
        stage('SCM Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
    }
}

```

```

    }

}

stage('SonarQube Analysis') {

    steps {

        withSonarQubeEnv('SonarQube') {

            sh 'mvn clean verify sonar:sonar'

        }

    }

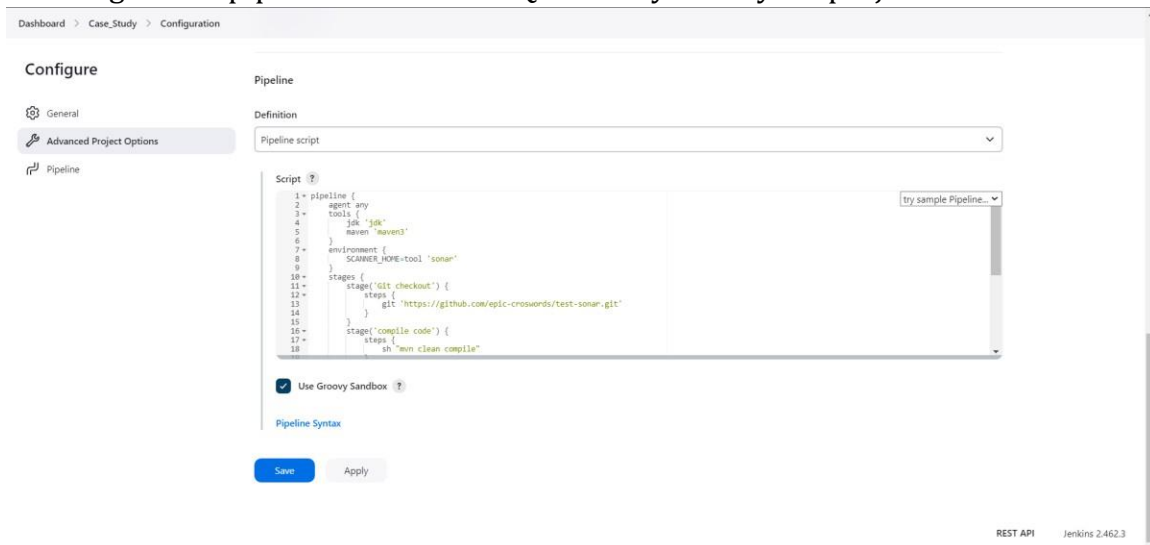
}

}

}

```

3. Configure the pipeline to run SonarQube analysis for your project.



Dashboard > Case_Study > Configuration

Configure

- General
- Advanced Project Options**
- Pipeline

Pipeline

Definition: Pipeline script

Script ?

```

1= pipeline {
2=   agent any
3=   tools {
4=     jdk 'jdk'
5=     maven 'maven3'
6=   }
7=   environment {
8=     SCANNER_HOME=tool 'sonar'
9=   }
10=  stages {
11=    stage('git checkout') {
12=      steps {
13=        git 'https://github.com/epic-crosswords/test-sonar.git'
14=      }
15=    }
16=    stage('compile code') {
17=      steps {
18=        sh 'mvn clean compile'

```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

[Save](#) [Apply](#)

REST API Jenkins 2.462.3

Step 6: Run the Jenkins Pipeline

1. Trigger the pipeline manually by clicking Build Now, or set it to run automatically after code push.
2. Monitor pipeline execution via Jenkins console output.

The screenshot displays the Jenkins interface for a pipeline named 'Case_Study'. The top section shows the console output, which includes logs for analysis report generation, compression, upload, and successful execution. The bottom section shows the 'Stage View' of the pipeline, which consists of five stages: Declarative: Tool Install, Git checkout, compile code, code test, and sonar analysis. The 'Stage View' also includes a table showing the average stage times and a 'Build History' section.

Console Output:

```
09:52:09.184 INFO Analysis report generated in 88ms, dir size=130.2 kB
09:52:09.204 INFO Analysis report compressed in 20ms, zip size=23.1 kB
09:52:09.234 INFO Analysis report uploaded in 29ms
09:52:09.237 INFO ANALYSIS SUCCESSFUL, you can find the results at: http://44.202.133.68:9000/dashboard?id=sonar-qube-analys
09:52:09.237 INFO Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
09:52:09.237 INFO More about the report processing at http://44.202.133.68:9000/api/ce/task?id=AZK4ynYHj60CLPvqqlr_
09:52:09.259 INFO Analysis total time: 6.459 s
09:52:09.259 INFO EXECUTION SUCCESS
09:52:09.260 INFO Total time: 7.948s
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Stage View:

	Declarative: Tool Install	Git checkout	compile code	code test	sonar analysis
Average stage times: (Average full run time: ~23s)	252ms	621ms	4s	8s	8s
Oct 23 15:22 No Changes	252ms	621ms	4s	8s	8s

Build History:

Build	Time
#3	Oct 23, 2024, 9:52 AM

Step 7: Analyze SonarQube Report

1. Access SonarQube dashboard at <http://<EC2-SonarQube-Public-IP>:9000>.
2. View code quality reports (bugs, code smells, vulnerabilities) and address issues accordingly.

The screenshot displays the SonarQube dashboard for a project named 'Sonarqube_CS'. The project status is 'Passed', and the last analysis was performed 17 minutes ago. The dashboard provides a comprehensive overview of code quality metrics, including Bugs, Vulnerabilities, Hotspots Reviewed, Code Smells, Coverage, Duplications, and Lines of code. The Quality Gate status is 'Passed', indicating that all conditions were met. The Measures section shows detailed metrics for New Code, Overall Code, and various code quality dimensions like Reliability, Security, Security Review, and Maintainability.

Filters

- Quality Gate**
 - Passed: 1
 - Failed: 0
- Reliability (Bugs)**
 - A: 1
 - B: 0
 - C: 0
 - D: 0
 - E: 0
- Security (Vulnerabilities)**
 - A: 1
 - B: 0
 - C: 0
 - D: 0
 - E: 0
- Security Review (Security Hotspots)**
 - A: ≥ 80%: 1
 - B: 70% - 80%: 0
 - C: 50% - 70%: 0
 - D: 30% - 50%: 0

1 projects Perspective: Overall Status Sort by: Name

Sonarqube_CS Passed Last analysis: 17 minutes ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	— A	0 A	0.0% F	0.0% F	8 XS Java

1 of 1 shown

SonarQube™ technology is powered by SonarSource SA
Community Edition - Version 9.0.1 (build 46107) - LGPL v3 - Community - Documentation - Plugins - Web API - About

Sonarqube_CS master Last analysis had 1 warning October 20, 2024 at 5:23 PM Version not provided

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

QUALITY GATE STATUS

Passed
All conditions passed.

MEASURES

New Code	Overall Code
Since October 20, 2024 Started 14 minutes ago	
0 New Bugs	Reliability A
0 New Vulnerabilities	Security A
0 New Security Hotspots	Security Review A
0 Added Debt	0 New Code Smells Maintainability A

UNIQUE FEATURES

1. Automated Slack Notifications:

- Configure Jenkins to send notifications to a Slack channel whenever the pipeline completes or issues are detected by SonarQube.
- Use the Slack Notification plugin in Jenkins.

2. GitHub Webhook Integration:

- Set up a GitHub webhook so that any new code pushed to the repository automatically triggers the Jenkins pipeline.

3. SonarQube Quality Gate Enforcement:

- Set up quality gates in SonarQube to automatically fail the Jenkins build if the code doesn't meet specific quality standards (e.g., a certain number of bugs, coverage percentage, or vulnerability issues).

4. Real-time Code Metrics Dashboard:

- Configure a Grafana Dashboard with SonarQube metrics to visualize real-time code quality data. This could include trend graphs of code coverage, technical debt, or issue count.