# **Jenkins Task -03 (Task 1)**

**Task 1  
Create one Jenkins job using the below code and create three stages:**

* **stage1: Git clone to download the source code.**
* **stage2: Sonarqube Integration to check the quality of code.**
* **stage3: Slack Integration to send the alerts to slack.**

**URL:**[**https://github.com/betawins/VProfile-1.git**](https://github.com/betawins/VProfile-1.git)

**Overall Task Title**

**Jenkins CI Pipeline with Git, SonarQube Code Quality Analysis, and Slack Notifications**

**Objective**

To design and implement a Jenkins job that automates the CI process by:

* Cloning source code from a GitHub repository
* Performing static code analysis using SonarQube to ensure code quality
* Sending real-time build and quality status notifications to a Slack channel

This task demonstrates an end-to-end Continuous Integration workflow.

**Prerequisites**

* Jenkins server installed and running
* Java (JDK 11/17) installed on Jenkins and SonarQube servers
* Git installed on Jenkins server
* SonarQube server installed and accessible on port 9000
* PostgreSQL database configured for SonarQube
* SonarQube Scanner plugin installed in Jenkins
* SonarQube authentication token generated
* Slack workspace with Jenkins CI app/integration enabled
* Slack Notification plugin installed in Jenkins
* Internet access from Jenkins server
* GitHub repository access
  + Repository URL:  
    https://github.com/betawins/VProfile-1.git

**Stage:1**

* **stage1: Git clone to download the source code.**

**Steps Performed**

* Opened Jenkins dashboard and clicked New Item.
* Entered the job name V-profile.
* Selected Freestyle project as the job type.
* Clicked OK to create the job.
* Navigated to Source Code Management section.
* Selected Git as the SCM option.
* Entered the Git repository URL:  
  https://github.com/betawins/VProfile-1.git
* Left Credentials as none (public repository).
* Set Branch Specifier to \*/master.
* Clicked Save to store the configuration.
* Clicked Build Now to trigger the job.
* Opened Console Output to verify execution.
* Confirmed that Jenkins successfully cloned the repository into the workspace and the build completed with SUCCESS.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a chat

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**stage2: SonarQube Integration to check the quality of code.**

**SonarQube Integration (Code Quality Check)**

**1. Launch EC2 Instance for SonarQube**

* Open AWS EC2 → Launch Instance
* Set Instance Name as sonarqube
* Select Amazon Linux 2023 AMI
* Choose instance type (minimum t3.medium / t3.large recommended)
* Configure storage (minimum 20 GB)
* Select or create a Security Group
* Launch the instance

**2. Configure Security Group for SonarQube**

* Edit inbound rules of the security group
* Add the following rules:
  + SSH → Port 22 → Source 0.0.0.0/0
  + HTTP → Port 80 → Source 0.0.0.0/0
  + Custom TCP → Port 9000 → Source 0.0.0.0/0 (SonarQube UI)
  + PostgreSQL → Port 5432 → Source 0.0.0.0/0 (or restricted to same SG)
* Save rules

**3. Connect to EC2 Instance**

* SSH into the instance using key pair
* Switch to root user

**4. Install Java (Required for SonarQube)**

* Update system packages
* Install Amazon Corretto Java 17
* Verify Java installation using java -version

**5. Install PostgreSQL Database**

* Install PostgreSQL server packages
* Initialize PostgreSQL database
* Enable and start PostgreSQL service
* Verify PostgreSQL service is running

**6. Configure PostgreSQL for SonarQube**

* Switch to postgres user
* Create database:
  + sonarqube
* Create database user:
  + Username: sonar
  + Password: sonar@123
* Grant all privileges on sonarqube database to sonar
* Set database owner to sonar
* Exit PostgreSQL shell

**7. Download and Extract SonarQube**

* Navigate to /opt
* Download SonarQube 6.6 zip file
* Unzip the package
* Rename directory from sonarqube-6.6 to sonar
* Verify SonarQube directory structure (bin, conf, data, logs, etc.)

**8. Configure SonarQube Database Settings**

* Edit file: /opt/sonar/conf/sonar.properties
* Configure PostgreSQL settings:
  + sonar.jdbc.username=sonar
  + sonar.jdbc.password=sonar@123
  + sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
* Configure web settings:
  + sonar.web.host=0.0.0.0
  + sonar.web.port=9000
* Save and exit configuration file

**9. Create Sonar User and Permissions**

* Create a dedicated Linux user sonar
* Change ownership of /opt/sonar to sonar user
* Switch to sonar user

**10. Start SonarQube Server**

* Navigate to /opt/sonar/bin/linux-x86-64
* Start SonarQube using sonar.sh start
* Check status using sonar.sh status
* Verify SonarQube is listening on port 9000
* Check logs under /opt/sonar/logs if needed

**11. Access SonarQube Web UI**

* Open browser
* Access SonarQube using:
  + http://<EC2-Public-IP>:9000
* Login with default credentials:
  + Username: admin
  + Password: admin
* Change admin password when prompted

**12. Create SonarQube Project**

* Go to Projects → Create Project
* Choose Create a local project
* Enter:
  + Project Key (e.g., Sabear)
  + Project Name
* Complete project creation

**13. Install SonarQube Plugins in Jenkins**

* Open Jenkins → Manage Jenkins → Plugins
* Search for:
  + SonarQube Scanner
  + Sonar Quality Gates
* Install required plugins
* Restart Jenkins if required

**14. Generate SonarQube Authentication Token**

* In SonarQube UI, go to Administrator → Security
* Generate a Global Analysis Token
* Copy the token (shown only once)

**15. Add SonarQube Token to Jenkins Credentials**

* Open Jenkins → Manage Jenkins → Credentials
* Add new credential:
  + Kind: Secret text
  + Secret: *SonarQube token*
  + ID: sonar-token
* Save credentials

**16. Configure SonarQube Server in Jenkins**

* Go to Manage Jenkins → Configure System
* Under SonarQube installations:
  + Name: sonarqube-server
  + Server URL: http://<SonarQube-Public-IP>:9000
  + Authentication token: sonar-token
* Save configuration

**17. Configure SonarQube Scanner Tool**

* Go to Manage Jenkins → Global Tool Configuration
* Add SonarQube Scanner
* Set name (e.g., sonarqube)
* Enable Install automatically
* Select scanner version
* Save configuration

**18. Configure Jenkins Job for SonarQube Analysis**

* Open Jenkins job (V-profile)
* Go to Configure
* Add Build Step → Execute SonarQube Scanner
* Add analysis properties:
  + sonar.projectKey
  + sonar.projectName
  + sonar.sources
  + sonar.java.binaries
  + sonar.junit.reportsPath
  + sonar.jacoco.reportPath
* Save job configuration

**19. Run Jenkins Job**

* Click Build Now
* Monitor Console Output
* Verify SonarQube scanner execution completes successfully

**20. Verify Results in SonarQube**

* Open SonarQube UI
* Navigate to the project (Sabear)
* Verify:
  + Code analysis completed
  + Metrics displayed (Bugs, Vulnerabilities, Code Smells)
  + Quality Gate status: Passed

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

A computer screen with text and numbers

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A computer screen with a black background

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer error

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A blurry image of a city

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

A black screen with white text

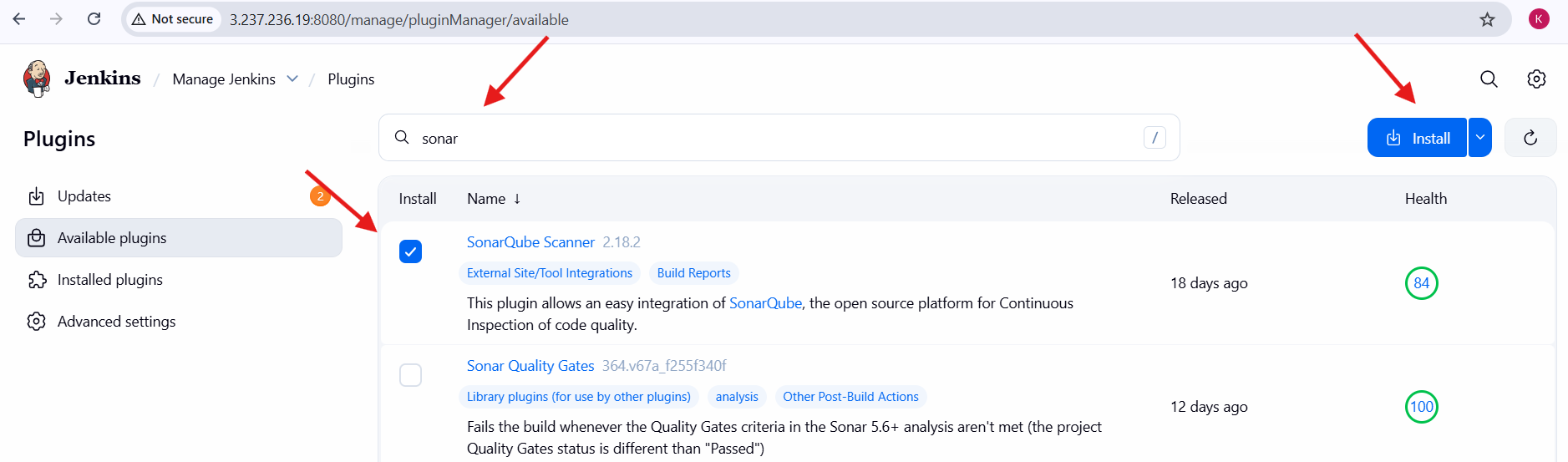
AI-generated content may be incorrect.

A black background with text on it

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Stage3: Slack Integration to send the alerts to slack.**

**Step-by-Step Implementation**

**Step 1: Verify Slack Workspace and Channel**

* Logged in to the Slack workspace TechieHorizon
* Identified and used the Slack channel:
  + #jenkins-integration

**Step 2: Add Jenkins CI App to Slack**

* Opened Slack → Apps
* Searched for Jenkins CI
* Opened the Jenkins CI app
* Clicked Add Jenkins CI Integration
* Selected channel #jenkins-integration
* Successfully added Jenkins CI integration to the channel

**Step 3: Generate Slack Integration Token**

* Opened Jenkins CI integration configuration page in Slack
* Copied the Integration Token generated by Slack
* Token is used as the authentication key for Jenkins → Slack communication

**Step 4: Install Slack Notification Plugin in Jenkins**

* Navigated to Jenkins Dashboard
* Opened Manage Jenkins → Manage Plugins
* Searched for Slack Notification
* Installed the Slack Notification Plugin
* Restarted Jenkins if required

**Step 5: Add Slack Token to Jenkins Credentials**

* Navigated to Manage Jenkins → Credentials
* Added new credentials:
  + Kind: Secret Text
  + Secret: Slack Integration Token (copied from Slack)
  + ID: slack
  + Description: slack
* Saved the credentials successfully

**Step 6: Configure Global Slack Settings in Jenkins**

* Opened Manage Jenkins → Configure System
* Navigated to Global Slack Notifier Settings
* Configured the following:
  + Team Subdomain: techiehorizon
  + Integration Token Credential ID: slack
  + Channel / Slack ID: #jenkins-integration
  + Enabled Custom Slack App Bot User
* Clicked Test Connection
* Connection verified successfully
* Saved configuration

**Step 7: Configure Slack Notifications in Jenkins Job**

* Opened Jenkins job V-profile
* Navigated to Configure
* Enabled Slack Notifications in post-build actions
* Configured notifications for:
  + Build Started
  + Build Success
* Saved job configuration

**Step 8: Trigger Jenkins Build**

* Manually triggered a Jenkins build
* Observed build execution completion

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Overall Task Validation steps**

* Jenkins job successfully clones the GitHub repository
* SonarQube analysis runs without errors in Jenkins console output
* Project appears in SonarQube dashboard with metrics (Bugs, Vulnerabilities, Code Smells)
* Quality Gate status shows **Passed**
* Slack channel receives notifications for:
  + Build started
  + Build success/failure
  + SonarQube analysis status

**Overall Task Errors Faced**

* **mvn: command not found**
  + Occurred due to Maven not being installed; resolved by using SonarQube Scanner directly
* **Could not find 'java' executable in JAVA\_HOME or PATH**
  + Resolved by correctly configuring JDK path in Jenkins Global Tool Configuration
* **SonarQube authentication failures**
  + Fixed by generating a new SonarQube token and adding it as a Jenkins secret text credential
* **Slack invalid\_auth error**
  + Resolved by using the correct Slack integration token and workspace subdomain
* **PostgreSQL authentication issues**
  + Resolved by updating pg\_hba.conf and restarting PostgreSQL service

**Overall Task Conclusion:**

This task successfully implemented a complete Jenkins CI pipeline with:

* Automated source code checkout from GitHub
* Static code quality analysis using SonarQube
* Real-time Slack notifications for build and analysis status

The pipeline improves code reliability, enforces quality gates, and enhances team communication by providing immediate feedback, aligning with CI/CD best practices.