# JENKINS – 03 TASK

**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>

**Title**

**End-to-End CI Integration using Jenkins, SonarQube, and Slack on Amazon EC2**

**Objective**

* To set up a complete Continuous Integration (CI) workflow
* To perform automated source code checkout and build using Jenkins
* To analyze code quality and enforce Quality Gates using SonarQube
* To send real-time build notifications (start, failure, success) to Slack
* To validate an end-to-end CI pipeline with visibility and alerts

**Prerequisites**

**Infrastructure**

* AWS EC2 Linux instance with public IP
* Security Group allowing:
  + Port 22 (SSH)
  + Port 9000 (SonarQube)
  + Port 5432 (PostgreSQL)

**Software**

* Java (Amazon Corretto 17)
* PostgreSQL Server
* Jenkins installed and running
* SonarQube installed and configured
* SonarQube Scanner CLI installed on Jenkins node

**Accounts & Access**

* GitHub repository (public)
* Slack workspace with permission to add apps
* Jenkins admin access
* SonarQube admin access

**Jenkins Plugins**

* SonarQube Scanner Plugin
* Slack Notification Plugin

**Errors Faced and How They Were Fixed**

**SonarQube Permission Denied Error**

**Error Observed**

Permission denied: SonarQube.pid

Permission denied: logs/sonar.log

**Root Cause**  
SonarQube was started using root user instead of a dedicated sonar user.

**Fix Applied**

* Created a sonar user
* Changed ownership of SonarQube directory

useradd sonar

chown -R sonar:sonar /opt/sonar

**SonarQube Web UI Not Accessible**

**Error Observed**

ERR\_CONNECTION\_REFUSED

**Root Cause**

* SonarQube service not running
* Port 9000 not listening or blocked

**Fix Applied**

* Verified SonarQube service status
* Opened port 9000 in EC2 security group
* Confirmed port binding

ss -tulnp | grep 9000

**SonarQube Database Connection Issue**

**Error Observed**

* SonarQube failed to start with embedded DB warnings

**Root Cause**  
PostgreSQL not configured in sonar.properties

**Fix Applied**

* Updated database configuration

sonar.jdbc.username=sonar

sonar.jdbc.password=Sonar@123

sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube

**Jenkins Unable to Authenticate with SonarQube**

**Error Observed**

* Analysis failed due to missing authentication

**Root Cause**  
SonarQube token not configured in Jenkins

**Fix Applied**

* Generated token in SonarQube
* Added token as Jenkins Secret Text credential
* Linked credential in Jenkins SonarQube configuration

**Slack Notifications Not Received Initially**

**Error Observed**

* No Jenkins messages in Slack channel

**Root Cause**  
Slack token not configured or not linked to Jenkins job

**Fix Applied**

* Installed Jenkins CI app in Slack
* Stored Slack integration token in Jenkins credentials
* Configured Slack globally in Jenkins
* Enabled Slack notifications in Jenkins job post-build actions

**Jenkins Build Failures in Early Runs**

**Error Observed**

stage\_1 – Failure after 15 sec

**Root Cause**  
Pipeline configuration incomplete during initial testing

**Fix Applied**

* Corrected SonarQube scanner properties
* Fixed source paths and binaries
* Re-ran the pipeline successfully

**Conclusion**

* Jenkins successfully cloned source code from GitHub
* SonarQube analyzed the code and applied Quality Gates
* Quality Gate status was **PASSED**
* Jenkins build completed successfully
* Slack received real-time notifications for:
  + Build start
  + Build failure
  + Build success
* End-to-end CI pipeline using Jenkins, SonarQube, and Slack was implemented and validated successfully

This setup ensures **code quality enforcement**, **pipeline visibility**, and **instant team notifications**, making it suitable for real-world DevOps and production CI environments.

**Steps Performed in Jenkins**

Opened **Jenkins** web UI using browser (http://<public-ip>:8080).

Clicked **New Item** from the Jenkins dashboard.

Entered the job name as **stage\_1**.

Selected **Freestyle project** as the job type.

Clicked **OK** to create the job.

In **General** section:

* + Added description: **stage\_1 task**
  + Enabled **Restrict where this project can be run**
  + Set **Label Expression** to **master**

Moved to **Source Code Management** section.

Selected **Git** as SCM.

Entered GitHub repository URL:

* + https://github.com/betawins/Vprofile-1.git

Left **Credentials** as **None** (public repository).

In **Branches to build**, set:

* + Branch Specifier: \*/master

Clicked **Save** to store the job configuration.

Returned to the job page **stage\_1**.

Clicked **Build Now** to trigger the job manually.

Jenkins executed the build on **Built-in Node (master)**.

Jenkins cloned the Git repository successfully.

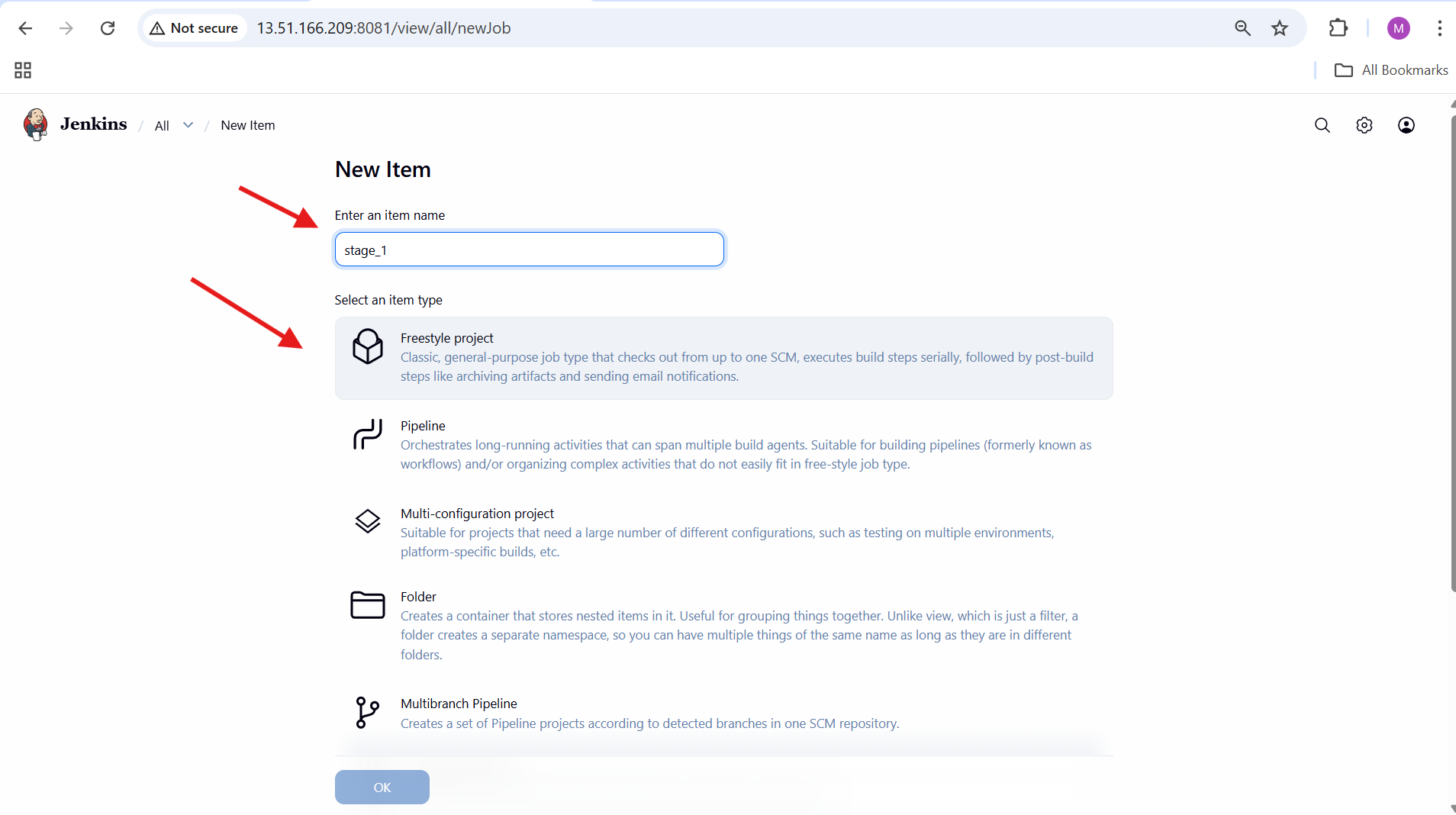
Git checked out the **master** branch.

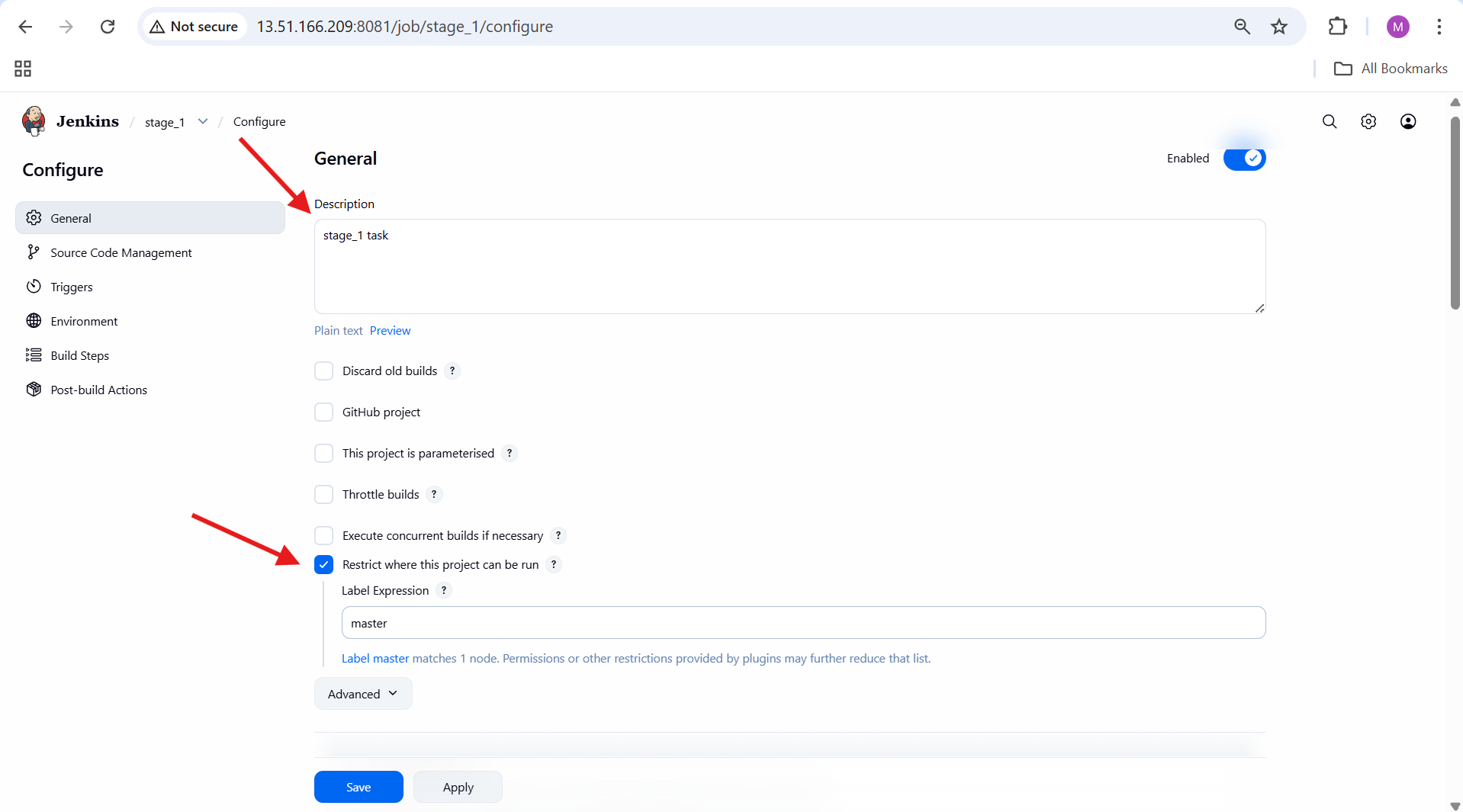
Console output showed:

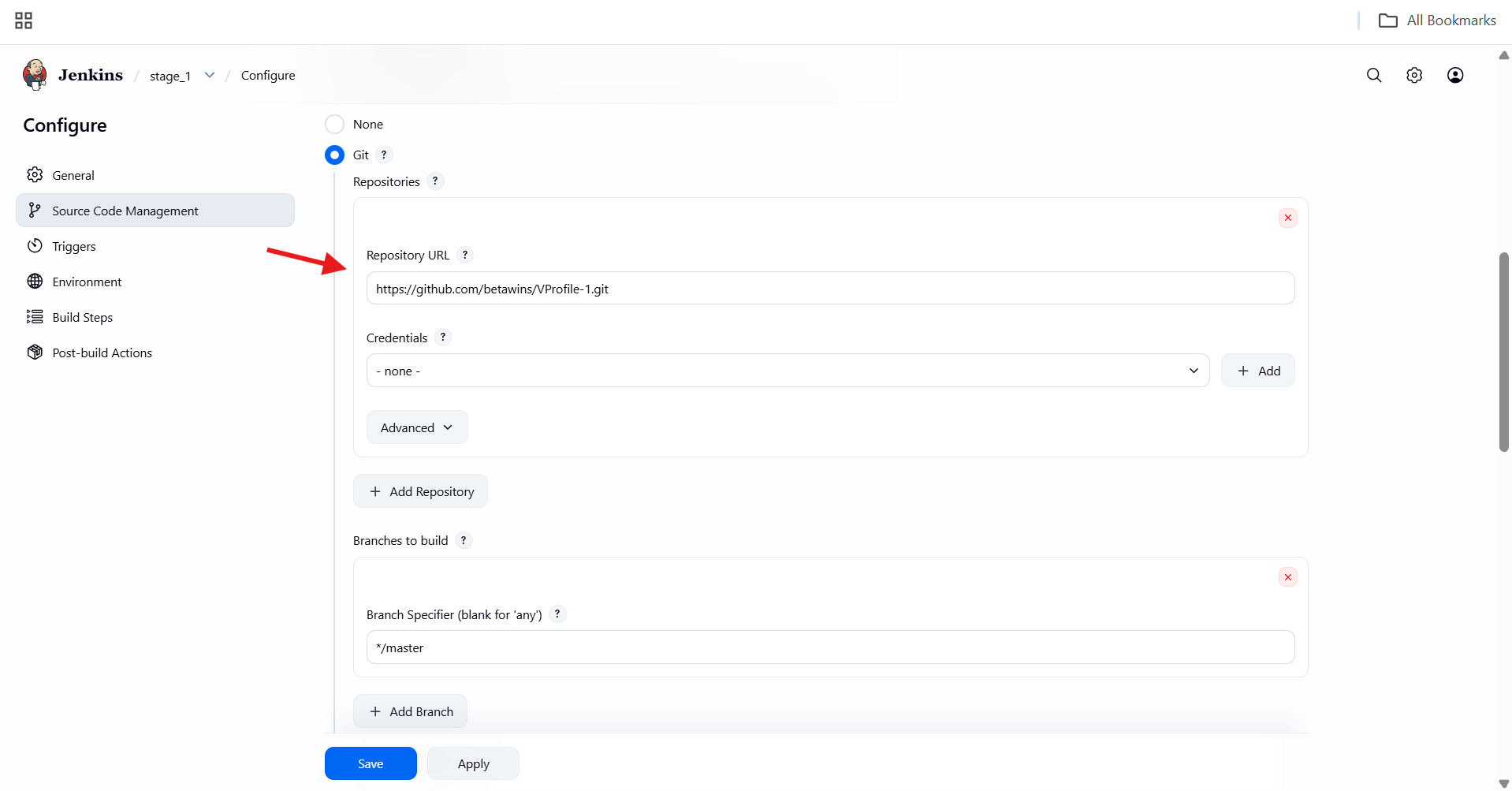
* + Repository cloned
  + Commit fetched
  + No build steps required
  + Build completed without errors

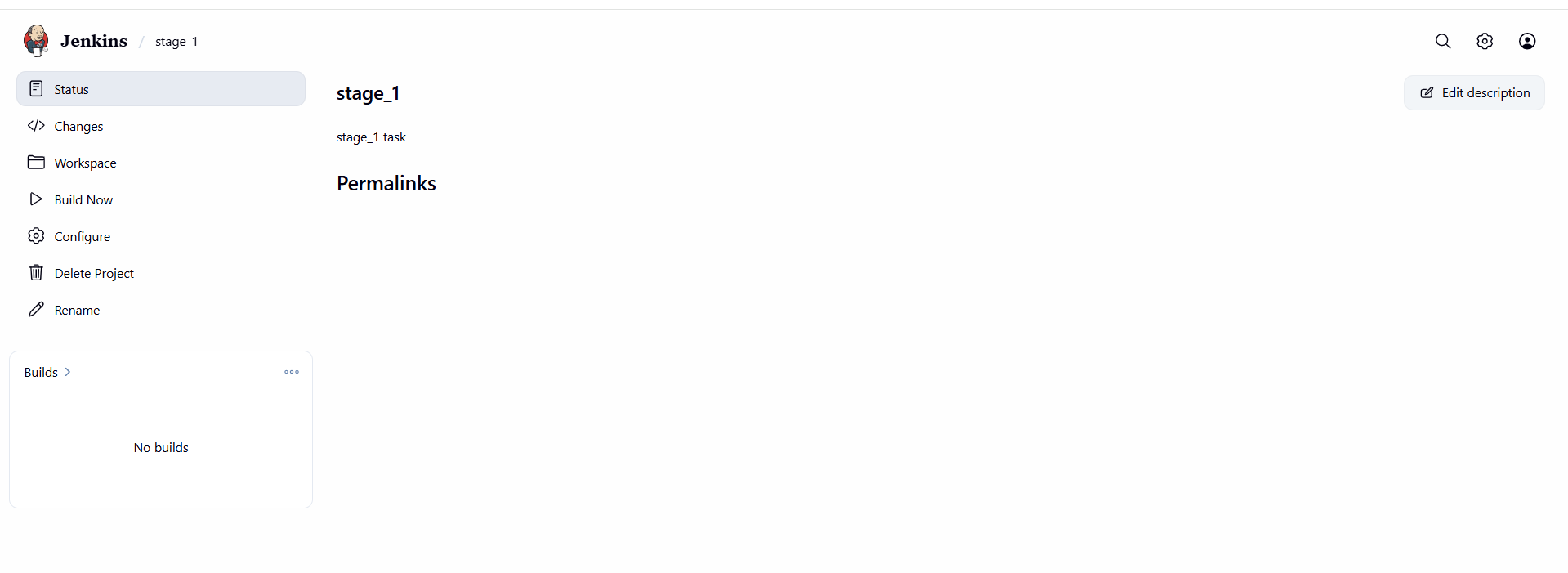
Build finished with status **SUCCESS**.

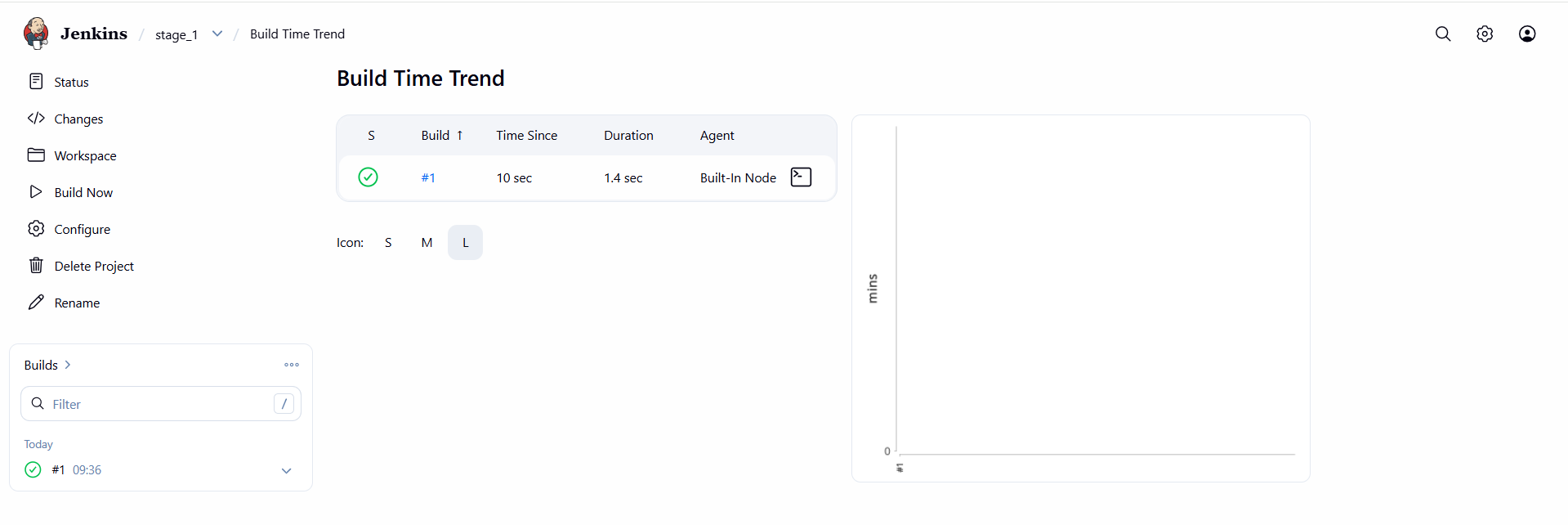
Viewed **Build Time Trend** showing successful build #1.

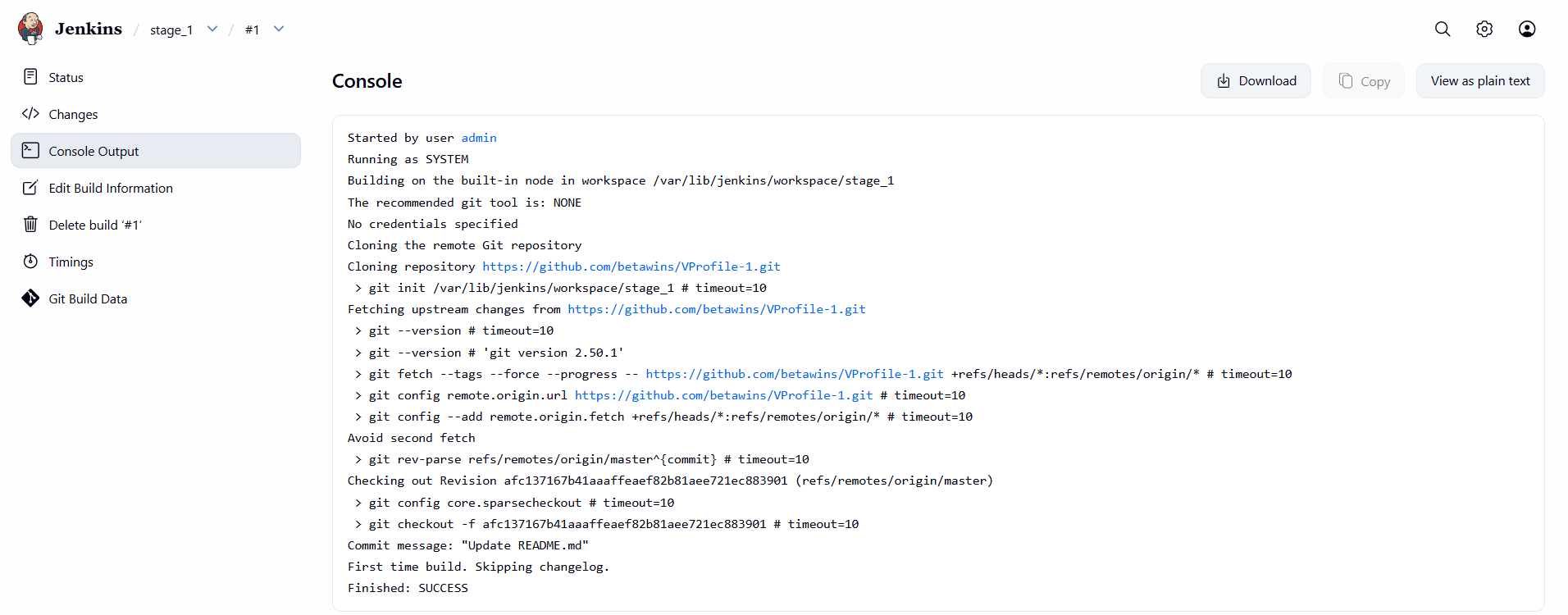












* **stage2**: Sonarqube Integration to check the quality of code.

**EC2 Instance Creation for SonarQube Server**

launched an EC2 instance to host SonarQube and PostgreSQL.

**Explanation**  
SonarQube needs a Linux server with enough CPU, memory, and disk. EC2 provides a flexible compute environment.

* Launched an instance on **Amazon EC2**
* Selected a compatible instance type
* Attached key pair
* Configured storage
* Noted Public IP and Private IP

**Security Group Configuration**

opened required ports for SonarQube and PostgreSQL.

**Explanation**  
SonarQube runs on port 9000 and PostgreSQL on 5432. These must be reachable.

**Inbound rules**

SSH 22 Your IP

Custom TCP 9000 0.0.0.0/0

Custom TCP 5432 0.0.0.0/0

**Java Installation (Amazon Corretto)**

installed Java because SonarQube runs on JVM.

**Command**

yum install java-17-amazon-corretto -y

**Verification**

java -version

**PostgreSQL Installation**

You installed PostgreSQL as SonarQube’s backend database.

**Command**

dnf install postgresql15 postgresql15-server -y

**Initialize database**

postgresql-setup --initdb

**Start and enable service**

systemctl start postgresql

systemctl enable postgresql

systemctl status postgresql

**PostgreSQL Database Configuration for SonarQube**

You created database and user for SonarQube.

**Switch to postgres user**

su - postgres

**Open PostgreSQL shell**

psql

**Commands executed**

CREATE DATABASE sonarqube;

CREATE USER sonar WITH ENCRYPTED PASSWORD 'Sonar@123';

GRANT ALL PRIVILEGES ON DATABASE sonarqube TO sonar;

ALTER DATABASE sonarqube OWNER TO sonar;

**Verify**

\l

\du

**Exit**

\q

exit

**SonarQube Download and Extraction**

You downloaded and extracted SonarQube binaries.

**Command**

cd /opt

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.6.0.59030.zip

unzip sonarqube-9.6.0.59030.zip

mv sonarqube-9.6.0.59030 sonar

**SonarQube Permission Issue and Fix**

You faced permission denied errors while starting SonarQube.

**Error**

Permission denied: SonarQube.pid

Permission denied: logs/sonar.log

**Fix**  
Create a dedicated sonar user and assign ownership.

**Commands**

useradd sonar

chown -R sonar:sonar /opt/sonar

chmod -R 775 /opt/sonar

**SonarQube Database Configuration**

You configured SonarQube to use PostgreSQL instead of embedded H2.

**File edited**

vi /opt/sonar/conf/sonar.properties

**Configuration added**

sonar.jdbc.username=sonar

sonar.jdbc.password=Sonar@123

sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube

sonar.web.port=9000

**SonarQube Startup**

You started SonarQube using the sonar user.

**Commands**

su - sonar

cd /opt/sonar/bin/linux-x86-64

./sonar.sh start

./sonar.sh status

**Port verification**

ss -tulnp | grep 9000

**Local test**

curl http://localhost:9000

**SonarQube Web Access**

You accessed SonarQube from the browser.

**URL**

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

**Result**

* SonarQube UI loaded successfully

**Jenkins Plugin Installation**

You installed SonarQube Scanner plugin in **Jenkins**.

**Path**

Manage Jenkins → Plugins → Available Plugins

**Action**

* Installed **SonarQube Scanner**
* Restarted Jenkins

**SonarQube Token Creation**

You created an authentication token for Jenkins.

**Path**

SonarQube → Administration → Security → Tokens

**Action**

* Generated token
* Copied immediately (one-time visibility)

**Jenkins Credentials Configuration**

You stored the SonarQube token securely.

**Path**

Manage Jenkins → Credentials → Global → Add Credentials

**Configuration**

* Kind: Secret Text
* Secret: SonarQube Token
* ID: sonarqube

**Sonar Scanner Installation on Jenkins Server**

You installed Sonar Scanner manually on Jenkins node.

**Commands**

cd /opt

wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.6.2.2472-linux.zip

unzip sonar-scanner-cli-4.6.2.2472-linux.zip

mv sonar-scanner-4.6.2.2472-linux sonar\_scanner

**Edit scanner config**

vi /opt/sonar\_scanner/conf/sonar-scanner.properties

**Configured**

sonar.host.url=http://<SonarQube-Public-IP>:9000

**Jenkins Global Tool Configuration**

You registered Sonar Scanner in Jenkins.

**Path**

Manage Jenkins → Tools → SonarQube Scanner

**Values**

* Name: sonar\_scanner
* SONAR\_RUNNER\_HOME: /opt/sonar\_scanner

**Jenkins Job Configuration for SonarQube Scan**

You added SonarQube analysis to Jenkins job.

**Path**

Job → Configure → Build Steps

**Build step**

* Execute SonarQube Scanner

**Analysis properties**

sonar.projectKey=Saber

sonar.projectName=Saber

sonar.projectVersion=1.0

sonar.sources=/var/lib/jenkins/workspace/$JOB\_NAME/src

sonar.java.binaries=src/main/java

**Jenkins Build Execution**

You triggered the job.

**Action**

Build Now

**Result**

* SonarQube scan executed
* Build succeeded

**SonarQube Quality Gate Result**

You verified Quality Gate from Jenkins.

**Result**

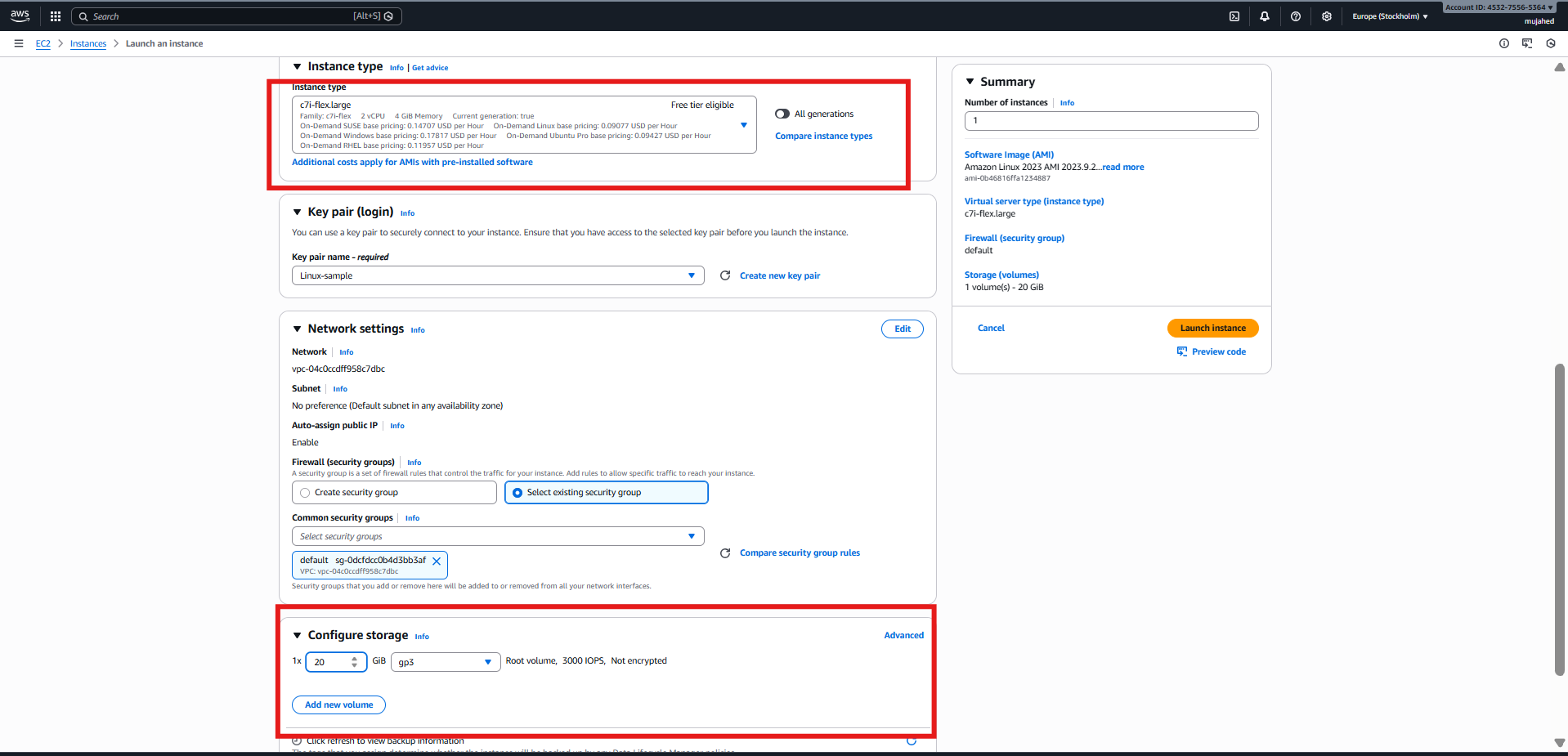
* Quality Gate: **PASSED**
* Server-side processing: **SUCCESS**

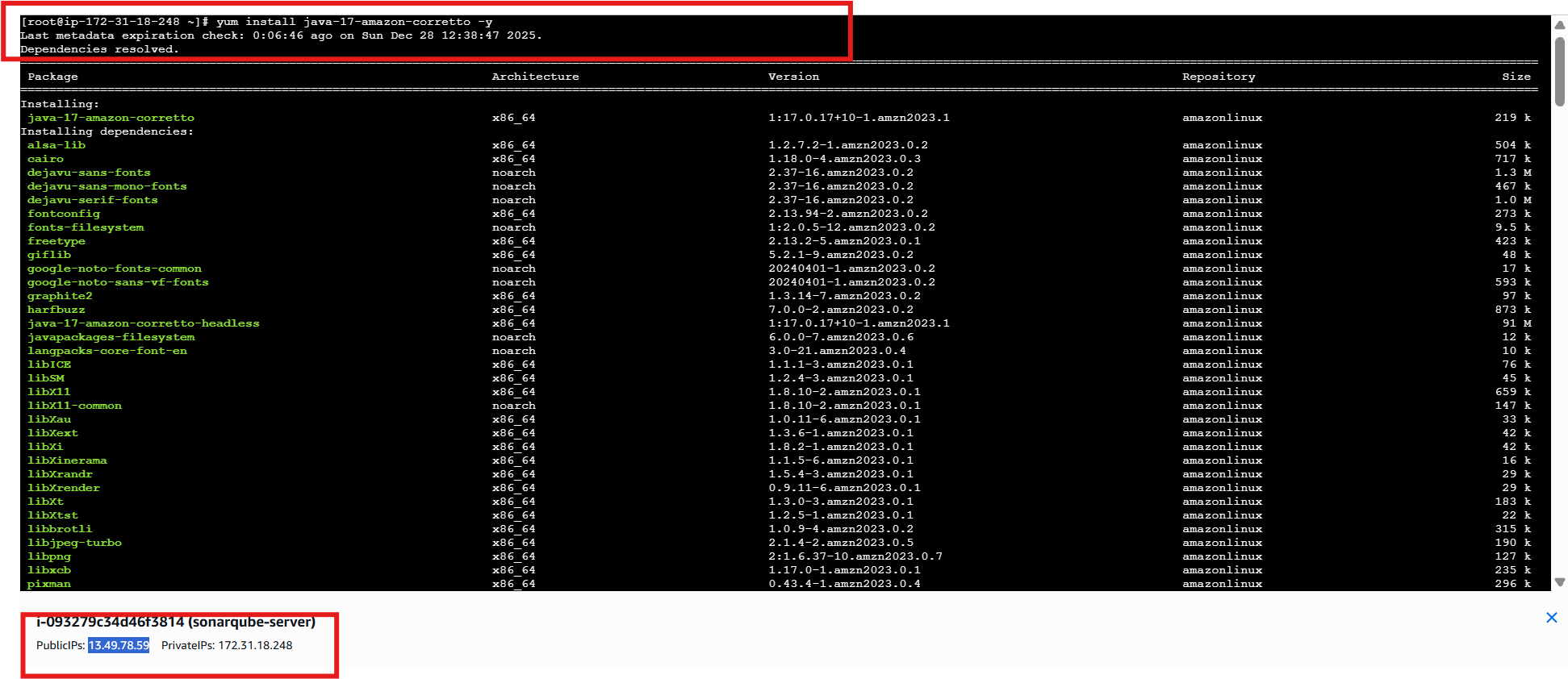
**SonarQube Dashboard Verification**

You reviewed the analysis in SonarQube UI.

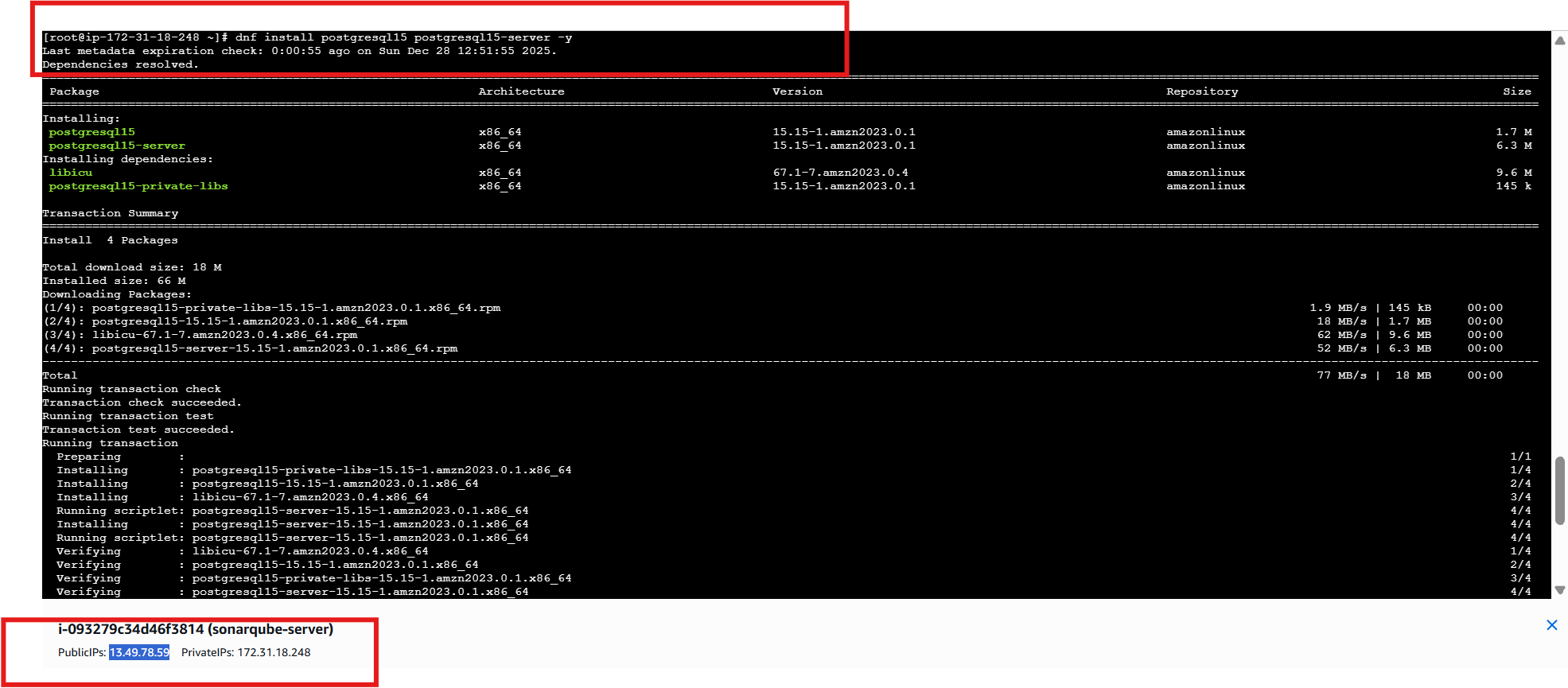
**Observed**

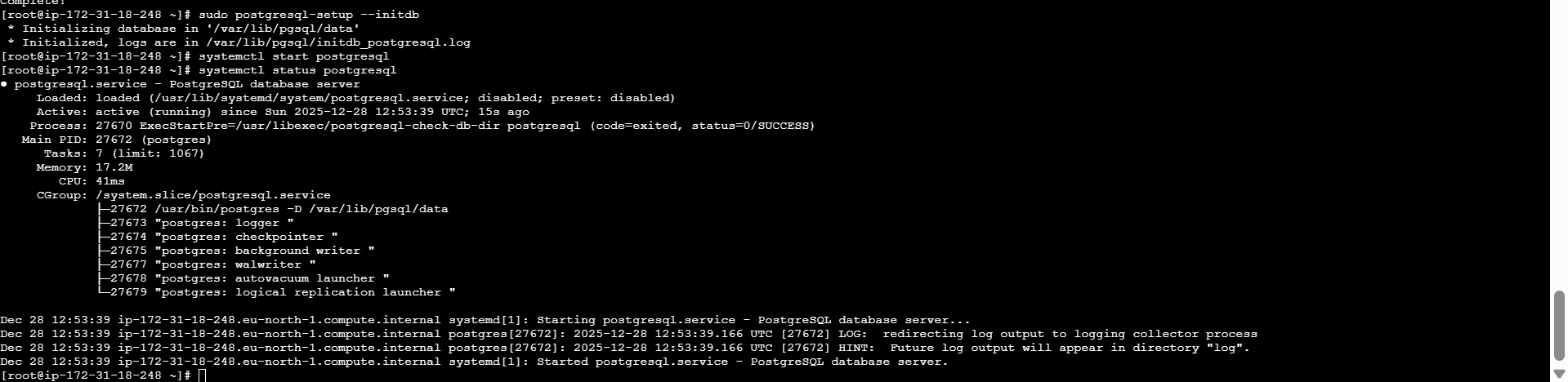
* Bugs detected
* Vulnerabilities: 0
* Code Smells present
* Maintainability rating passed
* Overall Quality Gate passed

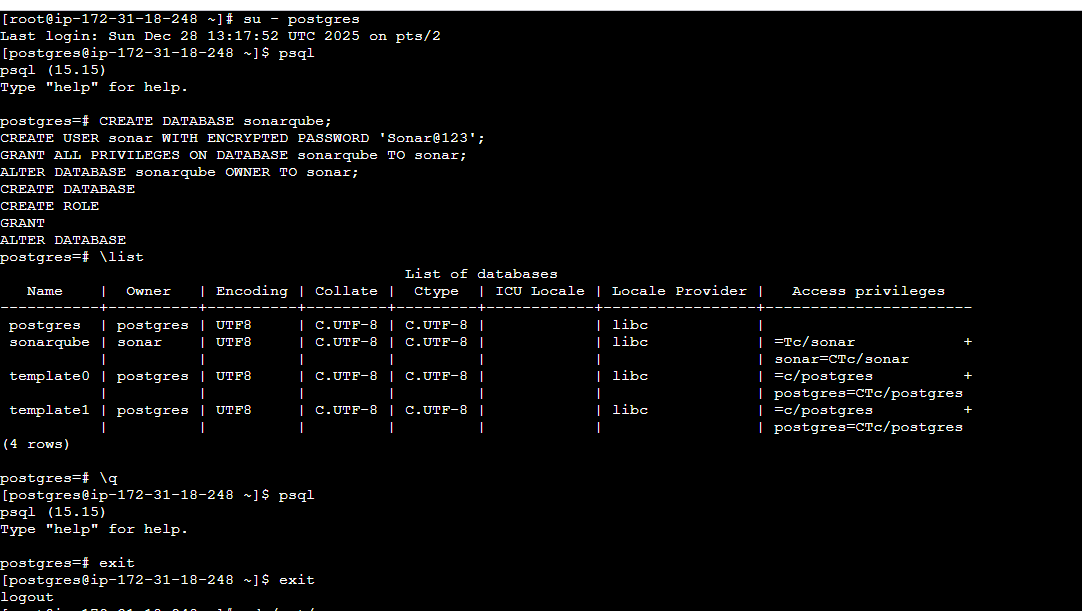


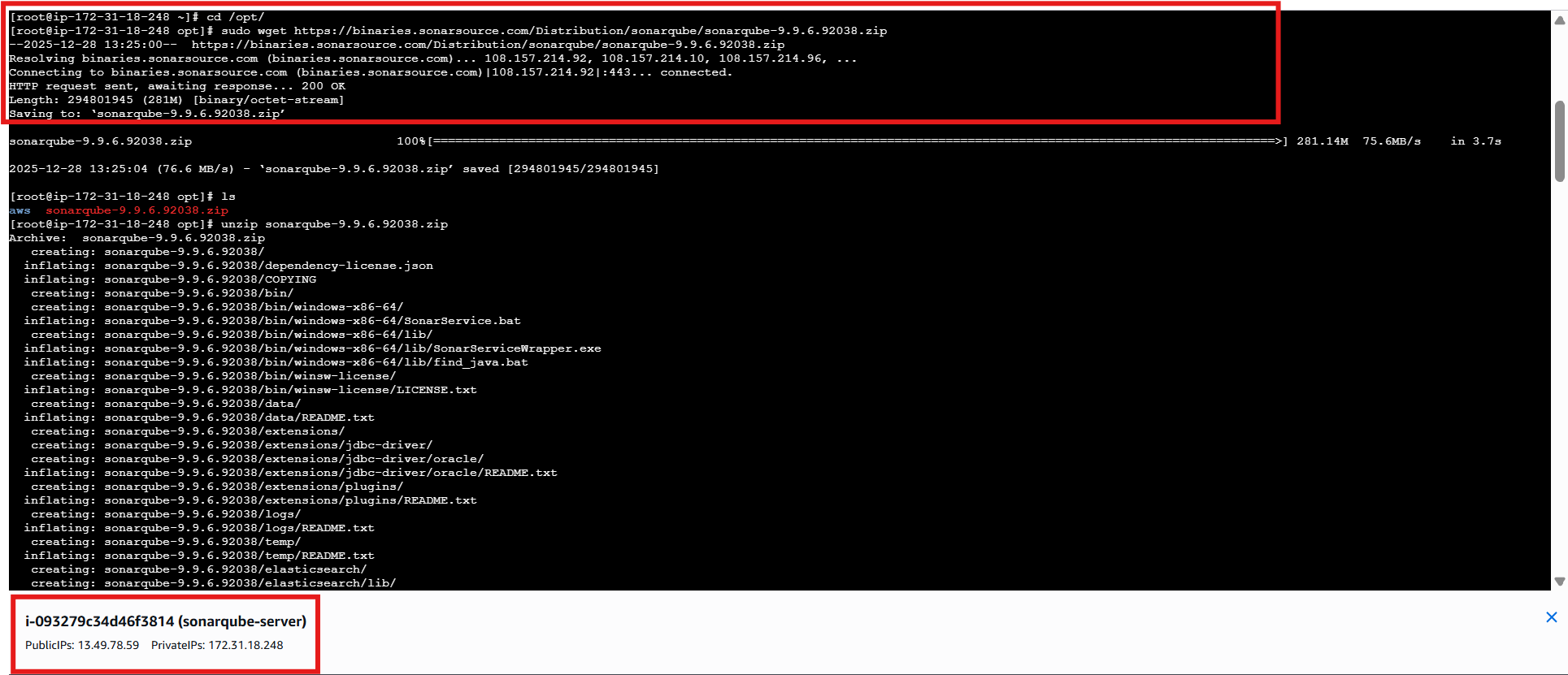


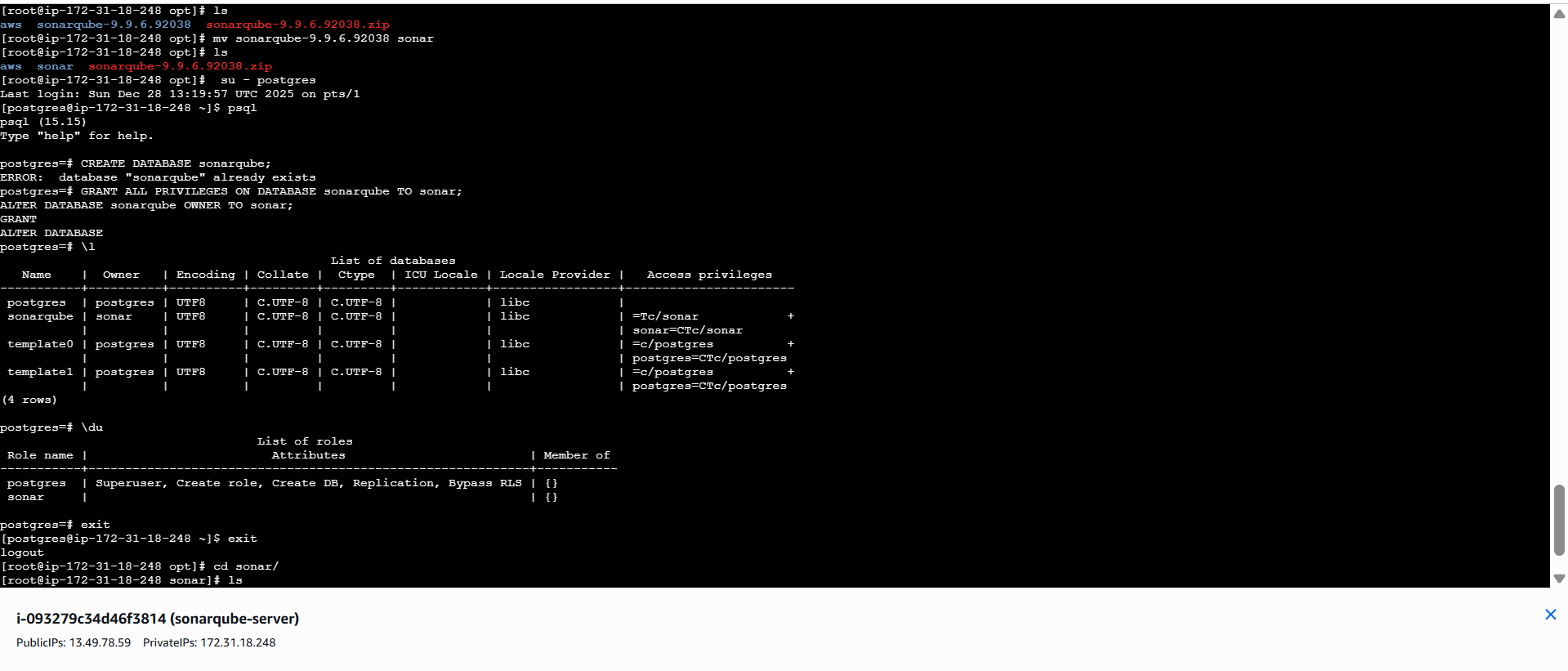


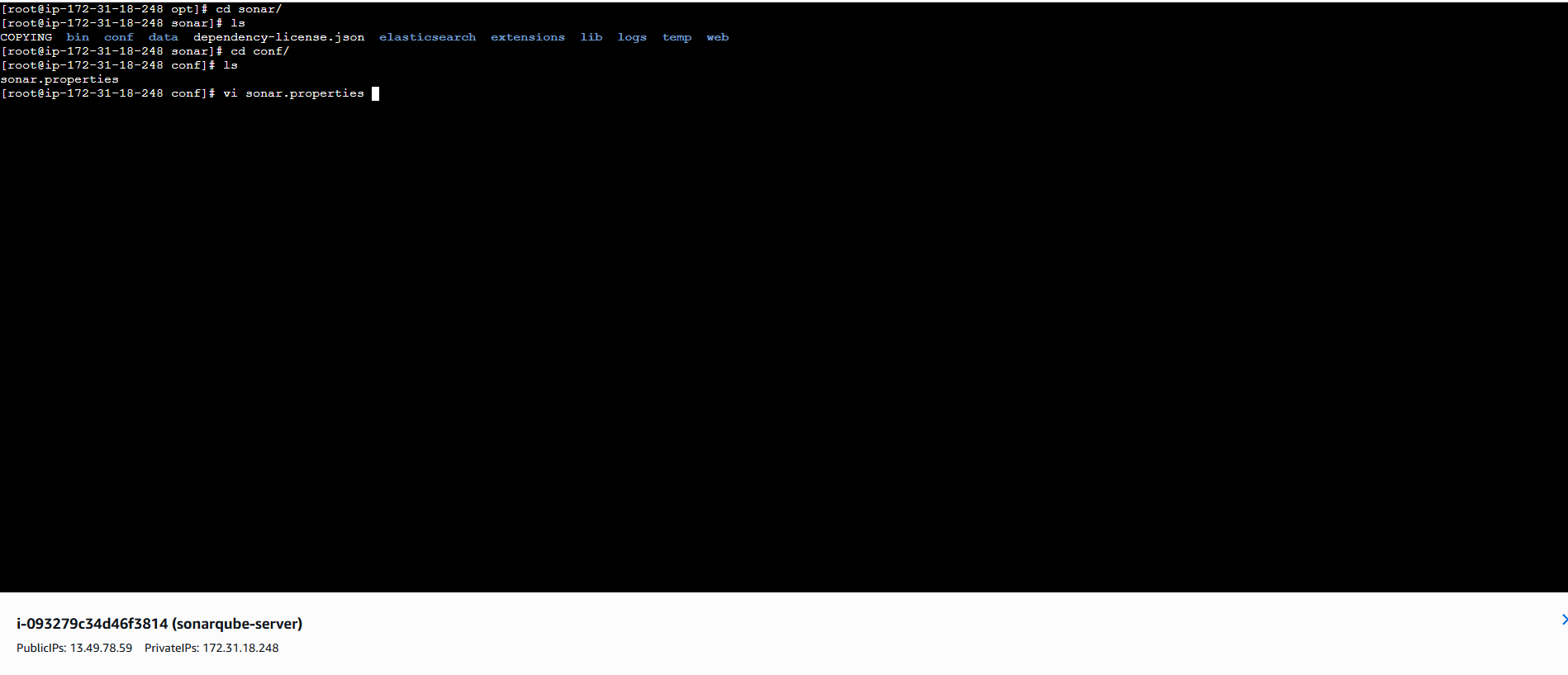


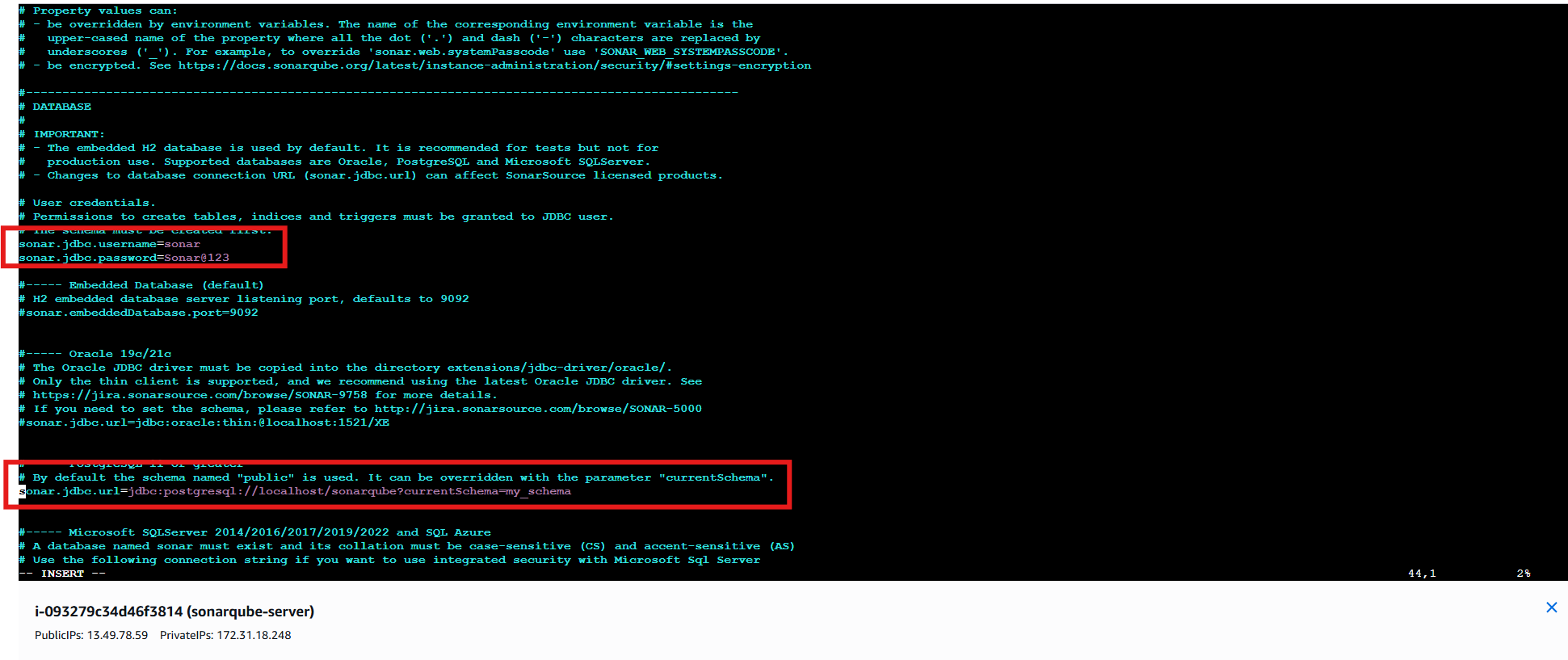


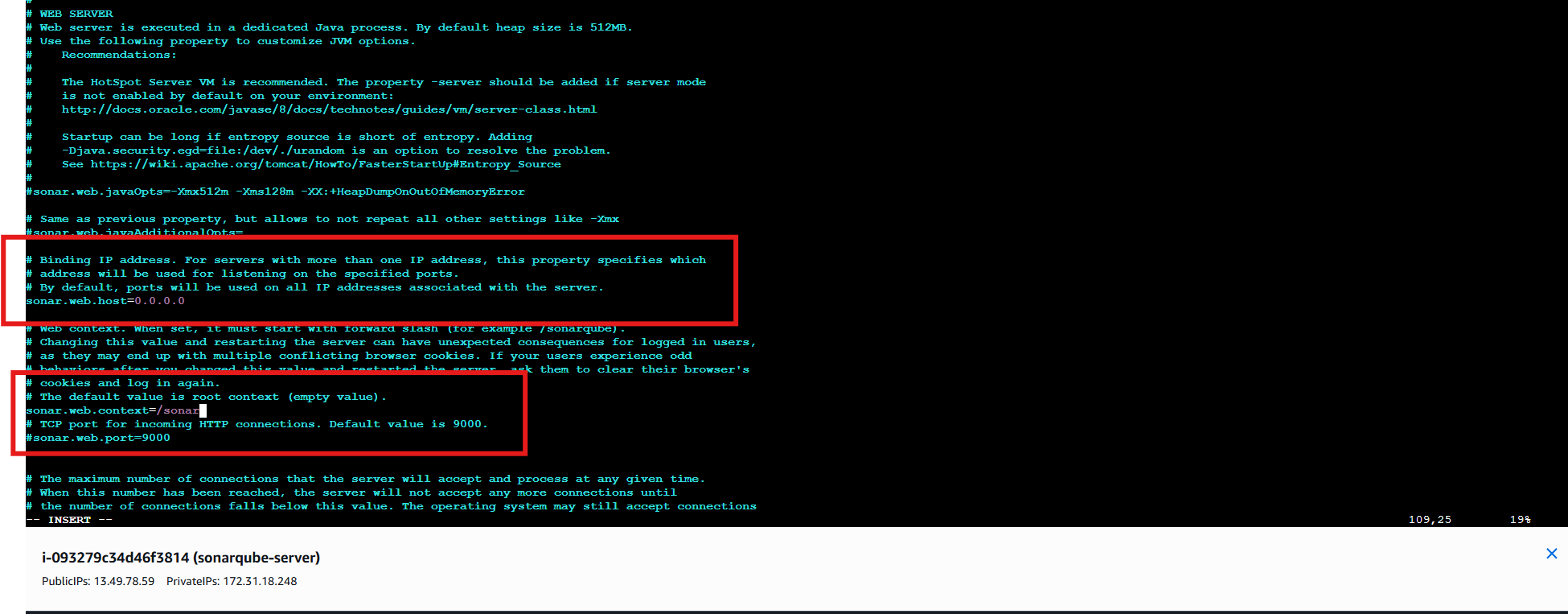


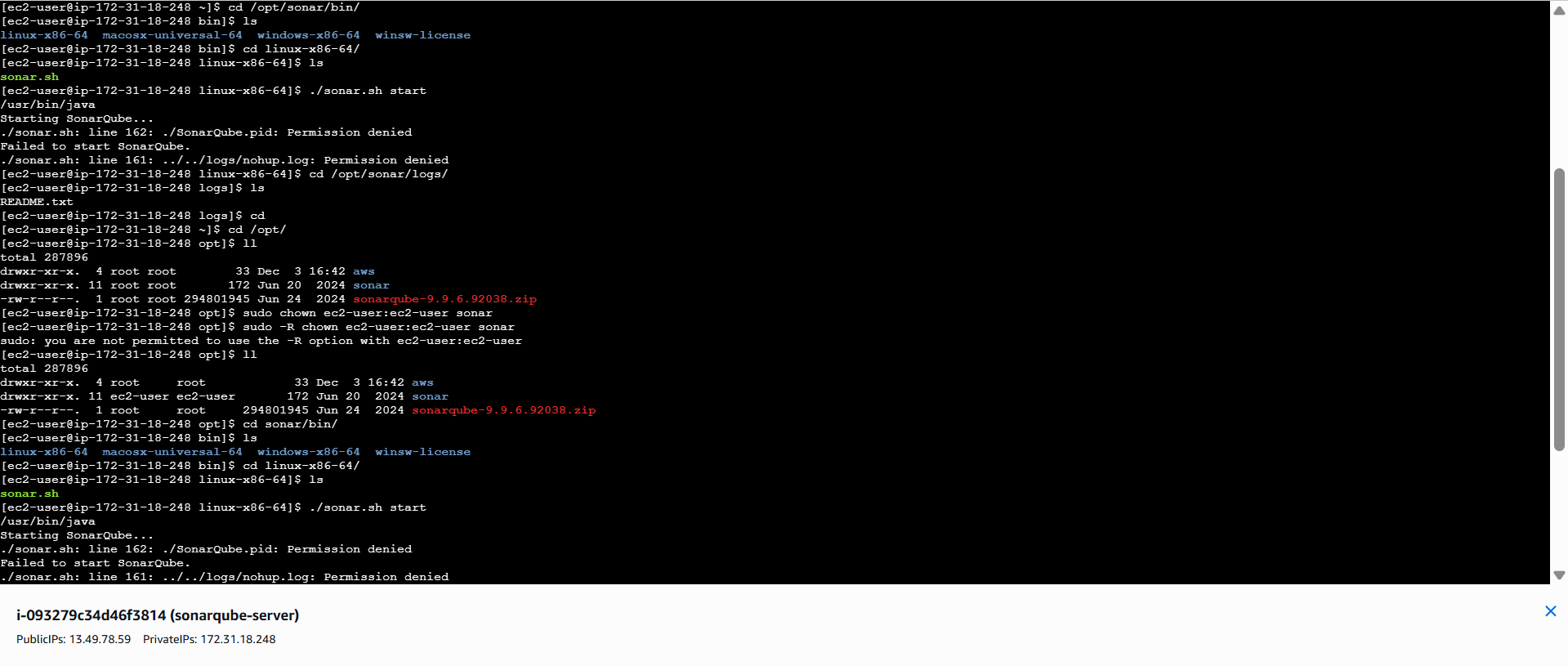


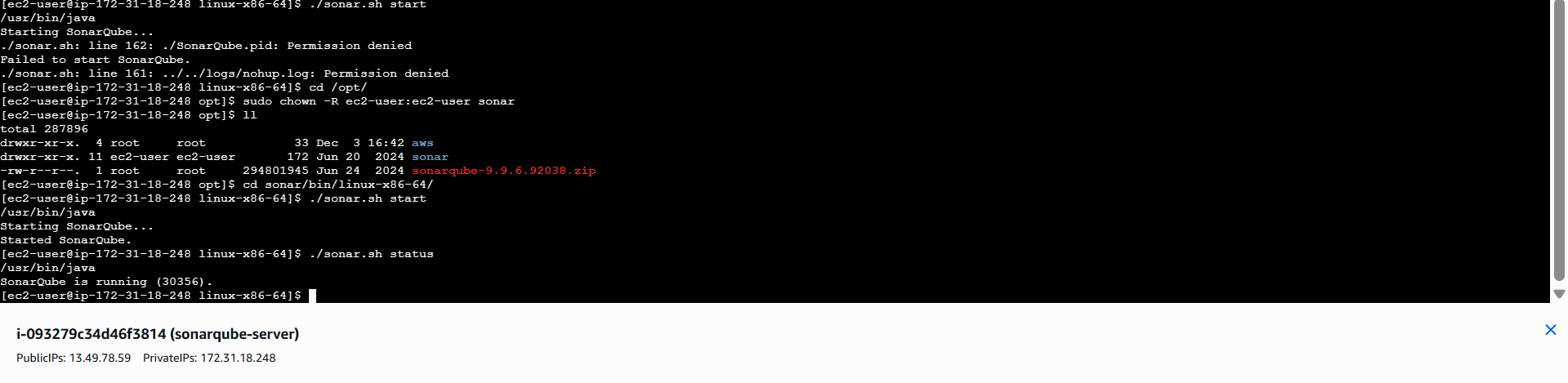


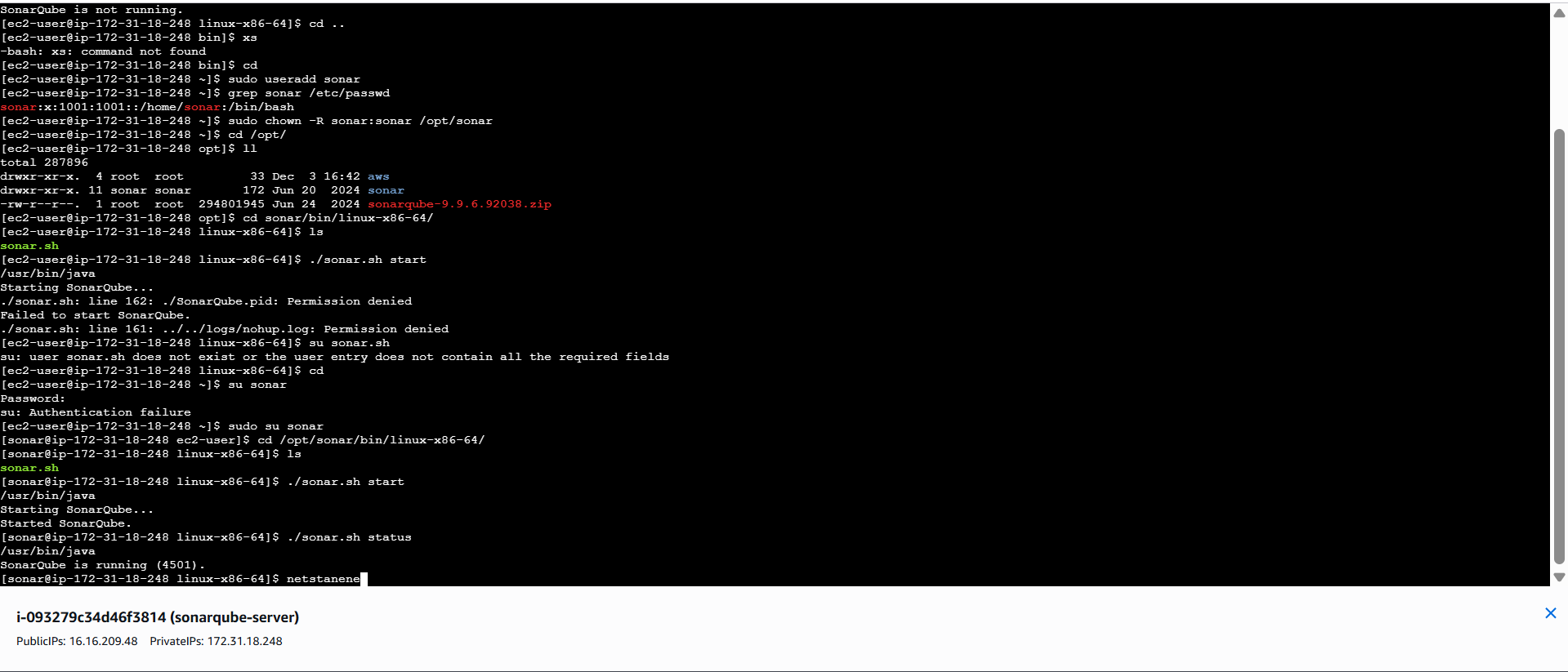


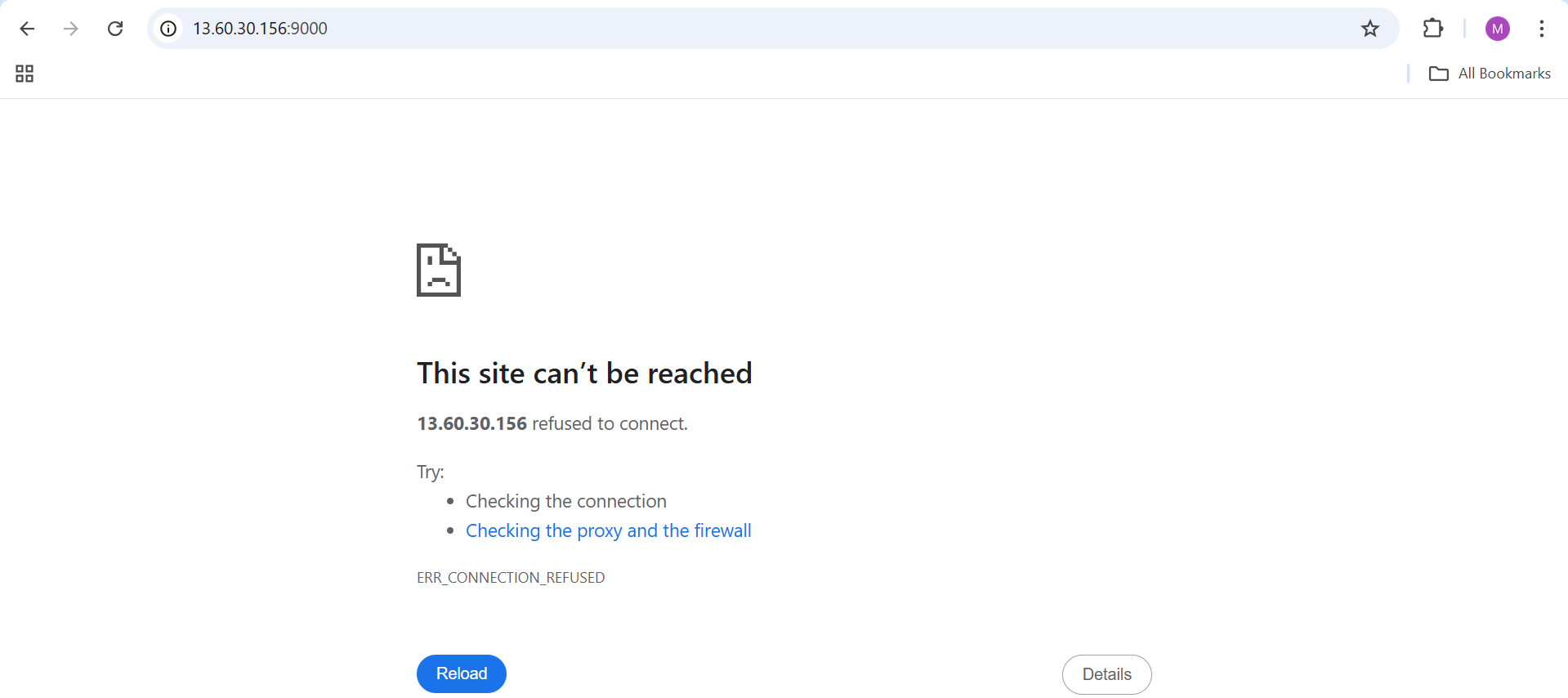


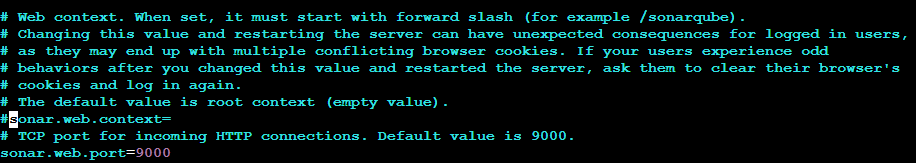


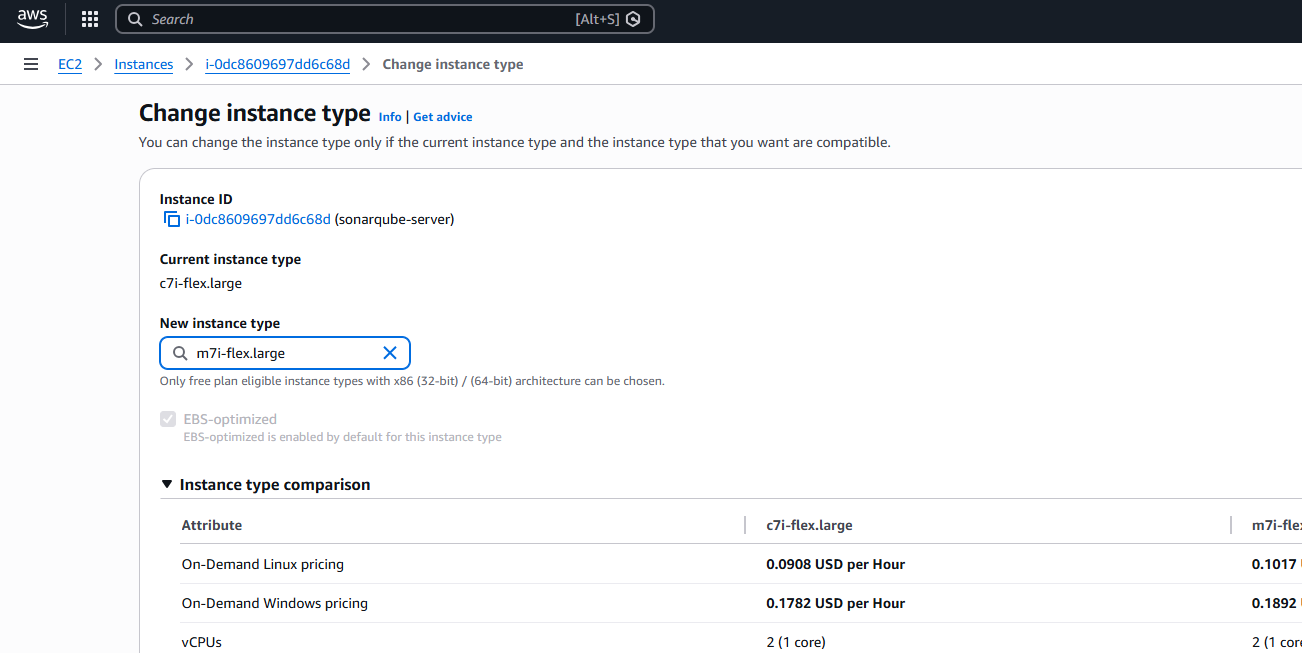


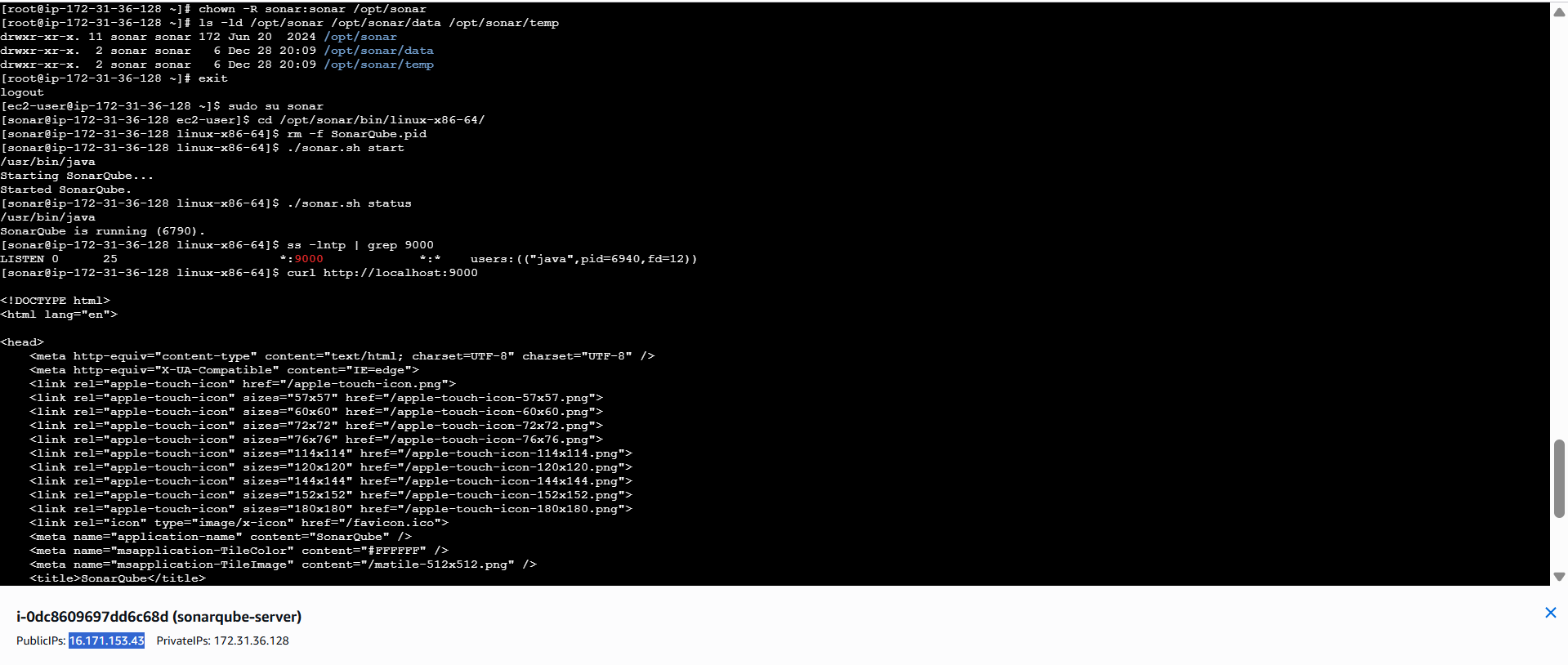


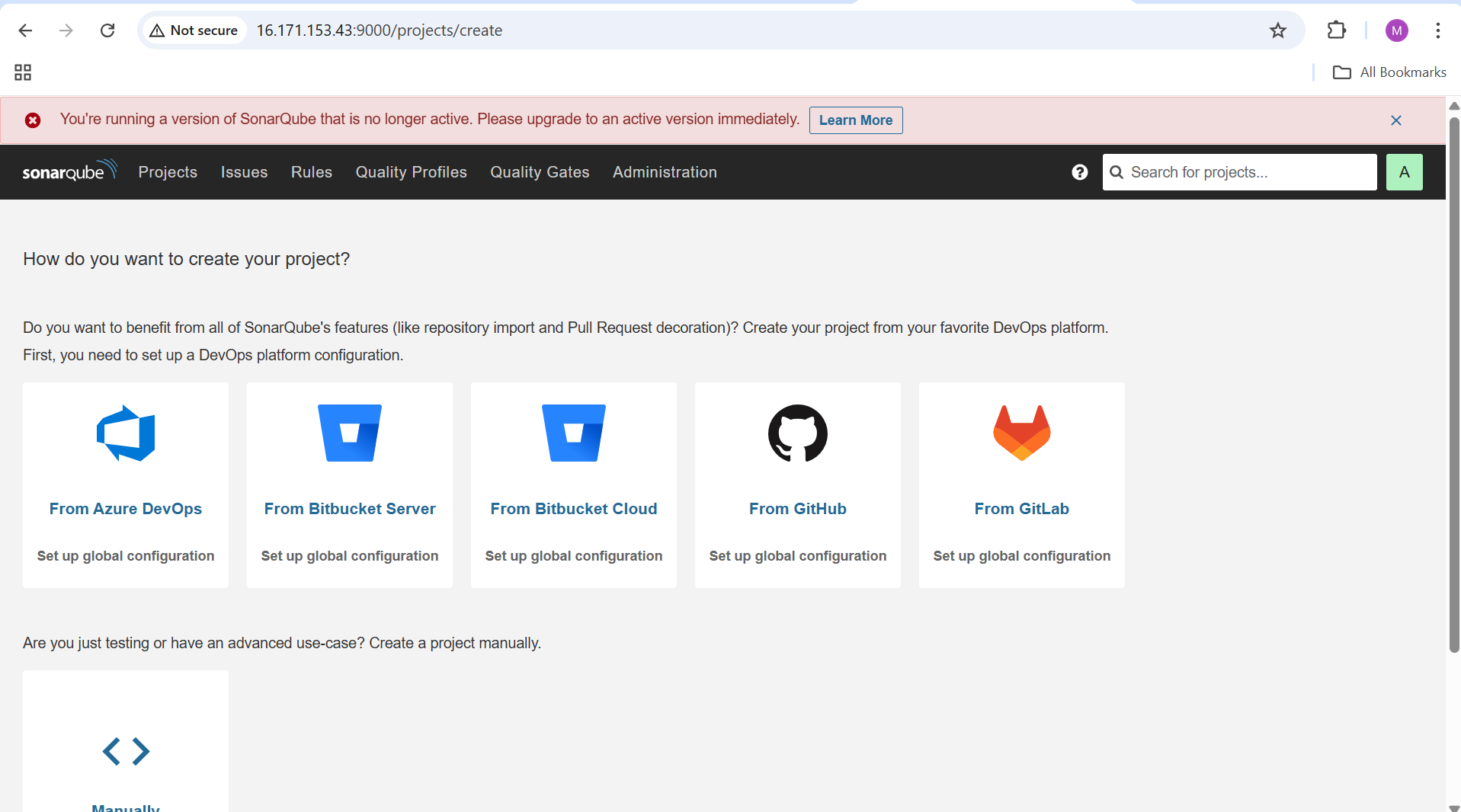


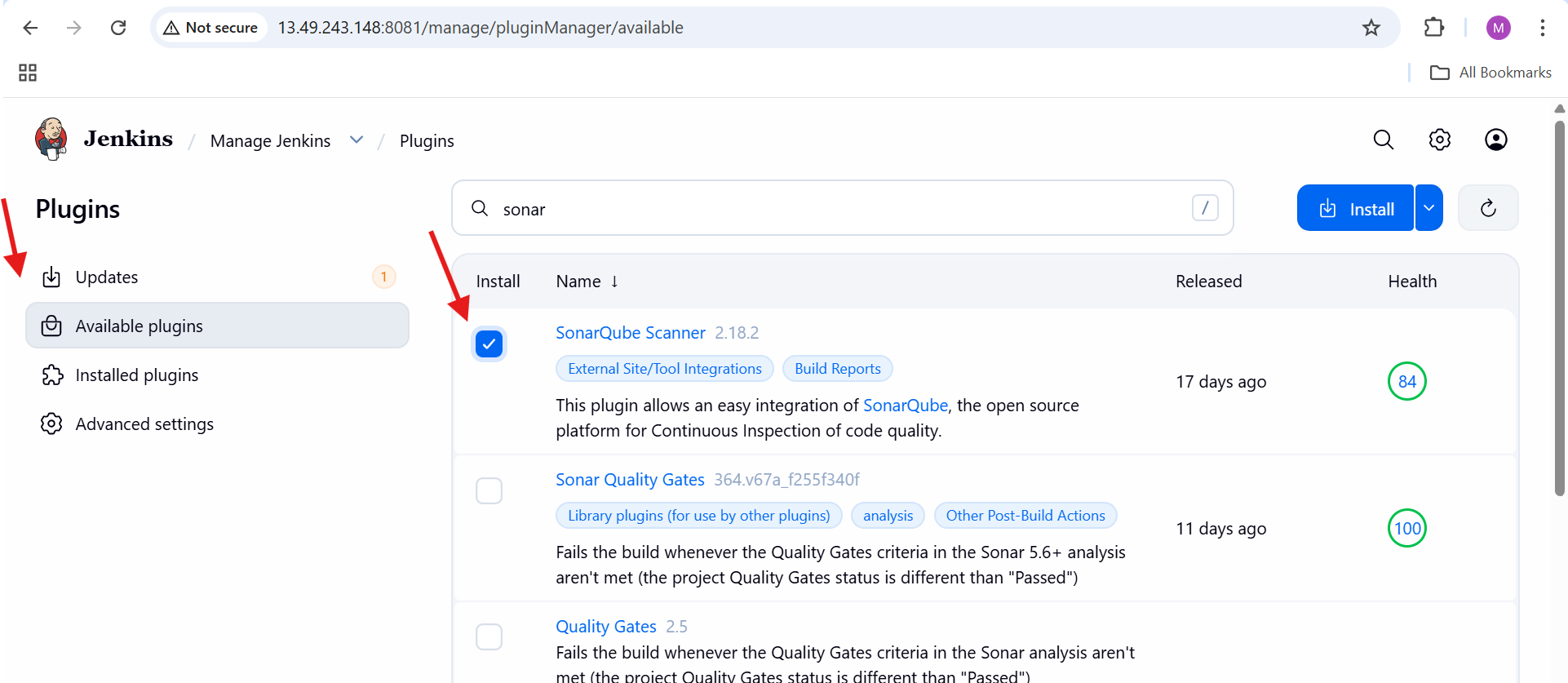


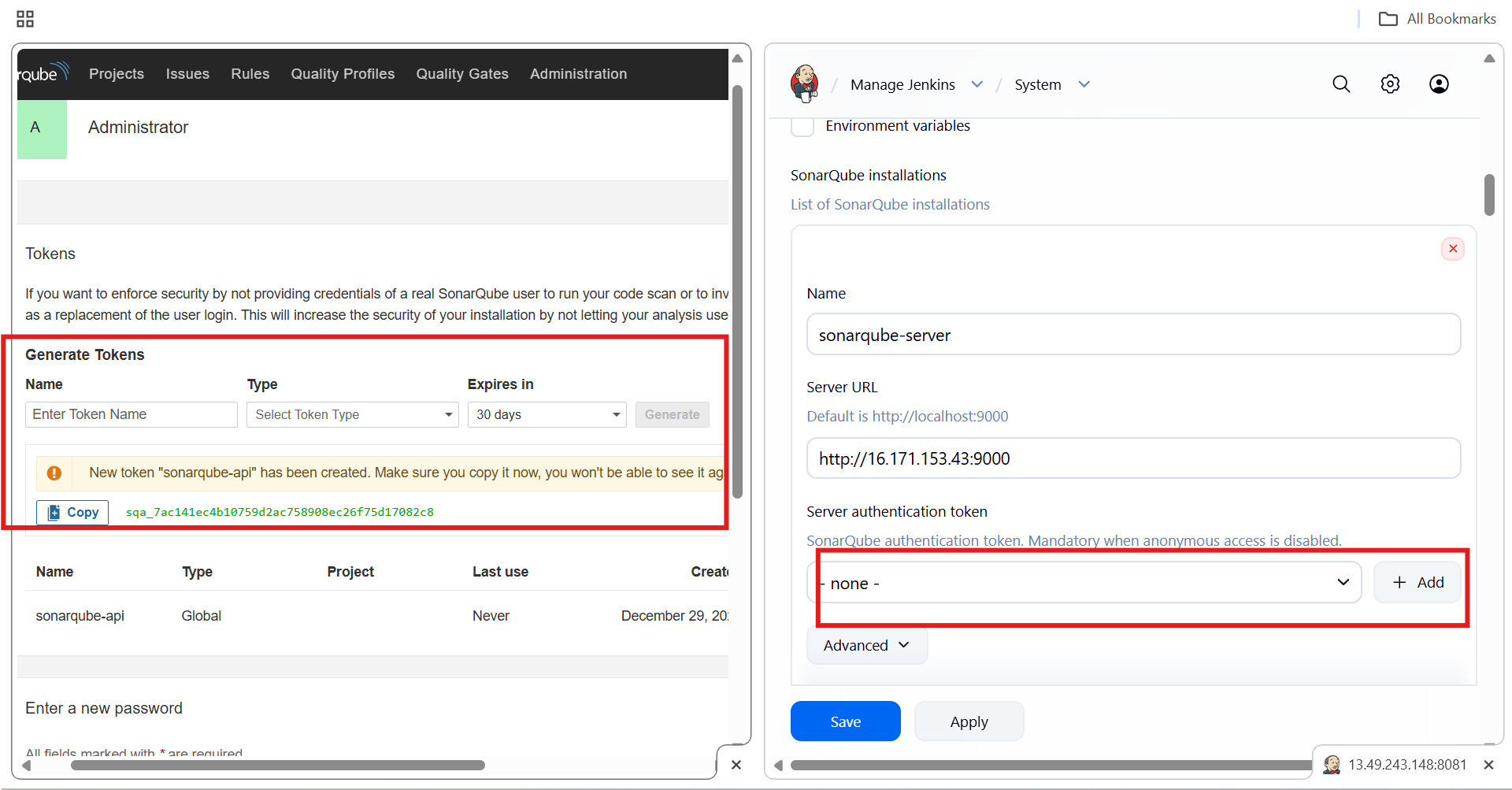


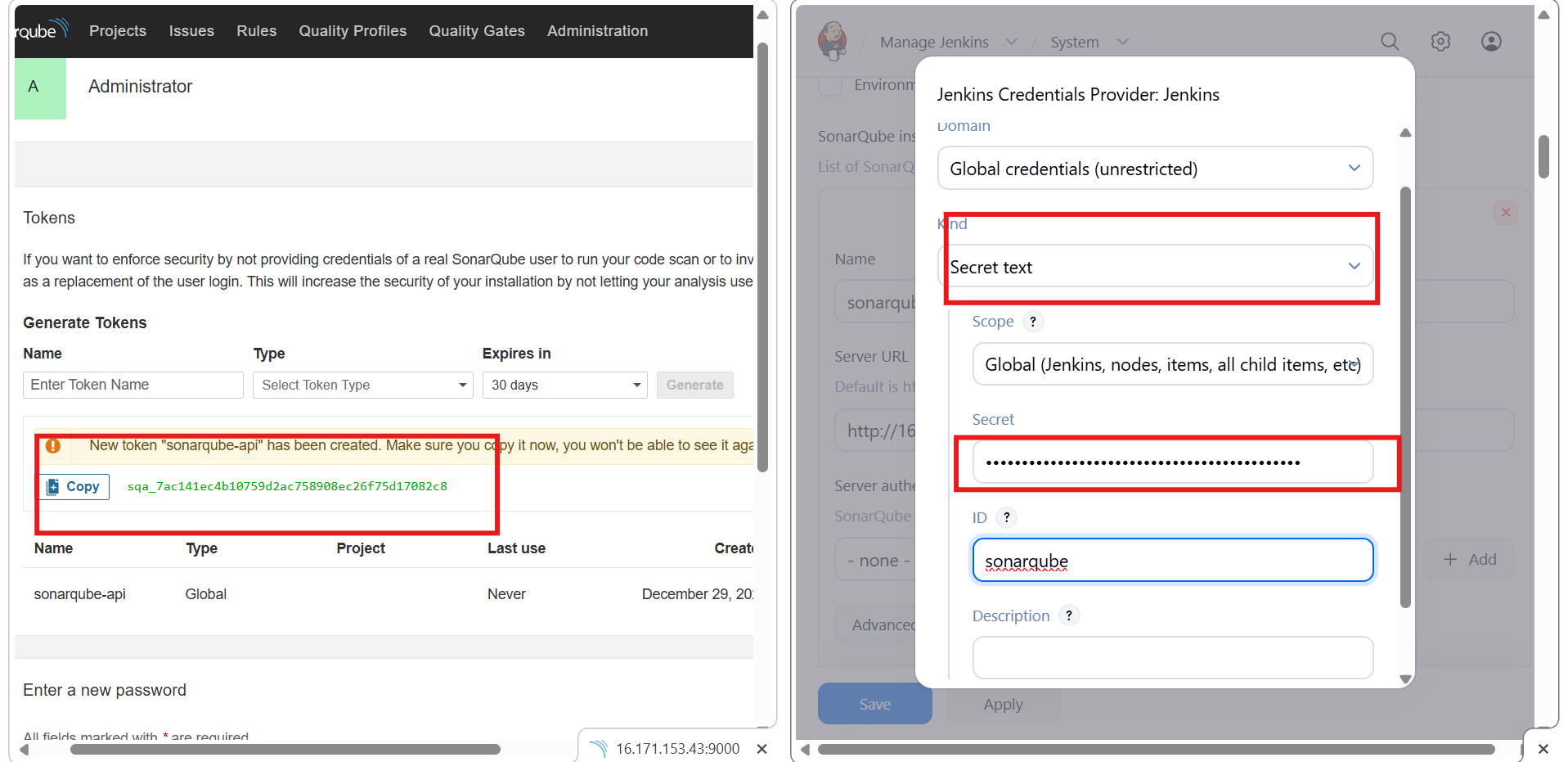


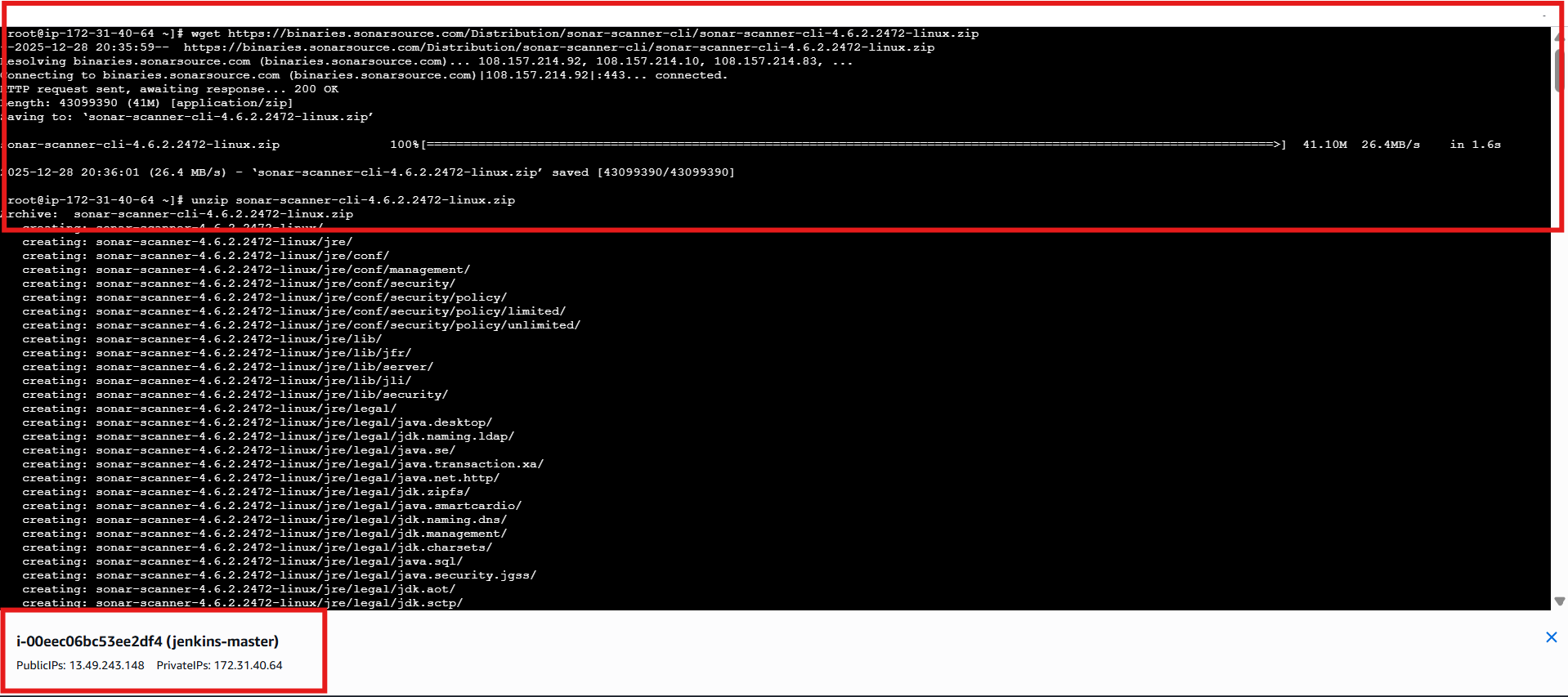




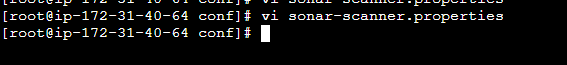


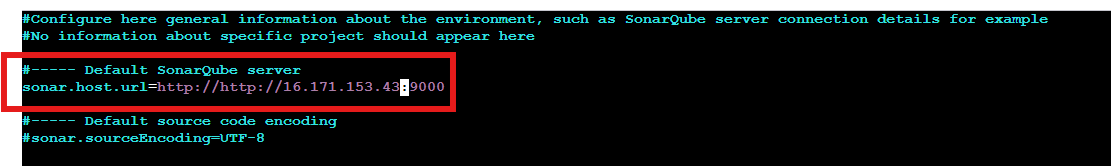


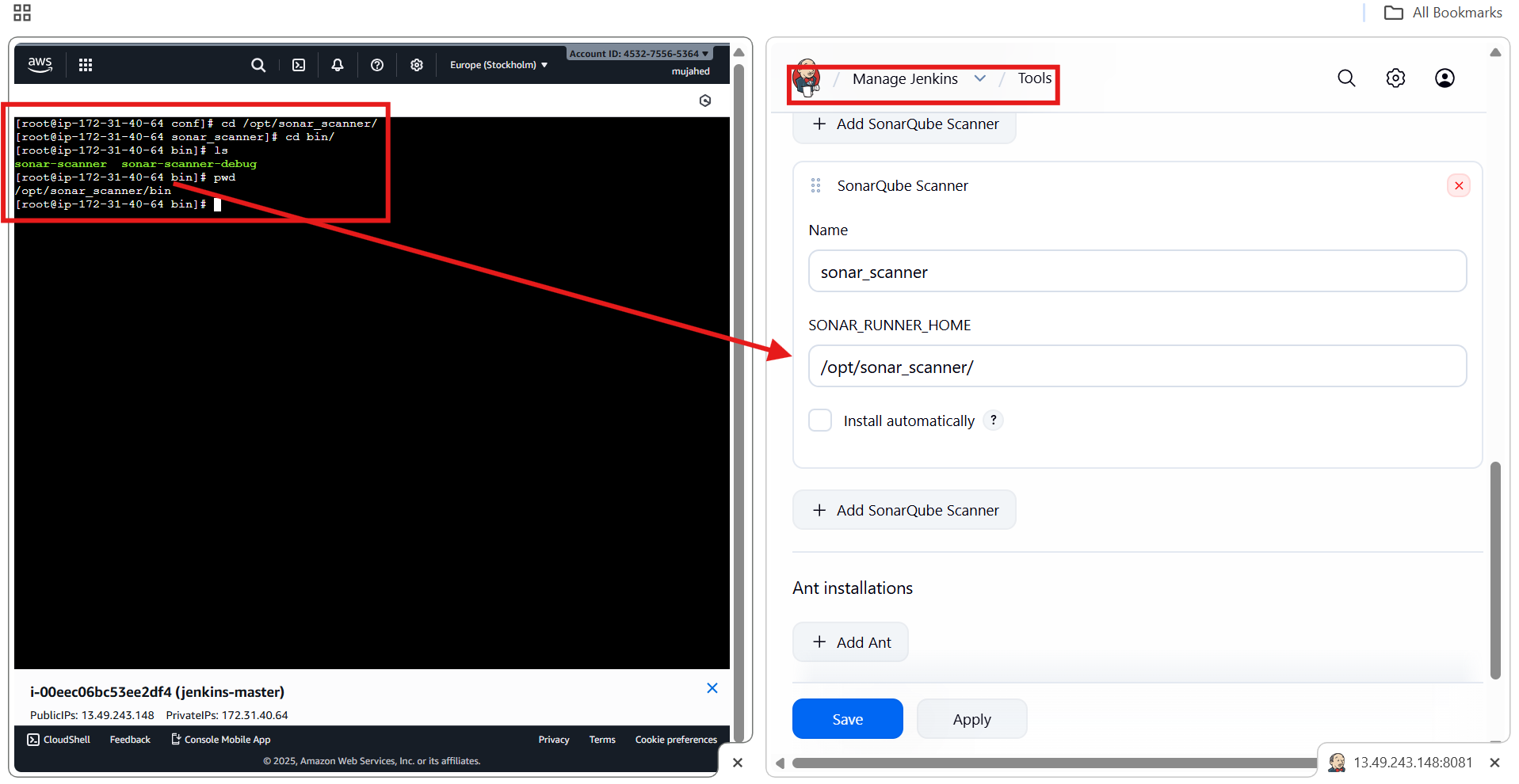


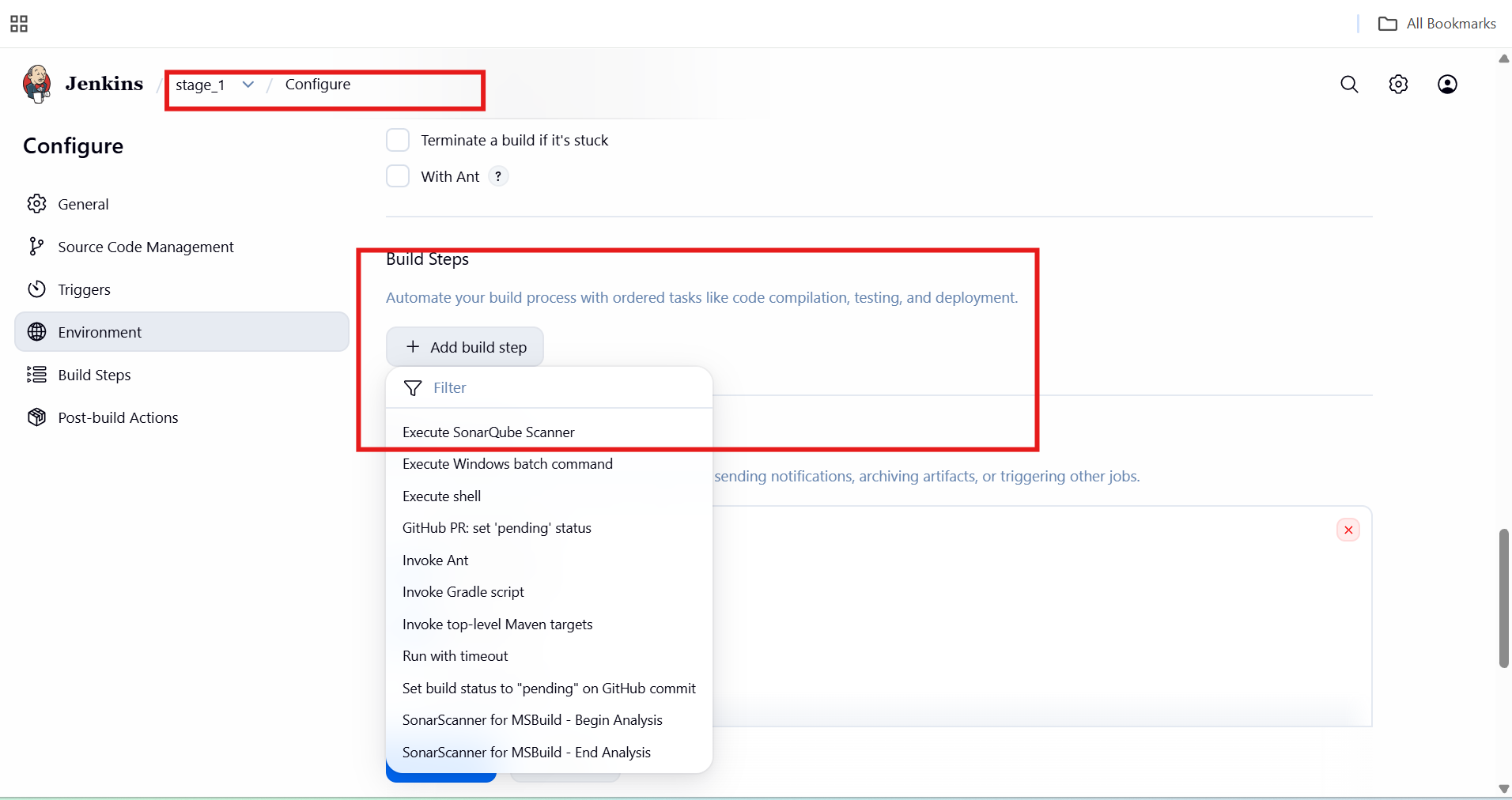


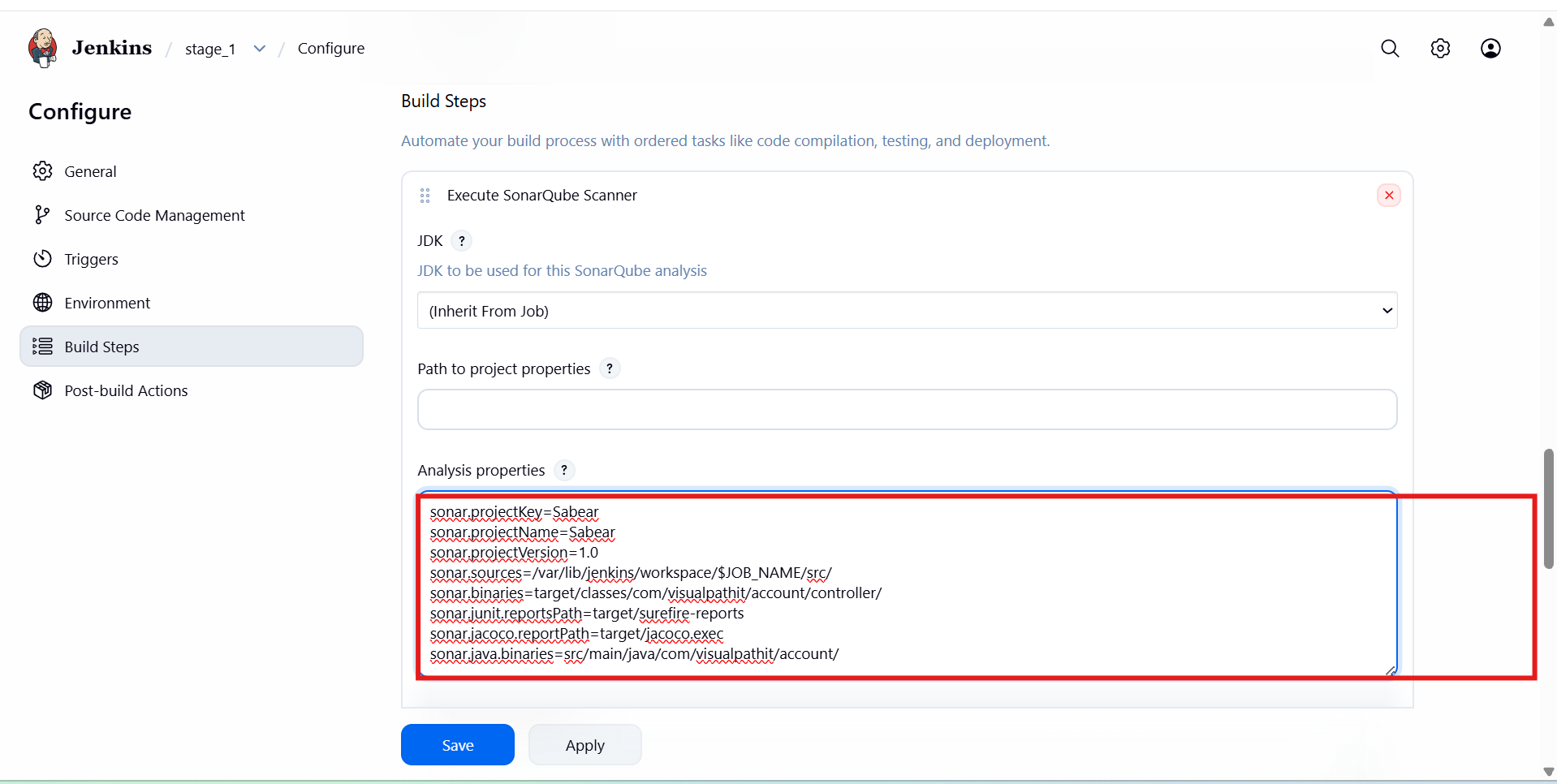


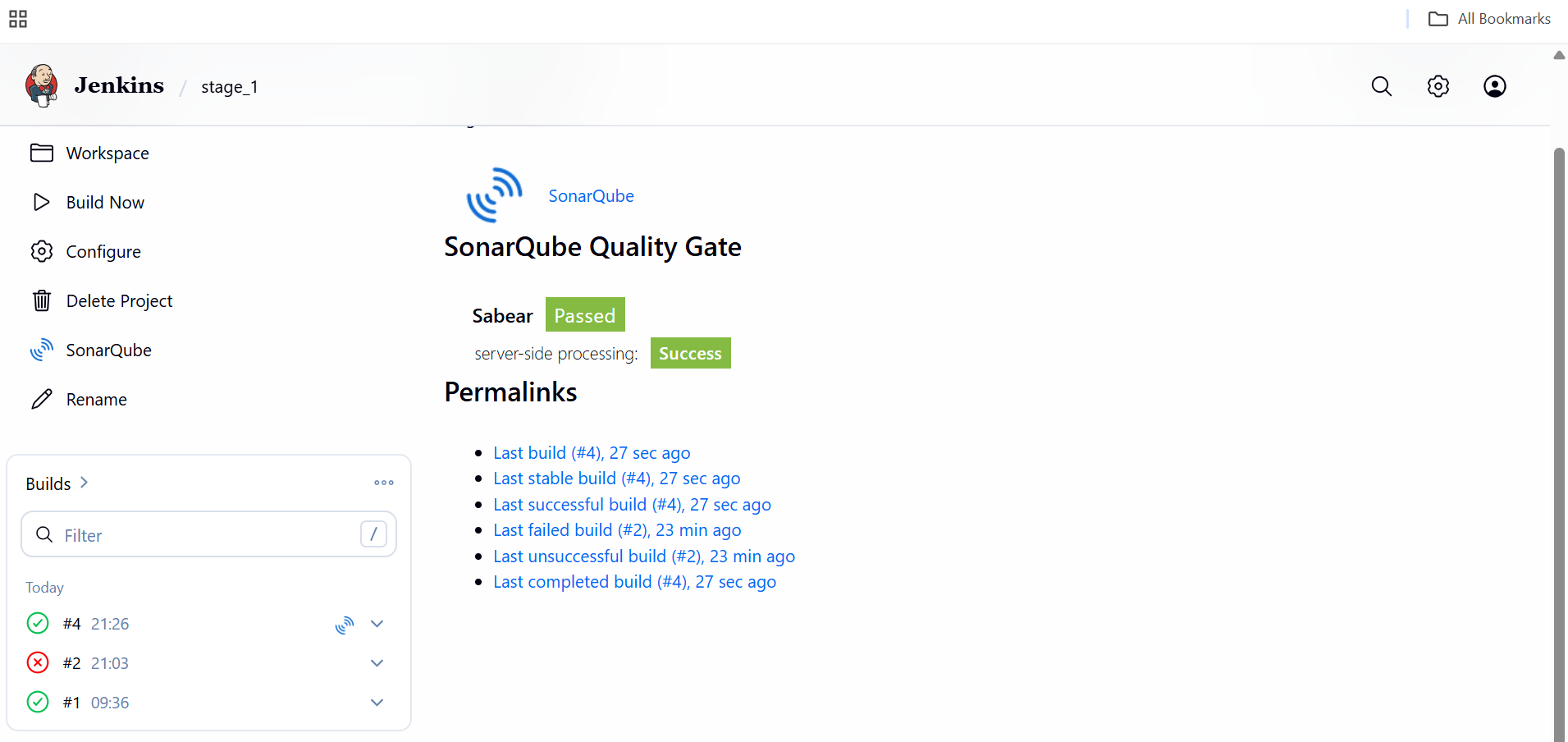


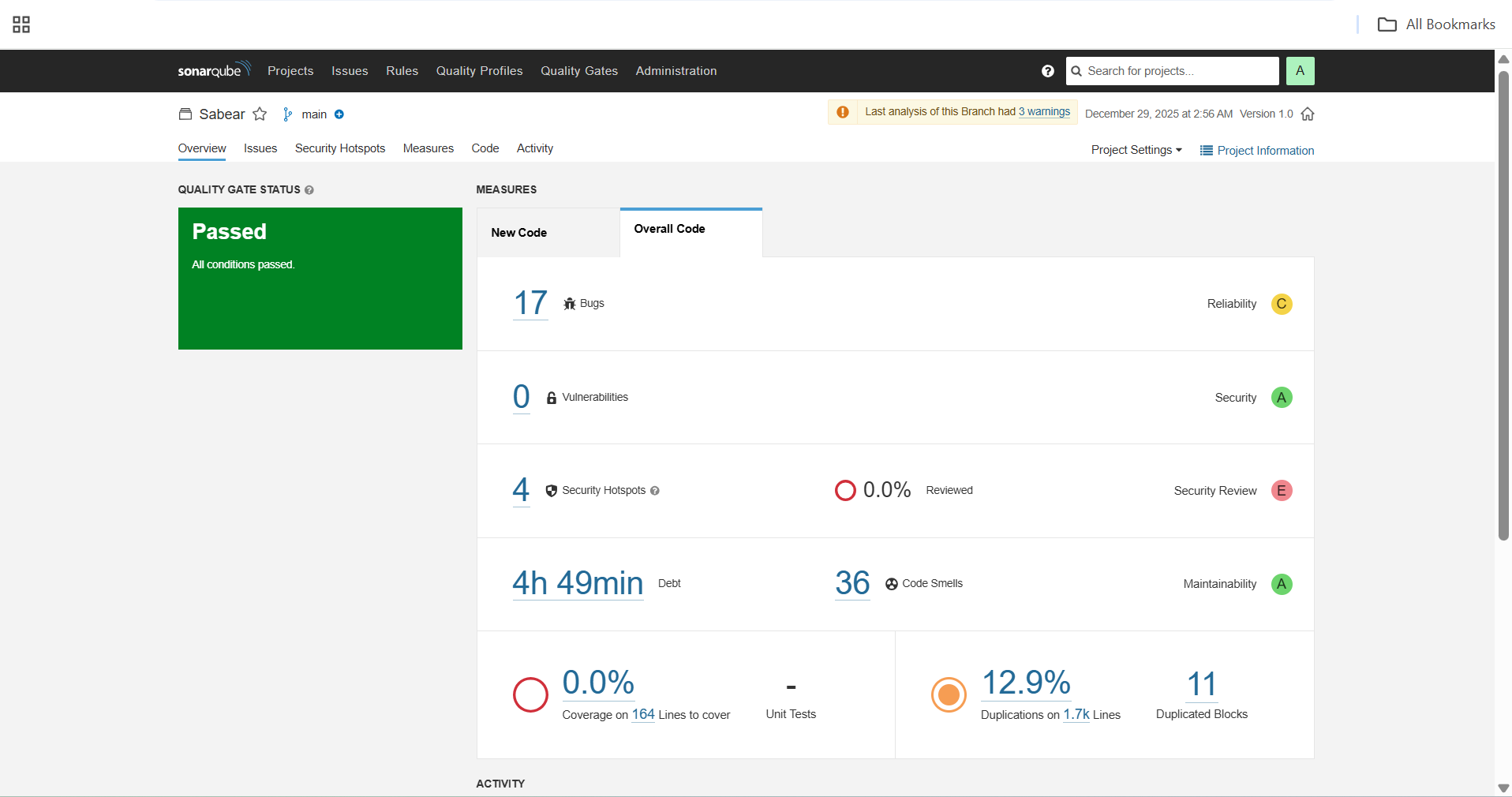












* **stage3**: Slack Integration to send the alerts to slack.

**Slack Workspace and Channel Preparation**

used an existing Slack workspace and created/used a dedicated channel for Jenkins alerts.

**Explanation**  
Slack is used to receive real-time Jenkins build notifications so the team can monitor pipeline status instantly.

**What you did**

* Logged into **Slack**
* Selected workspace techiehorizon
* Used channel #jenkins-integration

**Jenkins CI App Installation in Slack**

installed the Jenkins CI app from Slack Marketplace.

**Explanation**  
This app allows Jenkins to send build notifications directly to Slack channels.

**What you did**

* Opened **Browse apps** in Slack
* Searched for **Jenkins CI**
* Clicked **Add to Slack**
* Selected channel #jenkins-integration
* Completed Jenkins CI integration setup

**Jenkins Slack Notification Plugin Verification**

confirmed the Slack Notification Plugin is installed in Jenkins.

**Explanation**  
This plugin enables Jenkins to communicate with Slack using webhook/token-based authentication.

**Path**

Manage Jenkins → Plugins → Installed Plugins

**Observed**

* Slack Notification Plugin installed and enabled

**Slack Integration Token Generation**

generated an integration token required for Jenkins authentication.

**Explanation**  
The token securely allows Jenkins to post messages to Slack without exposing user credentials.

**What you did**

* Opened Jenkins CI app configuration in Slack
* Copied the generated integration token
* Noted that token is visible only once

**Jenkins Credentials Configuration for Slack**

securely stored the Slack token in Jenkins credentials.

**Explanation**  
Jenkins credentials store secrets securely and reuse them across jobs.

**Path**

Manage Jenkins → Credentials → Global → Add Credentials

**Configuration**

* Kind: Secret Text
* Scope: Global
* Secret: <Slack integration token>
* ID: slack

**Jenkins Global Slack Configuration**

configured Slack settings globally in Jenkins.

**Explanation**  
Global configuration allows all Jenkins jobs to use Slack without repeated setup.

**Path**

Manage Jenkins → System → Slack

**Values configured**

* Workspace: techiehorizon
* Credential: slack
* Default channel: #jenkins-integration

**Validation**

* Clicked **Test Connection**
* Verified success message

**Jenkins Job Post-Build Slack Configuration**

You enabled Slack notifications at the job level.

**Explanation**  
Post-build actions define when Jenkins should notify Slack (success, failure, start).

**Path**

Job (stage\_1) → Configure → Post-build Actions

**Slack notification options enabled**

* Notify Build Start
* Notify Success
* Notify Every Failure

**Jenkins Build Execution with Slack Alerts**

triggered Jenkins builds to test Slack integration.

**Explanation**  
Each build event sends messages to Slack in real time.

**Action**

Build Now

**Observed in Slack**

* Build started notification
* Build failure notification
* Build success notification
* Multiple build events correctly posted

**Slack Notification Validation**

verified Jenkins messages in Slack channel.

**Explanation**  
This confirms end-to-end integration between Jenkins and Slack.

**Observed**

* Jenkins bot posting messages
* Build number, status, and duration visible
* Links to Jenkins job included
* Success and failure messages displayed correctly

