

Documentation for Jenkins CI/CD Project

Pipeline Overview.....	2
Tools Integrated.....	2
Configuration Steps:.....	3
1. AWS EC2 Setup for Jenkins Server:.....	3
Detailed Pipeline Steps:.....	8
1. Checkout Code:.....	8
2. Build and Test:.....	8
3. JaCoCo Coverage Report:.....	8
4. SonarQube Analysis:.....	9
5. Sonar Quality Gate Scan:.....	9
6. Lizard Test:.....	11
7. OWASP Dependency Check:.....	11
8. Docker Build & Push:.....	12
9. Deploy to Container:.....	12
Troubleshooting Steps.....	12
1. Docker Permission Denied Error.....	12
2. Maven Build Failures.....	13
3. SonarQube Analysis Failures.....	13
4. Docker Build Issues.....	13
5. Test Failures.....	13

Pipeline Overview

The Jenkins pipeline performs the following tasks in a continuous integration/continuous deployment (CI/CD) workflow:

1. **Checkout Code:** Retrieves the latest code from the specified Git repository.
2. **Build and Test:** Builds the application using Maven and runs tests to validate code functionality.
3. **JaCoCo Coverage Report:** Generates and publishes a code coverage report using JaCoCo.
4. **SonarQube Analysis:** Analyzes the code quality with SonarQube to ensure that it adheres to the set standards.
5. **Sonar Quality Gate Scan:** Waits for the SonarQube quality gate to pass before proceeding.
6. **Lizard Complexity Analysis:** Analyzes the code complexity using Lizard.
7. **OWASP Dependency Check:** Scans the application dependencies for known vulnerabilities using OWASP Dependency Check.
8. **Docker Build & Push:** Builds a Docker image, tags it, and pushes it to Docker Hub.
9. **Deploy to Container:** Deploys the built Docker image to a container for testing or production purposes.
10. **Post Notification:** Integrated gmail as post notification for failure and success notification.

Tools Integrated

The following tools are integrated into the pipeline to ensure automated code quality checks, building, and deployment:

- **Git:** For version control and code management.
- **Maven:** For building and testing Java applications.
- **JaCoCo:** For generating code coverage reports.
- **SonarQube:** For analyzing and ensuring code quality.
- **Lizard:** For analyzing the complexity of the codebase.
- **OWASP Dependency Check:** For scanning project dependencies for known vulnerabilities.
- **Docker:** For building and pushing Docker images.
- **Email Notifications:** For sending success and failure alerts.


Configuration Steps:

1. AWS EC2 Setup for Jenkins Server:

- EC2 Instance: Created an EC2 instance with instance type t2.medium on AWS using Amazon ubuntu image to host Jenkins.






sg-03802a9b6e8750b93 - launch-wizard-1

Instances (1) [Info](#)

Last updated less than a minute ago  [Connect](#) [Instance state ▼](#) [Ac](#)

[All states ▼](#)

[Instance state = running](#) [Clear filters](#)

<input type="checkbox"/>	Name 	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Availability Zone ▼
<input type="checkbox"/>	Jenkins-Server	i-0019d198f7396b714	 Running  	t2.medium	 2/2 checks passed	View alarms +	ap-south-1a

- Generated a key-value pair for secure access purposes.
- Security Group: Configured inbound rules for HTTP (8080)/HTTPS(443) , SSH (port 22), Sonarqube port (9000) and Jenkins port (8080) to allow Jenkins access and server management.

Security groups

 [sg-03802a9b6e8750b93 \(launch-wizard-1\)](#)

▼ Inbound rules

<input type="text" value="Filter rules"/>			
Name	Security group rule ID	Port range	Protocol
–	sgr-0efba26c0d47d3230	80	TCP
–	sgr-0f2137b44f882a53c	5432	TCP
–	sgr-08afff8289f758973	9000	TCP
–	sgr-07ad9f4e848ad77f3	443	TCP
–	sgr-0ecd7abd76beec30f	8080	TCP
–	sgr-00f9321c97412b92e	22	TCP

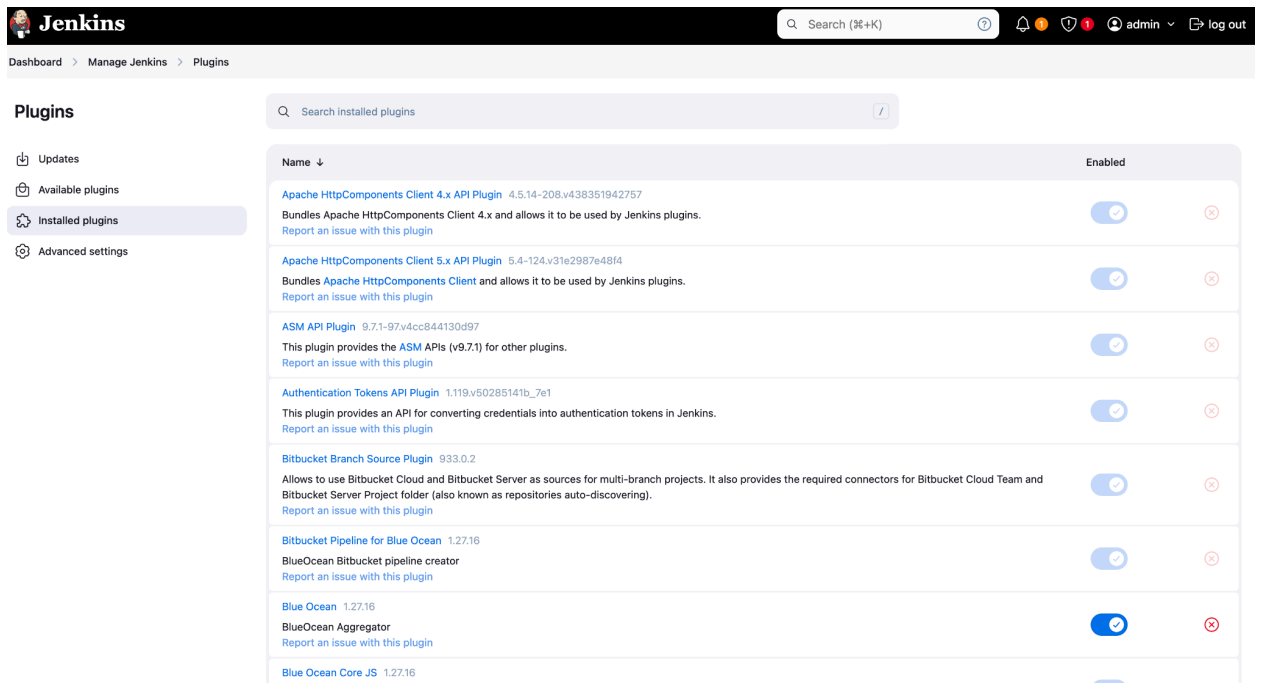
2. Jenkins Installation: Installed jenkins on ec2 machine.

```
-----
jbuntu@ip-172-31-36-75:~$ sudo systemctl status jenkins
jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Fri 2024-12-20 07:18:49 UTC; 3h 17min ago
     Main PID: 21971 (java)
        Tasks: 67 (limit: 4676)
      Memory: 958.1M (peak: 1.3G)
         CPU: 9min 53.916s
       CGroup: /system.slice/jenkins.service
               └─21971 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/c

Dec 20 08:43:08 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:43:08.026+0000 [id=1285] INFO c.c.
Dec 20 08:43:27 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:43:27.436+0000 [id=1324] INFO h.p.
Dec 20 08:43:27 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:43:27.442+0000 [id=1324] INFO h.p.
Dec 20 08:43:37 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:43:37.781+0000 [id=17] INFO o.s.s.
Dec 20 08:44:25 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:44:25.270+0000 [id=1324] INFO h.p.
Dec 20 08:44:25 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:44:25.272+0000 [id=1324] INFO h.p.
Dec 20 08:44:37 ip-172-31-36-75 jenkins[21971]: 2024-12-20 08:44:37.593+0000 [id=15] INFO o.s.s.
Dec 20 10:12:02 ip-172-31-36-75 jenkins[21971]: 2024-12-20 10:12:02.146+0000 [id=1565] INFO h.p.
Dec 20 10:12:02 ip-172-31-36-75 jenkins[21971]: 2024-12-20 10:12:02.150+0000 [id=1565] INFO h.p.
Dec 20 10:12:13 ip-172-31-36-75 jenkins[21971]: 2024-12-20 10:12:13.038+0000 [id=17] INFO o.s.s.
jbuntu@ip-172-31-36-75:~$
```

3. Jenkins Plugin:

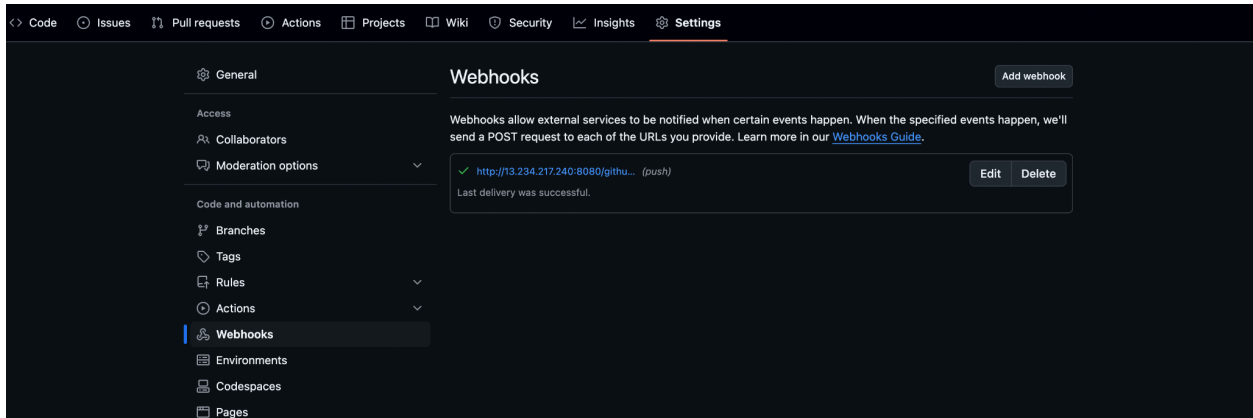
- Installed all the plugin required to integrate all the necessary tools.



The screenshot shows the Jenkins web interface. At the top, there's a header with the Jenkins logo, a search bar, and user information (admin). Below the header, the 'Plugins' page is displayed. On the left, there's a sidebar with navigation links: Updates, Available plugins, Installed plugins (selected), and Advanced settings. The main content area shows a list of installed plugins. Each plugin entry includes its name, version, a brief description, and a toggle switch in the 'Enabled' column. The plugins listed are:

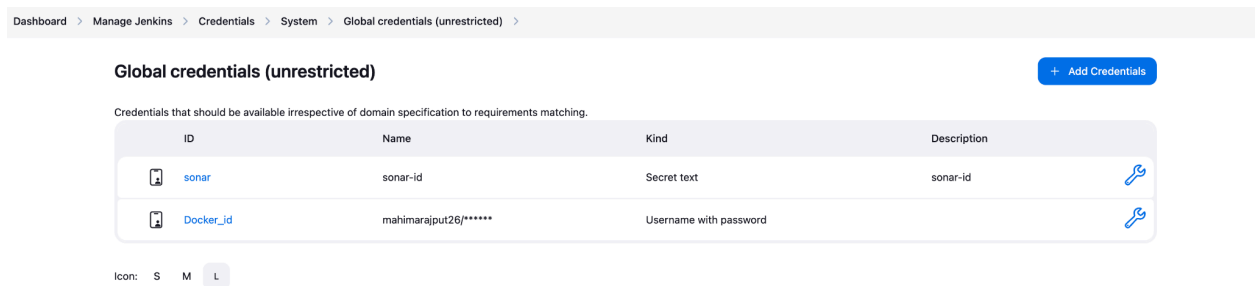
Name	Enabled
Apache HttpComponents Client 4.x API Plugin 4.5.14-208.v438351942757 Bundles Apache HttpComponents Client 4.x and allows it to be used by Jenkins plugins. Report an issue with this plugin	<input checked="" type="checkbox"/>
Apache HttpComponents Client 5.x API Plugin 5.4-124.v31e2987e48f4 Bundles Apache HttpComponents Client and allows it to be used by Jenkins plugins. Report an issue with this plugin	<input checked="" type="checkbox"/>
ASM API Plugin 9.7.1-97.v4cc844130d97 This plugin provides the ASM APIs (v9.7.1) for other plugins. Report an issue with this plugin	<input checked="" type="checkbox"/>
Authentication Tokens API Plugin 1.119.v50285141b_7e1 This plugin provides an API for converting credentials into authentication tokens in Jenkins. Report an issue with this plugin	<input checked="" type="checkbox"/>
Bitbucket Branch Source Plugin 933.0.2 Allows to use Bitbucket Cloud and Bitbucket Server as sources for multi-branch projects. It also provides the required connectors for Bitbucket Cloud Team and Bitbucket Server Project folder (also known as repositories auto-discovering). Report an issue with this plugin	<input checked="" type="checkbox"/>
Bitbucket Pipeline for Blue Ocean 1.27.16 BlueOcean Bitbucket pipeline creator Report an issue with this plugin	<input checked="" type="checkbox"/>
Blue Ocean 1.27.16 BlueOcean Aggregator Report an issue with this plugin	<input checked="" type="checkbox"/>
Blue Ocean Core JS 1.27.16	<input checked="" type="checkbox"/>

4. Github Repo setup with webhook:



5. Set Up Jenkins Credentials:

- Store sensitive information like API tokens, passwords, Sonar and Docker credentials securely in Jenkins' **Credentials Store**.



6. SonarQube Configuration:

- Configure SonarQube within Jenkins by adding the **SonarQube server** in the Jenkins Global Tool Configuration. Set up of Sonar-Scanner with the required version of **SonarScanner**.

SonarQube servers

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

Server URL

Default is http://localhost:9000

http://13.234.217.240:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

sonar-id

+ Add

SonarScanner for MSBuild installations

Add SonarScanner for MSBuild

SonarQube Scanner installations

SonarQube Scanner installations ^ Edited

Add SonarQube Scanner

SonarQube Scanner

Name

sonar-scanner

☒ Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 6.2.1.4610

Add Installer

7. Maven Configuration:

- Set up the Maven tool installation in Jenkins **Global Tool Configuration**, ensuring it's pointing to the correct Maven version.
-

8. Docker Configuration:

- Ensure that the Jenkins agent has Docker installed and is properly configured to build Docker images. It needs to have access to Docker's daemon and the necessary permissions to execute Docker commands.

9. Email Setup:

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix ?


Advanced ^ Edited

☒ Use SMTP Authentication ?

User Name

fourthmahima@gmail.com

Password

 Concealed

Change Password

☒ Use SSL ?

☐ Use TLS

SMTP Port ?

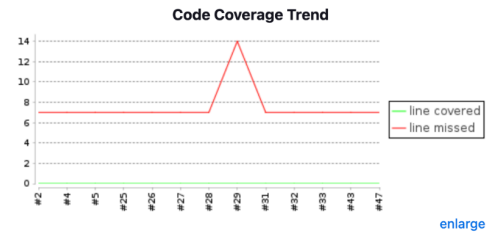
465

Reply-To Address

fourthmahima@gmail.com

Save Apply

Detailed Pipeline Steps:



Stage View

		Checkout Code	Build and Test	JaCoCo coverage report	SonarQube Analysis	Sonar Quality Gate Scan	Run Lizard	Docker Build & Push	Deploy to container	Declarative: Post Actions
Average stage times: (Average full run time: ~44s)		818ms	9s	127ms	9s	221ms	484ms	19s	590ms	3s
#50	Dec 20 13:30 No Changes	703ms	8s	119ms	8s	197ms (paused for 2s)	440ms	18s	1s	3s

1. Checkout Code:

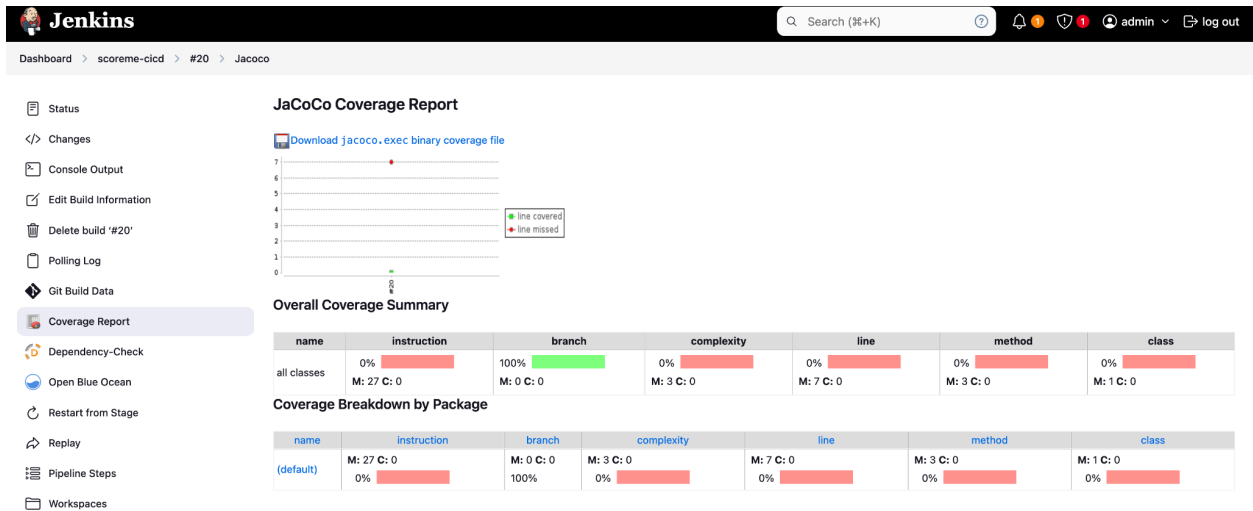
- The pipeline begins by checking out the latest code from the Git repository using the `git` step. This ensures the build always runs with the latest changes.

2. Build and Test:

- The pipeline uses Maven to build the project and run unit tests. The command `mvn clean package` builds the project, while `mvn clean test` runs the tests.
- A demo `.war` file is created during this process and moved into a designated directory (`pkg`).

3. JaCoCo Coverage Report:

- This stage generates a coverage report using JaCoCo and publishes it as part of the build results.

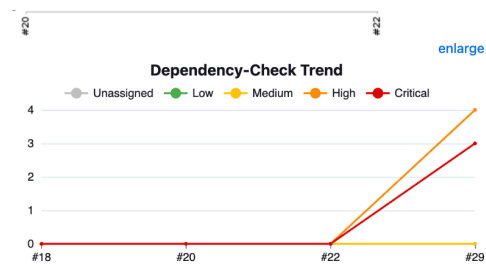


4. SonarQube Analysis:

- SonarQube analysis is triggered using the sonar-scanner tool. The pipeline uses SonarQube's sonar-scanner to analyze code quality and send the results to the SonarQube server.

5. Sonar Quality Gate Scan:

- This step waits for the quality gate to pass before moving to the next stage. If the quality gate fails, the pipeline will stop.



	Declarative: Checkout SCM	Checkout Code	Build and Test	JaCoCo coverage report	SonarQube Analysis	Sonar Quality Gate Scan	Run Lizard	Docker Build & Push	Deploy to container	Declarative: Post Actions
times:	1s	672ms	9s	109ms	11s	232ms	93ms	91ms	94ms	3s
	1s	672ms	9s	109ms	11s	232ms (paused for 2s) failed	93ms failed	91ms failed	94ms failed	3s

Quality Gates

Create

sonar way

DEFAULT

BUILT-IN

cicd-qualityGate

This quality gate complies with **Clean as You Code**

This quality gate complies with the [Clean as You Code](#) methodology, so that you benefit from the most efficient approach to delivering Clean Code. It ensures that:

- No new bugs are introduced
- No new vulnerabilities are introduced
- All new security hotspots are reviewed
- New code has limited technical debt
- New code has limited duplication
- New code is properly covered by tests

Conditions

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	8.0%
Maintainability Rating	is worse than	A (Technical debt ratio is less than 5.0%)
Reliability Rating	is worse than	A (No bugs)
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A (No vulnerabilities)

You may click unlock to edit this quality gate. Adding extra conditions to a compliant quality gate can result in drawbacks. Are you reconsidering [Clean as You Code](#)? We strongly recommend this methodology to achieve a Clean Code status.

Unlock editing

Projects

WithWithoutAll

Q Search

☒

testciold
testciold

Permissions

Users with the global 'Administer Quality Gates' permission and those listed below can manage this Quality Gate.

sonar-administrators

Dashboard

>

scoreme-cicd

>

#19

```

[Pipeline] stage
[Pipeline] { (Sonar Quality Gate Scan)
[Pipeline] timeout
Timeout set to expire in 2 min 0 sec
[Pipeline] {
[Pipeline] waitForQualityGate
Checking status of SonarQube task 'AZPhFGoJ_U7s9zdwHhnU' on server 'sonar-server'
SonarQube task 'AZPhFGoJ_U7s9zdwHhnU' status is 'PENDING'
SonarQube task 'AZPhFGoJ_U7s9zdwHhnU' status is 'SUCCESS'
SonarQube task 'AZPhFGoJ_U7s9zdwHhnU' completed. Quality gate is 'ERROR'
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Lizard Test)
Stage "Lizard Test" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (OWASP Dependency Check)
Stage "OWASP Dependency Check" skipped due to earlier failure(s)
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] mail
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: Pipeline aborted due to quality gate failure: ERROR
Finished: FAILURE
```

6. Lizard Test:

- Lizard is used to analyze the complexity of the code. This helps ensure that the codebase isn't overly complex or hard to maintain.

```
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Run Lizard)
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ /home/ubuntu/lizard-env/bin/lizard . --output report.txt
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Owasp Dependency)
[Pipeline] dependencyCheck
[INFO] Checking for updates
[INFO] Skipping the NVD API Update as it was completed within the last 240 minutes
```

7. OWASP Dependency Check:

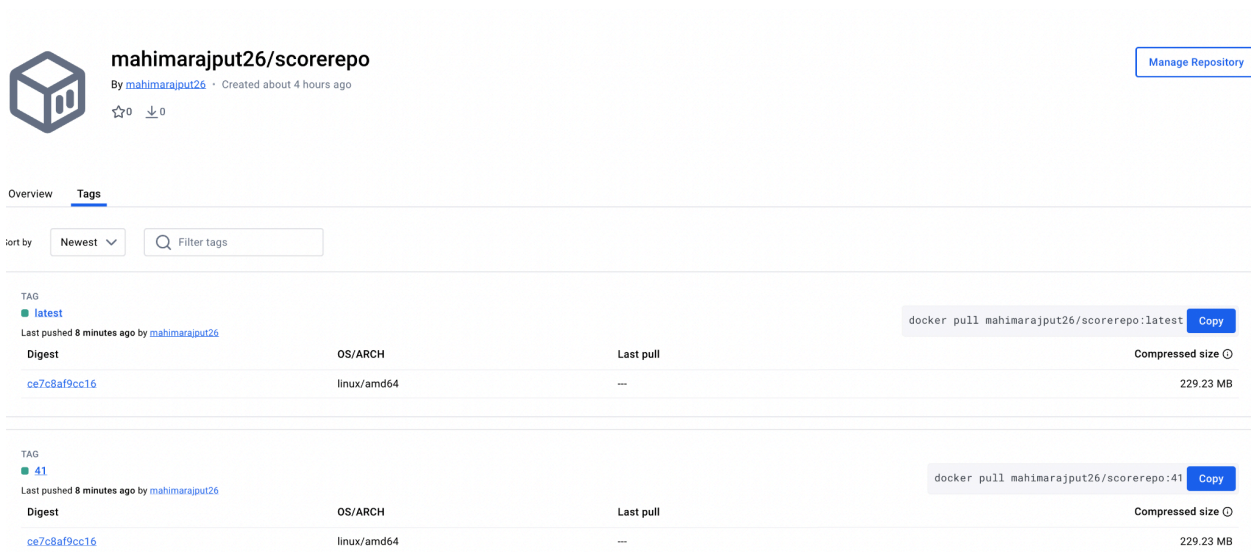
- This stage uses OWASP Dependency Check to scan project dependencies for known vulnerabilities.

The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search bar, and user information (admin). The breadcrumb trail indicates the path: Dashboard > scoreme-cicd > #20 > Dependency-Check. On the left sidebar, various build options are listed, with 'Dependency-Check' highlighted. The main content area is titled 'Dependency-Check Results' and features a 'SEVERITY DISTRIBUTION' section. A blue banner at the top of this section states 'No Vulnerabilities Found'. Below this is a table with columns for 'File Name', 'Vulnerability', 'Severity', and 'Weakness'. The table is currently empty, displaying 'No results' in the center.

File Name	Vulnerability	Severity	Weakness
No results			

8. Docker Build & Push:

- The pipeline builds a Docker image and pushes it to Docker Hub. The image is tagged with both the build number and the latest tag.



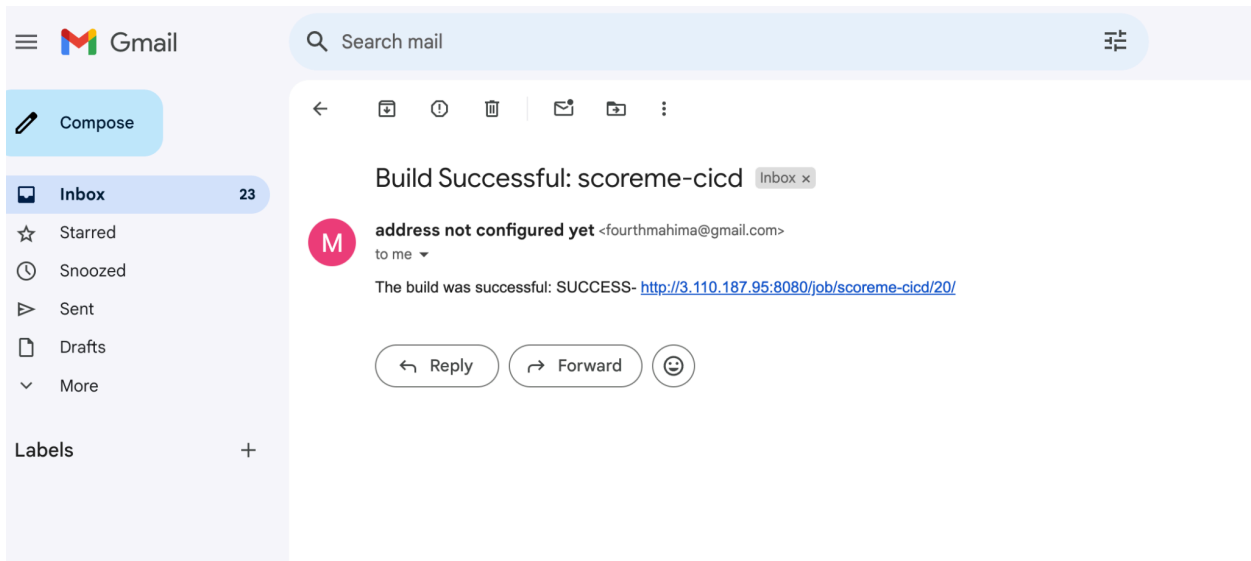
The screenshot shows the Docker Hub repository page for **mahimarajput26/scorerepo**. The repository was created about 4 hours ago. The **Tags** tab is selected, showing two tags: **latest** and **41**. Both tags were pushed 8 minutes ago by **mahimarajput26**. The table below lists the details for each tag.

TAG	Digest	OS/ARCH	Last pull	Compressed size
latest Last pushed 8 minutes ago by mahimarajput26	ce7c8af9cc16	linux/amd64	---	229.23 MB
41 Last pushed 8 minutes ago by mahimarajput26	ce7c8af9cc16	linux/amd64	---	229.23 MB

9. Deploy to Container:

- This stage runs the Docker container and exposes the application on port 8000.

10. Email Notification:



The screenshot shows a Gmail inbox with an email notification titled **Build Successful: scoreme-cicd**. The email is from **address not configured yet** (fourthmahima@gmail.com) and is addressed to the user. The body of the email states: "The build was successful: SUCCESS- <http://3.110.187.95:8080/job/scoreme-cicd/20/>". The email interface includes a left sidebar with navigation options (Compose, Inbox, Starred, Snoozed, Sent, Drafts, More) and a right sidebar with action buttons (Reply, Forward, Smile).

Troubleshooting Steps

Below are some common issues and troubleshooting steps:

1. Docker Permission Denied Error

- **Symptom:** permission denied while trying to connect to the Docker daemon socket.

Solution: Ensure the Jenkins user has the necessary permissions to interact with the Docker daemon. Add the Jenkins user to the Docker group:

```
sudo usermod -aG docker jenkins
```

2. Maven Build Failures

- **Symptom:** Errors like missing dependencies or plugin errors.
- **Solution:** Verify that Maven is properly configured and that all dependencies are available. Also, check the repository for any issues with dependencies or plugins.

3. SonarQube Analysis Failures

- **Symptom:** SonarQube reports are not generated or are incomplete.
- **Solution:** Ensure that the sonar-scanner is correctly configured and that the SonarQube server is accessible. Also, verify that the necessary authentication tokens are correctly set in Jenkins' **Credentials Store**.

4. Docker Build Issues

- **Symptom:** Docker build fails with errors related to missing files or incorrect Dockerfile.
- **Solution:** Check that the Dockerfile is in the correct location and properly configured. Ensure all required files (e.g., demo.war) are available and correctly named in the Docker context.

5. Test Failures

- **Symptom:** Tests fail during the Maven build.
- **Solution:** Review the test logs to identify the cause of the failure. Common causes include incorrect test configurations or dependencies. Ensure that the tests are passing locally before triggering them in Jenkins.