

Continuous Integration Using Jenkins, Nexus, SonarQube, Slack

Benefits

Short MTTR - Mean Time To Repair
Fault Isolation
Agile
No Human Intervention Required

Tools used

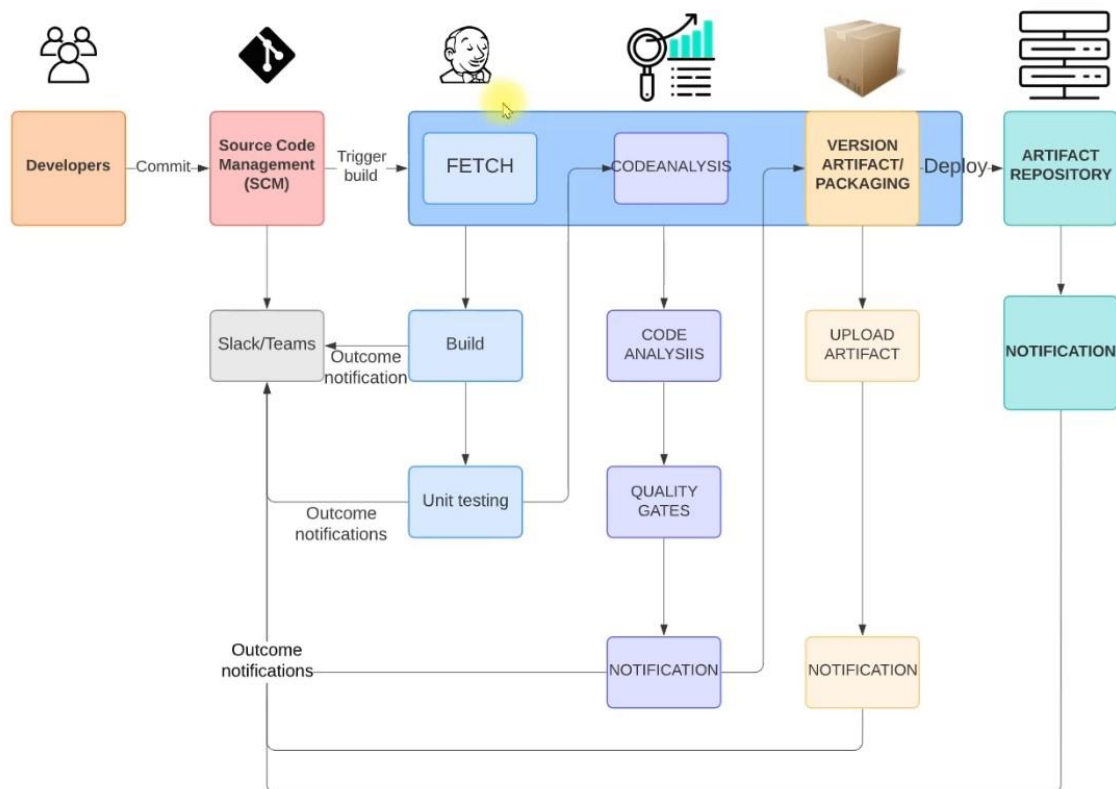
Jenkins, GIT, Maven, Checkstyle, Slack, Sonatype Nexus, Sonarqube, AWS EC2

Goals Achieved

Fast TAT on Feature changes
Less Disruptive

Workflow

The below flowchart illustrates, how the code passes through every stage.



- 1) Developers Commits the Code into GIT(Source Code Management). This application is built on Java, so we are using Maven as build tool.
- 2) Jenkins fetches the code and builds(compiles) it, runs Unit Tests on it and write the outcome as a notification in Slack/Teams
- 3) Next it runs the Code Analysis using Sonarqube and puts the code through Quality Gates and if it passes the quality thresholds it will send the notification to Slack/Teams

- 4) If it is not passing the quality thresholds - It will be passed to Developer to fix the bugs and the process begins from scratch
- 5) After passing through Code Analysis, the artifact will be packaged and uploaded to the artifact repository that is Nexus and a notification will be sent through Slack/Teams

Flow of Execution

- ❖ Login to AWS Account
- ❖ Create login key
- ❖ Create Security Group
 - Jenkins
 - Nexus
 - Sonar
- ❖ Create EC2 instances with user data
 - Jenkins
 - Nexus
 - Sonar
- ❖ Jenkins Post Installation
- ❖ Nexus Repository Setup
- ❖ SonarQube post installation
- ❖ Jenkins Steps
 - Build Job
 - Setup Slack Notification
 - Check style Code Analysis Job
 - Setup Sonar integration
 - Sonar code analysis job
 - Artifact upload Job
- ❖ Connect all jobs together with BuildPipeline
 - Set Automatic Build Trigger
- ❖ Test with VSCode or IntelliJ

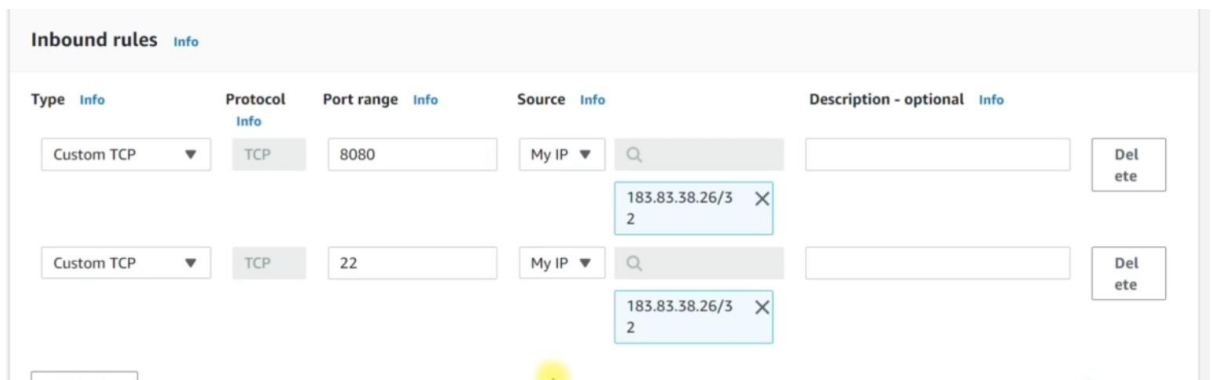
Detailed Description of Steps to follow

- Login into AWS Management Console. Open the EC2 Pane
- Before creating the EC2 instances, we need to create Key pairs and Security Groups for the EC2 instances
- EC2 → Key pairs
 - Give a name for the key pair
 - File format - .pem

- EC2 → Security Groups
 - Give names for security group (Below are examples)
 - dprofile-jenkins-sg
 - dprofile-nexus-sg
 - dprofile-sonar-sg

dprofile-jenkins-sg

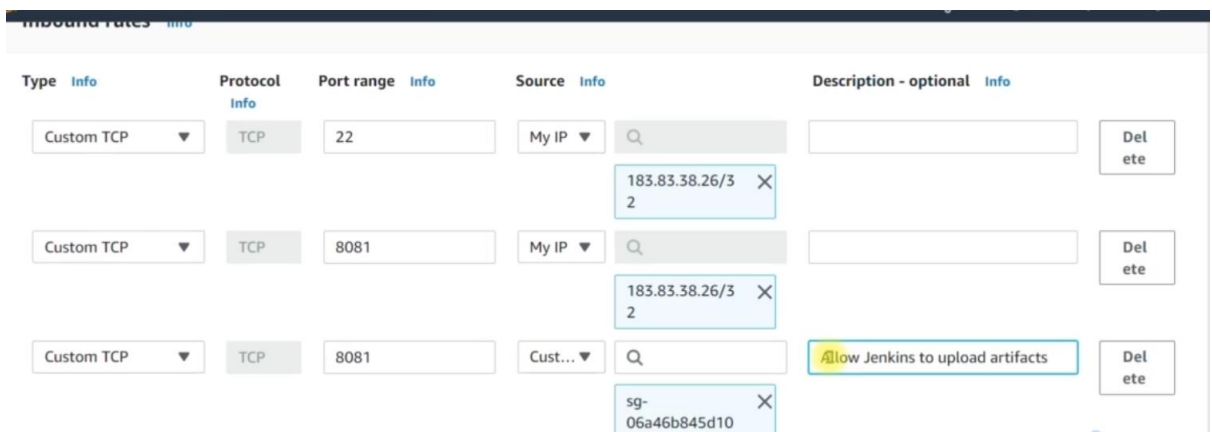
- ✓ Provide the Group Name (dprofile-jenkins-sg) and Short Description.
- ✓ Create Inbound Rules –
 - Port 8080 and 22 from the machine IP address, where you will manage the instances. All Traffic or Port 80 from SonarQube security group (Allow Sonar to access Jenkins for quality gates result)
- ✓ Create Outbound Rules –
 - Allow All traffic



Type	Protocol	Port range	Source	Description - optional	Info
Custom TCP	TCP	8080	My IP 183.83.38.26/32		Info
Custom TCP	TCP	22	My IP 183.83.38.26/32		Info

dprofile-nexus-sg

- ✓ Provide the Group Name (dprofile-nexus-sg) and Short Description.
- ✓ Create Inbound Rules –
 - Port 8081 and 22 from the machine IP address, where you will manage the instances. Source Port (8081) from Jenkins Security Group - To allow the upload of artifacts
- ✓ Create Outbound Rules –
 - Allow All traffic



Type	Protocol	Port range	Source	Description - optional	Info
Custom TCP	TCP	22	My IP 183.83.38.26/32		Info
Custom TCP	TCP	8081	My IP 183.83.38.26/32		Info
Custom TCP	TCP	8081	Cust... sg-06a46b845d10	Allow Jenkins to upload artifacts	Info

dprofile-sonar-sg

- ✓ Provide the Group Name (dprofile-sonar-sg) and Short Description.
- ✓ Create Inbound Rules –
Port 8081 and 22 from the machine IP address, where you will manage the instances.
Source Port Range (8081) from Jenkins Security Group - To allow the upload of artifacts
- ✓ Create Outbound Rules –
Allow All traffic

We have to create tags in Security Groups page.

javaprofile-jenkins-sg

javaprofile-nexus-sg

javaprofile-sonar-sg

Launch EC2 instances

1. Provision the instances using userdata. Start with Jenkins instance

Github → UserData Folder → jenkins-setup.sh, nexus-setup.sh and sonar-setup.sh

2. Clone the Github repository – ci-jenkins. After cloning, always run “git pull” to update the latest changes. Switch to the branch – “git checkout ci-jenkins”
3. Navigate to “userdata” folder. Shell scripts are available to create the EC2 instances for Jenkins, Nexus and Sonarqube
4. Install all roles using Ubuntu 22.04.
5. AWS Portal → EC2 → Launch an instance → Select t2.small (which will have some charges)
6. Fill the details. Under Advanced Details section → User Data → fill it with the shell script for nexus from userdata. Repeat the steps 5 and 6 for Jenkins and Sonarqube servers
7. Provide proper tags to identify the instances
Name – Nexus Server
Project – Javaprofile
8. Select the respective security groups and security key pairs created in the beginning of this tutorial.

Post installation of Jenkins

1. To complete post installation of Jenkins, access the Jenkins portal with <public IP>:8080



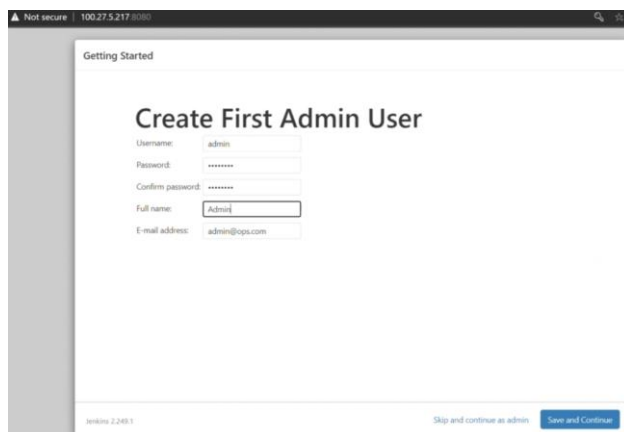
2. To get the initial password for administrator account, we need to access the server through SSH with default user name “ubuntu” and with the key file to login into the AWS CLI
3. Switch to root user – “sudo -i”
4. To get the password – open the file “cat /var/lib/jenkins/secrets/initialAdminPassword
5. Paste the password into the portal and continue the setup



6. Select “Install suggested plugins”



7. Create the Admin User



8. Confirm Instance Configuration and complete the installation.
Before we configure a job, we need to perform Nexus repository.

Post installation of Nexus

1. Access the VM with the public IP address with port 8081. <public ip>:8081
2. To sign in, we need to SSH into the Nexus VM with the key file and the default user is “admin”
3. The password is in the file “/opt/nexus/sonatype-work/nexus3/admin.password”.
4. Post installation wizard opens. Follow the instructions to finish the post installation steps.
5. We need to create three maven repositories. Administration → Repository → Repositories → Create Repository
 - a. Select maven2(hosted) **javaprofile-release** (This repository will have artifacts which are well tested and deployed to servers)
 - b. Select maven2(proxy) **javaprofile-maven-central**. We have provide the remote repository address to download all dependencies (This repository will have all the dependencies downloaded from the maven public repository)

Proxy

Remote storage:

Location of the remote repository being proxied, e.g. <https://repo1.maven.org/maven2/>

<https://repo1.maven.org/maven2/>

Use the Nexus truststore:

- c. Select maven2(group) **javaprofile-maven-group**. Make the above two repositories as a member repositories under this group repository.

Group

Member repositories:

Select and order the repositories that are part of this group

Available

Members

6. We can create a snapshot repository (optional). Select maven2(hosted) and select “Snapshot” under Version Policy, Add this snapshot repository also to the “javaprofile-maven-group” as a member.

The screenshot shows the 'Create Repository' form for a maven2 (hosted) repository. The 'Name' field is required and empty. The 'Online' checkbox is checked. The 'Version policy' is set to 'Snapshot'. The 'Layout policy' is set to 'Validate that all paths are maven artifact or metadata paths'.

7. To configure maven to download the dependencies and place those under “**javaprofile-maven-central**”, we need to configure the settings.xml. We will pass these variables from maven build job that we are going to configure as a Jenkins job.

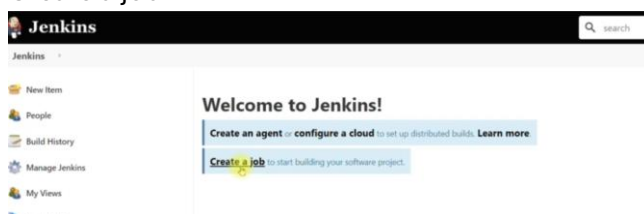
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0 http://maven.apache.org/xsd/
   settings-1.1.0.xsd">
5
6   <servers>
7     <server>
8       <id>${SNAP-REPO}</id>
9       <username>${NEXUS-USER}</username>
10      <password>${NEXUS-PASS}</password>
11    </server>
12    <server>
13      <id>${RELEASE-REPO}</id>
14      <username>${NEXUS-USER}</username>
15      <password>${NEXUS-PASS}</password>
16    </server>
17    <server>
18      <id>${CENTRAL-REPO}</id>
19      <username>${NEXUS-USER}</username>
20      <password>${NEXUS-PASS}</password>
21    </server>
22    <server>
23      <id>${NEXUS-GRP-REPO}</id>
24      <username>${NEXUS-USER}</username>
25      <password>${NEXUS-PASS}</password>
26    </server>
27  </servers>
28
29  <profiles>
30  </profiles>
31 </settings>

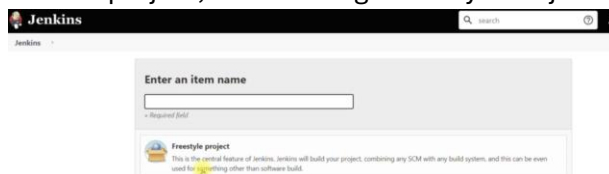
```

Nexus Integration Job

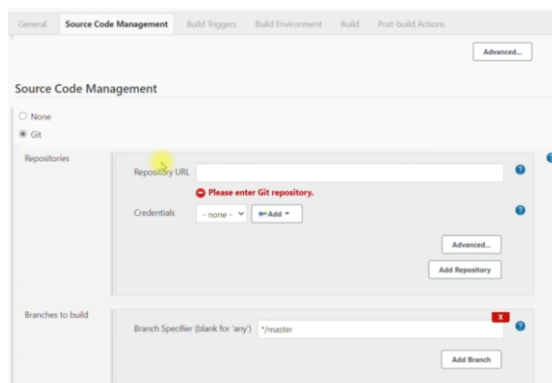
1. Login into Jenkins - <publicip>:8080
2. Create a job



3. For this project, we are using “Freestyle Project”



4. Give unique name for the project, description, provide the git url and branch name to fetch source code



5. In Build section, select “Invoke top-level Maven targets” and for goals mention “install -DskipTests”. This is will use the pom.xml file, which is already in the source code. For Settings, use the filesystem option and mention the path of the settings.xml file.

6. Assign variables for the settings.xml

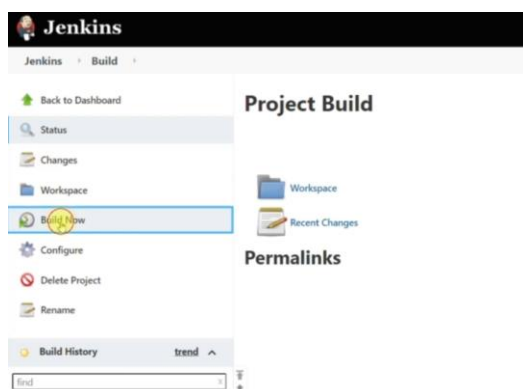
```

1 SNAP-REPO=
2 NEXUS-USER
3 NEXUS-PASS
4 RELEASE-REPO
5 CENTRAL-REPO
6 NEXUS-GRP-REPO
7 NEXUSIP
8 NEXUSPORT

```

7. After configuring the variables, paste it under “properties”

8. Save this job and build the job.



9. Build job is successful.

```

Downloaded from srepo-maven-central: http://172.31.45.141:8082/repository/srepo-maven-group/org/apache/plexus/plexus
[INFO]
Progress (3): 200/230 kB
Progress (3): 200/230 kB
Progress (3): 200/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Progress (3): 227/230 kB
Downloaded from srepo-maven-central: http://172.31.45.141:8082/repository/srepo-maven-group/org/apache/plexus/plexus
[INFO]
[REX-Shell@0000] Installing /var/lib/jenkins/workspace/Build/target/plexus-v2.jar to /var/lib/jenkins/.m2/repository
[REX-Shell@0000] Installing /var/lib/jenkins/workspace/Build/pom.xml to /var/lib/jenkins/.m2/repository/com/ctoolops
[REX-Shell@0000] [INFO]-----[INFO]
[REX-Shell@0000] [INFO]BUILD SUCCESS[INFO]-----[INFO]
[REX-Shell@0000] [INFO]-----[INFO]
[REX-Shell@0000] Total time: 12.448 s
[REX-Shell@0000] Finished at: 2020-09-30T07:46:49Z
[REX-Shell@0000] [INFO]-----[INFO]

```

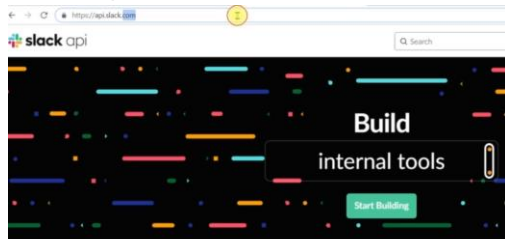

Slack Integration

Build job is successful and the developers should be notified regarding this. So we are going to integrate with Slack to notify the developers and managers.

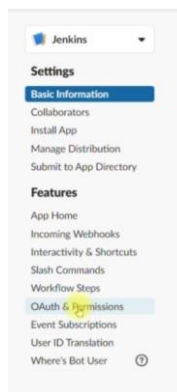
1. Create a slack account.
2. Create a workspace
3. Create a channel – provide a name and description as per requirement

Create an API for slack

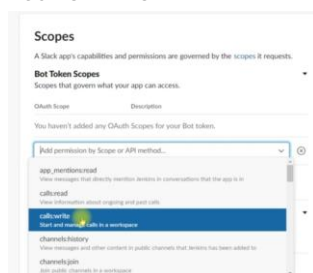
1. Go to “api.slack.com” and click on “Start building”



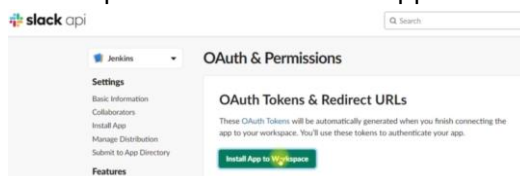
2. Click on “Create an app”, Give a name and select the workspace, and then “Create app”
3. Go to settings of the app and click on “Oauth & Permissions”



4. Scroll down to see “Scopes” and under “Bot Token Scopes” add the permission for “calls:write”

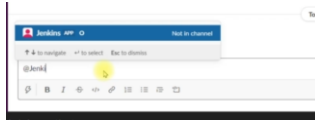


5. Scroll up and click on “Install App to Workspace”

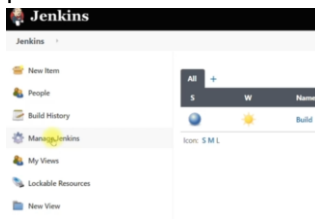


6. You will be provided with a token. Copy the token and save it in the notepad.

7. We need to add the app into the channel and invite the app to the channel.



8. To make use of the slack bot, we need to install a plugin in Jenkins. Head to Jenkins portal and click on “Manage Jenkins”



9. Manage Jenkins → Manage Plugins → Available → Search for “Slack” → “Install without Restart”



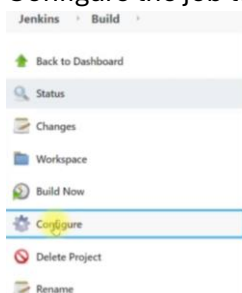
10. To configure the token, Navigate to Manage Jenkins → Manage Credentials → Store ‘Jenkins’ → ‘Global Credentials’ → Add Credentials → Change the kind to “secret text”



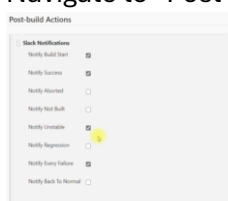
11. Now we will integrate with Slack with Jenkins. Navigate to Manage Jenkins → Configure System → Scroll to the end for Slack settings → Put checkmark on “Custom Slack app bot user” and save the settings.



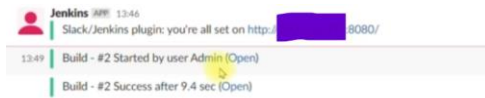
12. Configure the job to send notifications.



13. Navigate to “Post build Actions”



14. Click on Advanced, Enter the workspace name, credential (slack-token), channel/member id - #jenkins → Test Connection
15. Go back to slack channel and you would have got a notification.



Unit Tests, Code Analysis with Sonarqube

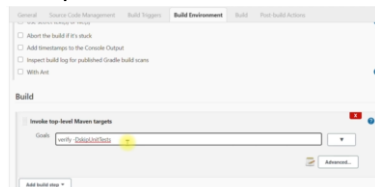
1. We are going to create a job in Jenkins called as “Test” and as a Freestyle Project and copy from the Build job that we created earlier.
2. In the Build phase, provide the goals as “test”, rest of the settings copied from “build” job.



3. Run the test job.
4. This test job should automatically run after the build job, so to configure this, Navigate to “Build” job → Configure → “Build” Tab → Add post-build action → Select “Build Other projects” → Projects to build as “Test”

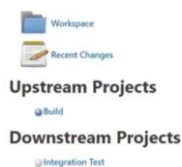


5. Configure Integration job. Freestyle Job → “Build Environment” tab → Goals → “verify - DskipUnitTests” → Rest of the settings comes from “Build” job.



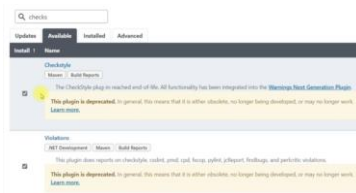
6. This integration job should automatically run after the test job, so to configure this, Navigate to “Build” job → Configure → “Build” Tab → Add post-build action → Select “Build Other projects” → Projects to build as “Integration Test”
7. So upstream job is “Build” and downstream job is “Integration Test”

Project Test

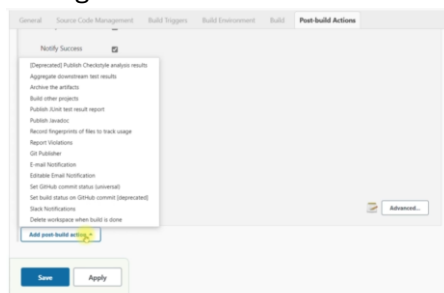


8. To perform simple code analysis, we have to install a plugin called “checkstyle”

9. Manage Jenkins → Manage Plugins → Available → Search for “checks” → Select “Checkstyle” and “Violations” → “Install without Restart”



- 10.** Checkstyle will analyze the code and give the report. If the failures goes beyond threshold defined by quality gates, Violations will make the job unstable, so it won't be promoted to next step.
- 11.** Create a job in Jenkins for code analysis. Create new job and copy the settings of build job. For goals, enter "checkstyle:checkstyle".
- 12.** Configure "Post Build Actions" → "Publish Checkstyle analysis results"



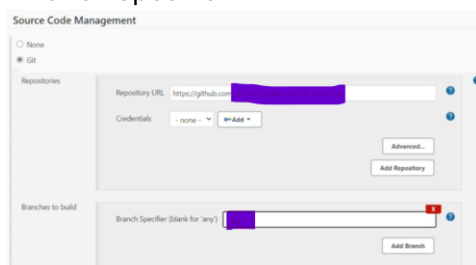
- 13.** Run the job and results will be saved in the workspace

```
[H1] 34a3f00[n] 03a-----0[m]
[CHECKSTYLE] Collecting checkstyle analysis files...
[CHECKSTYLE] Searching for all files in /var/lib/jenkins/workspace/code_analysis that match the pattern **/checkstyle-result.xml
[CHECKSTYLE] Parsing 1 file in /var/lib/jenkins/workspace/code_analysis
[CHECKSTYLE] Successfully parsed file /var/lib/jenkins/workspace/code_analysis/target/checkstyle-result.xml with 96 unique warnings and 0 duplicates.
```

14. Create a new job as “Code_Analysis” and copy the settings of “Build” job
15. In “Build Environment” tab → Invoke top-level Maven targets → Goals
“checkstyle:checkstyle”

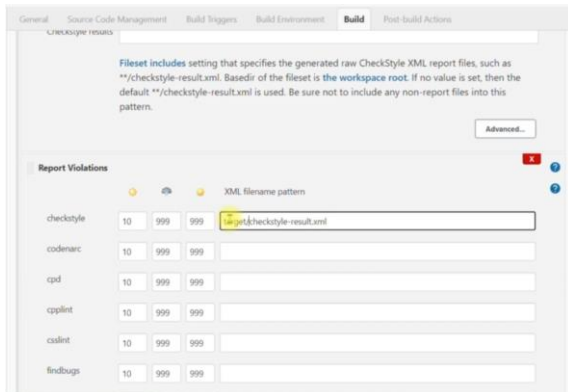


- 16.** If you want to perform the Quality Gates checks for the source code in a different branch Under “Source Code Management” → Select “Git” → Enter Repository Url → Then “Branch Specifier”



17. If you want to perform the quality checks for code in the branch, either create a `settings.xml` file in the branch or use the default maven settings.

18. Under build tab, we configure the violations and stop the code to get promoted to next level if the quality checks are not passed.

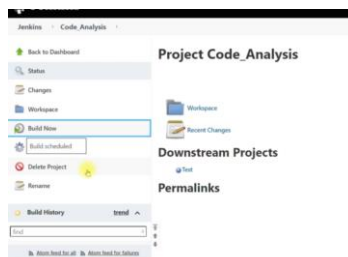


19. After configuring and testing the job, we can integrate the job. Jenkins Home → Configure → Integration Test → Make the “Code_Analysis” job as the downstream job of “Integration Test” job using Post Build Actions

	W	Name	Last Success	Last Failure	Last Duration
		Build	22 min - 42	N/A	0.4 sec
		Code_Analysis	43 sec - #6	8 min 21 sec - #3	0.3 sec
		Integration Test	14 min - #1	N/A	17 sec
		Test	20 min - #1	N/A	15 sec

20. Now we are going to analyse the code with checkstyle

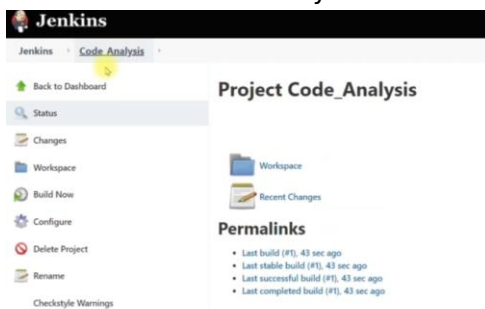
21. Run the “Code Analysis” job to the publish the results



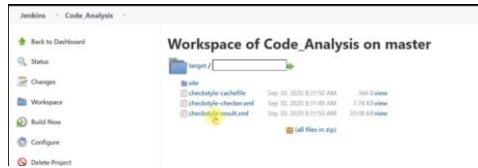
22. Click “Console Output” and there we can find the location of the published results in .xml format

```
[H1]Maven[INFO] Finished at: 2020-09-30T08:31:50Z
[H1]Maven[INFO] B[1]e-----B[1]e
[CHECKSTYLE] Collecting checkstyle analysis files...
[CHECKSTYLE] Searching for all files in /var/lib/jenkins/workspace/Code_Analysis that match the pattern **/checkstyle-result.xml
[CHECKSTYLE] Parsing 1 file in /var/lib/jenkins/workspace/Code_Analysis
[CHECKSTYLE] Successfully parsed file /var/lib/jenkins/workspace/Code_Analysis/target/checkstyle-result.xml with 96 unique warnings and 0 duplicates.
```

23. We can find this checkstyle-result.xml in our Code Analysis job’s workspace



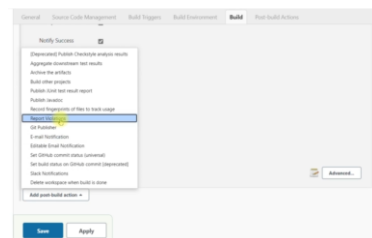
24. Under Workspace, we can see the result.xml



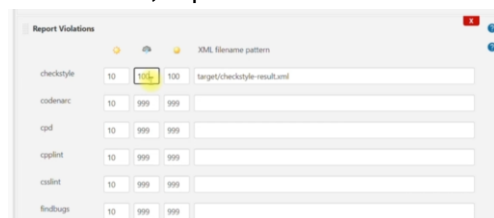
25. We get a graph with the data from violations in checkstyle-result.xml, the graph shows the violations between first job and second job. If we have fixed some violations, we will see a decrease in this graph.



26. We are going to give the checkstyle-result.xml file for scanning. In Post Build Action → Add post-build section → Report violations



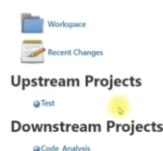
27. We can use other code analysis tools as you see in the below screenshot like “codenarc”, “cpd” etc. But for our use case we are using “checkstyle”



28. According to the organization’s policies, we can set the toleration. In our new case, we have set the maximum toleration to 100.

29. We have tested the “Code Analysis” Job and now we can configure this job as a downstream job for “Integration Tests” job.

Project Integration Test

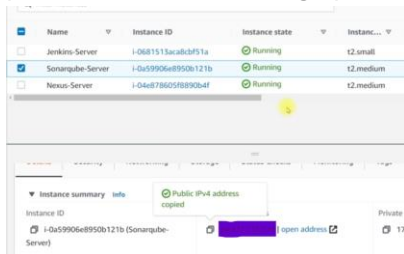


30. We now have four jobs connected with each other. First is Build, Second is Test, Third is Integration Test and the last one will be Code_Analysis

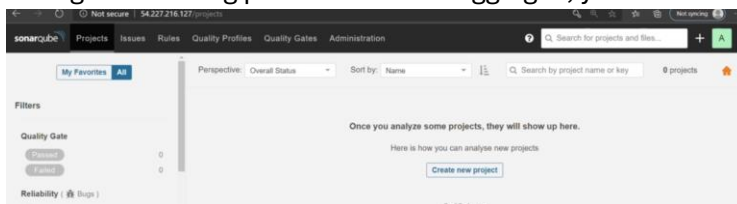
S	W	Name	Last Success	Last Failure	Last Duration
1	Build	Build	23 min - #2	N/A	0.4 sec
2	Code_Analysis	Code_Analysis	1 min 31 sec - #6	9 min 10 sec - #3	0.3 sec
3	Integration Test	Integration Test	14 min - #1	N/A	17 sec
4	Test	Test	20 min - #1	N/A	15 sec

31. We have one more code analysis with Sonarqube and we are going to publish the results from checkstyle and sonarqube to Sonarqube dashboard.

32. We should have Sonarqube server up and running. We will access this server using its public IP address through port 80 with a browser.

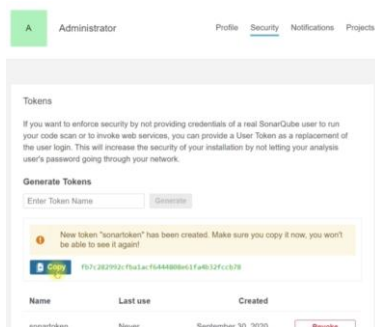


33. Default username is admin and password is admin. Before putting it into production, change to a strong password. After logging in, you can see the projects page



34. To integrate Jenkins with Sonarqube, Jenkins needs to authenticate to Sonarqube Server, for that we need to generate a token.

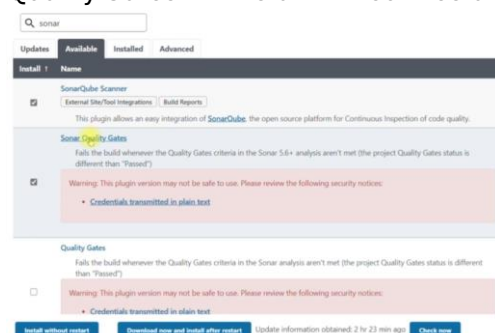
35. Top right corner → Click the profile → My Account → Security → Enter a token name and click on Generate. Copy the token and keep it safe to configure Jenkins.



36. Login into Jenkins → Manage Plugins



37. Click on “Available” Tab → Search for “sonar” → Select “SonarQube Server” and “Sonar Quality Gates” → Install without Restart



- 38.** Manage Jenkins → Global Tool Configuration → Scroll down to “SonarQube Scanner” → Click on “Add SonarQube Scanner”



- 39.** Select the version and provide the name for the scanner