

## Continuous integration:

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for release day to merge their changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

## Continuous delivery:

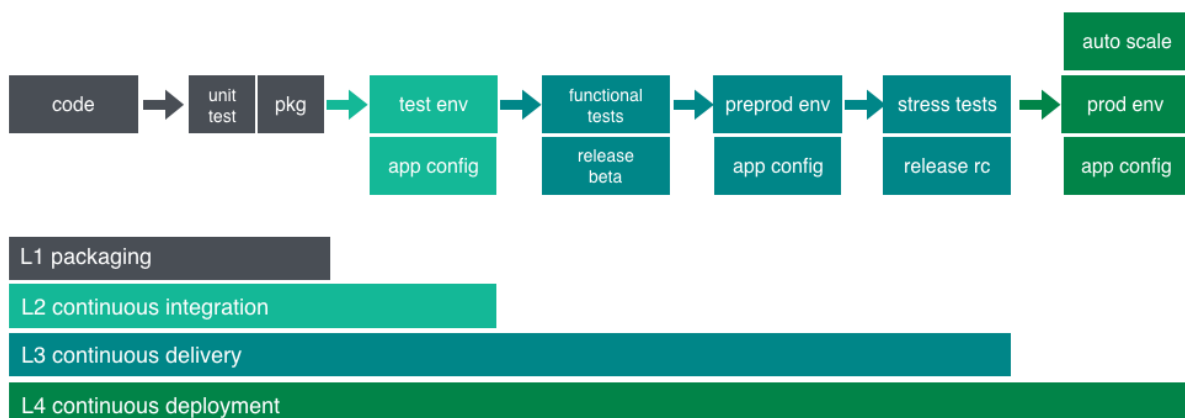
Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches, that are easy to troubleshoot in case of a problem.

## Continuous deployment:

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a Release Day anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.



## What is Jenkins?

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

### Prerequisites:

Minimum hardware requirements:

256 MB of RAM

1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

### Recommended hardware configuration for a small team:

1 GB+ of RAM

20 GB+ of drive space

### Software requirements:

Java 8 - either a Java Runtime Environment (JRE) or a Java Development Kit (JDK) is fine.

In this tutorial we install Jenkins on CentOS 7:

### Install JAVA:

- Install jdk8 RPM package from the following URL  
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- By using winscp we can copy the RPM package(jdk-8u151-linux-x64.rpm) to the centos machine.
- Now, login as a root to the centos machine and check for the execution permissions of the rpm package  
[root ~] ls -l jdk-8u151-linux-x64.rpm  
If the execution permissions are not there change with  
[root ~] chmod +x jdk-8u151-linux-x64.rpm
- Now, execute the rpm package to install jdk on the machine  
[root ~] rpm -Uvh jdk-8u151-linux-x64.rpm
- Once the jdk is installed successfully, now set up the java path as  
[root~] alternatives --install /usr/bin/java java /usr/java/latest/bin/java 200000  
  
[root~] alternatives --install /usr/bin/javac javac /usr/java/latest/bin/javac 200000  
  
[root~] alternatives --install /usr/bin/jar jar /usr/java/latest/bin/jar 200000
- Now, export the path in  
[root~] nano /etc/rc.local

Add the following line at the end of the file  
export JAVA\_HOME="/usr/java/latest"

- Now, check the java with the java, javac command

### Install Jenkins:

```
[root~] wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
[root~] rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
```

```
[root~] yum install -y jenkins
```

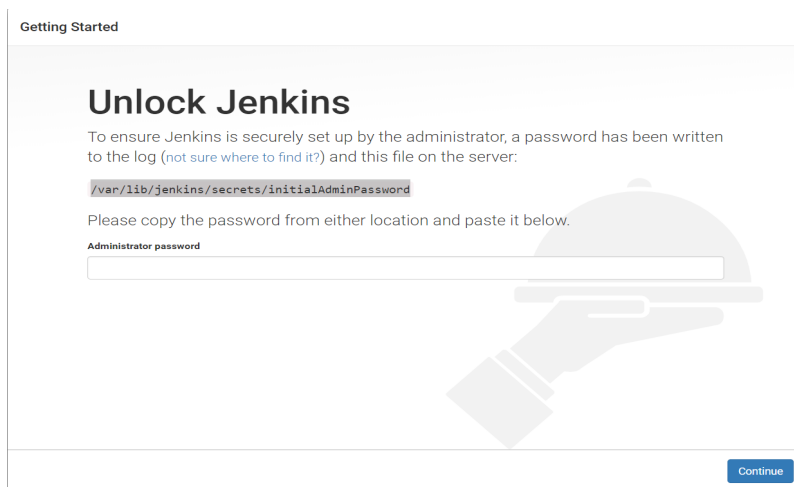
```
[root~] systemctl start jenkins
```

```
[root~] systemctl enable Jenkins
```

After the Jenkins installation process go to the URL as,

<http://<machine-ip>:8080>

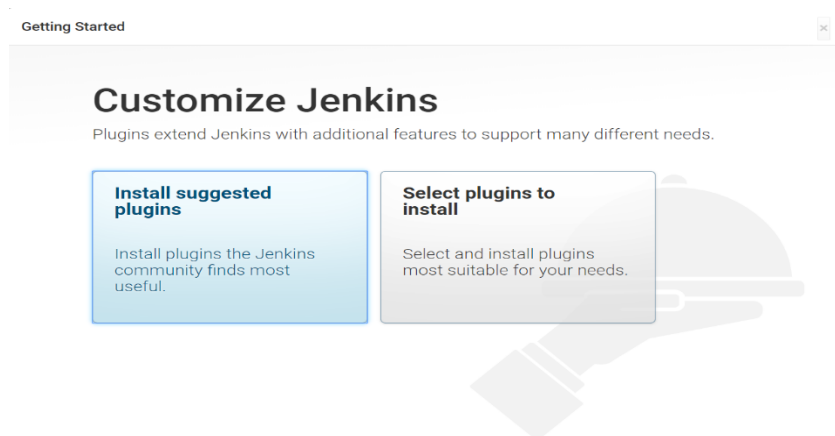
It shows the Jenkins start up page as,



To get the initial password type the following command in the terminal and enter the administrator password here

```
[root~] cat /var/lib/jenkins/secrets/initialAdminPassword
```

In the next you get the options to install the plugins, select install suggested plugins



Once all the plugins are installed, create the first admin user in the next page

Getting Started

## Create First Admin User

Username:

Password:

Confirm password:

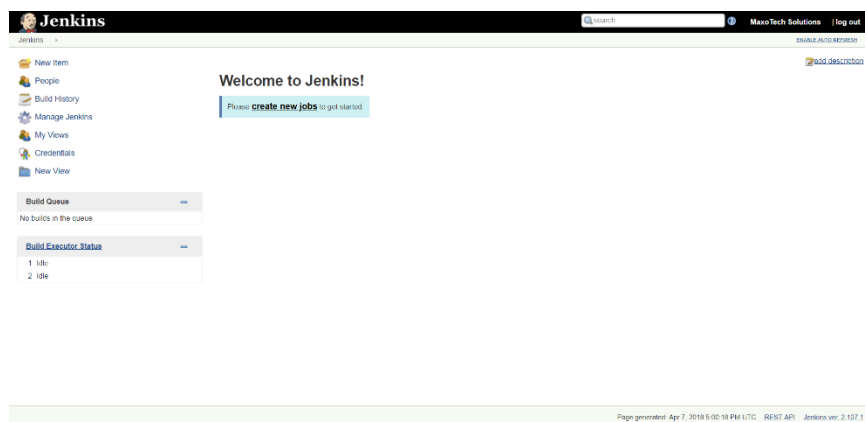
Full name:

E-mail address:

Jenkins 2.107.1

[Continue as admin](#) [Save and Finish](#)

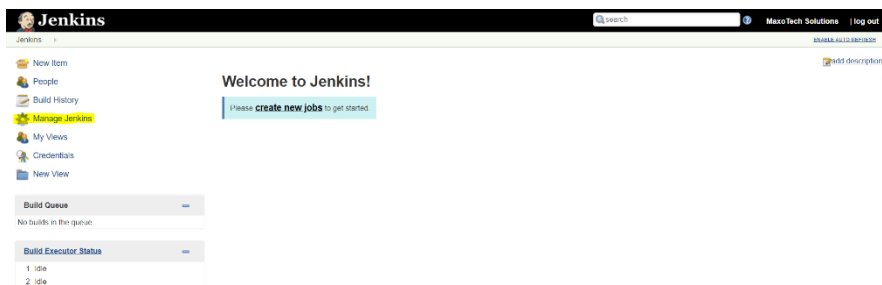
Now, you can start using the Jenkins



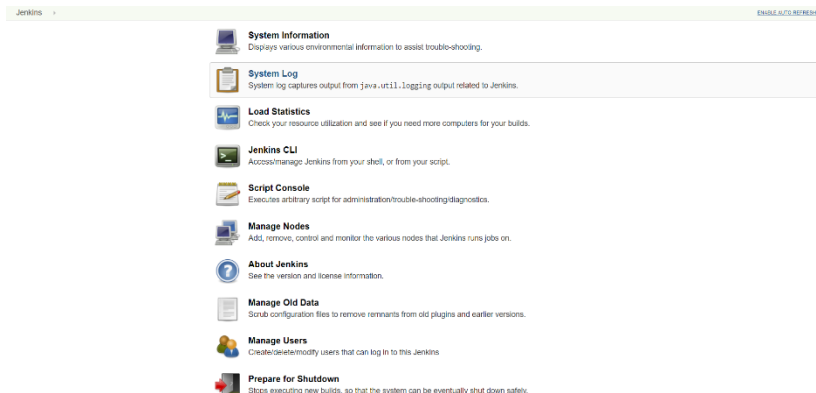
## Adding a slave node:

- First in the Jenkins master terminal, change the user as Jenkins with  
[root~] su Jenkins -s /bin/bash  
[jenkins~] ssh-keygen
- Then copy the generated key to the remote machine with  
[root~] ssh-copy-id user@<remote-ip>  
Enter the password.  
To check the authentication ssh user@<remote-ip>
- Install java and setup the path
- Once the authentication is done go to Jenkins home page and click on manage Jenkins

Go to the Jenkins page and click on Manage Jenkins tab



In the manage Jenkins tab you will find the manage nodes option



Now, click on the New Node on the left side and enter the name of the node, select as a permanent agent.

Jenkins

search

MaxoTech Solutions | log out

Jenkins > Nodes >

Back to Dashboard

Manage Jenkins

New Node

Configure

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Node name

dev-node

Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

OK

Configure the node with the details and save the node (make sure you have the remote root directory available in node)

Jenkins

search

MaxoTech Solutions | log out

Jenkins > Nodes > dev-node

Back to List

Status

Delete Agent

Configure

Build History

Load Statistics

Log

Build Executor Status

Name

dev-node

Description

Development Server Node

# of executors

2

Remote root directory

/home/user/jenkins

Labels

dev-node

Usage

Use this node as much as possible

Launch method

Launch slave agents via SSH

Host

172.31.118.213

Credentials

user

Host Key Verification Strategy

Known hosts file Verification Strategy

Advanced...

Availability

Keep this agent online as much as possible

Node Properties

Environment variables

Save

Now, you can see the node is connected.

Jenkins

search

MaxoTech Solutions | log out

Jenkins > Nodes >

Back to Dashboard

Manage Jenkins

New Node

Configure

Build Queue

No builds in the queue.

Build Executor Status

master

1 Idle

2 Idle

dev-node

1 Idle

2 Idle

S

Name

Architecture

Clock Difference

Free Disk Space

Free Swap Space

Free Temp Space

Response Time

	dev-node	Linux (amd64)	In sync	14.09 GB	2.00 GB	14.09 GB	203ms
	master	Linux (amd64)	In sync	13.81 GB	2.00 GB	13.81 GB	0ms
	Data obtained	56 min	56 min	56 min	56 min	56 min	56 min

Refresh status

- To download the source code of the project we use the version control system Git in slave node  
Install git with the command,  
[root~] yum install -y git
- For the project we will download the source code from the following GitHub link,  
<https://github.com/vemular1/jenkins-maven-pipeline.git>  
[root~] git clone <https://github.com/vemular1/jenkins-maven-pipeline.git>

To create a pipeline, we need to configure SonarQube, Nexus and Maven with Jenkins

### Configure Maven:

To setup the Maven, go to Jenkins home page click on Manage Jenkins->Global Tool Configuration->Maven->Add Maven

Here, you have the option to choose you maven installation automatically or you manually configure the existing maven installation path.

Select “Install Automatically” and apply, save it

The screenshot shows the Jenkins 'Global Tool Configuration' page. Under the 'Maven' section, there is a 'Maven installations' list. A new installation is being added with the name 'MAVEN'. The 'Install automatically' checkbox is checked. The 'Install from Apache' dropdown is set to 'Version 3.5.3'. There are buttons for 'Add Maven', 'Delete Installer', and 'Delete Maven'. At the bottom, there are 'Save' and 'Apply' buttons.

Now we need to set up the Sonar Qube for code analysis.

### Configure SonarQube:

**SonarQube is an open source tool for quality system development. It is written in Java and supports multiple databases. It provides capabilities to continuously inspect code, show the health of an application, and highlight newly introduced issues. It contains code analyzers which are equipped to detect tricky issues. It also integrates easily with DevOps.**

#### Prerequisites

- A Vultr 64-bit CentOS 7 server instance with at least 2 GB RAM.
- A sudo user.
- Install and configure Java

## Install PostgreSQL:

- Install PostgreSQL repository by typing:  
`sudo rpm -Uvh https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/pgdg-centos96-9.6-3.noarch.rpm`
- Install PostgreSQL database server by running:  
`sudo yum -y install postgresql96-server postgresql96-contrib`
- Initialize the database:  
`sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb`
- Edit the `/var/lib/pgsql/9.6/data/pg_hba.conf` to enable MD5-based authentication.  
`sudo nano /var/lib/pgsql/9.6/data/pg_hba.conf`
- Find the following lines and change peer to trust and ident to md5.  

# TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only				
local	all	all		peer
# IPv4 local connections:				
host	all	all	127.0.0.1/32	ident
# IPv6 local connections:				
host	all	all	::1/128	ident
- Once updated, the configuration should look like the one shown below.  

# TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only				
local	all	all		trust
# IPv4 local connections:				
host	all	all	127.0.0.1/32	md5
# IPv6 local connections:				
host	all	all	::1/128	md5
- Start PostgreSQL server and enable it to start automatically at boot time by running:  
`sudo systemctl start postgresql-9.6`  
`sudo systemctl enable postgresql-9.6`
- Change the password for the default PostgreSQL user.  
`sudo passwd postgres`
- Switch to the postgres user.  
`su - postgres`
- Create a new user by typing:  
`createuser sonar`
- Switch to the PostgreSQL shell.



psql

- Set a password for the newly created user for SonarQube database.  
ALTER USER sonar WITH ENCRYPTED password 'StrongPassword';
- Create a new database for PostgreSQL database by running:  
CREATE DATABASE sonar OWNER sonar;
- Exit from the psql shell:  
\q
- Switch back to the sudo user by running the exit command.

### Download and configure SonarQube

- Download the SonarQube installer files archive.  
wget https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-6.4.zip

You can always look for the link to the latest version of the application on the SonarQube [download page](#).

- Install unzip by running:  
sudo yum -y install unzip
- Unzip the archive using the following command.  
sudo unzip sonarqube-6.4.zip -d /opt
- Rename the directory:  
sudo mv /opt/sonarqube-6.4 /opt/sonarqube
- Open the SonarQube configuration file using your favorite text editor.  
sudo nano /opt/sonarqube/conf/sonar.properties
- Find the following lines.  
#sonar.jdbc.username=  
#sonar.jdbc.password=  
  
• Uncomment and provide the PostgreSQL username and password of the database that we have created earlier. It should look like:  
sonar.jdbc.username=sonar  
sonar.jdbc.password=StrongPassword
- Next, find:  
#sonar.jdbc.url=jdbc:postgresql://localhost/sonar
- Uncomment the line, save the file and exit from the editor.

### Configure Systemd service

- SonarQube can be started directly using the startup script provided in the installer package. As a matter of convenience, you should setup a Systemd unit file for SonarQube.

`sudo nano /etc/systemd/system/sonar.service`

- Populate the file with:

[Unit]

Description=SonarQube service

After=syslog.target network.target

[Service]

Type=forking

ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start

ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop

User=root

Group=root

Restart=always

[Install]

WantedBy=multi-user.target

- Start the application by running:

`sudo systemctl start sonar`

- Enable the SonarQube service to automatically start at boot time.

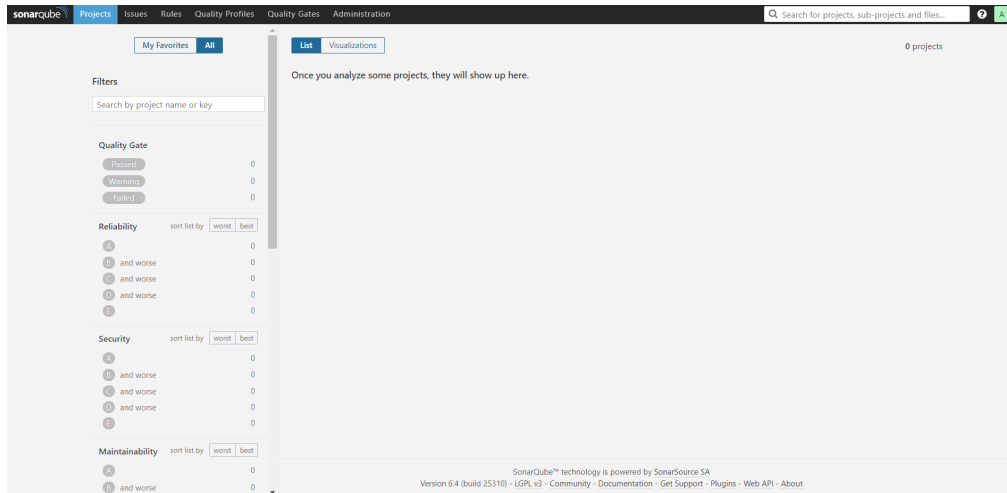
`sudo systemctl enable sonar`

- To check if the service is running, run:

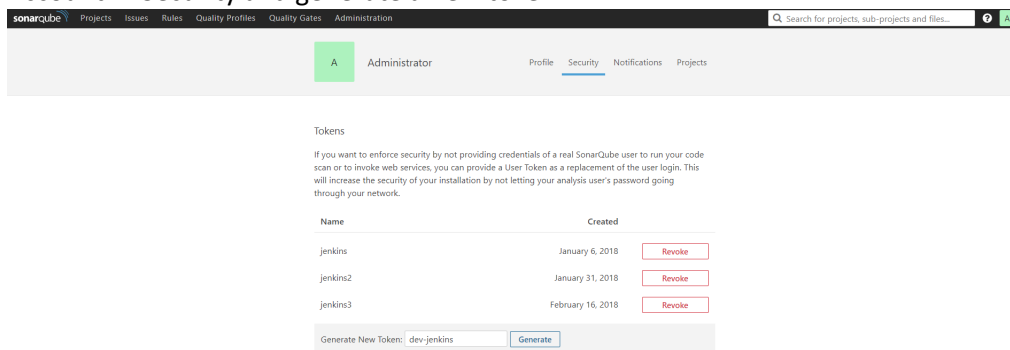
`sudo systemctl status sonar`

- SonarQube is installed on your server, access the dashboard at the following address.

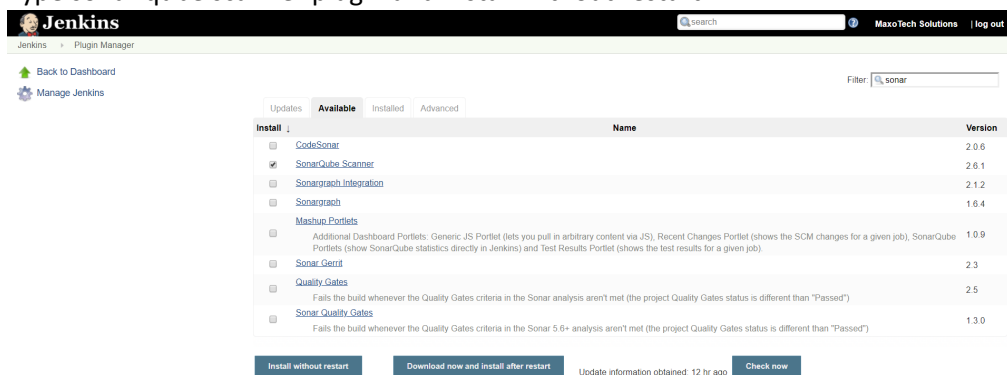
<http://<server-ip>:9000>



- To integrate SonarQube with Jenkins we need to generate the authentication token from My Account -> Security and generate a new token.



- Now copy the token and go to the Jenkins home page and click on Manage Jenkins -> Manage Plugins -> Available
- Type sonar qube scanner plugin and install without restart



**Jenkins** MaxoTech Solutions [log out](#)

Jenkins > Update Center ENABLE AUTO REFRESH

## Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

jQuery Success

SonarQube Scanner Success

[Go back to the top page](#)  
(you can start using the installed plugins right away)

[Restart Jenkins when installation is complete and no jobs are running](#)

- **Configure the SonarQube with Jenkins**

Jenkins > configuration

**master**  
1 Idle  
2 Idle

**dev-node**  
1 Idle  
2 Idle

☐ Restrict project naming

**Global properties**

☐ Environment variables

☐ Tool Locations

**SonarQube servers**

☐ Enable injection of SonarQube server configuration as build environment variables  
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

Environment variables

SonarQube installations

Name

Server URL   
Default is http://localhost:9000

Server version

Server authentication token   
Configuration fields depend on the SonarQube server version.

SonarQube account login   
SonarQube authentication token. Mandatory when anonymous access is disabled.

SonarQube account password   
SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.

[Advanced...](#)

[Delete SonarQube](#)

[Add SonarQube](#)

List of SonarQube installations

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

- **Add the SonarQube Scanner Manage Jenkins -> Global Tool Configuration ->Add SonarQube Scanner with Install Automatically or Configure manually.**

Jenkins > Global Tool Configuration

**Gradle**

Gradle installations [Add Gradle](#)  
List of Gradle installations on this system

**SonarQube Scanner for MSBuild**

SonarQube Scanner for MSBuild installations [Add SonarQube Scanner for MSBuild](#)  
List of SonarQube Scanner for MSBuild installations on this system

**SonarQube Scanner**

SonarQube Scanner installations

☐ SonarQube Scanner

Name

☒ Install automatically

☐ Install from Maven Central  
Version

[Delete Installer](#)

[Add Installer](#)

[Delete SonarQube Scanner](#)

[Add SonarQube Scanner](#)

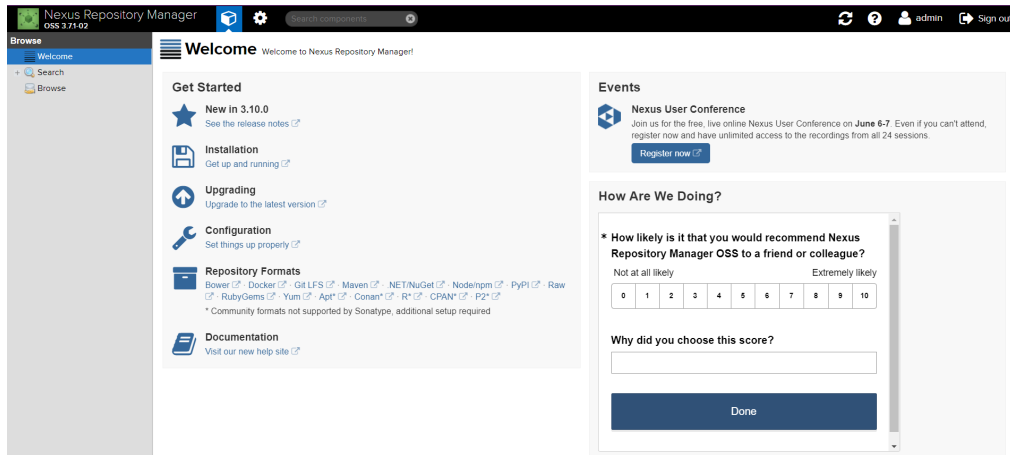
List of SonarQube Scanner installations on this system

**Ant**

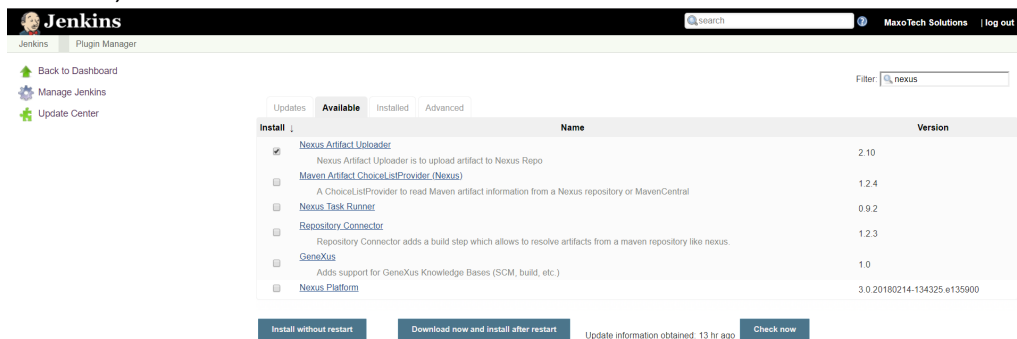
Ant installations [Add Ant](#)  
List of Ant installations on this system

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

- **Install Nexus from the following URL**  
<https://devopscube.com/how-to-install-latest-sonatype-nexus-3-on-linux/>
- **Nexus Home Page**



To integrate nexus with Jenkins, install the nexus plugin Manage Jenkins -> Manage Plugins -> Available, search for Nexus



- Once it is installed, configure the Nexus in the Jenkins server by editing the file in the location  

```
sudo nano /var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/MAVEN/conf/settings.xml
```

add the following lines before the </servers> tag, (watch out for <!-- -->)

```
<server>
  <id>deploymentRepo</id>
  <username>admin</username>
  <password>your nexus password here</password>
</server>
```

```

<!-- servers
| This is a list of authentication profiles, keyed by the server-id used within the system.
| Authentication profiles can be used whenever maven must make a connection to a remote server.
-->
<servers>
  <!-- server
  | Specifies the authentication information to use when connecting to a particular server, identified by
  | a unique name within the system (referred to by the 'id' attribute below).
  |
  | NOTE: You should either specify username/password OR privateKey/passphrase, since these pairings are
  | used together.
  -->
  <server>
    <id>deploymentRepo</id>
    <username>repouser</username>
    <password>repopwd</password>
  </server>
  -->

  <!-- Another sample, using keys to authenticate.
  <server>
    <id>siteServer</id>
    <privateKey>/path/to/private/key</privateKey>
    <passphrase>optional; leave empty if not used.</passphrase>
  </server>
  -->

  <server>
    <id>deploymentRepo</id>
    <username>admin</username>
    <password>your nexus passowrd here</password>
  </server>
</servers>

```

- Now edit your POM.xml file to add the distribution management to upload the artifacts to NEXUS.

```

<distributionManagement>
  <repository>
    <id>deploymentRepo</id>
    <name>Internal Releases</name>
    <url>http://<nexus-ip>:8081/repository/maven-releases/</url>
  </repository>

  <snapshotRepository>
    <id>deploymentRepo</id>
    <name>Internal Releases</name>
    <url>http://<nexus-ip>:8081/repository/maven-snapshots/</url>
  </snapshotRepository>
</distributionManagement>

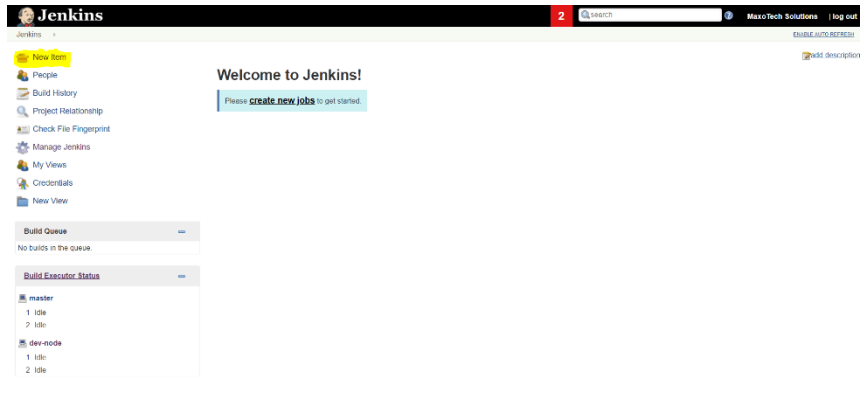
```

### Jenkins Build Pipeline Jobs:

**GIT → SONARQUBE → MAVEN → JUNIT → JAR → NEXUS**

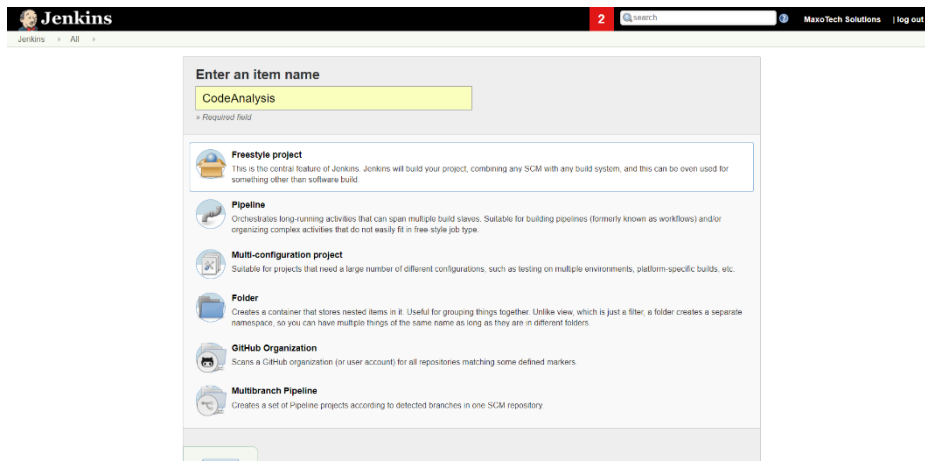
In the Jenkins console create your first job with Code Analysis and configure the steps as below.

- First click on the new item

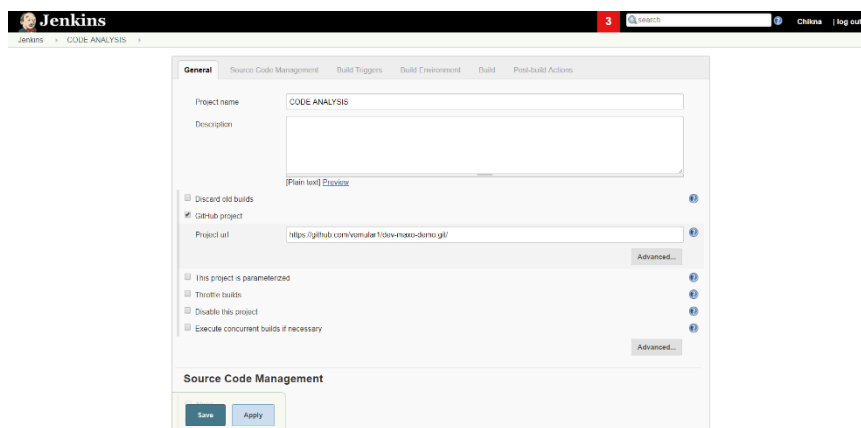


## Code Analysis:

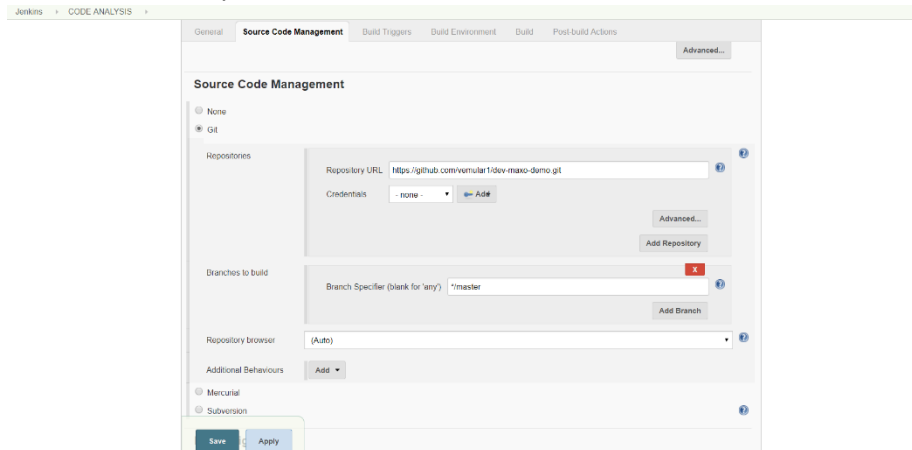
- Enter an Item name and select a free style job. (For all the remaining jobs we select the same)



- In the first step we select the GitHub project and enter the URL of our project which is <https://github.com/vemular1/dev-maxo-demo>

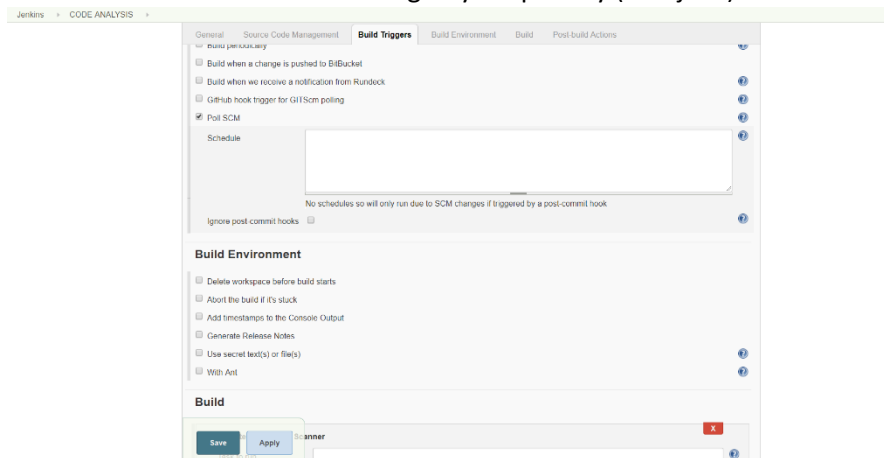


- Now paste the same github project and enter the credentials if it is a private repository and select the branch you want to build.



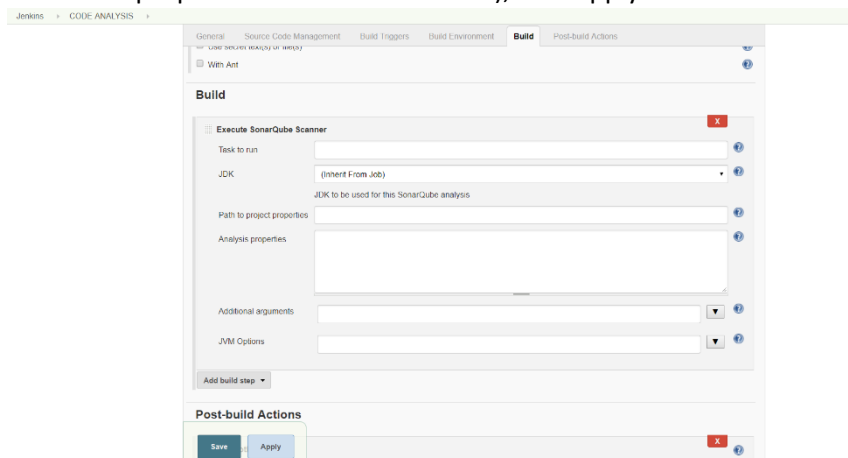
The image shows the 'Source Code Management' configuration page in Jenkins. The 'Git' option is selected under 'Source Code Management'. The 'Repository URL' is set to 'https://github.com/venkatar1dev/mxao-demo.git'. The 'Credentials' dropdown is set to 'none'. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' set to '\*/master'. The 'Repository browser' is set to '(Auto)'. There are 'Save' and 'Apply' buttons at the bottom.

- Now schedule the builds according to your priority (cronjobs)



The image shows the 'Build Triggers' configuration page in Jenkins. The 'Poll SCM' option is checked. The 'Schedule' field is empty. There are 'Save' and 'Apply' buttons at the bottom.

- In this step select the build step and in the drop down find the sonarqube scanner. (You can define the properties here or leave blank), click apply and save



The image shows the 'Build' configuration page in Jenkins. The 'Execute SonarQube Scanner' option is checked. The 'Task to run' is set to 'JDK'. The 'JDK' dropdown is set to '(Inherit From Job)'. The 'Path to project properties' and 'Analysis properties' fields are empty. There are 'Save' and 'Apply' buttons at the bottom.

- Now go to the job page and click on build now to verify whether the job runs successfully.



```
INFO: No source files to be analyzed
INFO: Java Test Files AST scan (done) | time=12ms
INFO: Sensor JavaRefSensor [java] (done) | time=2044ms
INFO: Sensor SurefireRefSensor [java]
INFO: parsing /var/lib/jenkins/workspace/CODE_ANALYSIS/target/surefire-reports
INFO: Sensor SurefireRefSensor [java] (done) | time=1ms
INFO: Sensor JAcCoSensor [java]
WARN: Property 'sonar.jacoco-reportPath' is deprecated. Please use 'sonar.jacoco.reportPath' instead.
INFO: JaCoCo UT report not found: 'target/jacoco.exec'
INFO: Sensor JAcCoSensor [java] (done) | time=1ms
INFO: Sensor SonarJavaAllFileIndexSensor [java]
INFO: Sensor SonarJavaAllFileIndexSensor [java] (done) | time=0ms
INFO: Sensor Analyzer for "php.ini" files [php]
INFO: Sensor Analyzer for "php.ini" files [php] (done) | time=3ms
INFO: Sensor Zero Coverage Sensor
INFO: 2/2 source files have been analyzed
INFO: 0/0 source files have been analyzed
INFO: Sensor Zero Coverage Sensor (done) | time=42ms
INFO: Sensor CPD Block Indexer
INFO: Sensor CPD Block Indexer (done) | time=23ms
INFO: 2 files had no CPD blocks
INFO: Calculating CPD for 0 files
INFO: CPD calculation finished
INFO: Analysis report generated in 145ms, size 22 KB
INFO: Analysis reports compressed in 15ms, size 12 KB
INFO: Analysis report uploaded in 65ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://35.225.44.211:8080/dashboard/index/er-samarneh-mann
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: Now about the report processing at http://35.225.44.211:8080/ai/cas/askId8WZm2Wtmgicvly-c5de
INFO: Task total time: 6.669 s
INFO:
INFO: EXECUTION SUCCESS
INFO:
INFO: Final Memory: 49M/117M
INFO:
INFO: Finished: SUCCESS
```

- Now create the next stage in the pipeline as Compile to compile the code. To create the compile stage we follow the same steps as above and change the build step as compile.

- Click apply and save the job, now build the job to check whether runs successfully.

- In this job we specify the same and we in place of test we specify as package to package the application.

Jenkins > Package > #36

General Source Code Management Build Triggers **Build Environment** Build Post-build Actions

☐ GitHub hook trigger for GITSCm polling  
☐ Poll SCM

**Build Environment**

☐ Delete workspace before build starts  
☐ Abort the build if it's stuck  
☐ Add timestamps to the Console Output  
☐ Generate Release Notes  
☐ Use secret text(s) or file(s)  
☐ With Ant

**Build**

☐ Invoke top-level Maven targets

Maven Version: Maven  
Goals: package  
Advanced...

Add build step

Post-build Actions

Save Apply

- Now check the build status with build now.

Jenkins > Package > #36

```
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ maxotech ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /var/lib/jenkins/workspace/Package/target/test-classes
[WARNING] /var/lib/jenkins/workspace/Package/src/test/java/TestBankAccount.java: /var/lib/jenkins/workspace/Package/src/test/java/TestBankAccount.java uses or overrides a deprecated API.
[WARNING] /var/lib/jenkins/workspace/Package/src/test/java/TestBankAccount.java: Recompile with -Xlint:deprecation for details.
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ maxotech ---
[INFO] Surefire report directory: /var/lib/jenkins/workspace/Package/target/surefire-reports

T E S T S
-----
Running com.maxo.TestBankAccount
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.271 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- jacoco-maven-plugin:0.7.8:report (post-unit-test) @ maxotech ---
[INFO] Loading execution data file /var/lib/jenkins/workspace/Package/target/jacoco.exec
[INFO] Analyzed bundle 'maxotech' with 1 classes
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ maxotech ---
[INFO] Building jar: /var/lib/jenkins/workspace/Package/target/maxotech-1.36.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.904 s
[INFO] Finished at: 2018-04-11T22:21:42Z
[INFO] Final Memory: 23M/54M
[INFO] -----
Finished: SUCCESS
```

## Deploy to Nexus

- Now we will deploy our generated artifacts to Nexus, we create the job same as previous ones and change the build step as below.

Jenkins > DepTONexus > #36

General Source Code Management Build Triggers **Build Environment** Build Post-build Actions

☐ Generate Release Notes  
☐ Use secret text(s) or file(s)  
☐ With Ant

**Build**

☐ Invoke top-level Maven targets

Maven Version: Maven  
Goals: deploy  
Advanced...

Add build step

**Post-build Actions**

Add post-build action

Save Apply

- Now we will build the job with build now and check the output on the console.

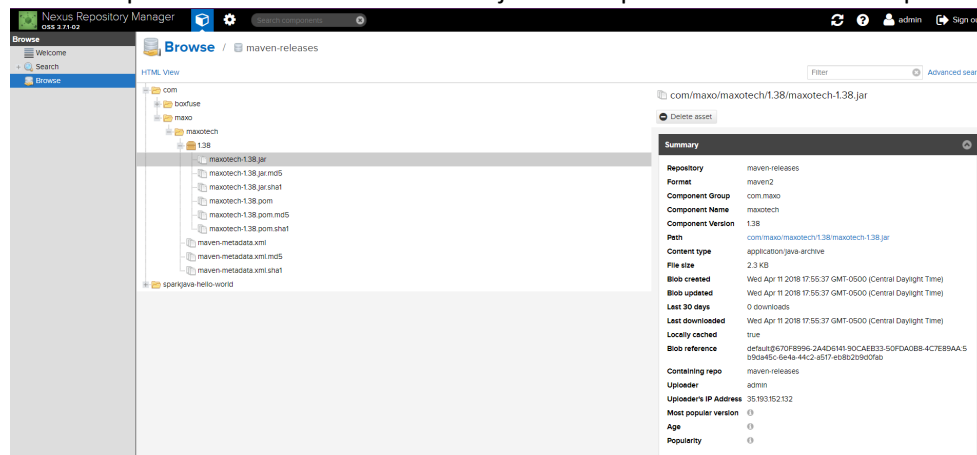
```

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- jacoco-maven-plugin:0.7.8:report (post-unit-test) @ maxotech ---
[INFO] Loading execution data file /var/lib/jenkins/workspace/DepTOexus/target/jacoco.exec
[INFO] Analyzed bundle 'maxotech' with 1 classes
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ maxotech ---
[INFO] Building jar: /var/lib/jenkins/workspace/DepTOexus/target/maxotech-1.38.jar
[INFO] --- maven-install-plugin:2.4:install (default-install) @ maxotech ---
[INFO] Installing /var/lib/jenkins/workspace/DepTOexus/target/maxotech-1.38.jar to /var/lib/jenkins/.m2/repository/com/maxo/maxotech/1.38/maxotech-1.38.jar
[INFO] Installing /var/lib/jenkins/workspace/DepTOexus/pom.xml to /var/lib/jenkins/.m2/repository/com/maxo/maxotech/1.38/maxotech-1.38.pom
[INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ maxotech ---
[INFO] Uploading to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/1.38/maxotech-1.38.jar
Progress (1): 2.0/2.4 kB
Progress (1): 2.4 kB
Uploaded to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/1.38/maxotech-1.38.jar (2.4 kB at 2.4 kB/s)
[INFO] Uploading to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/1.38/maxotech-1.38.pom
Progress (1): 1.4 kB
Uploaded to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/1.38/maxotech-1.38.pom (1.4 kB at 2.6 kB/s)
[INFO] Downloading from deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/maven-metadata.xml
[INFO] Uploading to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/maven-metadata.xml
Progress (1): 294 B
Uploaded to deploymentRepo: http://35.193.152.132:8081/repository/maven-releases/com/maxo/maxotech/maven-metadata.xml (294 B at 963 B/s)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.589 s
[INFO] Finished at: 2018-04-11T22:55:39Z
[INFO] Final Memory: 24M/50M
[INFO] -----
Finished: SUCCESS

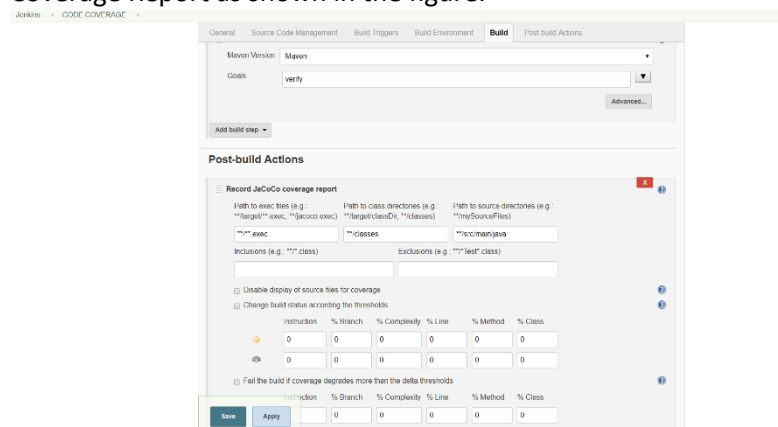
```

In the output screen we can see that the jar file is uploaded to the nexus repo location.



## Code Coverage:

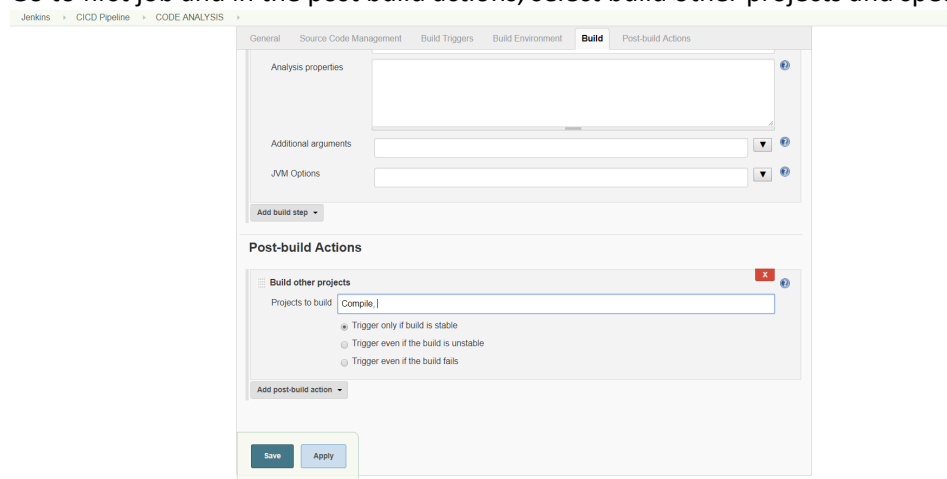
- Create another job called “Code Coverage” and configure it same as previous jobs and in the build step we specify it as “maven verify” and configure the post build actions as Record JaCoCo Coverage Report as shown in the figure.



- To create a Pipeline view, go to the manage plugins page and search for “Build Pipeline Plugin” and install without restart.

- In order to execute the jobs in pipeline we specify the upstream and downstream projects in the job configurations.
- In each job post build actions, we specify the next job to be executed in pipeline.
- **Code Analysis → Compile → UnitTest → Package → DeptoNexus → CodeCoverage**

Go to first job and in the post build actions, select build other projects and specify as “Compile”



The screenshot shows the Jenkins job configuration page for 'CODE ANALYSIS'. The 'Post-build Actions' tab is active. Under 'Post-build Actions', the 'Build other projects' action is selected. The 'Projects to build' field contains the text 'Compile'. Below this field, three radio button options are visible: 'Trigger only if build is stable' (which is selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. At the bottom of the configuration, there are 'Save' and 'Apply' buttons.

Go to compile job and now specify build other projects as “Unit Test”

Jenkins » Compile »

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Build

Invoke top-level Maven targets

Maven VersionMaven

Goalscompile

Advanced...

Add build step

Post-build Actions

Build other projects

Projects to buildUnit Test

☒ Trigger only if build is stable  
☐ Trigger even if the build is unstable  
☐ Trigger even if the build fails

Add post-build action

Save

Apply

Go to Unit Test job and now specify build other projects as “Package”

Jenkins » Unit Test »

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Post-build Actions

Build other projects

Projects to buildPackage

☒ Trigger only if build is stable  
☐ Trigger even if the build is unstable  
☐ Trigger even if the build fails

Publish JUnit test result report

Test report XMLStarget\surefire-reports.xml

Fileset<includes> setting that specifies the generated raw XML report files, such as myproject\target\test-reports.xml. Basedir of the fileset is the workspace root.

☐ Retain long standard output/error

Health report amplification factor1.0

Allow empty results☐ Do not fail the build on empty test results

Add post-build action

Save

Apply

Go to Package job and now specify build other projects as “DepToNexus”

Jenkins » Package »

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Build

Invoke top-level Maven targets

Maven VersionMaven

Goalspackage

Advanced...

Add build step

Post-build Actions

Build other projects

Projects to buildDepToNexus

☒ Trigger only if build is stable  
☐ Trigger even if the build is unstable  
☐ Trigger even if the build fails

Add post-build action

Save

Apply

Go to DepToNexus job and now specify build other projects as “Code Coverage”

Jenkins > DepTONexus >

General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions

Invoke top-level Maven targets
Maven Version: Maven
Goals: deploy
Advanced...

Add build step

Post-build Actions

Build other projects
Projects to build: CODE\_COVERAGE
No such project 'C'. Did you mean 'job1'?
Trigger only if build is stable
Trigger even if the build is unstable
Trigger even if the build fails

Add post-build action

Save
Apply

Finally, to create a pipeline view by clicking the “+” sign on the Jenkins home page

Jenkins >

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Open Blue Ocean
Credentials
New View

Build Queue

Build Executor Status

All
CICD Pipeline
Maven Pipeline
Maven Project

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
		AcceptanceTest	2 mo 3 days - #32	N/A	15 ms	
		CODE_ANALYSIS	4 hr 15 min - #39	N/A	9.7 sec	
		CODE_COVERAGE	7 min 52 sec - #33	N/A	11 sec	
		Compile	4 hr 5 min - #35	N/A	8.1 sec	
		DepTONexus	2 hr 45 min - #38	2 hr 57 min - #37	14 sec	
		DocToQA	2 mo 3 days - #32	N/A	9 ms	
		job1	3 mo 0 days - #9	N/A	6.5 sec	
		JobtoNexus	3 mo 0 days - #1	N/A	9.2 sec	
		Maven Analysis	2 mo 29 days - #4	N/A	9.4 sec	
		Maven Compile	2 mo 29 days - #4	N/A	6.4 sec	
		Maven Package	2 mo 20 days - #14	3 mo 0 days - #11	8.2 sec	
		Maven UI	2 mo 29 days - #8	3 mo 1 day - #2	7.2 sec	
		MavenToNexus	2 mo 20 days - #15	3 mo 0 days - #12	11 sec	
		mongo-dfs-proj	N/A	N/A	N/A	
		Package	3 hr 19 min - #30	3 mo 1 day - #9	13 sec	
		Sample proj	N/A	N/A	N/A	
		Unit Test	3 hr 34 min - #35	N/A	12 sec	

In the next page give a name to the pipeline view and select build pipeline view

Jenkins >

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Open Blue Ocean
Credentials
New View

Build Queue

Build Executor Status

View name: dev-pipeline

Build Pipeline View
Deployment Dashboard
List View
My View

OK

In the next page give the Description of the view, Select the flow and specify the initial job as “Code Analysis”

Jenkins - dev pipeline

New Item  
People  
Build History  
Edit View  
Delete View  
Project Relationship  
Check File Fingerprint  
Manage Jenkins  
My Views  
Credentials  
New View

Build Queue

No builds in the queue

Build Executor Status

1 Idle  
2 Idle

Name: Dev-Pipeline  
Description: Sample Development Pipeline  
[Plain text] [Expand](#)

Filter build queue: ☐  
Filter build executors: ☐  
Build Pipeline View title: Dev Pipeline

Pipeline Flow: Based on upstream/downstream relationship  
This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.  
Upstream / downstream config: Select Initial Job: CODE ANALYSIS  
Trigger Options: Build Cards: Standard build card  
Use the default build cards  
Restrict triggers to most recent successful builds: ☐ Yes ☒ No  
Always allow manual trigger on pipeline steps: ☐ Yes ☒ No  
Display Options: No. of displayed builds: 2

OK Apply

We can also specify no of displayed builds. Here we specify as “2”

Display Options

No Of Displayed Builds: 2

Row Headers: Just the pipeline number  
Show just the build pipeline number

Column Headers: No header  
Do not show any column headers

Refresh frequency (in seconds): 3

URL for custom CSS files:

Console Output Link Style: Lightbox

OK Apply

The final view is,

