CI/CD using Jenkins Pipeline as code

JULY 23, 2018 ANUSHA SHARMA 3 COMMENTS

Jenkins Pipeline

Pipeline annexes a strong set of automation tools onto Jenkins. It assists use cases covering from simple to comprehensive continuous integration/delivery pipelines. This blog will provide easy steps to implement CI/CD using Jenkins Pipeline as code. Furthermore it will integrate Jenkins, Github, SonarQube and JFrog Artifactory.

Job –> Pipeline

As Job is a defined process. Build is outcome of that process. Pipeline can be thought of as a series of jobs. It orchestrates various phases (e.g. build, test, deploy) involved in the life-cycle of a software. Above all, using Jenkins pipeline as code, entire process can be automated by writing scripts for each module.

Types of DSL syntax

Two types of DSL syntax are available: Scripted Pipeline and Declarative Pipeline. Groovy DSL is used to code scripted pipelines, while Declarative Pipeline provides predefined structures and models. It allows fast, stable & compact pipelines creation by users with or without learning Groovy.

3 ways to create pipeline

As enumerated by their website , there exists 3 ways to create pipeline:

- Through Blue Ocean—after setting up a Pipeline project in Blue Ocean, the Blue Ocean UI helps to write Pipeline’s Jenkinsfile and commit it to source control.
- Through the classic UI— a basic Pipeline can be entred directly in Jenkins through the classic UI.
- In SCM— Jenkins can be written manually and then can be committed to project’s source control repository.

*Note : It is the best practice to define pipeline in Jenkinsfile and store it in source control (Github) along with the other code check-in. Hence allowing Jenkins to load it directly from SCM and execute the scripted stages.

Using Jenkinsfile has the following advantages (as stated by them):

a). Firstly, code review/iteration on the Pipeline

b). Secondly, audit trail for the Pipeline

c). Lastly, it act as a single source of truth for the Pipeline, therefore can be viewed and edited by multiple members of the project.

Enough with the background theory, let's get start with the procedure step-by-step :

CI/CD using Jenkins Pipeline as code

Following are the steps employing declarative pipeline script to automate the Jenkins job:

Step 1. Start Jenkins :

a). If downloaded as a .zip file & installed by running jenkins.msi :

- Run the Jenkins on default browser ***http://localhost:8080/***
- Supply the login id & password

b). However, if downloaded as a jenkins.war file:

Go to command prompt. Browse to the directory containing jenkins.war. Run the following command:

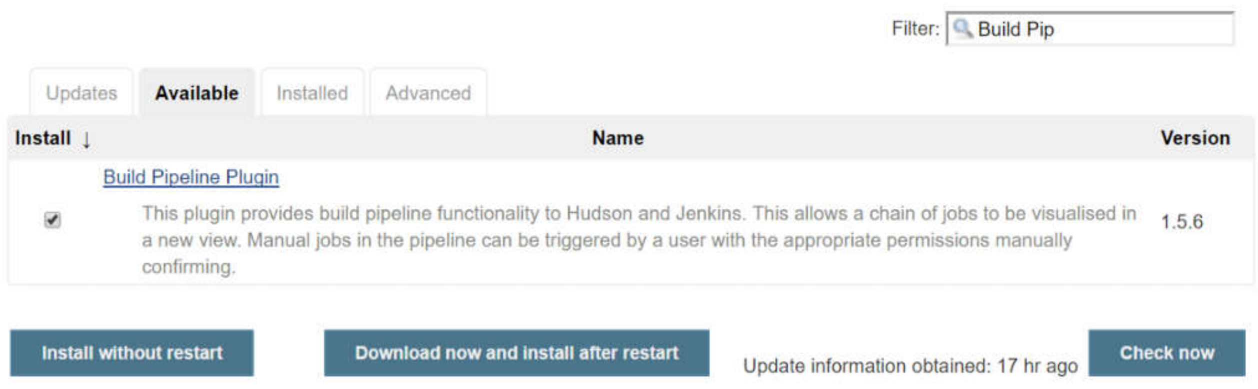
```
java -jar jenkins.war
```

After Jenkins is fully up & running. Go to the default browser. Log in with the credentials

Step 2. Create & configure Jenkins Pipeline Project/Job:

a). Install the plugin from Manage Jenkins

Go to Jenkins Dashboard. Manage Jenkins. Manage Plugins. Available. search for 'Build Pipeline'. Click on 'Install without restart'

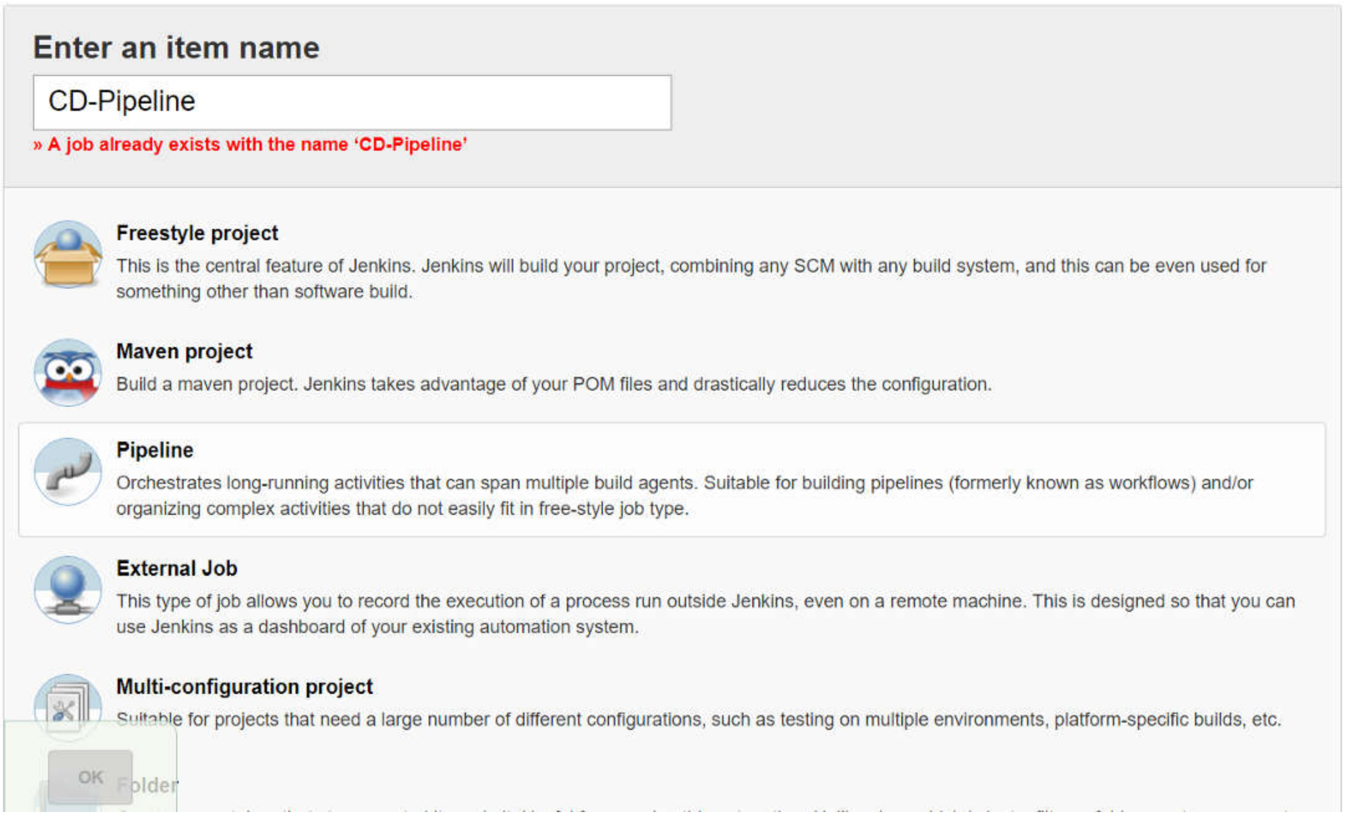


Install ↓	Name	Version
✓	Build Pipeline Plugin This plugin provides build pipeline functionality to Hudson and Jenkins. This allows a chain of jobs to be visualised in a new view. Manual jobs in the pipeline can be triggered by a user with the appropriate permissions manually confirming.	1.5.6

Update information obtained: 17 hr ago

Plugin Installation : Jenkins

b). Go to Jenkins Dashboard. Click on "New Item". Type the name of the project/job. Select pipeline. Click OK.



Enter an item name

CD-Pipeline

» A job already exists with the name 'CD-Pipeline'

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

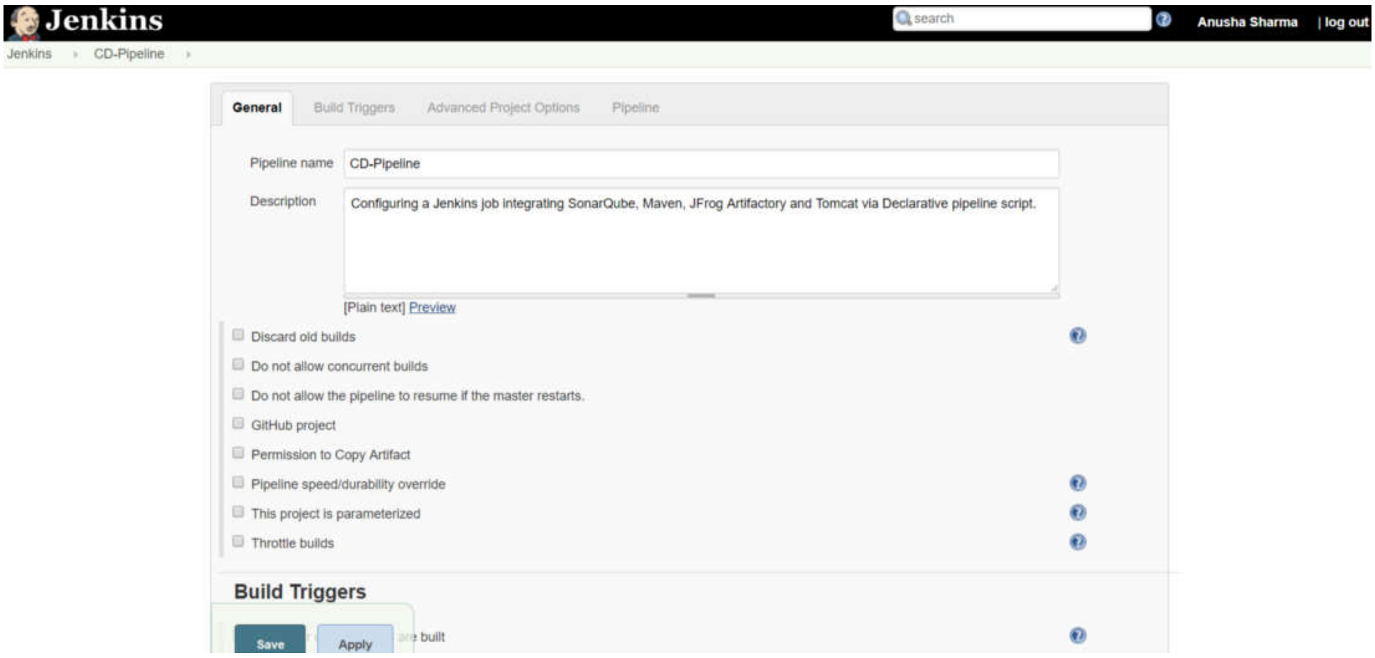
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Job creation : Jenkins

c). Edit the Project Configuration Window. You can provide the Description (though optional).



Pipeline name: CD-Pipeline

Description: Configuring a Jenkins job integrating SonarQube, Maven, JFrog Artifactory and Tomcat via Declarative pipeline script.

[Plain text] Preview

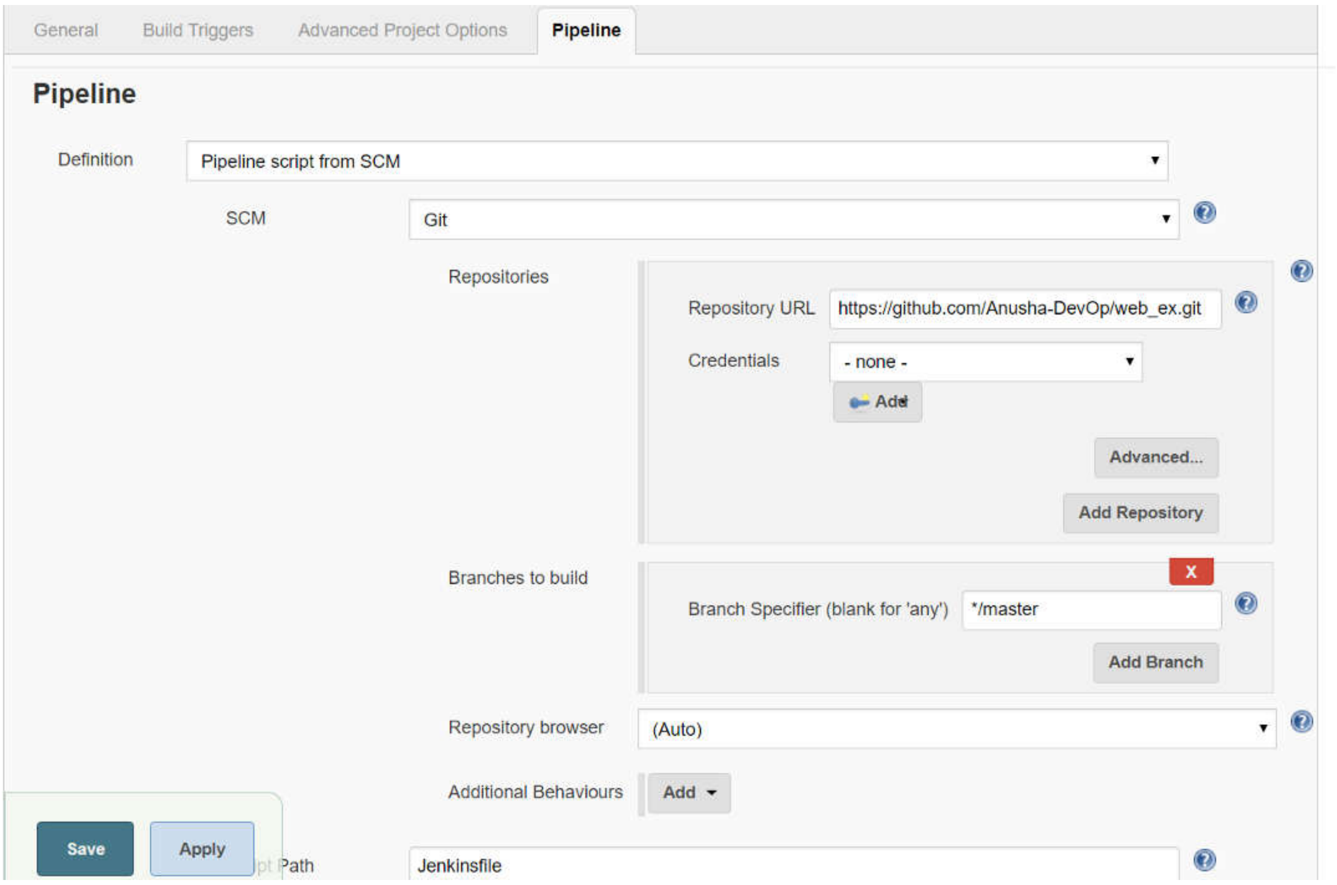
- ☐ Discard old builds
- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the master restarts.
- ☐ GitHub project
- ☐ Permission to Copy Artifact
- ☐ Pipeline speed/durability override
- ☐ This project is parameterized
- ☐ Throttle builds

Build Triggers

Job configuration Window : Jenkins

Go to the Pipeline section :

- Select "Pipeline script from SCM" from definition.
- Select Git as SCM
- Provide the URL of the github repository, where you have checked-in your source code, pom.xml and ***Jenkinsfile (Wait. Don't panic!! we will create one in a bit).***


General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/Anusha-DevOp/web_ex.git

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Script Path: Jenkinsfile

Configuration Window : Jenkins

- Provide the script path, i.e. name of your Jenkinsfile (by default its name is Jenkinsfile, without any extension).
- Click Apply & Save.

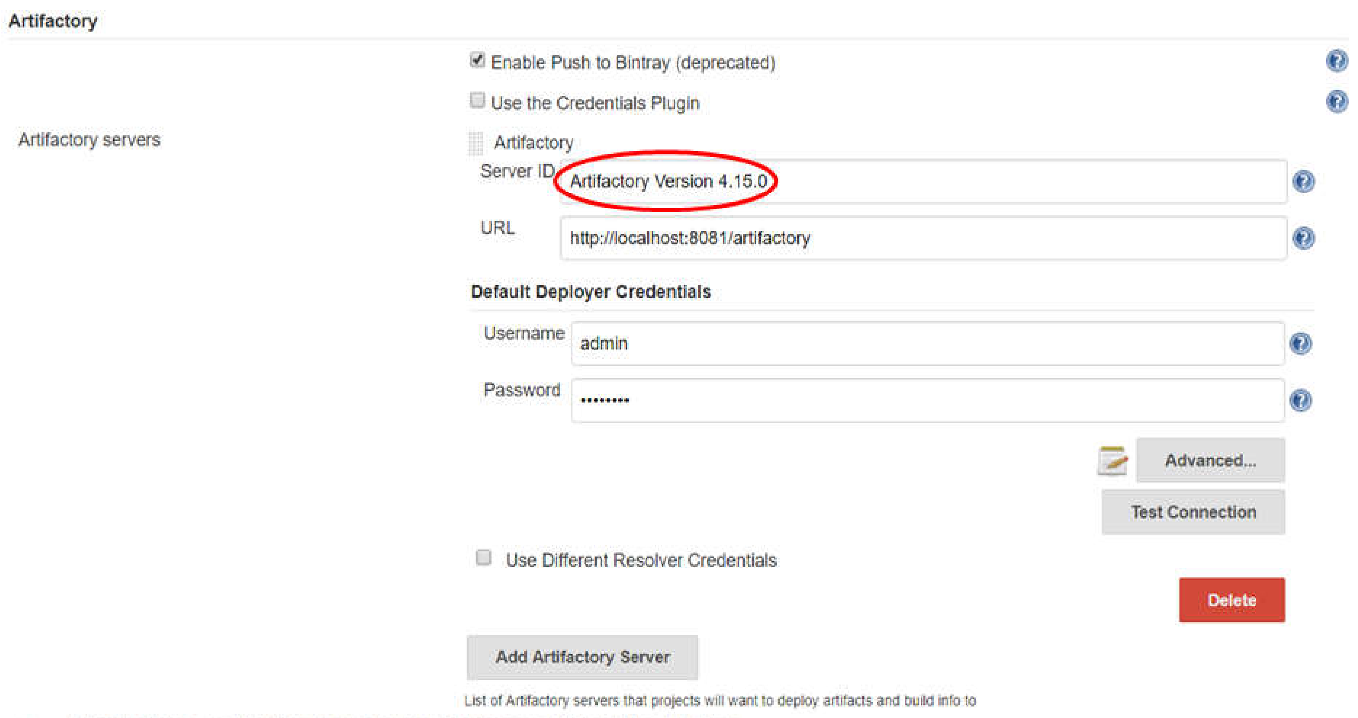
Step 3. Create a basic maven project (maven-archetype-webapp or maven-archetype-quickstart).

Step 4. Create a Jenkinsfile:

Go to the root directory of your maven project. Create a new text file with name: Jenkinsfile and edit it with the following code snippet:

a). Create Artifactory Server Instance :

Make sure Artifactory Server is installed and configured (see here) in Jenkins under Manage Jenkins → Configure System



Artifactory

☑ Enable Push to Bintray (deprecated)

☐ Use the Credentials Plugin

Artifactory servers

Artifactory

Server ID: Artifactory Version 4.15.0

URL: http://localhost:8081/artifactory

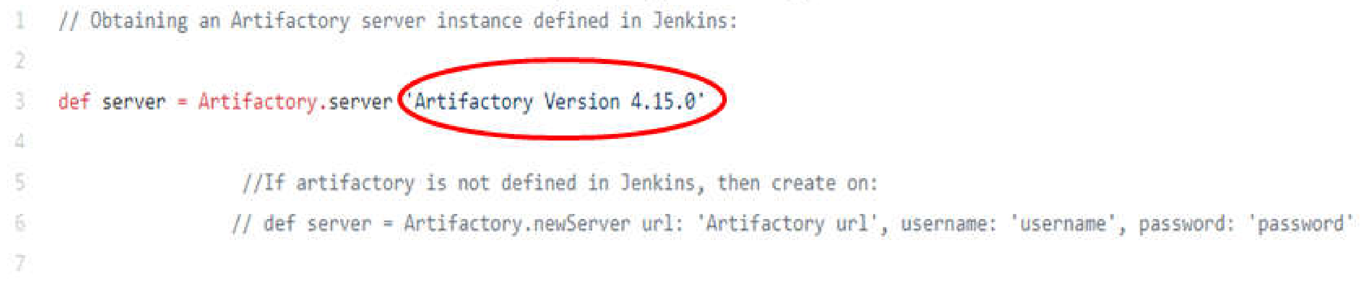
Default Deployer Credentials

Username: admin

Password: ••••••••

☐ Use Different Resolver Credentials

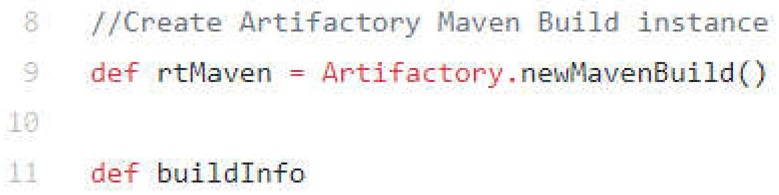
List of Artifactory servers that projects will want to deploy artifacts and build info to



```
// Obtaining an Artifactory server instance defined in Jenkins:

def server = Artifactory.server 'Artifactory Version 4.15.0'

                    //If artifactory is not defined in Jenkins, then create on:
                   // def server = Artifactory.newServer url: 'Artifactory url', username: 'username', password: 'password'

```

**Provide the server-id same as set in the Jenkins Configuration System.*

b). Create Artifactory Maven Build instance.



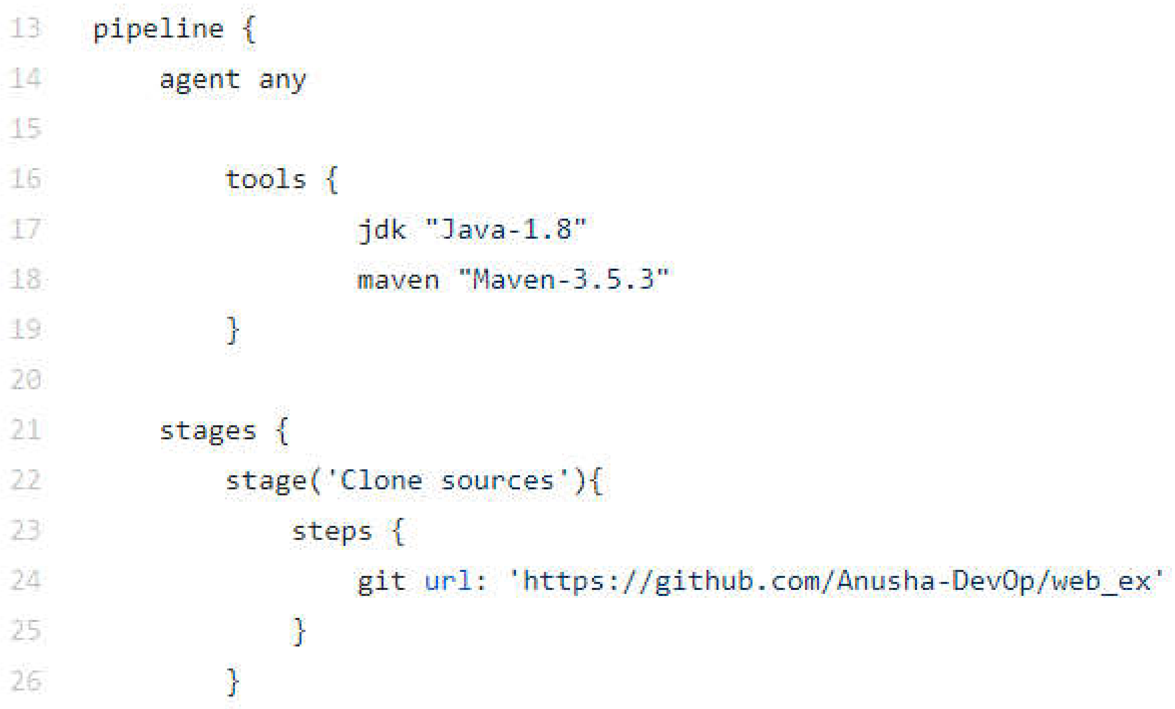
```
//Create Artifactory Maven Build instance
def rtMaven = Artifactory.newMavenBuild()

def buildInfo
```

*Maven builds can resolve dependencies, deploy artifacts and publish build-info to Artifactory.

*Let’s define buildInfo Object too.

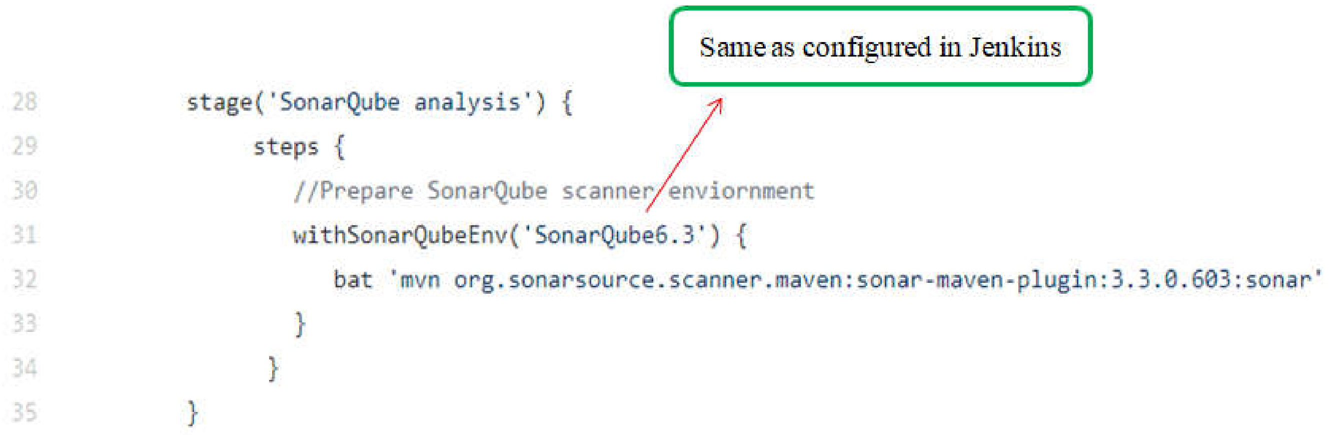
c). Start Declarative pipeline script (get basics here) with different stages.

Stage (i). ‘Clone sources’ : to tell Jenkins to checkout source code from a particular repository (github in this case).



```
pipeline {
    agent any

        tools {
                jdk "Java-1.8"
                maven "Maven-3.5.3"
        }

    stages {
        stage('Clone sources'){
            steps {
                git url: 'https://github.com/Anusha-DevOp/web_ex'
            }
        }
```

Stage (ii). ‘SonarQube analysis’ :



```
28            stage('SonarQube analysis') {
29                steps {
30                  //Prepare SonarQube scanner enviornment
31                  withSonarQubeEnv('SonarQube6.3') {
32                    bat 'mvn org.sonarsource.scanner.maven:sonar-maven-plugin:3.3.0.603:sonar'
33                  }
34                }
35            }
```

Line no. **28** : The dedicated stage to run SonarQube analysis.

Line no. **31** : Since version 2.5 of the SonarQube scanner for Jenkins, there is an official support of Jenkins pipeline. They provide ‘withSonarQubeEnv’ block that allow to select the SonarQube server instance you want to interact with.

Line no. **32** : Running & configuring scanner—triggering SonarQube analysis on maven projects, with the help of sonar-maven-plugin, available in **maven central repository**.

Stage (iii). ‘Artifactory configuration’ : this is the stage responsible for uploading the artifacts to artifactory.

Maven Builds with Artifactory:

Maven builds can resolve dependencies, deploy artifacts and can also publish build-info to Artifactory. To run Maven builds with artifactory from your pipeline script,

(1). Create Artifactory server instance (done in the beginning).

```
stage('Artifactory configuration'){

steps {

script {
rtMaven.tool ='Maven-3.5.3'

rtMaven.deployer releaseRepo: 'libs-release-local', 'libs-snapshot-local',
server: server

rtMaven.resolver releaseRepo:'libs-release', snapshotRepo: 'libs-snapshot',
server: server

rtMaven deployer.artifactDeploymentPatterns.addExclude("pom.xml")

buildInfo = Artifactory.newBuildInfo()

buildInfo.retention maxBuilds: 10, maxDays: 7, deleteBuildArtifacts: true

buildInfo.env.capture = true
}
}
```

(2) Create an Artifactory Maven Build instance, as well as define the location to deploy Maven build artifacts (jar, war, ear) into. Also, the location to download dependencies from.

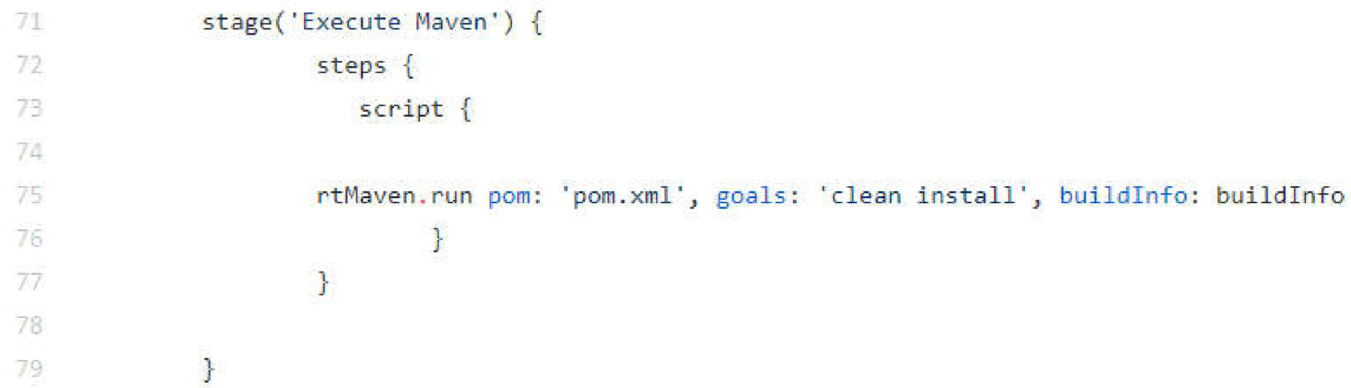
* For instance, release dependencies to be resolved from the ‘libs-release’ repository & the snapshot dependencies from the ‘libs-snapshot’ repository.

*Though by default all the build artifacts are deployed to Artifactory, filter can also be applied based on their names, using the ‘addInclude’ & ‘addExclude’ method.

(3) We can also capture the enviornment variables and publish the build information in artifactory. Set build-Info object to automatically capture enviornment variables while downloading & uploadng files.

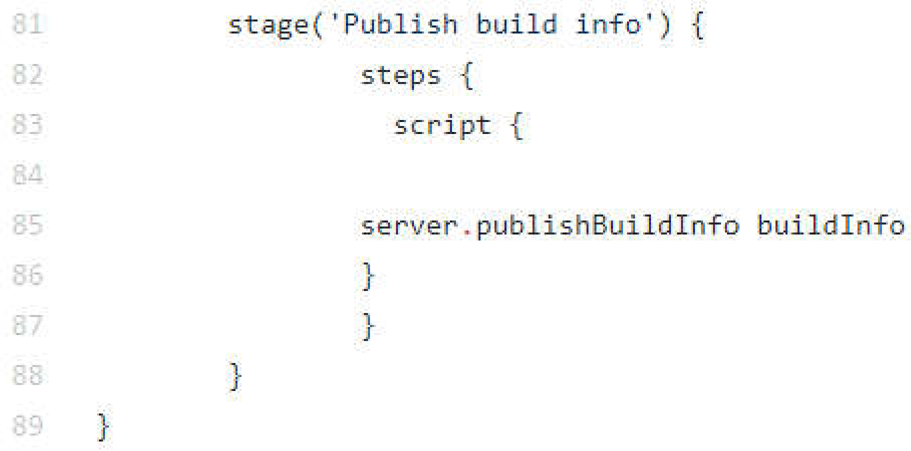
*By default enviornment variables, like ‘password’, ‘secret’, or ‘key’ are excluded & will not be published to Artifactory.

Stage (iv). Executing Maven goals



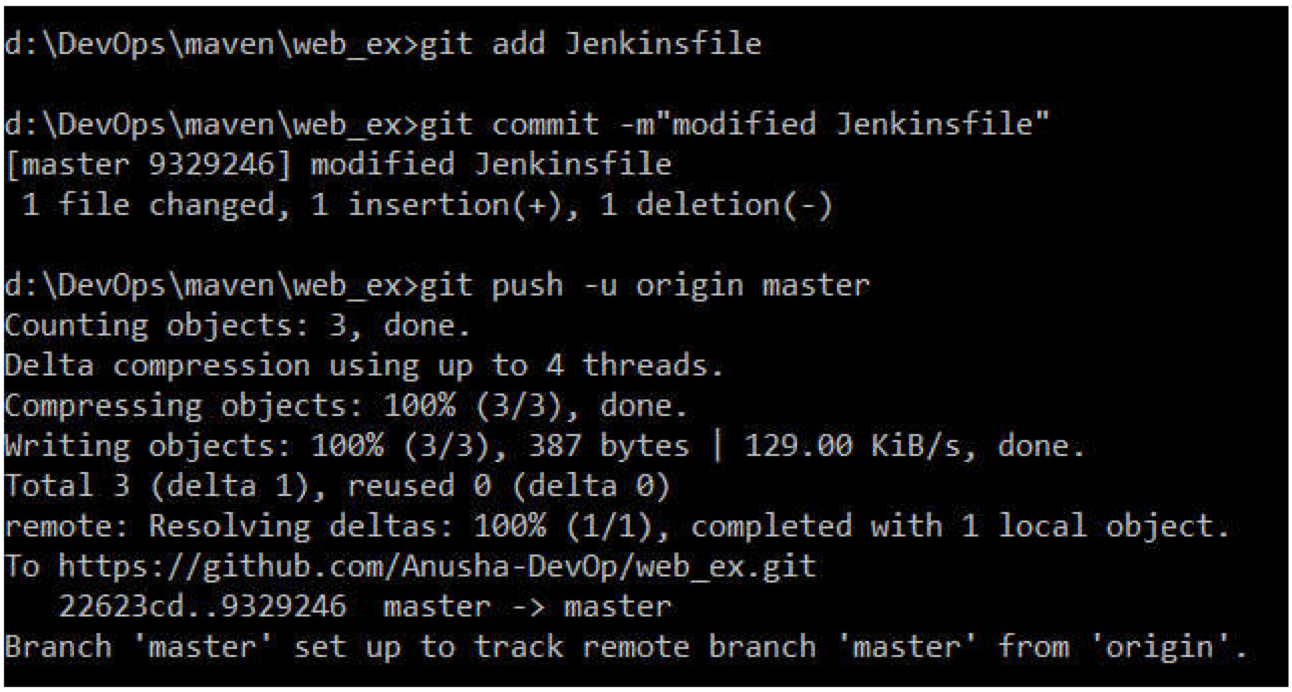
```
        stage('Execute Maven') {
                steps {
                   script {

                rtMaven.run pom: 'pom.xml', goals: 'clean install', buildInfo: buildInfo
                        }
                }

        }
```

Stage (v). Publishing Build Information:



```
        stage('Publish build info') {
                steps {
                  script {

                server.publishBuildInfo buildInfo
                }
                }
        }
}
```

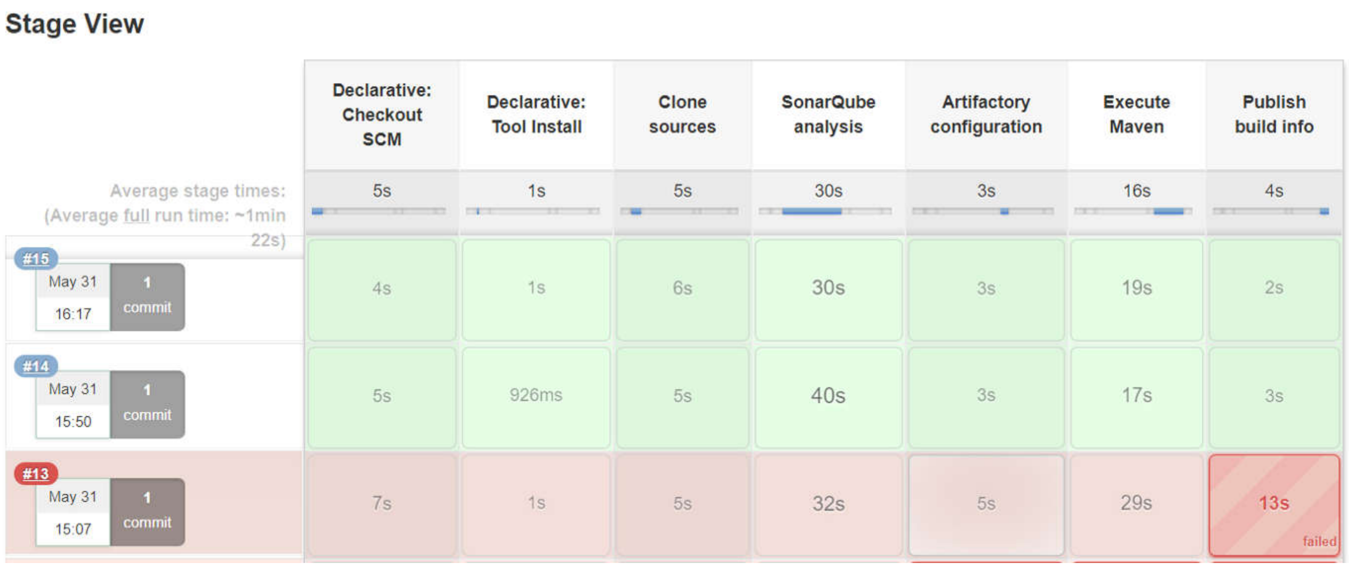
Step 5. Commit & push the Jenkinsfile in the source code repository specified in Stage (i), with the git (Use git add, git commit & git push command).



```
d:\DevOps\maven\web_ex>git add Jenkinsfile

d:\DevOps\maven\web_ex>git commit -m"modified Jenkinsfile"
[master 9329246] modified Jenkinsfile
 1 file changed, 1 insertion(+), 1 deletion(-)

d:\DevOps\maven\web_ex>git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 387 bytes | 129.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Anusha-DevOp/web_ex.git
   22623cd..9329246  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Step 6. Build Project/job:

Go to Jenkins dashboard. Select your project ("CD-pipeline"). Click on "Build Now".



Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Clone sources	SonarQube analysis	Artifactory configuration	Execute Maven	Publish build info
Average stage times: (Average full run time: ~1min 22s)	5s	1s	5s	30s	3s	16s	4s
#15 May 31 16:17 1 commit	4s	1s	6s	30s	3s	19s	2s
#14 May 31 15:50 1 commit	5s	926ms	5s	40s	3s	17s	3s
#13 May 31 15:07 1 commit	7s	1s	5s	32s	5s	29s	13s failed

Pipeline Execution : Jenkins

*Note 1: Each stage defined in the script can be visualized in the stage view, along with the details (time spent & logs).

*Note 2: Moreover it is easy to visualize and track the build process. It helps to detect the point/stage where & which build failed.

*Note 3: Further, we can notify the developers regarding the failed build.

Congratulations !! You have finally automated your pipeline in Jenkins.