SCOTIA BANK\_JENKINS

Why do we need pipeline as code?

Ans –‘Pipeline as Code’ describes a set of features that allow Jenkins users to define pipelined job processes with code, stored and versioned in a source repository. These features allow Jenkins to discover, manage, and run jobs for multiple source repositories and branches — eliminating the need for manual job creation and management.

* With the introduction of the [Pipeline plugin](https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Plugin), users now can implement a project’s entire build/test/deploy pipeline in a Jenkinsfile and store that alongside their code, treating their pipeline as another piece of code checked into source control.
* The Pipeline plugin introduces a domain-specific language (DSL) that helps Jenkins users to model their software delivery pipelines as code, which can be checked in and version-controlled along with the rest of their project's source code.
* There are 2 ways of doing helping Jenkins users to model their software delivery pipelines as code that –

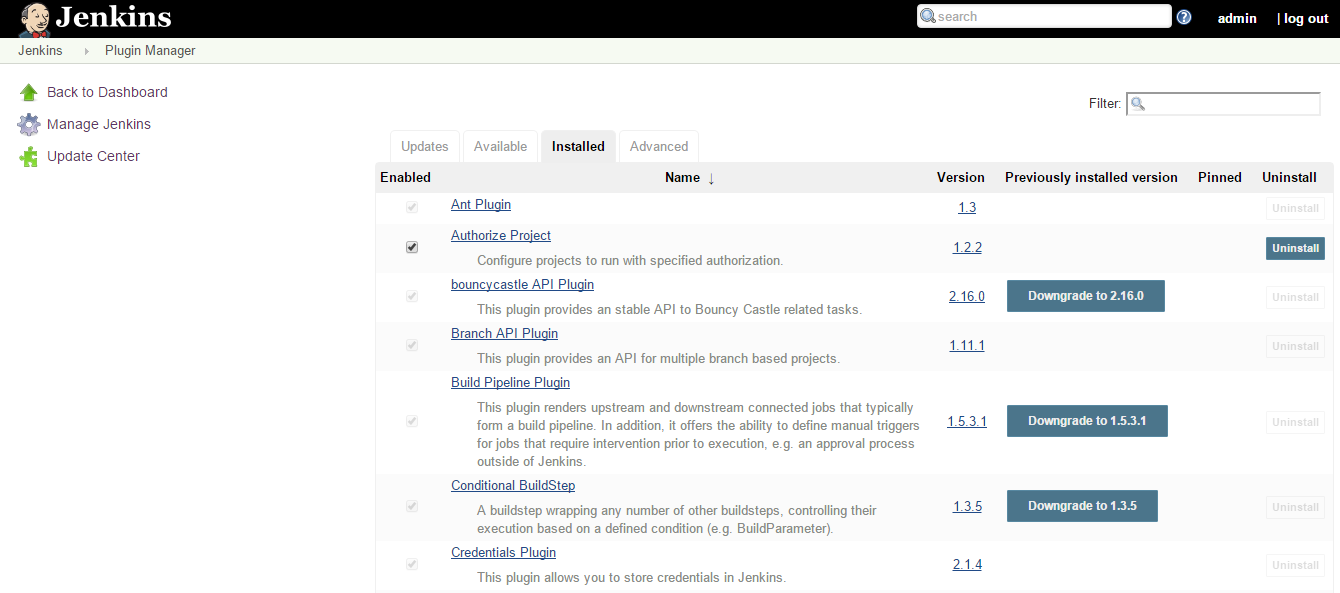
1. By creating jenkinsfile & using Pipeline Script from SCM inside Jenkins job
2. By using Pipeline Script inside Jenkins job

Firstly, we have to install Pipeline plugin in Jenkins:-

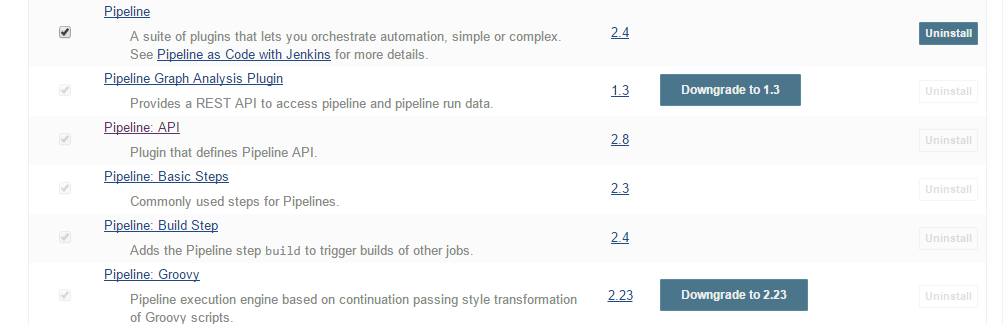
To get the ‘pipeline’ option in the New Item list 🡪 we want to install many dependency plugins for that🡪 to get the list of all those dependency plugins search for the required plugin i.e, Pipeline plugin and do install from jenkins UI i.e, from the shown below page

Go to Manage Jenkins 🡪 Manage plugins 🡪 you will get a page like shown below (Fig.1.1)

🡪Click on ‘Available tab’ 🡪 in the search bar search ‘Pipeline plugin’ 🡪Then, you will be able to see a page (Fig.1.2)

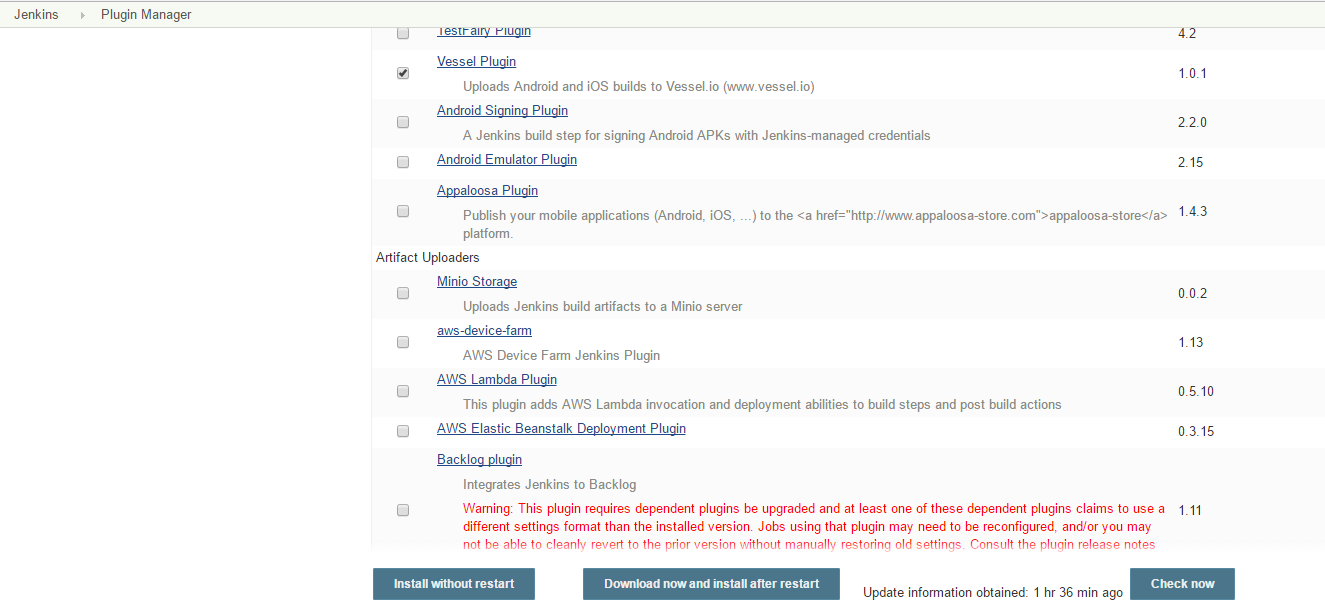


(Fig.1.1)

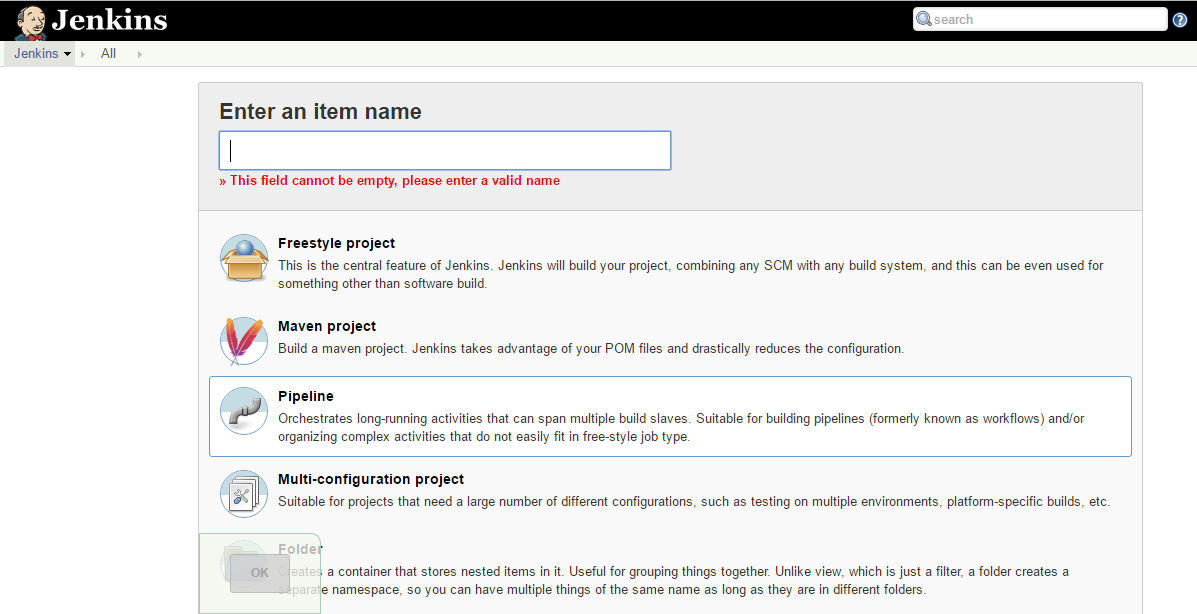


(Fig.1.2)

Now, check the checkbox for ‘pipeline plugin’ 🡪 click on ‘Install without restart’ as shown below -



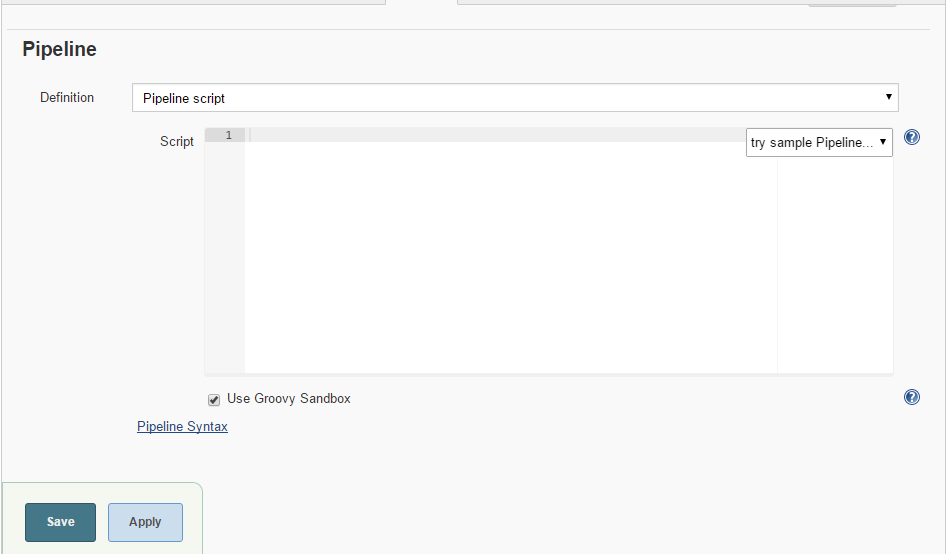
If it is successfully installed in your Jenkins then, we will be able to see the ‘pipeline’ option in New Items –



For example, I have created one pipeline job (i.e, Pipeline).

(Refer Fig.1.3 below) Inside that job we will be able to find Pipeline section, in which Definition field, there is a dropdown list having two options –

* Pipeline Script from SCM or,
* Pipeline Script



(Fig.1.3)

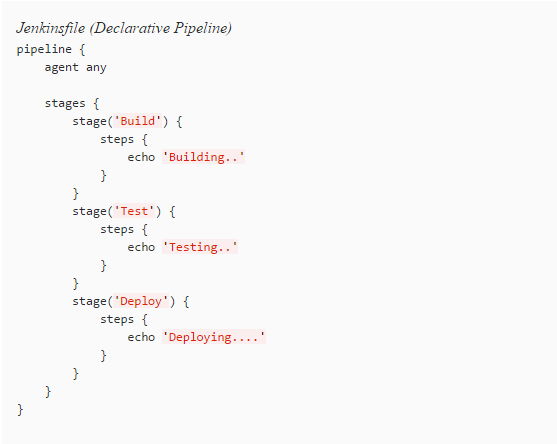
* By using ‘Pipeline Script from SCM’ inside Jenkins job :-

Steps :-

1. First create a jenkinsfile 🡪

A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline and is checked into source control.

Consider the following Pipeline, which implements a basic three-stage continuous delivery pipeline.



Not all Pipelines will have these same three stages, but it is a good starting point to define them for most projects. Using a text editor, ideally one which supports [Groovy](http://groovy-lang.org/) syntax highlighting, create a new Jenkinsfile.

* **Build:-**
* For many projects, the beginning of "work" in the Pipeline would be the "build" stage. Typically, this stage of the Pipeline will be where source code is assembled, compiled, or packaged. The Jenkinsfile is **not** a replacement for an existing build tool such as GNU/Make, Maven, Gradle, etc., but rather can be viewed as a glue layer to bind the multiple phases of a project’s development lifecycle (build, test, deploy, etc.) together.
* Jenkins has a number of plugins for invoking practically any build tool in general use, but this example will simply invoke make from a shell step (sh). The sh step assumes the system is Unix/Linux-based, for Windows-based systems the bat could be used instead.

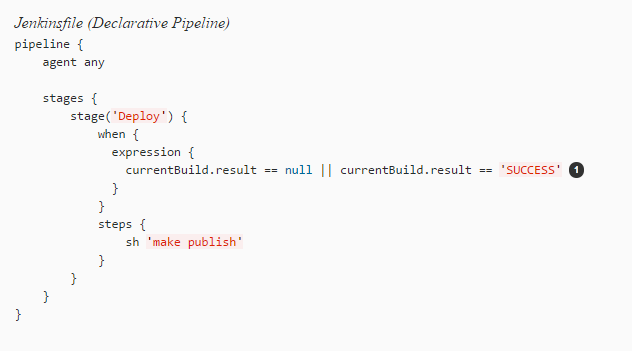


|  |  |
| --- | --- |
|  | * The sh step invokes the make command and will only continue if a zero exit code is returned by the command. Any non-zero exit code will fail the Pipeline. |
|  | * archiveArtifacts captures the files built matching the include pattern (\*\*/target/\*.jar) and saves them to the Jenkins master for later retrieval. |
|  | Test :-  * Running automated tests is a crucial component of any successful continuous delivery process. As such, Jenkins has a number of test recording, reporting, and visualization facilities provided by a [number of plugins](https://plugins.jenkins.io/?labels=report). At a fundamental level, when there are test failures, it is useful to have Jenkins record the failures for reporting and visualization in the web UI. The example below uses the junit step, provided by the [JUnit plugin](https://plugins.jenkins.io/junit). * In the example below, if tests fail, the Pipeline is marked "unstable", as denoted by a yellow ball in the web UI. Based on the recorded test reports, Jenkins can also provide historical trend analysis and visualization. |

* Using an inline shell conditional (sh 'make || true') ensures that the sh step always sees a zero exit code, giving the junit step the opportunity to capture and process the test reports. Alternative approaches to this are covered in more detail in the [Handling Failures](https://jenkins.io/doc/book/pipeline/jenkinsfile/#handling-failures) section below.
* junit captures and associates the JUnit XML files matching the inclusion pattern (\*\*/target/\*.xml)

### Deploy :-

* Deployment can imply a variety of steps, depending on the project or organization requirements, and may be anything from publishing built artifacts to an Artifactory server, to pushing code to a production system.
* At this stage of the example Pipeline, both the "Build" and "Test" stages have successfully executed. In essense, the "Deploy" stage will only execute assuming previous stages completed successfully, otherwise the Pipeline would have exited early.



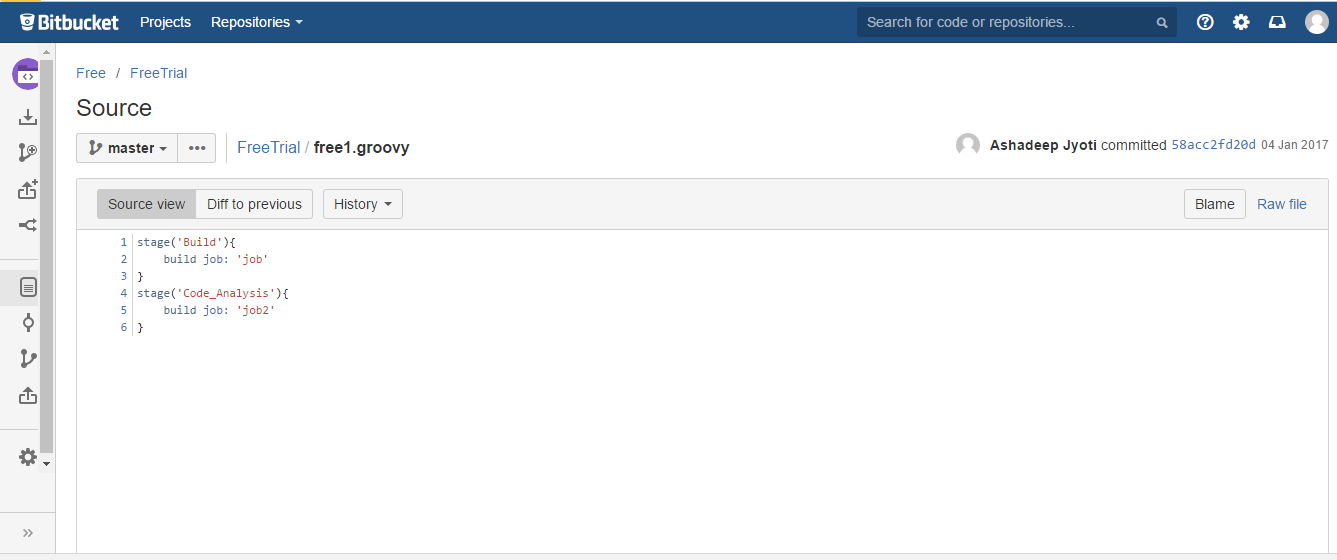
* Accessing the currentBuild.result variable allows the Pipeline to determine if there were any test failures. In which case, the value would be UNSTABLE.

NOTE: - A Scripted Pipeline can include conditional tests (shown above), loops, try/catch/finally blocks and even functions.

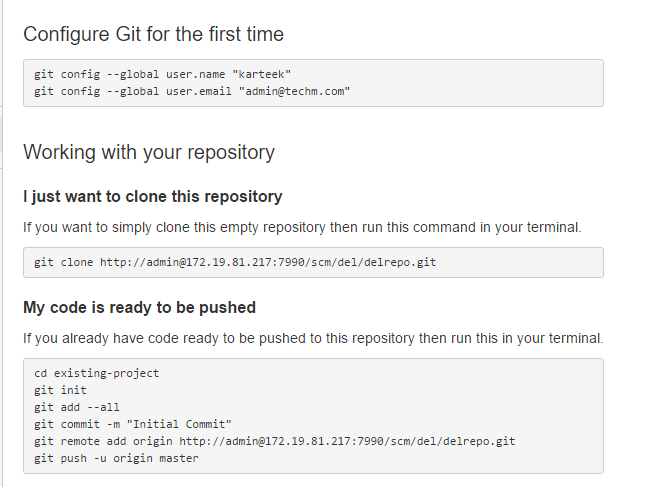
**WHAT I HAVE DONE** for ‘Pipeline Script from SCM’ ?

I have created one free1.groovy file (with extension .groovy) using text editor i.e, Notepad++ where I have kept my pipeline script there.

Then I created one repository in Bitbucket as - FreeTrial as shown below-

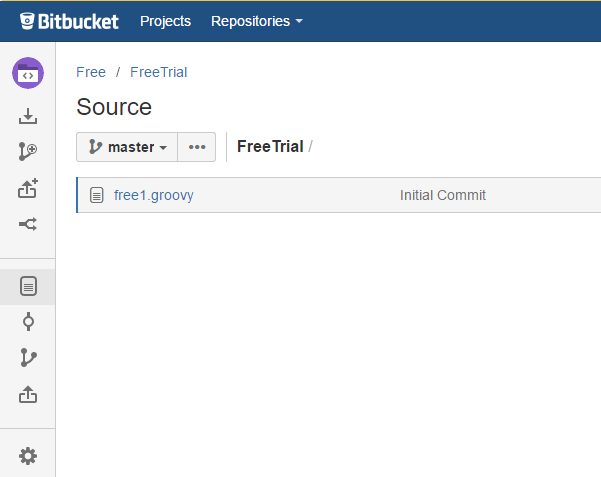


Now you have an Empty repository 🡪 pushed the free1.groovy file to that repository using following steps –

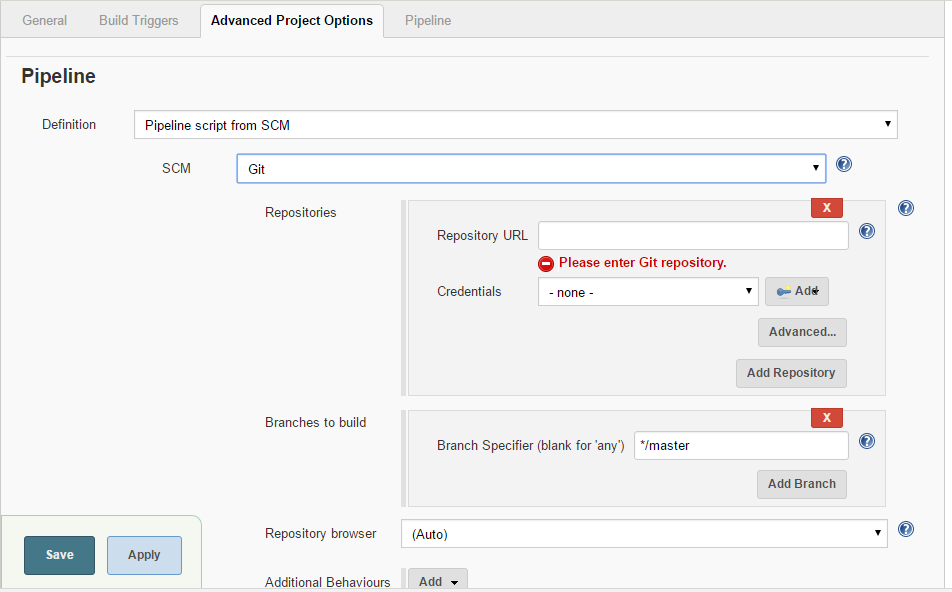




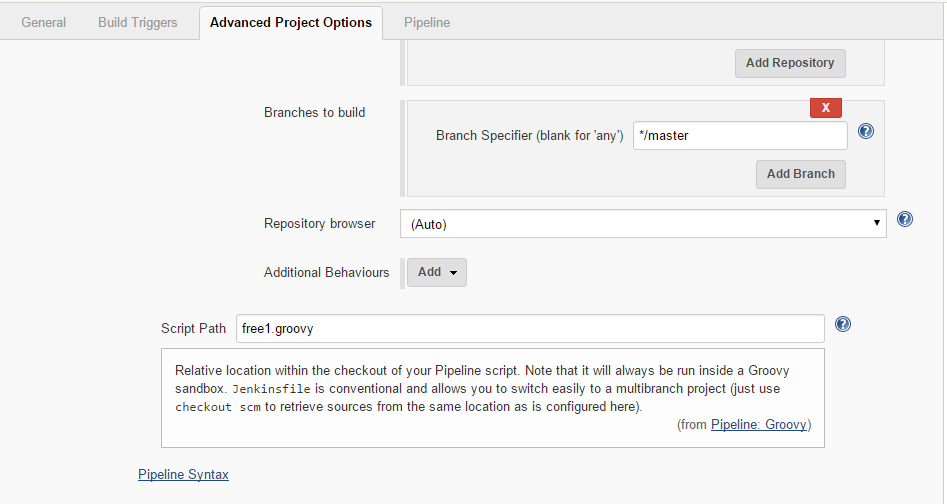
Now you will be able to see that file inside the repo –



Now Go to your Jenkins UI🡪 create a New Item of Job type ‘Pipeline’ 🡪 click on ‘Pipeline ‘tab 🡪 In Definition field Select ‘Pipeline Script from SCM’ 🡪 Enter your repository URL and credentials 🡪



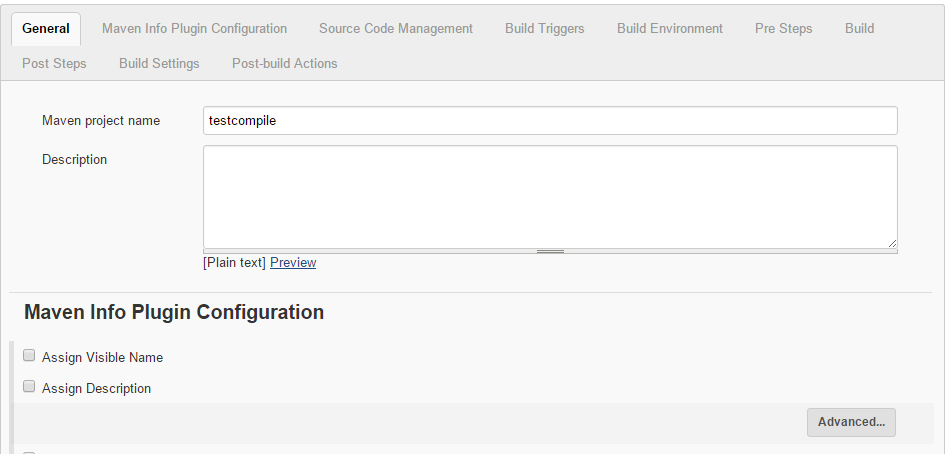
And, give the Jenkins filename there with that name by what it’s there in the repo i.e, free1.groovy

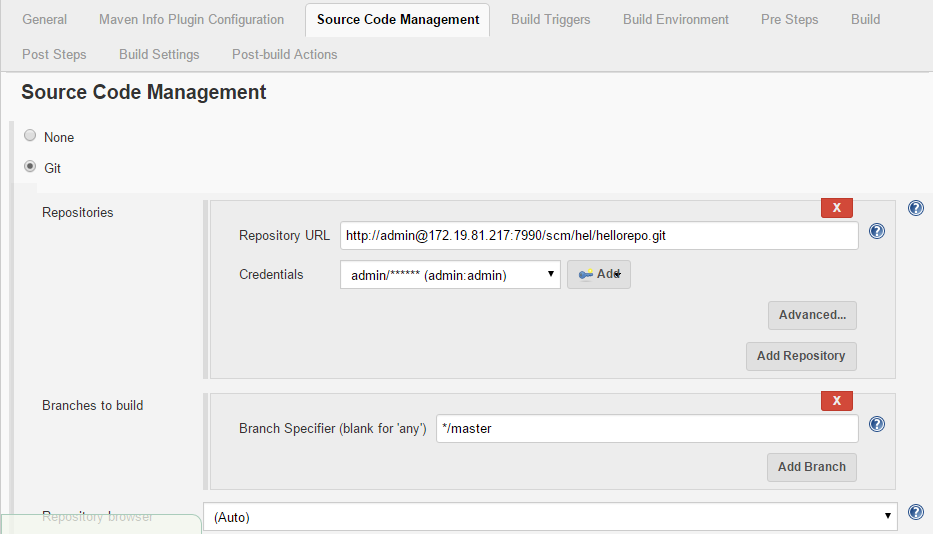


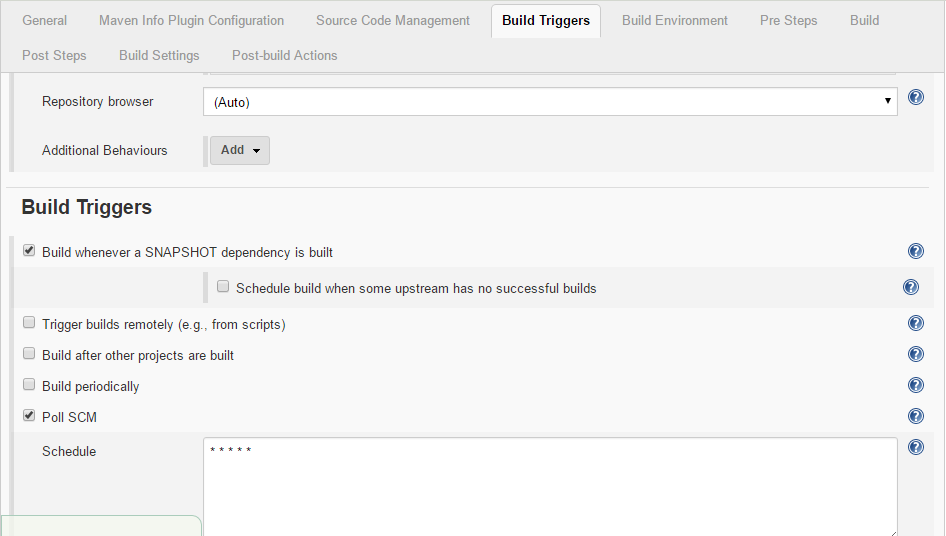
**WHAT I HAVE DONE** for ‘Pipeline Script’?

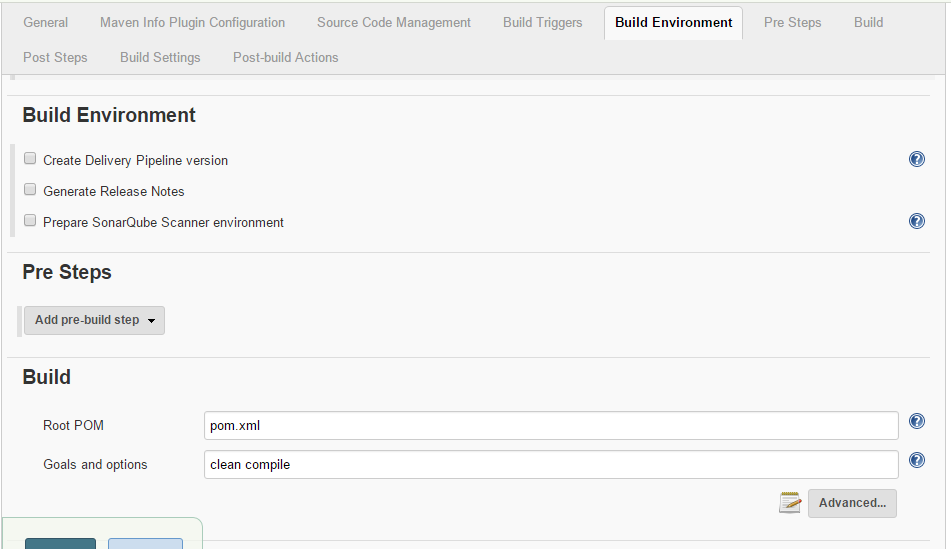
I have created 3 jobs for – compile, code analysis and package shown below –

1. testcompile – compile job name

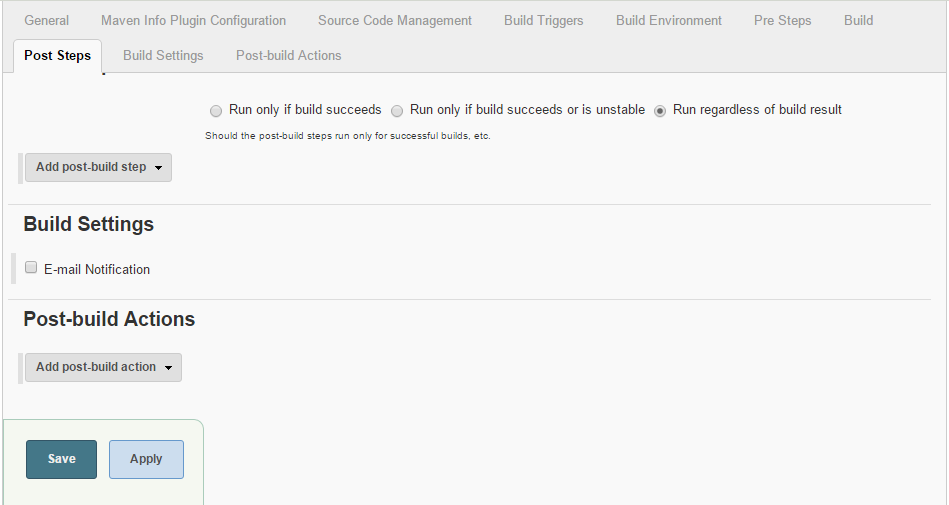




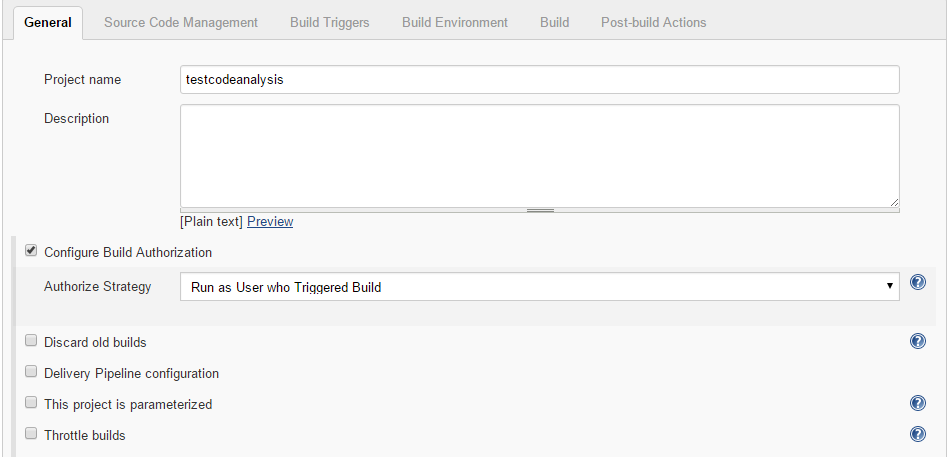


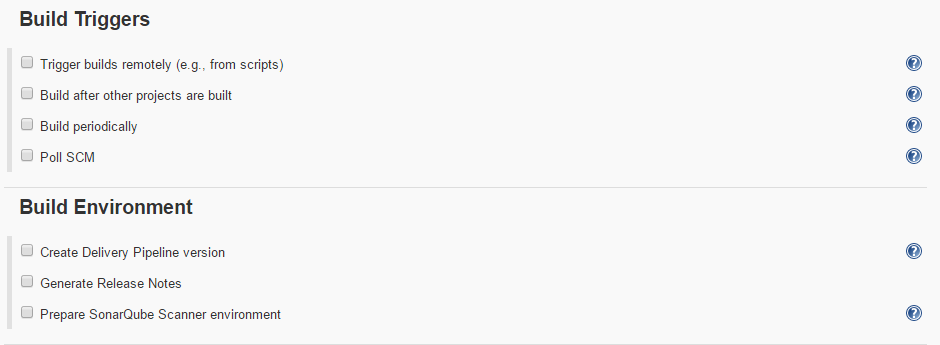


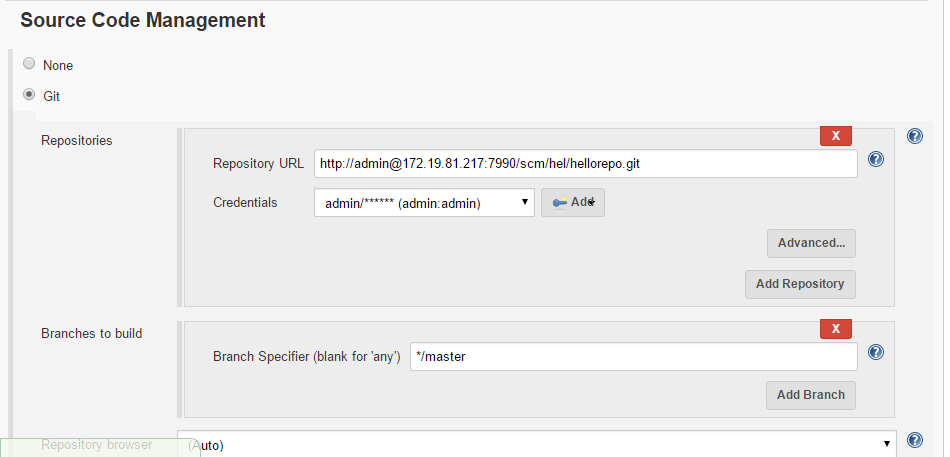
Here below, you can see no need of giving Add Post Steps as we ‘re using pipeline script the script will automatically connect all the jobs and will create a pipeline.

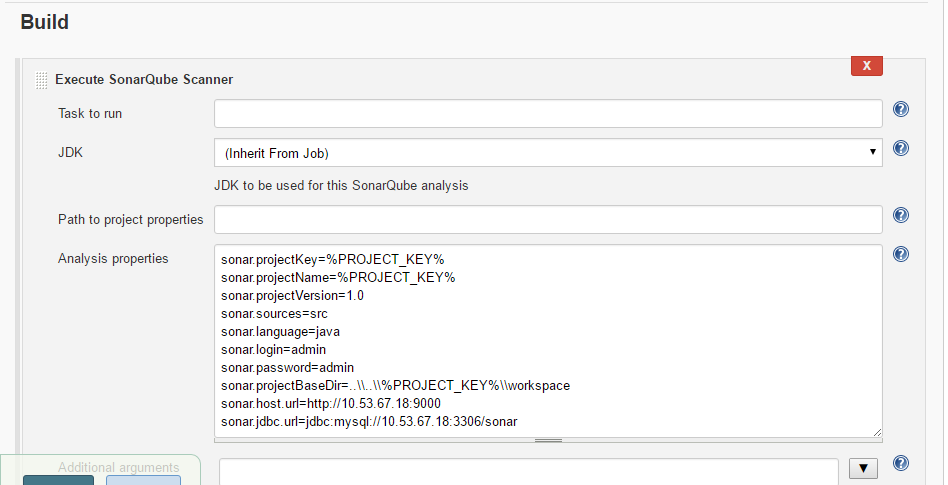


1. testcodeanalysis – code analysis job

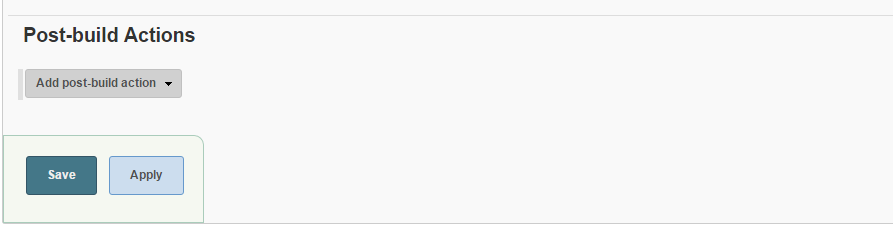




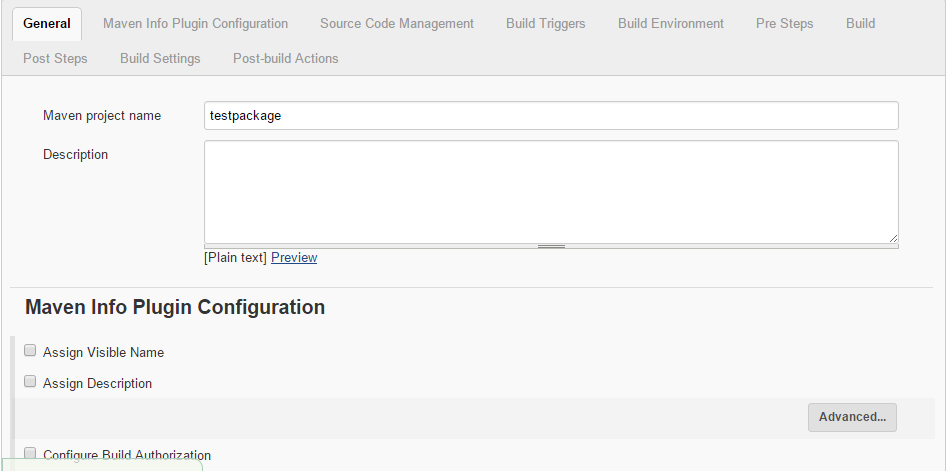


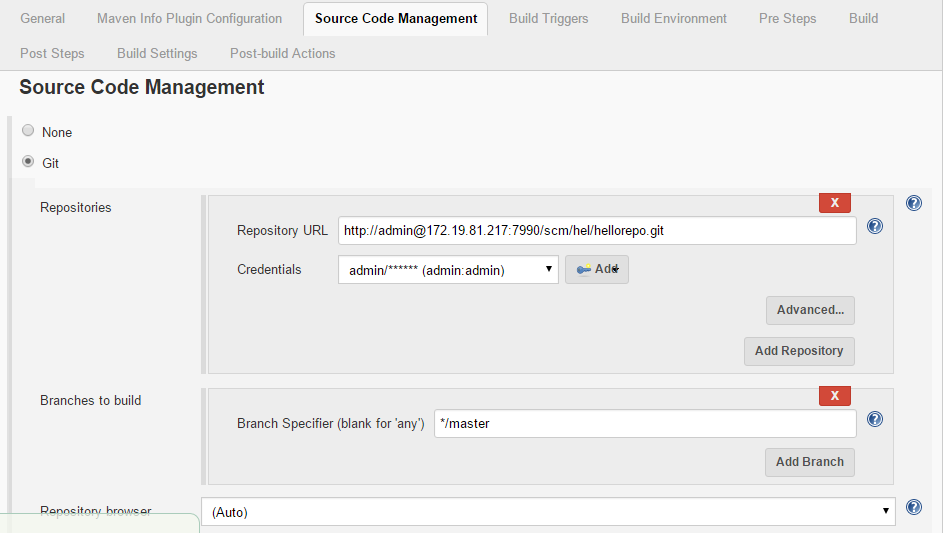


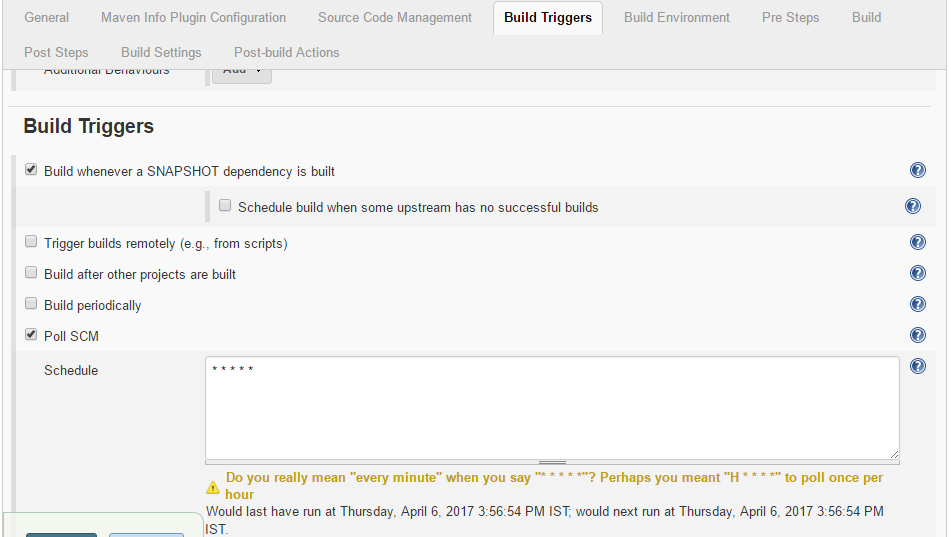
Here below, you can see no need of giving Add Post Steps as we ‘re using pipeline script the script will automatically connect all the jobs and will create a pipeline.

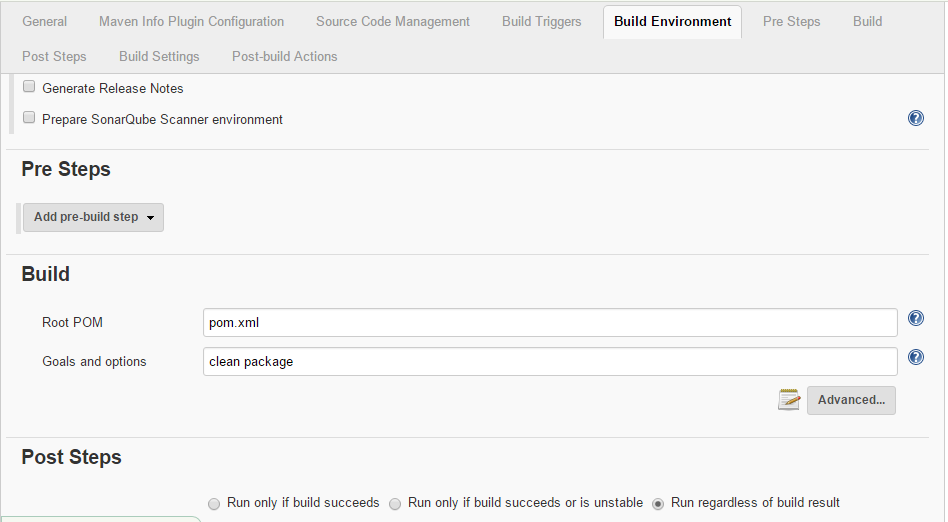


3.testpackage – package job

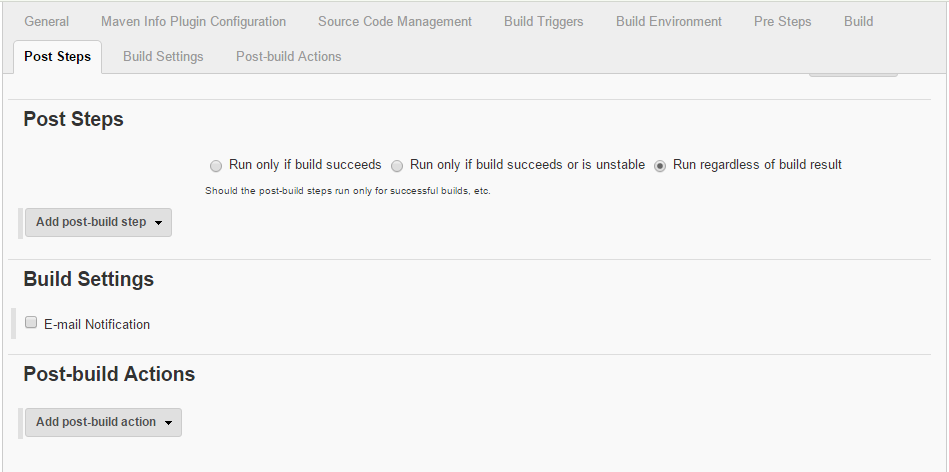




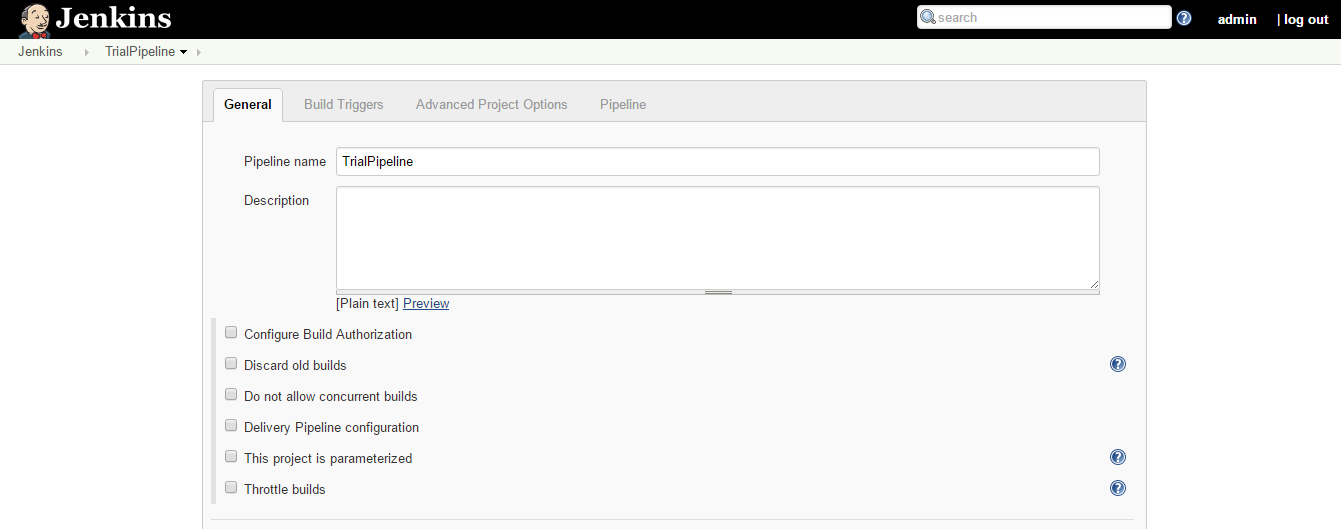




Here, you can see no need of giving Add Post Steps as we‘re using pipeline script the script will automatically connect all the jobs and will create a pipeline.

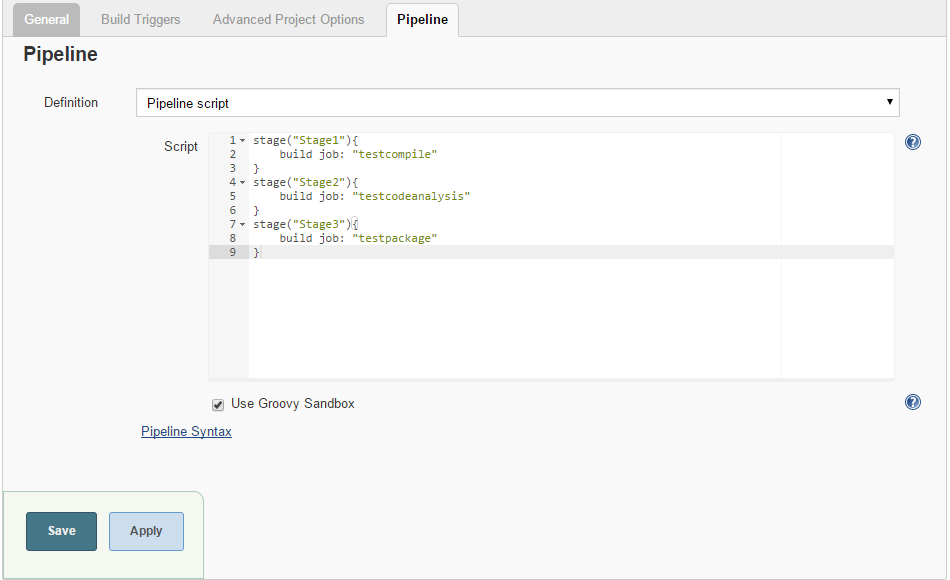


Now I have created a pipeline job named ‘TrialPipeline’ as shown below –



This Pipeline job has one configuration called Pipeline in that I have put my script as shown below –

where Stage 1 will execute first and the build job: contains a jobname which will be our by-default initial job(testcompile) and like that it goes on next will be our second job(testcodeanalysis)and last one will be our third job(testpackage)

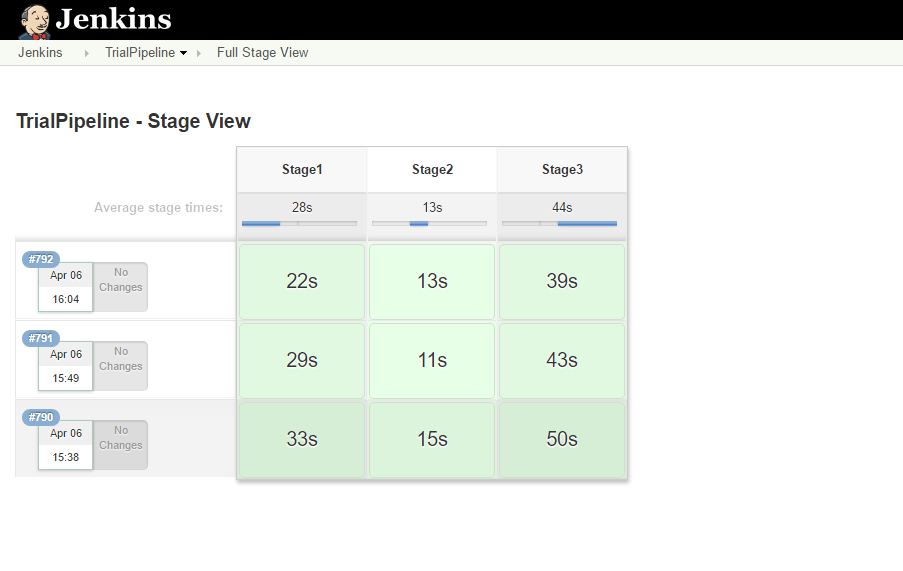


Now, if we will Build this pipeline job then it will create a pipeline for us as shown below –

Green Color-SUCCESS

Red Color-FAILURE

Left side shows us the Build number along with at what time it gets triggered.



## **Advanced Syntax for Pipeline :-**

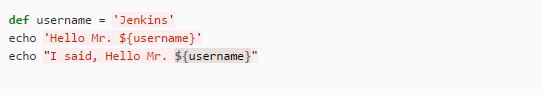
### String Interpolation:-

Jenkins Pipeline uses rules identical to [Groovy](http://groovy-lang.org/) for string interpolation. Groovy’s String interpolation support can be confusing to many newcomers to the language. While Groovy supports declaring a string with either single quotes, or double quotes, for example:

Only the latter string will support the dollar-sign ($) based string interpolation, for example:



Would result in:



Hello Mr. ${username}

I said, Hello Mr. Jenkins

Understanding how to use string interpolation is vital for using some of Pipeline’s more advanced features.

### Working with the Environment

Jenkins Pipeline exposes environment variables via the global variable env, which is available from anywhere within a Jenkinsfile. The full list of environment variables accessible from within Jenkins Pipeline is documented at [localhost: 8080/pipeline-syntax/globals#env](http://localhost:8080/pipeline-syntax/globals#env), assuming a Jenkins master is running on localhost: 8080, and includes:

**BUILD\_ID**

The current build ID, identical to BUILD\_NUMBER for builds created in Jenkins versions 1.597+

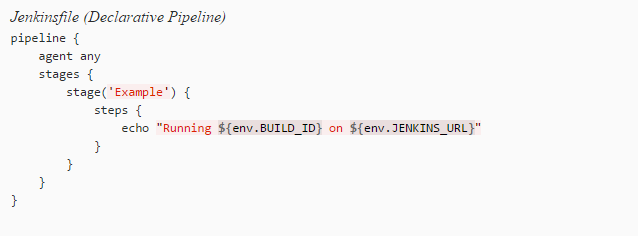
**JOB\_NAME**

Name of the project of this build, such as "foo" or "foo/bar".

**JENKINS\_URL**

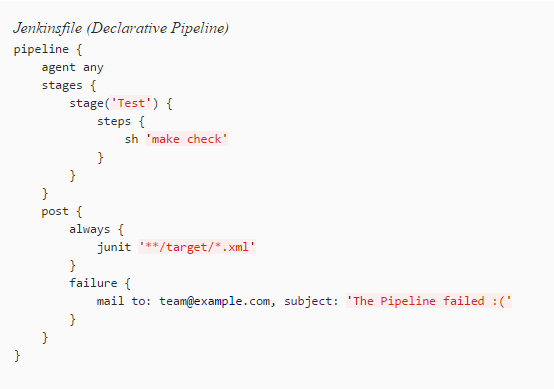
Full URL of Jenkins, such as [example.com:port/jenkins/](http://example.com:port/jenkins/) (NOTE: only available if Jenkins URL set in "System Configuration")

Referencing or using these environment variables can be accomplished like accessing any key in a Groovy [Map](http://groovy-lang.org/syntax.html#_maps), for example:



### Handling Failures:-

Declarative Pipeline supports robust failure handling by default via its [post section](https://jenkins.io/doc/book/pipeline/syntax/#post) , which allows declaring a number of different "post conditions" such as: always, unstable, success, failure, and changed. The [Pipeline Syntax](https://jenkins.io/doc/book/pipeline/jenkinsfile/#syntax) section provides more detail on how to use the various post conditions.



### Using multiple agents:-

In all the previous examples, only a single agent has been used. This means Jenkins will allocate an executor wherever one is available, regardless of how it is labeled or configured. Not only can this behavior be overridden, but Pipeline allows utilizing multiple agents in the Jenkins environment from within the same Jenkinsfile, which can helpful for more advanced use-cases such as executing builds/tests across multiple platforms.

In the example below, the "Build" stage will be performed on one agent and the built results will be reused on two subsequent agents, labelled "linux" and "windows" respectively, during the "Test" stage.





* The stash step allows capturing files matching an inclusion pattern (\*\*/target/\*.jar) for reuse within the same Pipeline. Once the Pipeline has completed its execution, stashed files are deleted from the Jenkins master.
* The parameter in agent/node allows for any valid Jenkins label expression. Consult the [Pipeline Syntax](https://jenkins.io/doc/book/pipeline/syntax/) section for more details
* unstash will retrieve the named "stash" from the Jenkins master into the Pipeline’s current workspace.
* The bat script allows for executing batch scripts on Windows-based platforms.

### Advanced Scripted Pipeline :-

Scripted Pipeline is a domain-specific language [[3](https://jenkins.io/doc/book/pipeline/jenkinsfile/#_footnote_3)] based on Groovy, most [Groovy syntax](http://groovy-lang.org/semantics.html) can be used in Scripted Pipeline without modification.

#### **Executing in parallel**

The example in the [section above](https://jenkins.io/doc/book/pipeline/jenkinsfile/#using-multiple-nodes) runs tests across two different platforms in a linear series. In practice, if the make check execution takes 30 minutes to complete, the "Test" stage would now take 60 minutes to complete!

Fortunately, Pipeline has built-in functionality for executing portions of Scripted Pipeline in parallel, implemented in the aptly named parallel step.

Refactoring the example above to use the parallel step:

