Jenkins Pipeline : Declarative and IaC approaches for DevOps

Kawtar Oukil

Table des matières

I.Introducttion:

How we can write pipeline scripts and Jenkins using the UI in a declarative way and how we can also write Jenkins files and have that file committed to source code management in the tools such as get help and to have all of our pipelines committed as code.

Prerequires:

Jenkins server

java stalled DACA

choco

maven

a. Task 1: What is a pipeline?

we're going to focus a lot just on pipelines and how we can use that to automate all of our processes and so without further ado what exactly is a pipeline?

Jenkins pipeline is nothing more than a collection of plugins that supports our continuous delivery procedures from source code management down to end user and this is all concerned with the product development lifecycle, which starts from developers submitting code and having to go through all of those processes and the variations of different outcomes or different, for example, like a success or a failure.

Second point, your application code goes through a complex web of process is on its way to being released as a final product.

So we have these different environments, testing, staging deployment, um, and other environments as well and different stages.

We need to be able to automate, the processes that happen at each of these actual stages, which takes occurred from the repo level to release level.

The definition of a pipeline can be written either directly as a pipeline script which is written directly in the Ui. The Jenkins ui itself very popular way to actually define our pipelines and secondly, can also be written as a Jenkins file, which would contain some variation of the Ui version of the pipeline script essentially is file that's committed usually at root level along with the source code and a pipeline can be written either is declarative or scripted format.

just looking at the benefits of Jenkin file, Jenkins file, That process really immerses us in the whole infrastructure as code world.

And the pipeline is considered a part of the project code and it’s treated as such by being committed along with the application code. Of course, the pipeline code is part of the project code and having it at that level.

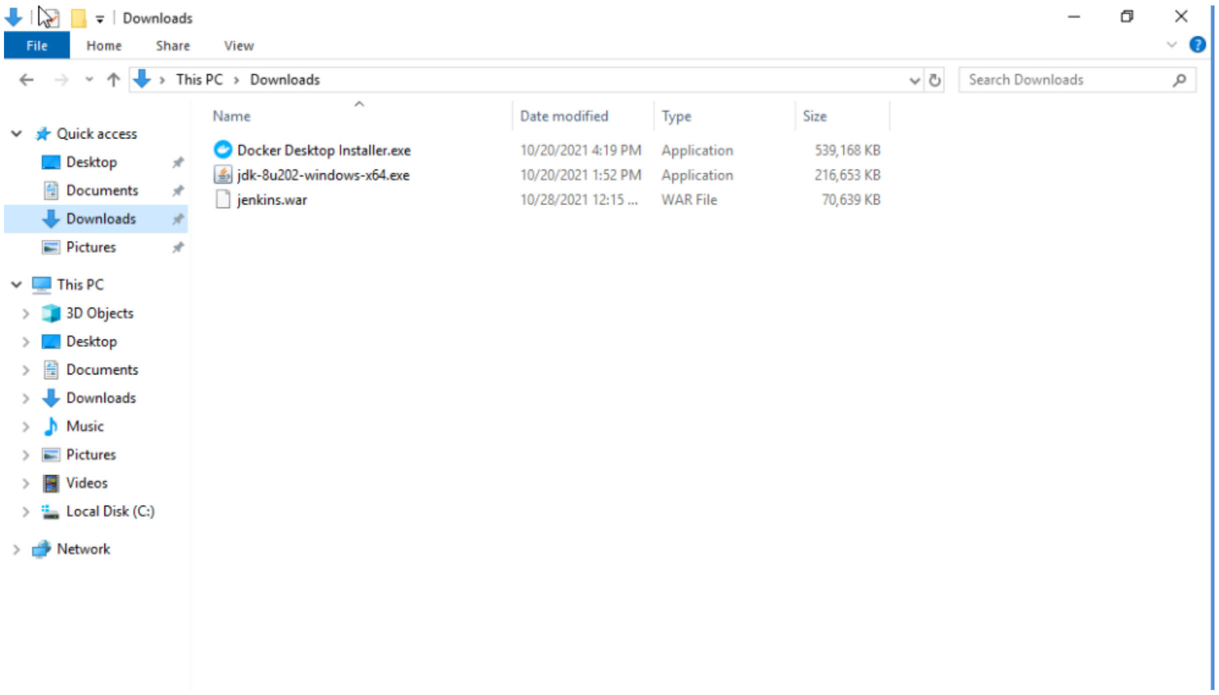
It could be visualized by all members of the team that it concerns um whether you're working in development, whether you're working in, in in the product team or if you're working in the operations team, for example or the testing team. everyone can have a look and see to what part of the pipeline they're concerned, which can be very useful.

b. Task 2: Pipeline Script format and global environment variables

In this task we gonna start writing our first pipeline scripts inside the Jenkins Ui, and also have a look at how we're able to inject environment variables into our code itself. So, we're gonna have a look at the environment variables that are available out the box and how we can actually inject that into our scripts.

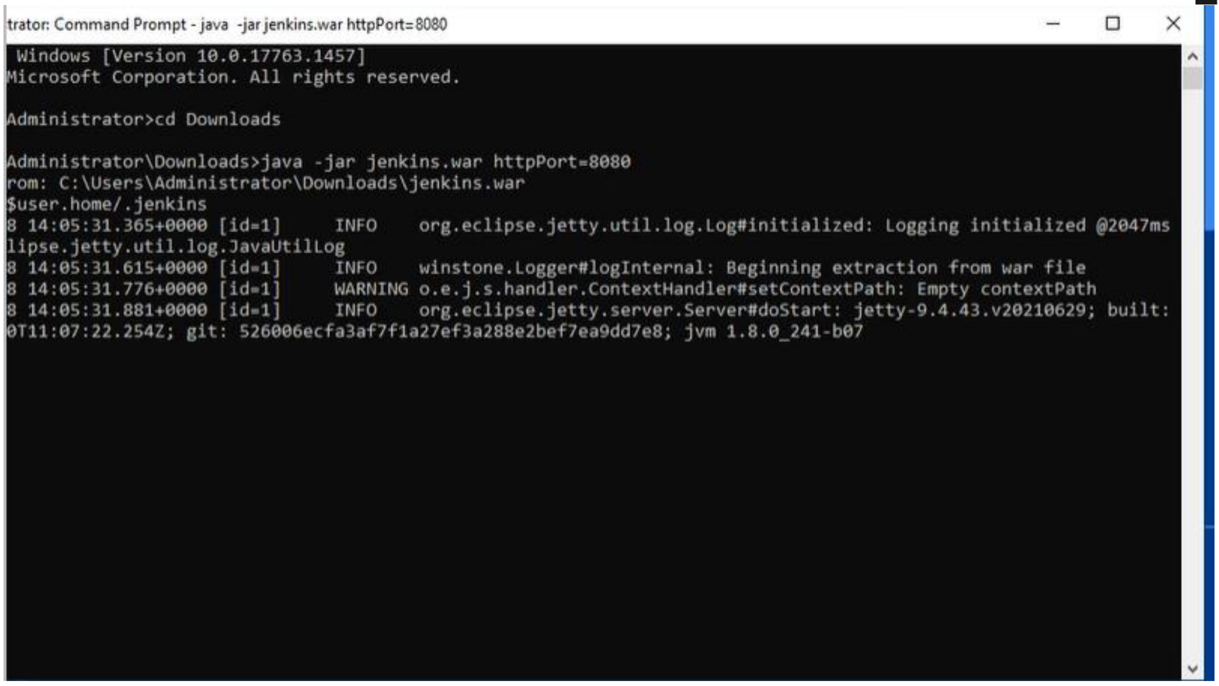
Firstly, we need to actually get our server up and running.

We should have installed the Jenkins. War in the first of course



Name	Date modified	Type	Size
Docker Desktop Installer.exe	10/20/2021 4:19 PM	Application	539,168 KB
jdk-8u202-windows-x64.exe	10/20/2021 1:52 PM	Application	216,653 KB
jenkins.war	10/28/2021 12:15 ...	WAR File	70,639 KB

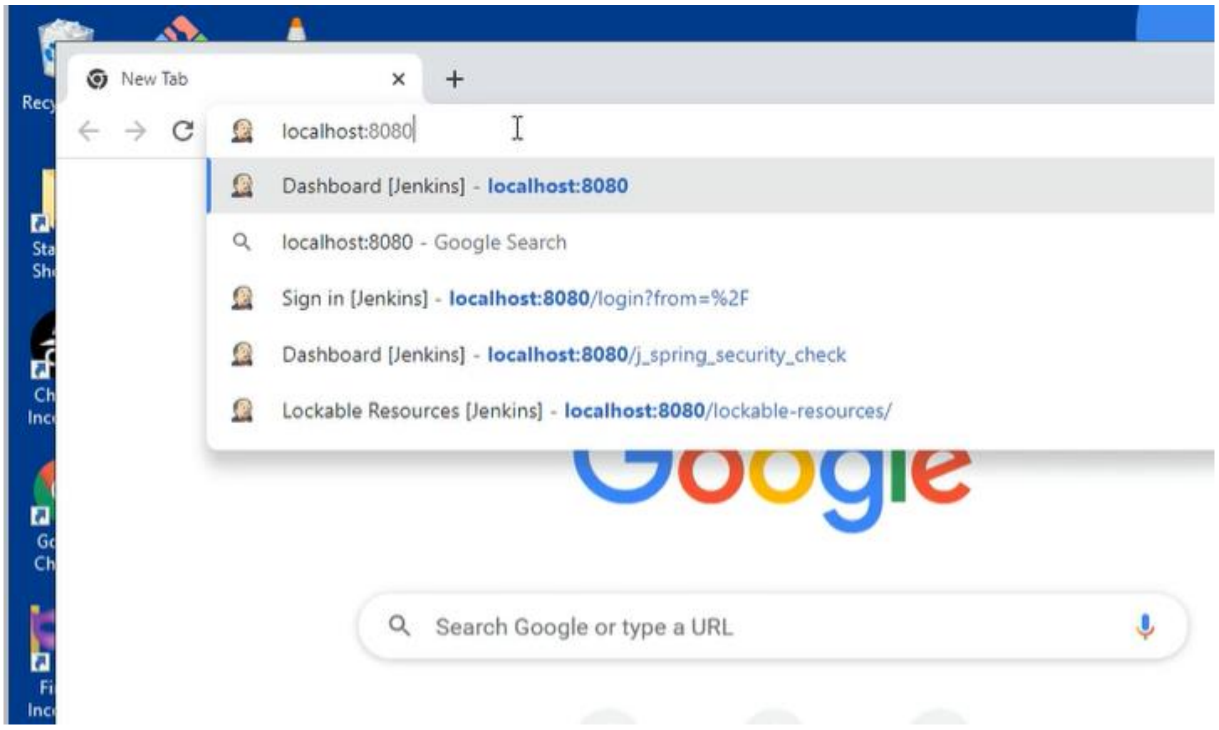
We open command prompts and then we want to see the into downloads and type the command:



```
trator: Command Prompt - java  -jar jenkins.war httpPort=8080
 Windows [Version 10.0.17763.1457]
Microsoft Corporation. All rights reserved.

Administrator>cd Downloads

Administrator\Downloads>java -jar jenkins.war httpPort=8080
rom: C:\Users\Administrator\Downloads\jenkins.war
$user.home/.jenkins
8 14:05:31.365+0000 [id=1]     INFO    org.eclipse.jetty.util.log.Log#initialized: Logging initialized @2047ms
lipse.jetty.util.log.JavaUtilLog
8 14:05:31.615+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginning extraction from war file
8 14:05:31.776+0000 [id=1]     WARNING o.e.j.s.handler.ContextHandler#setContextPath: Empty contextPath
8 14:05:31.881+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#doStart: jetty-9.4.43.v20210629; built:
0T11:07:22.254Z; git: 526006ecfa3af7f1a27ef3a288e2bef7ea9dd7e8; jvm 1.8.0_241-b07
```

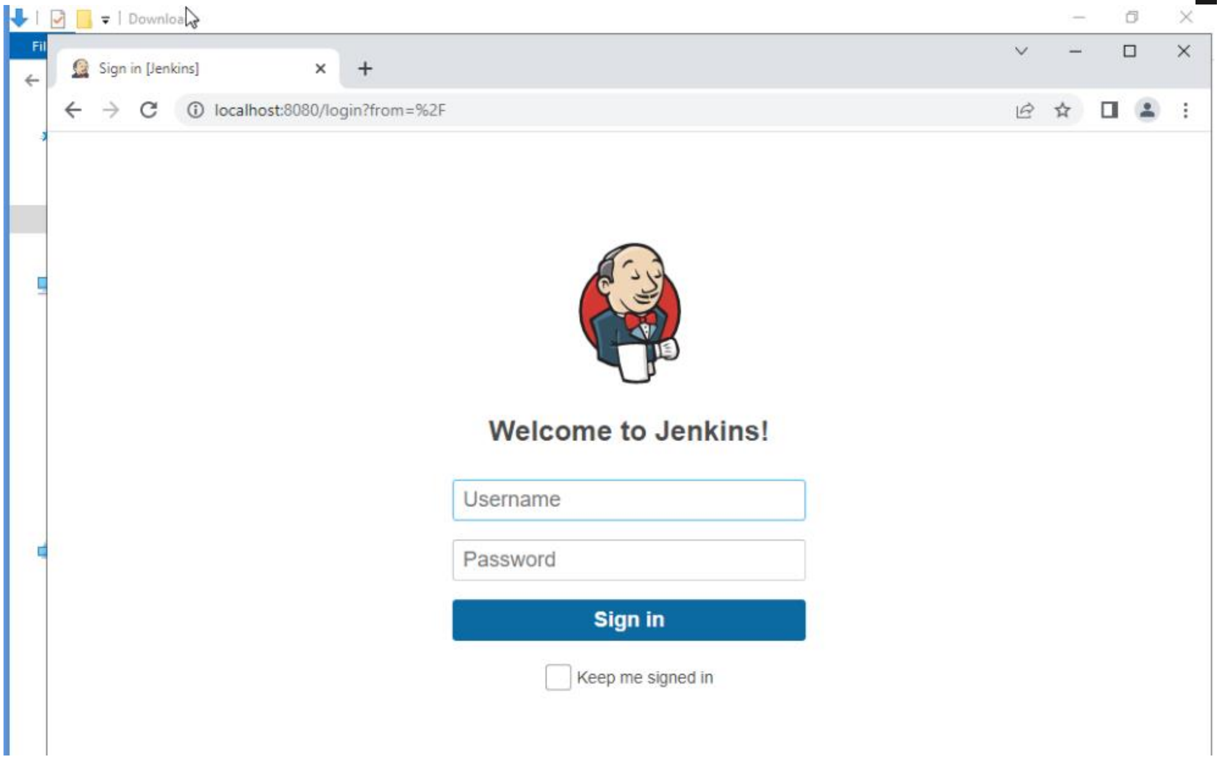
we have the default ports for Jenkins server of 8080

So, it's gonna expose a Jenkins server on this port of our local host With type in localhost:8080:

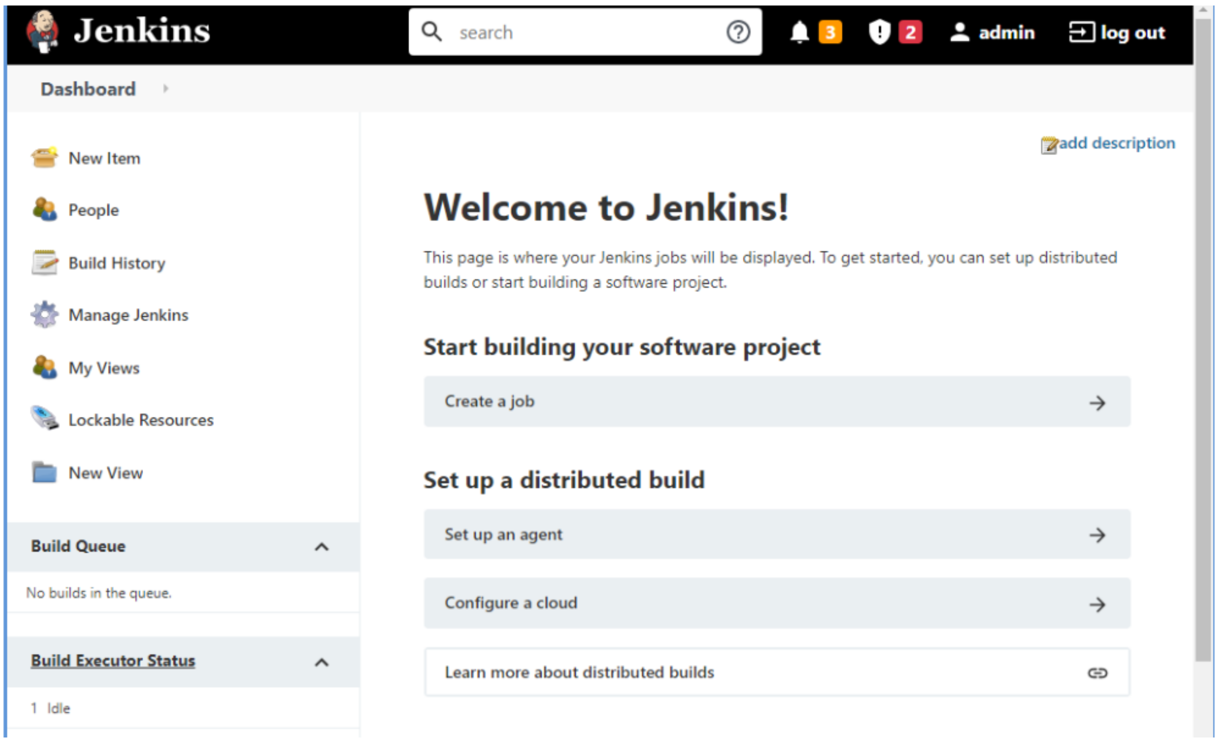


```
localhost:8080
Dashboard [Jenkins] - localhost:8080
localhost:8080 - Google Search
Sign in [Jenkins] - localhost:8080/login?from=%2F
Dashboard [Jenkins] - localhost:8080/j_spring_security_check
Lockable Resources [Jenkins] - localhost:8080/lockable-resources/
```

So, we should be brought here to the log in page, so we gonna type the name of user and the password. Here we type: User='admin' and password='admin'



This is now bringing us here to the Jenkins dashboard.

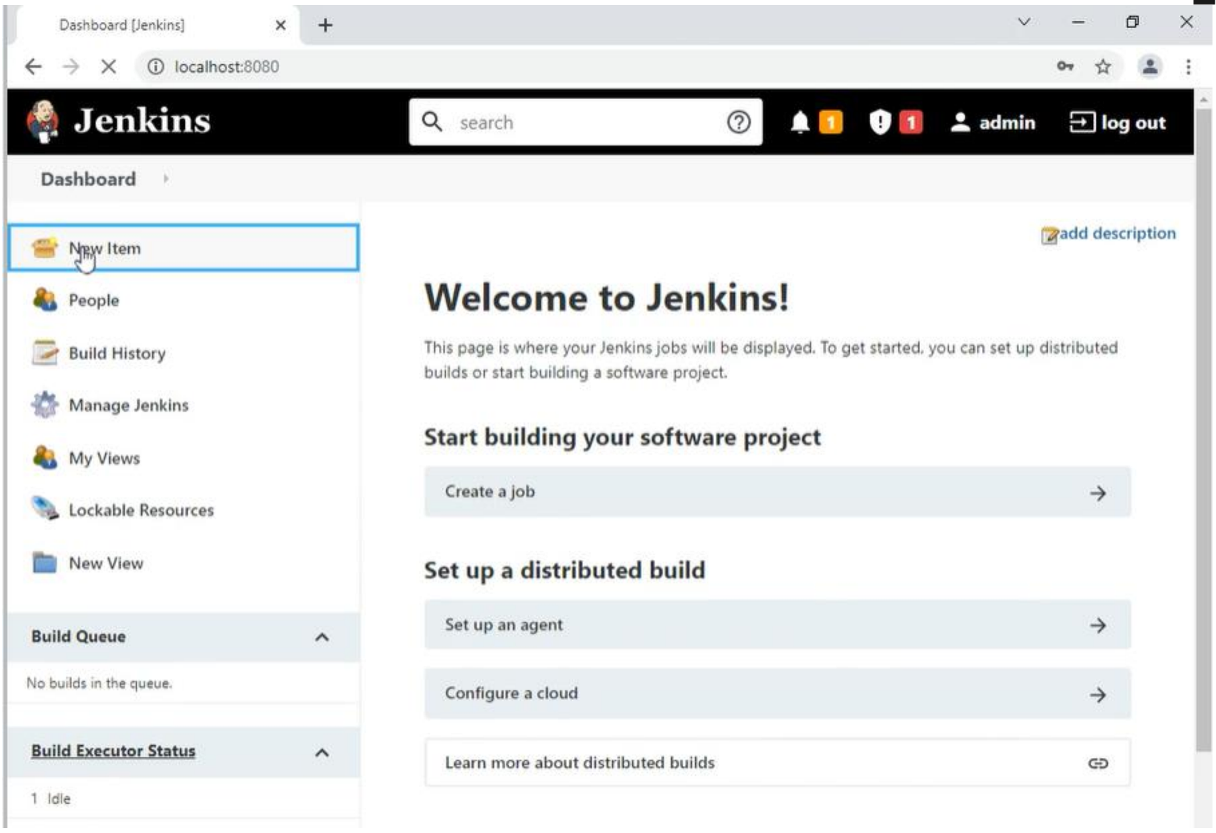


Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

No builds in the queue.

So let's click on 'new item' here


Dashboard [Jenkins]

localhost:8080

Jenkins

search

admin

log out

Dashboard

New Item

People

Build History

Manage Jenkins

My Views

Lockable Resources

New View

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

add description

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

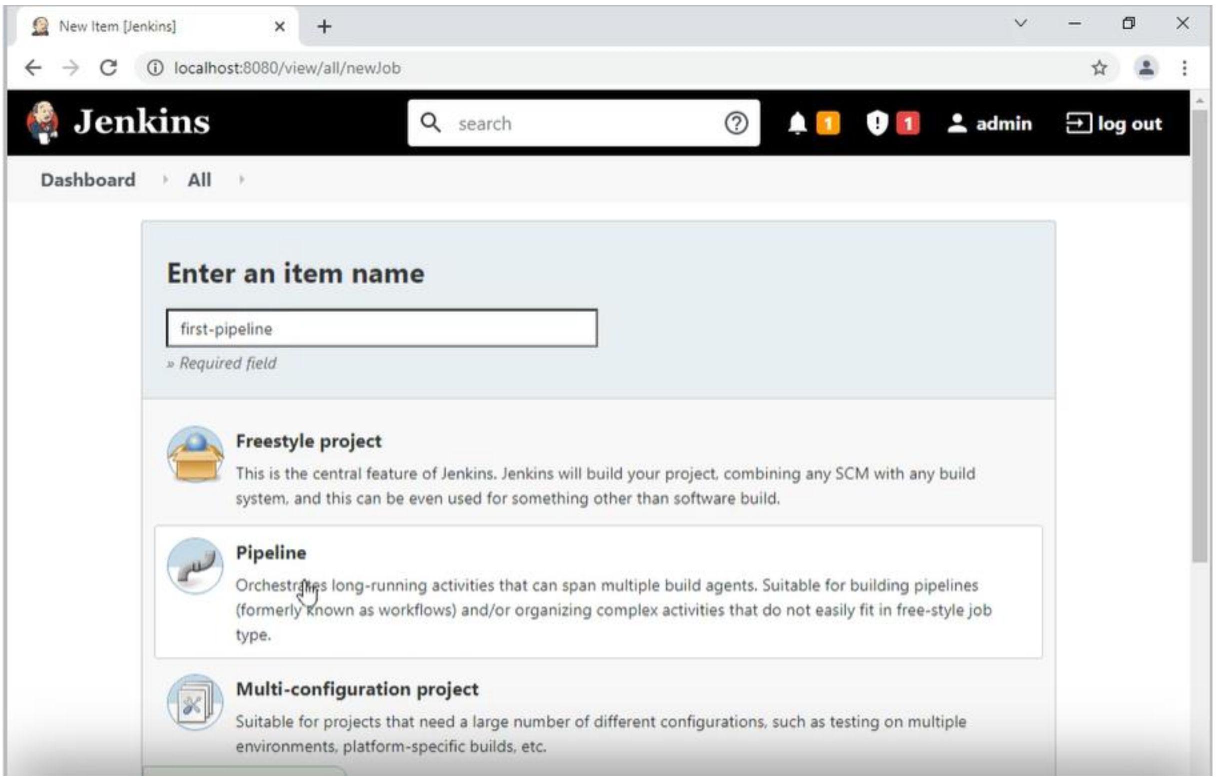
Set up a distributed build

Set up an agent

Configure a cloud

Learn more about distributed builds

So we tape the name of the item and click on Pipeline projects:


New Item [Jenkins]

localhost:8080/view/all/newJob

Jenkins

search

admin

log out

Dashboard › All

Enter an item name

first-pipeline

» Required field

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.

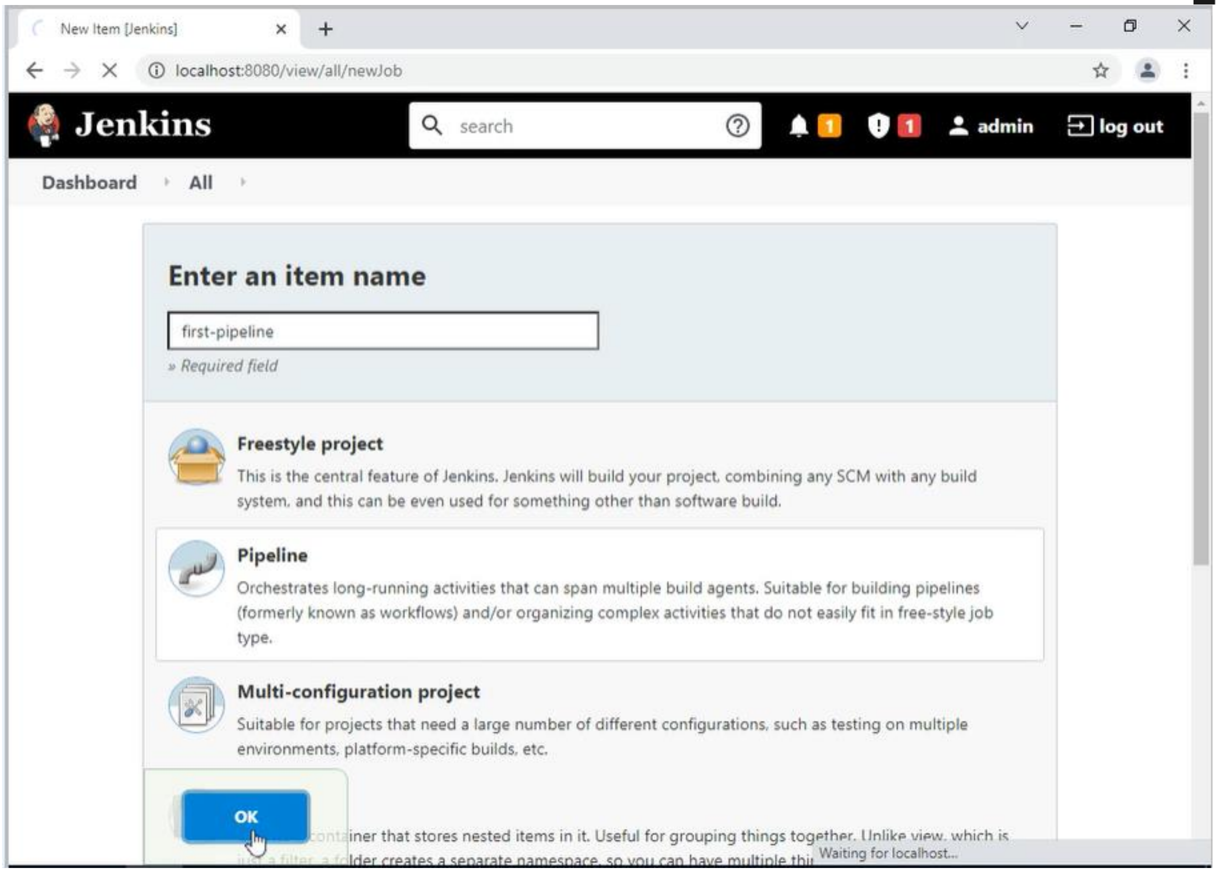
Pipeline

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.

Multi-configuration project

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

Now let’s clock on ‘OK’:


New Item [Jenkins]

localhost:8080/view/all/newJob

Dashboard › All ›

Enter an item name

first-pipeline

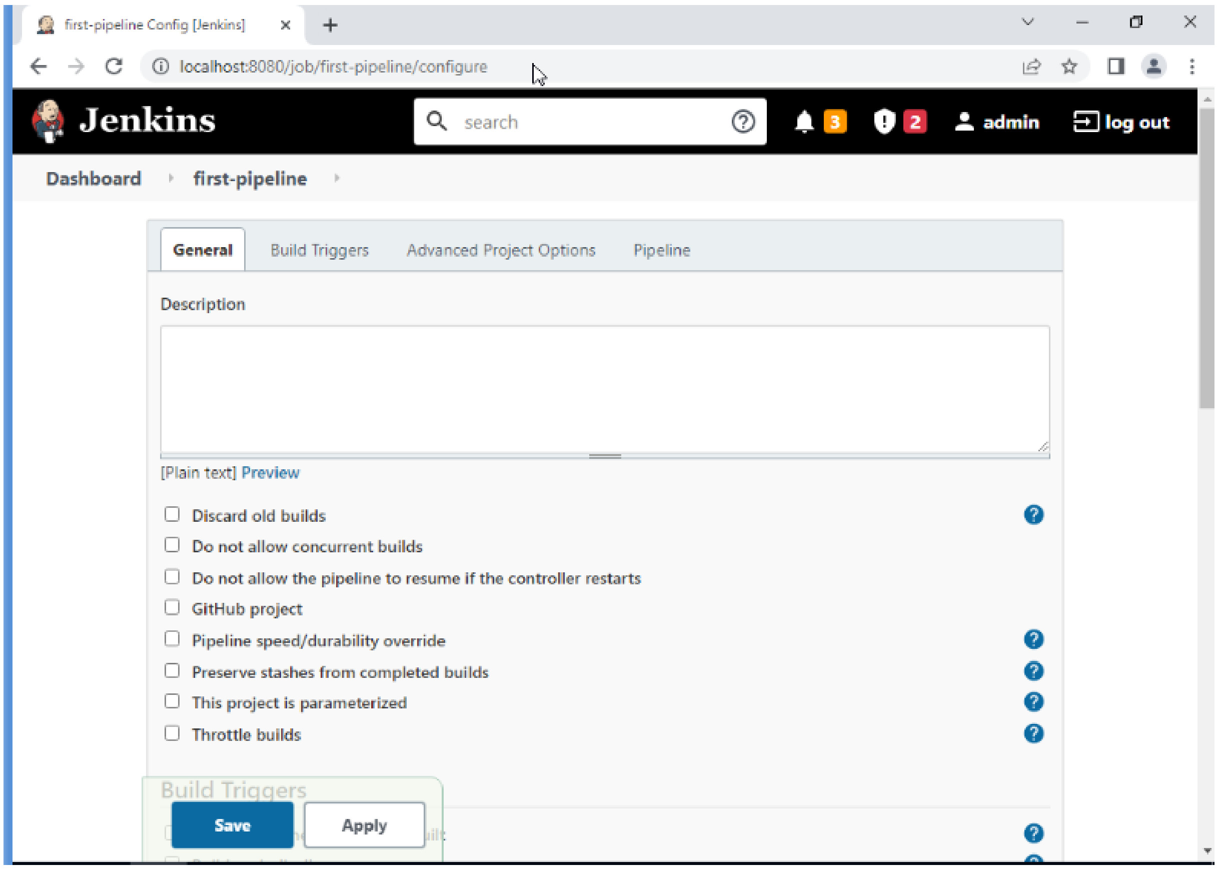
» Required field

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.

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.

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

it's gonna bring us to the configuration page and at this point we're not actually connecting this to some kind of source code.


first-pipeline Config [Jenkins]

localhost:8080/job/first-pipeline/configure

Dashboard › first-pipeline ›

General | Build Triggers | Advanced Project Options | Pipeline

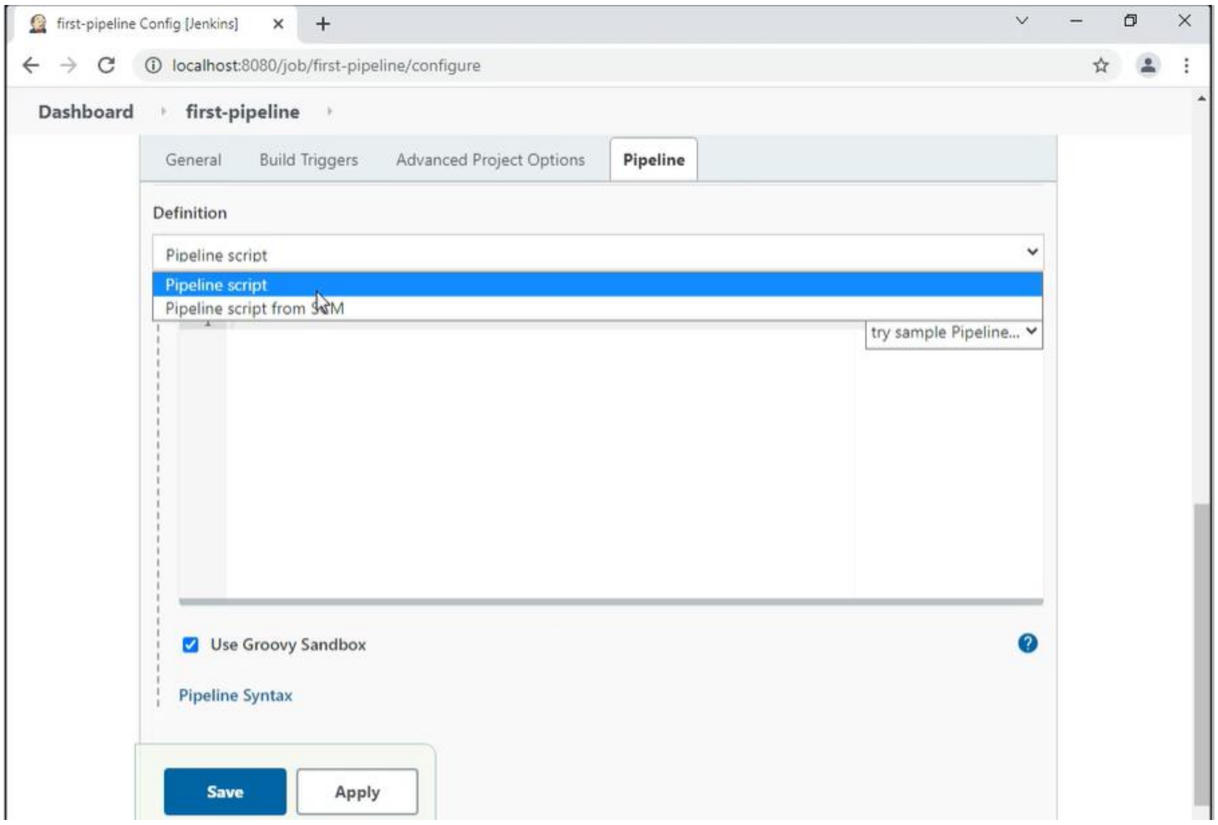
Description

[Plain text] Preview

- ☐ Discard old builds
- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the controller restarts
- ☐ GitHub project
- ☐ Pipeline speed/durability override
- ☐ Preserve stashes from completed builds
- ☐ This project is parameterized
- ☐ Throttle builds

Build Triggers

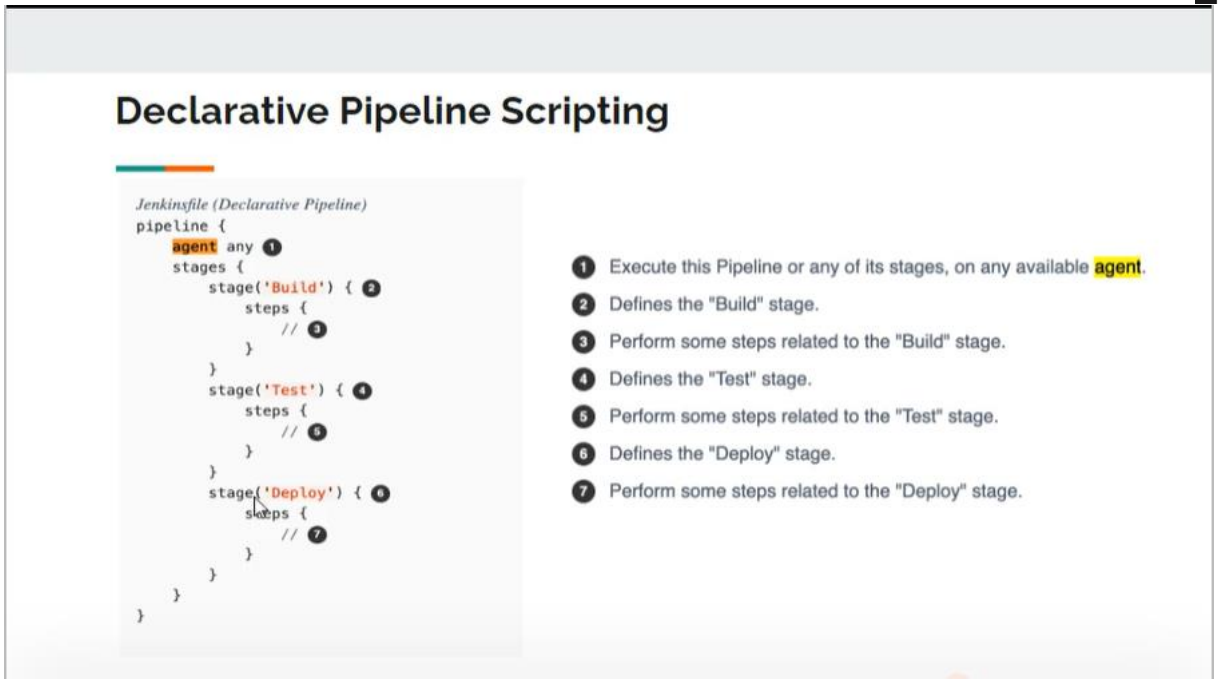
So if you click pipeline here, bring you down to this pipeline script section and then here we have the option here on this drop down of providing a pipeline script or pipeline script from source code management.



And this pipeline script here is the one that we're actually going to start writing here ourselves. And so to be able to actually write our script, we need to take an aside and that she start having a look at the formatting of a pipeline script.

And Jenkins and as we know, the pipeline is focused around continuous delivery And most importantly, with Jenkins, it's flexible when it's a user defined model of a continuous delivery pipeline.

So, we can have a typical example where the build test and deploy stage, we could have a staging stage, we could have a preparation stage, we could have a post stage which handles things after the actual pipeline has been completed.



Declarative Pipeline Scripting

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any ❶
    stages {
        stage('Build') { ❷
            steps {
                // ❸
            }
        }
        stage('Test') { ❹
            steps {
                // ❺
            }
        }
        stage('Deploy') { ❻
            steps {
                // ❼
            }
        }
    }
}
```

❶ Execute this Pipeline or any of its stages, on any available agent.
❷ Defines the "Build" stage.
❸ Perform some steps related to the "Build" stage.
❹ Defines the "Test" stage.
❺ Perform some steps related to the "Test" stage.
❻ Defines the "Deploy" stage.
❼ Perform some steps related to the "Deploy" stage.

So, if we have a look over here at the script, pipeline scripts are actually written in groovy syntax

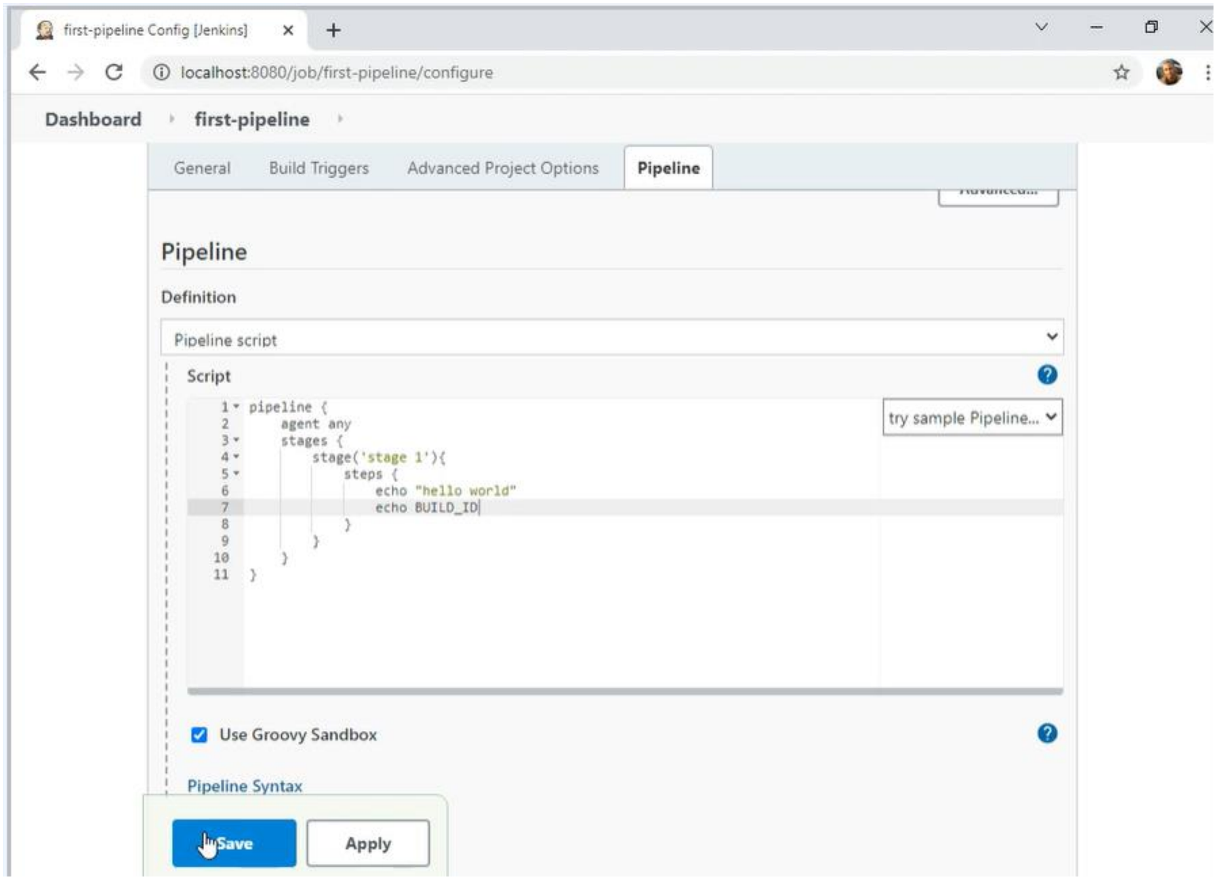
So no parentheses, no quotations is just literally pipeline open block and the first thing you'll see here is agent

And the agent in this case is more or less to do

The steps represent a single task and it tells Jenkins what to do at a particular point in time or a particular step in the process.

c. Task 3: Write our first Pipeline script and inject envars

I'm gonna get to start writing our first pipelines groups and inject into that some environment variables.



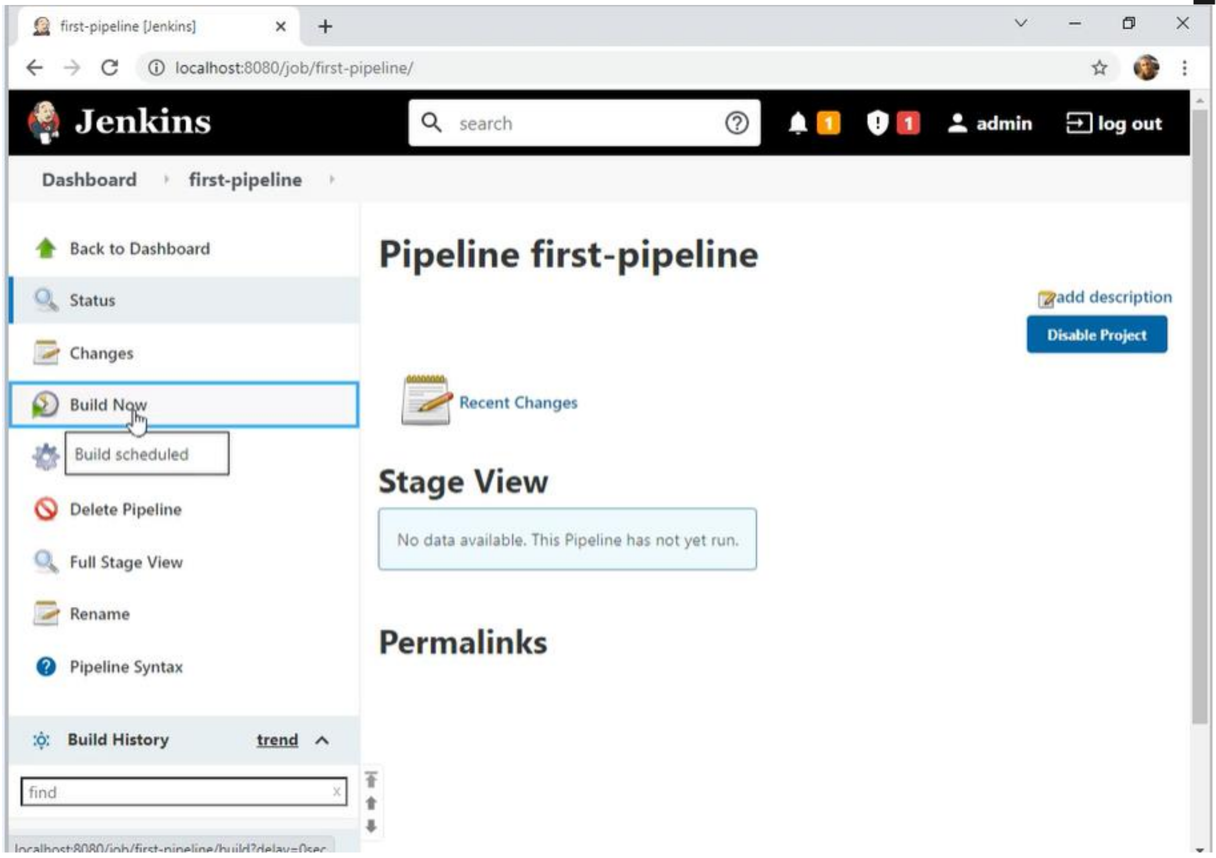
Pipeline

Definition: Pipeline script

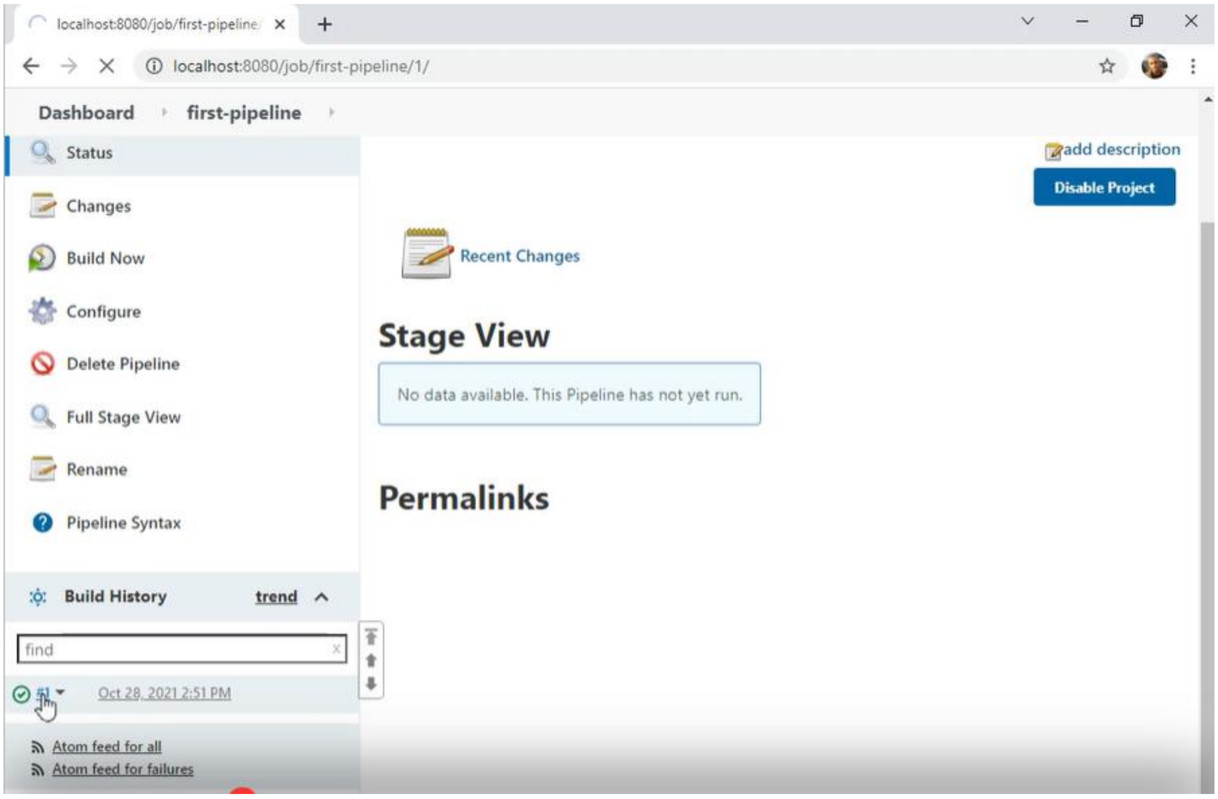
Script

```
pipeline {
    agent any
    stages {
        stage('stage 1'){
            steps {
                echo "hello world"
                echo BUILD_ID
            }
        }
    }
}
```

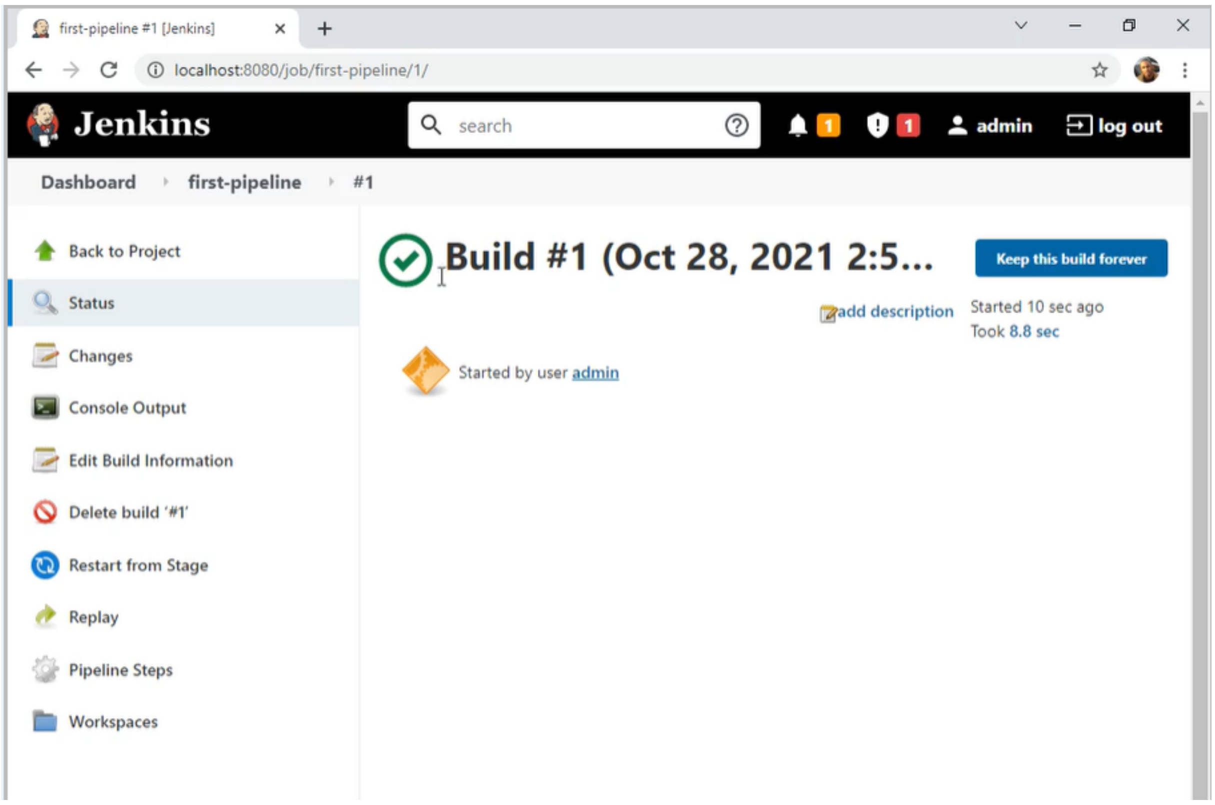
After saving the script, Let’s try to run our first pipeline on click on **‘Build Now’**



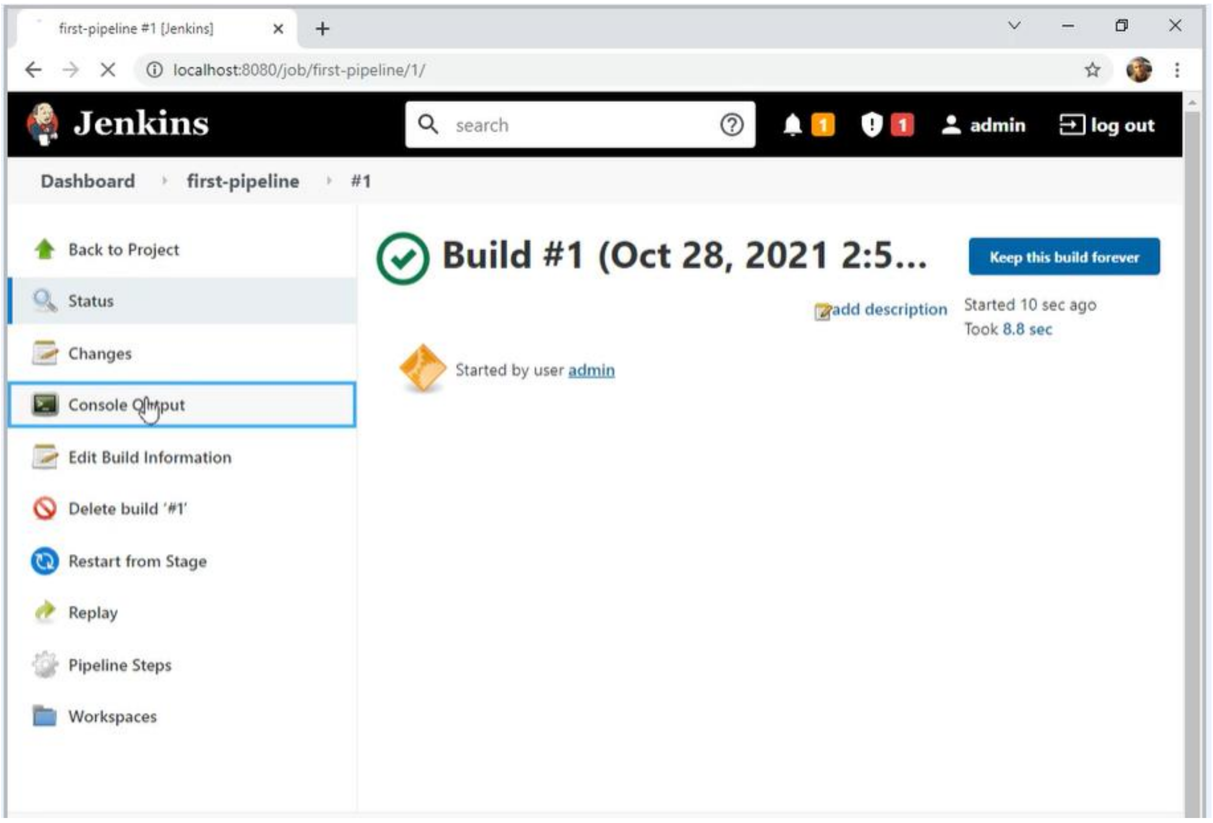
And if we scroll down, we gonna see inside to '**Build History'** the number of the build. So, when we click on:



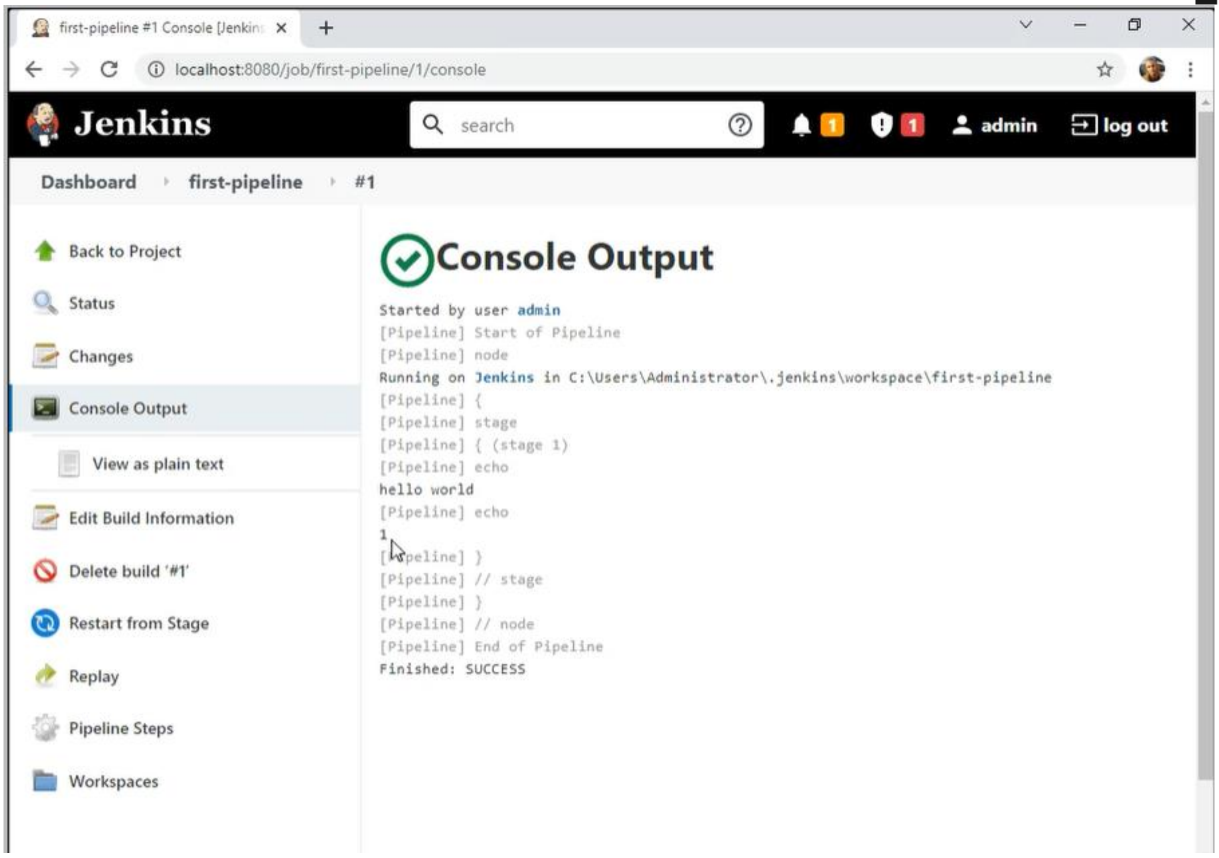
We go to this page below:



And we go to a **'Console Output'**

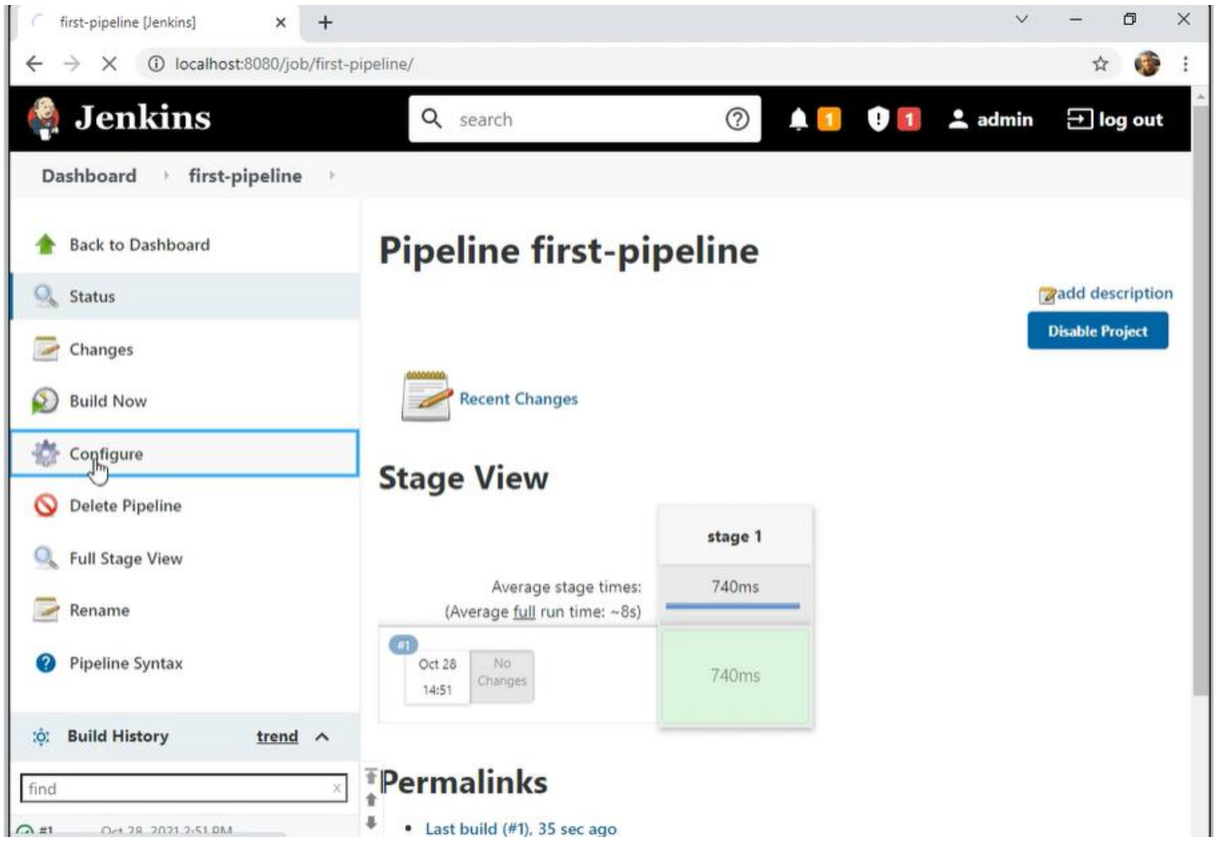


So, we can see here Hello world, and echo, and the ID of the build is number 1



```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\Users\Administrator\.jenkins\workspace\first-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (stage 1)
[Pipeline] echo
hello world
[Pipeline] echo
1
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

If we want to back of the project, we can click on 'Configure' for updating or changing something in the pipeline:



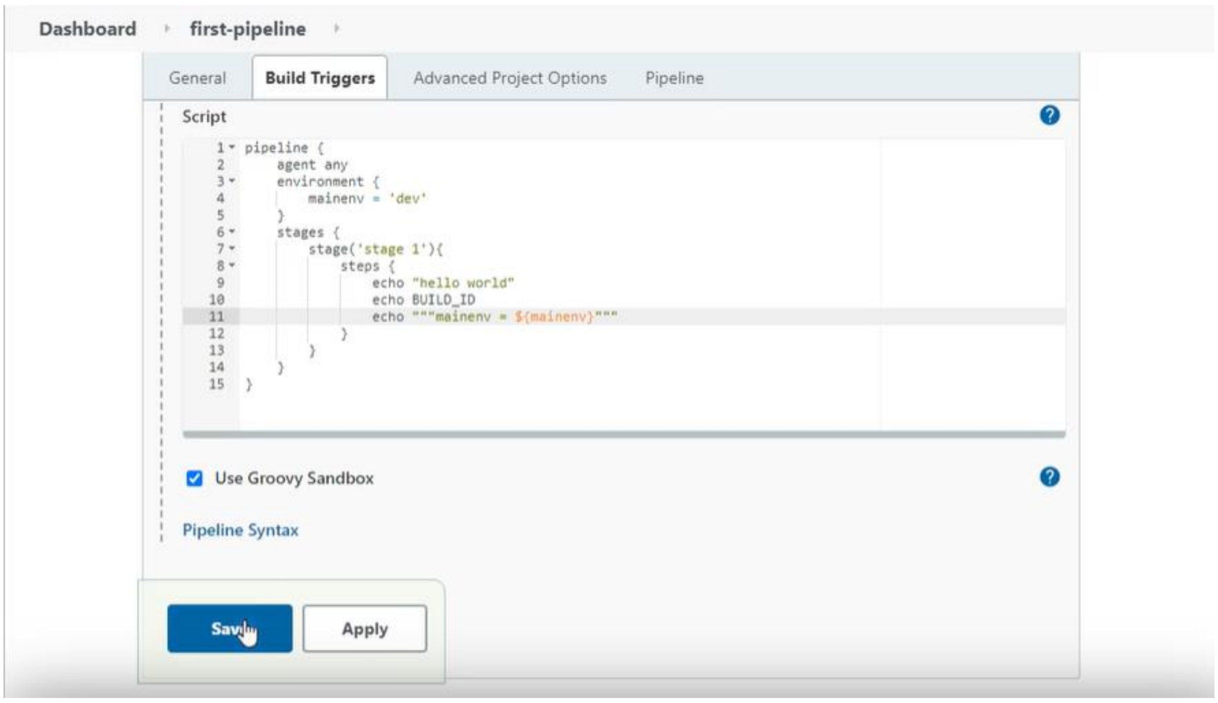
So we've already covered the concept of using the globally available environment variables. But we can also inject environment variables of our own into our code to be able to use throughout the pipeline process or life cycle.

And to do that, we use the Environment directive and the environment directive usually goes in the top level block and if it does in this case it will apply to every single step inside or every single stage inside of our pipeline.

So I can go ahead and simply define our variables. I'm gonna define a variable called **'mainenv'**

One thing which we can do here in groovy is we can provide three double quotation marks here too.

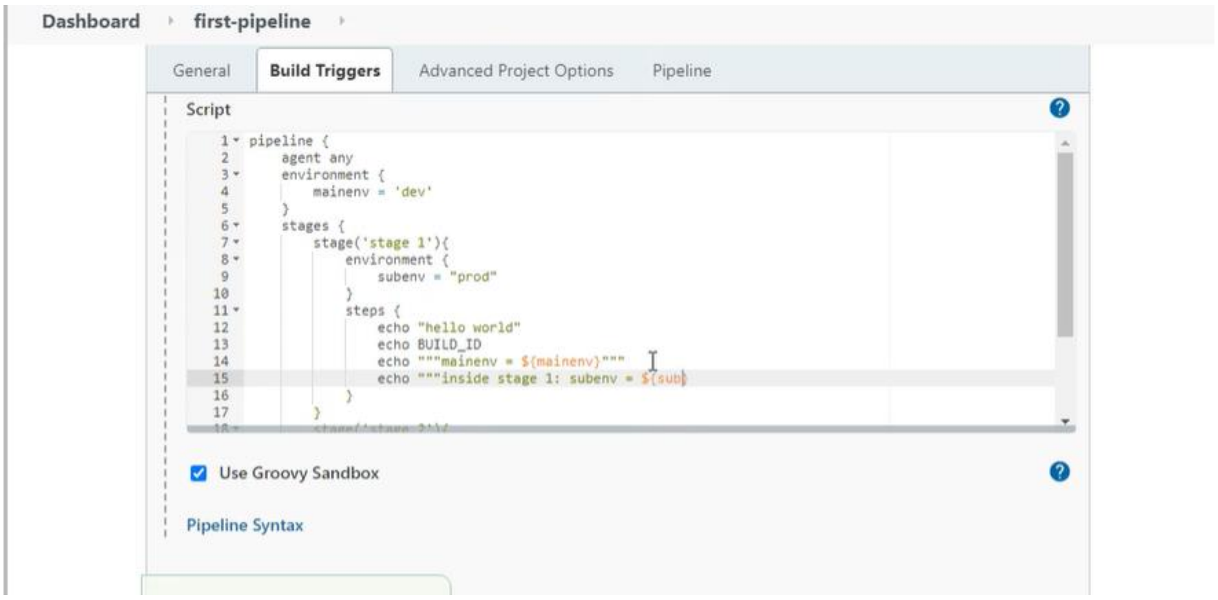
And now So let's save that here.



```
pipeline {
    agent any
    environment {
        mainenv = 'dev'
    }
    stages {
        stage('stage 1'){
            steps {
                echo "hello world"
                echo BUILD_ID
                echo """mainenv = ${mainenv}"""
            }
        }
    }
}
```

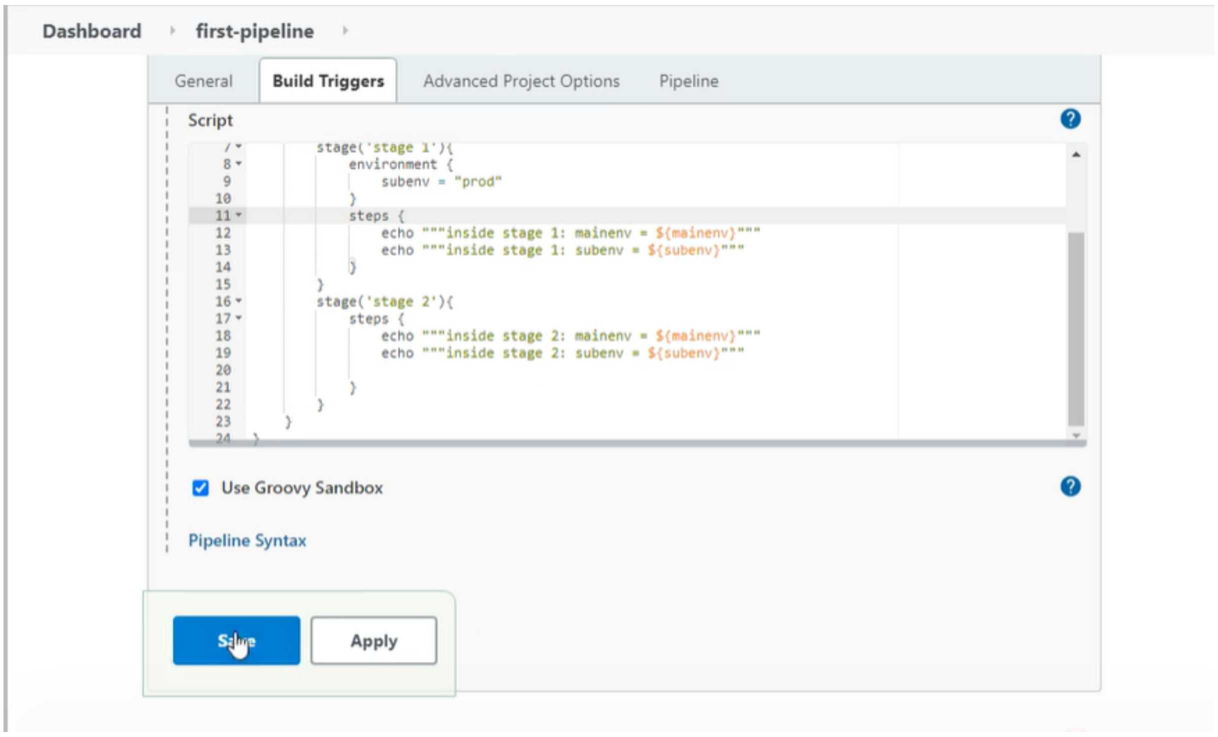
And when we build this now, wait for the bill to start here.

Let's add a variable on the stage 1 called 'subenv' :



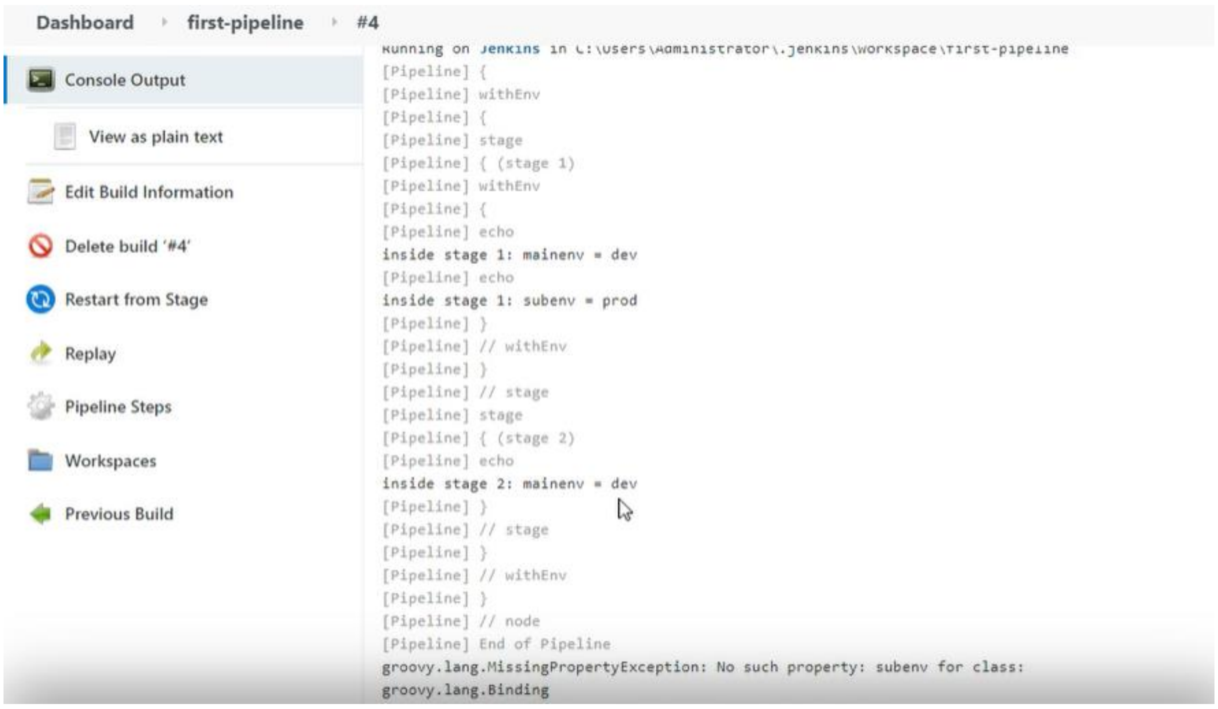
```
pipeline {
    agent any
    environment {
        mainenv = 'dev'
    }
    stages {
        stage('stage 1'){
            environment {
                subenv = "prod"
            }
            steps {
                echo "hello world"
                echo BUILD_ID
                echo """mainenv = ${mainenv}"""
                echo """inside stage 1: subenv = ${sub
            }
        }
        stage('stage 2'){
```

And let’s build too same steps in the both stages:



```
        stage('stage 1'){
            environment {
                subenv = "prod"
            }
            steps {
                echo """inside stage 1: mainenv = ${mainenv}"""
                echo """inside stage 1: subenv = ${subenv}"""
            }
        }
        stage('stage 2'){
            steps {
                echo """inside stage 2: mainenv = ${mainenv}"""
                echo """inside stage 2: subenv = ${subenv}"""
            }
        }
    }
}
```

And if we see the Console Output, we note that the steps of the stage 1 was executed , but for the steps of stage 2, only the first step was executed because in stage 2 we cannot access to the variable ‘subenv’ because it scoped to the block of stage 1

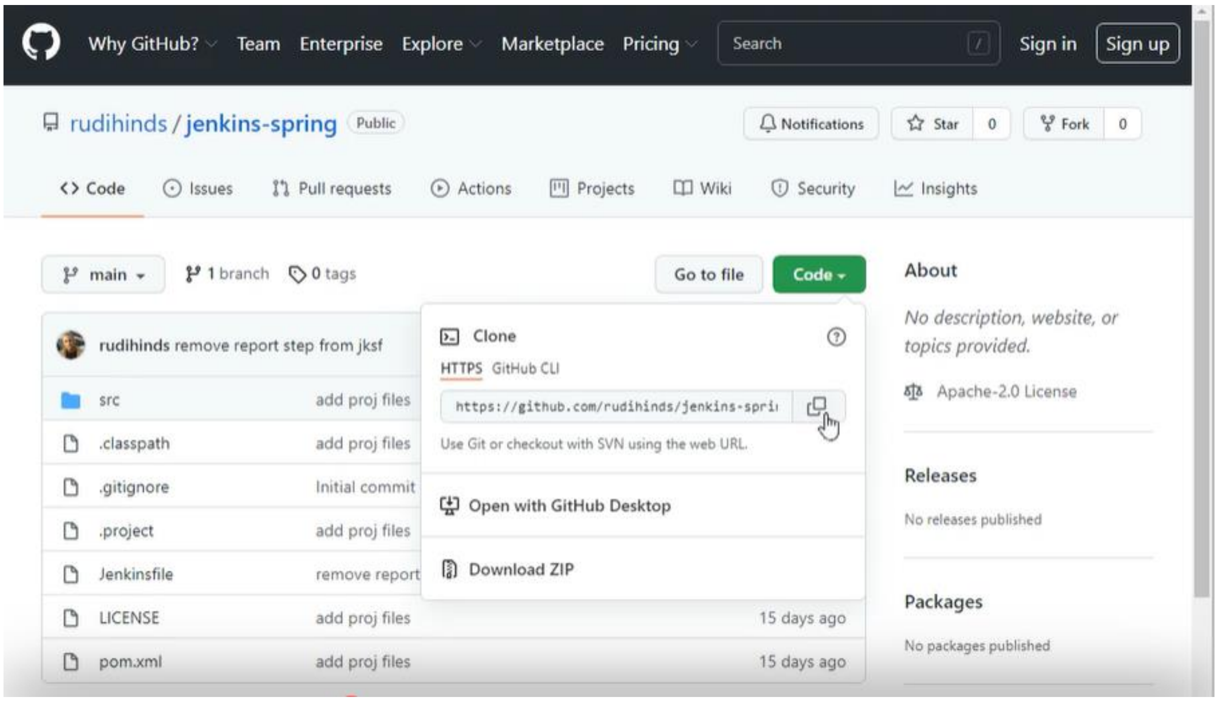


```
Running on Jenkins in C:\Users\Administrator\.jenkins\workspace\first-pipeline
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (stage 1)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] echo
inside stage 1: mainenv = dev
[Pipeline] echo
inside stage 1: subenv = prod
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (stage 2)
[Pipeline] echo
inside stage 2: mainenv = dev
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
groovy.lang.MissingPropertyException: No such property: subenv for class:
groovy.lang.Binding
```

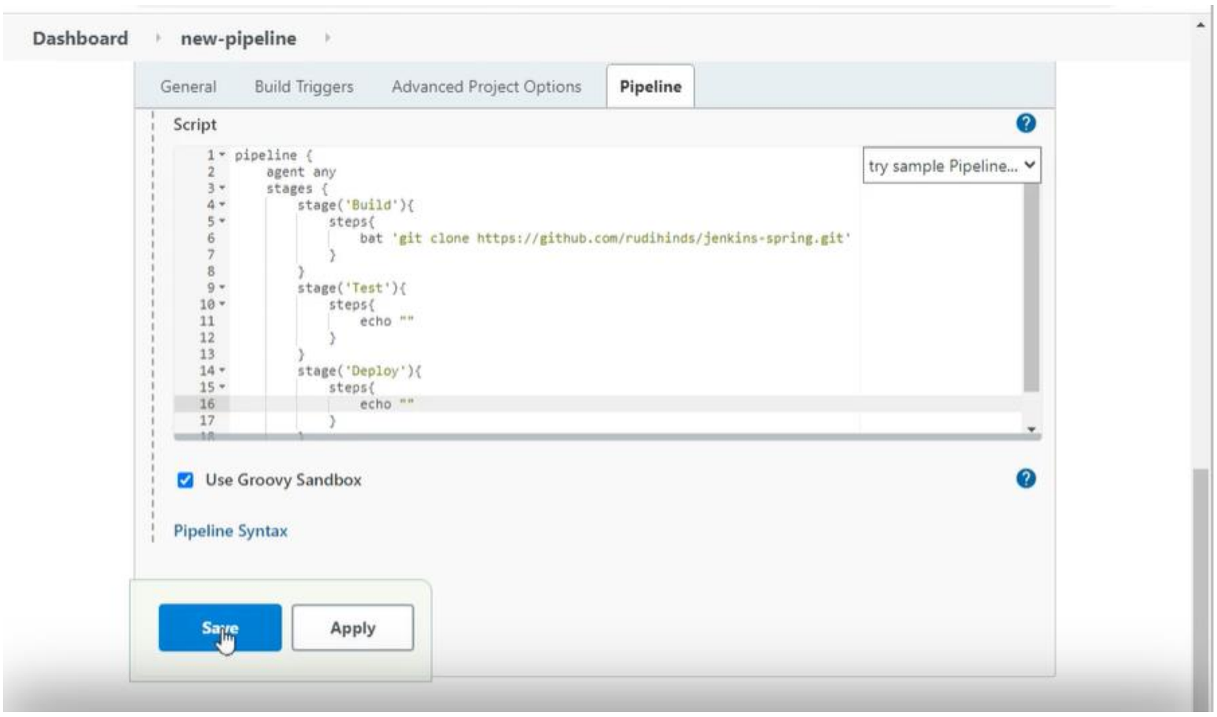
d. Task 4: Pipeline Script using real GitHub repo and build steps

It gives us a pipeline here with a very simple Hello World example.

So, we gonna copy the URL of a project in GitHub:

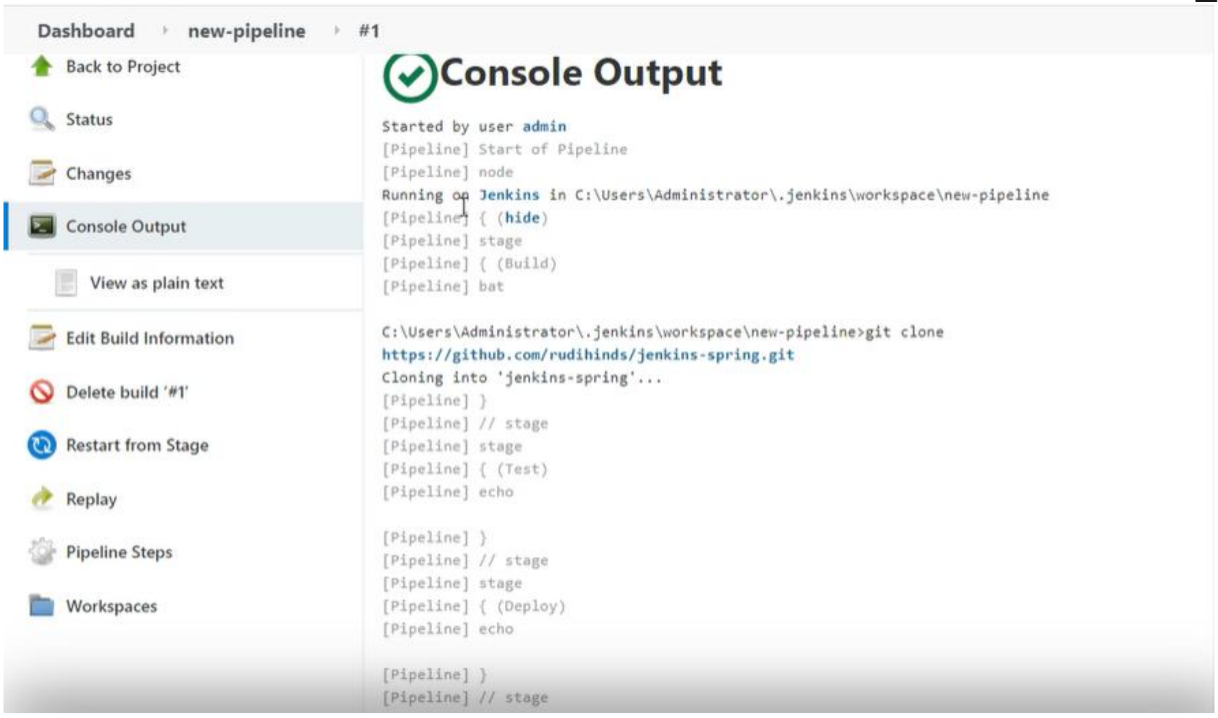


and clone it in pipeline script:



```
pipeline {
    agent any
    stages {
        stage('Build'){
            steps{
                bat 'git clone https://github.com/rudihinds/jenkins-spring.git'
            }
        }
        stage('Test'){
            steps{
                echo ""
            }
        }
        stage('Deploy'){
            steps{
                echo ""
            }
```

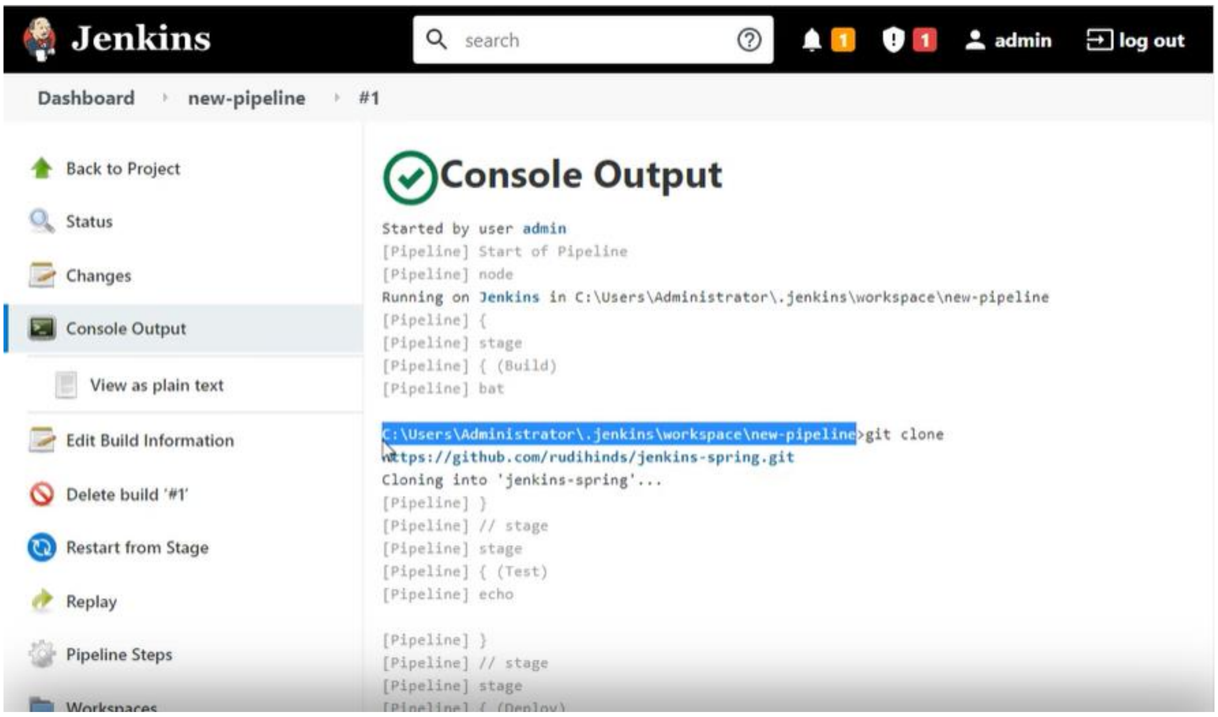
So, that the process was successfully executed



Console Output

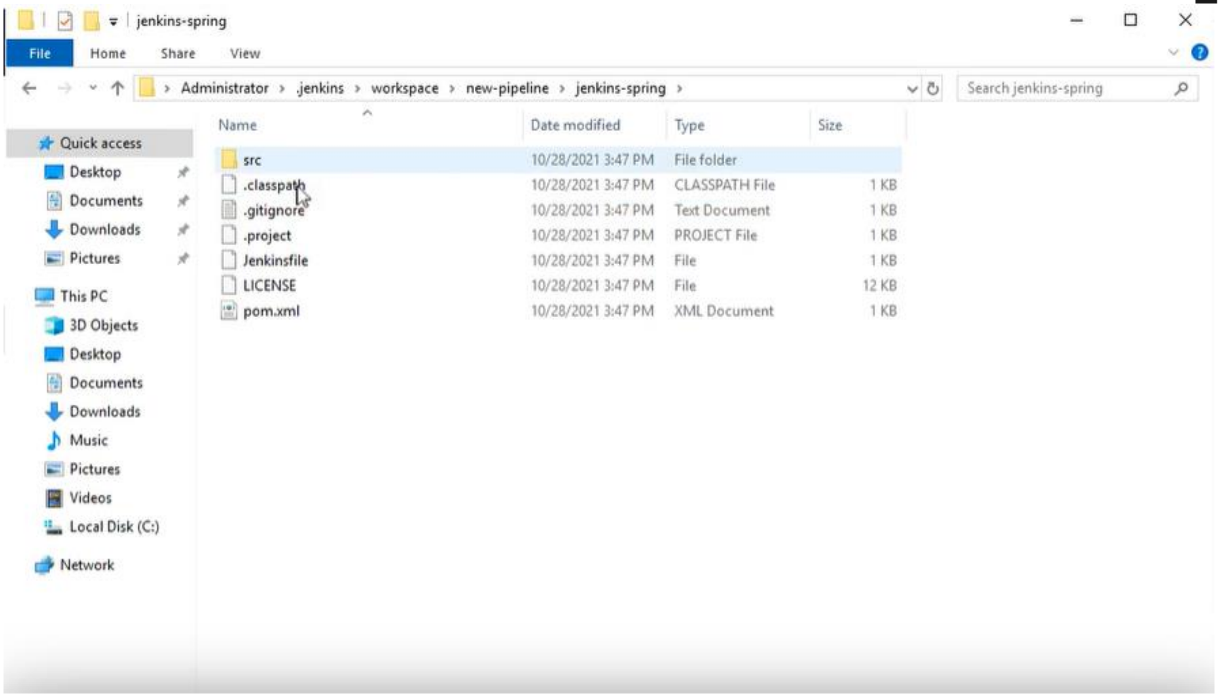
```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\Users\Administrator\.jenkins\workspace\new-pipeline
[Pipeline] { (hide)
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] bat

C:\Users\Administrator\.jenkins\workspace\new-pipeline>git clone
https://github.com/rudihinds/jenkins-spring.git
Cloning into 'jenkins-spring'...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo

[Pipeline] }
[Pipeline] // stage
```

If we see this file location in folders



Console Output

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\Users\Administrator\.jenkins\workspace\new-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] bat

C:\Users\Administrator\.jenkins\workspace\new-pipeline>git clone
https://github.com/rudihinds/jenkins-spring.git
Cloning into 'jenkins-spring'...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
```



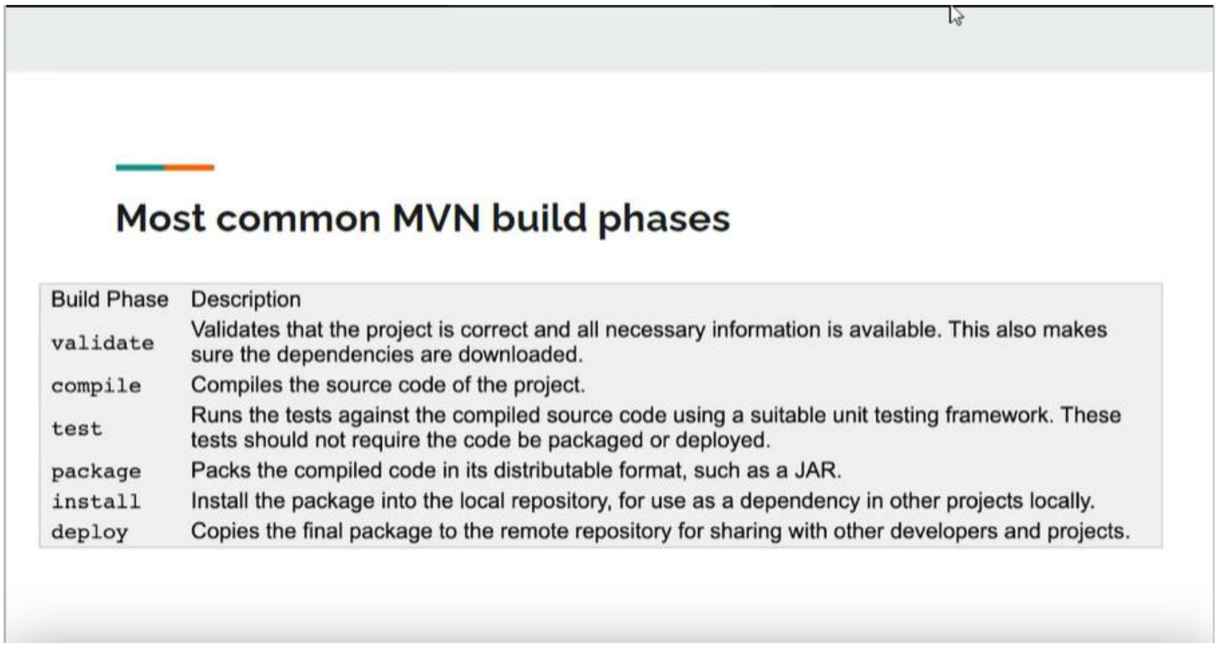
Name	Date modified	Type	Size
src	10/28/2021 3:47 PM	File folder	
.classpath	10/28/2021 3:47 PM	CLASSPATH File	1 KB
.gitignore	10/28/2021 3:47 PM	Text Document	1 KB
.project	10/28/2021 3:47 PM	PROJECT File	1 KB
Jenkinsfile	10/28/2021 3:47 PM	File	1 KB
LICENSE	10/28/2021 3:47 PM	File	12 KB
pom.xml	10/28/2021 3:47 PM	XML Document	1 KB

So, we see that because it's been cloned successfully on our local system the formats of files look similar to the format inside of GitHub.

In this second step, we're gonna use the popular built or Maven to run. And what I want to do with Maven is to provide the command Maven Clean which actually execute Maven and then cleans the target directory of all the old files that were generated from the previous build before it goes ahead and run that particular in particular stage itself.

So, we certainly must to have a look from the most common Maven build phases:

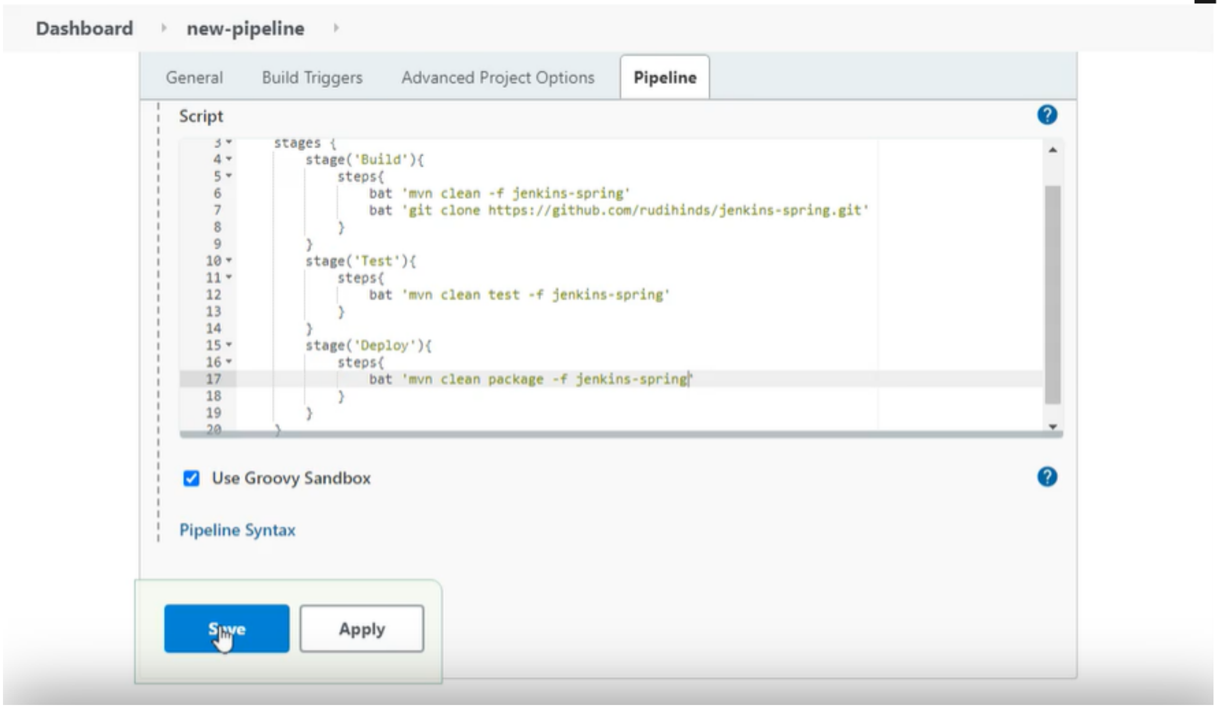
We provide the maven clean command along with **validate** compiled test packages to deploy



Most common MVN build phases

Build Phase	Description
`validate`	Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded.
`compile`	Compiles the source code of the project.
`test`	Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
`package`	Packs the compiled code in its distributable format, such as a JAR.
`install`	Install the package into the local repository, for use as a dependency in other projects locally.
`deploy`	Copies the final package to the remote repository for sharing with other developers and projects.

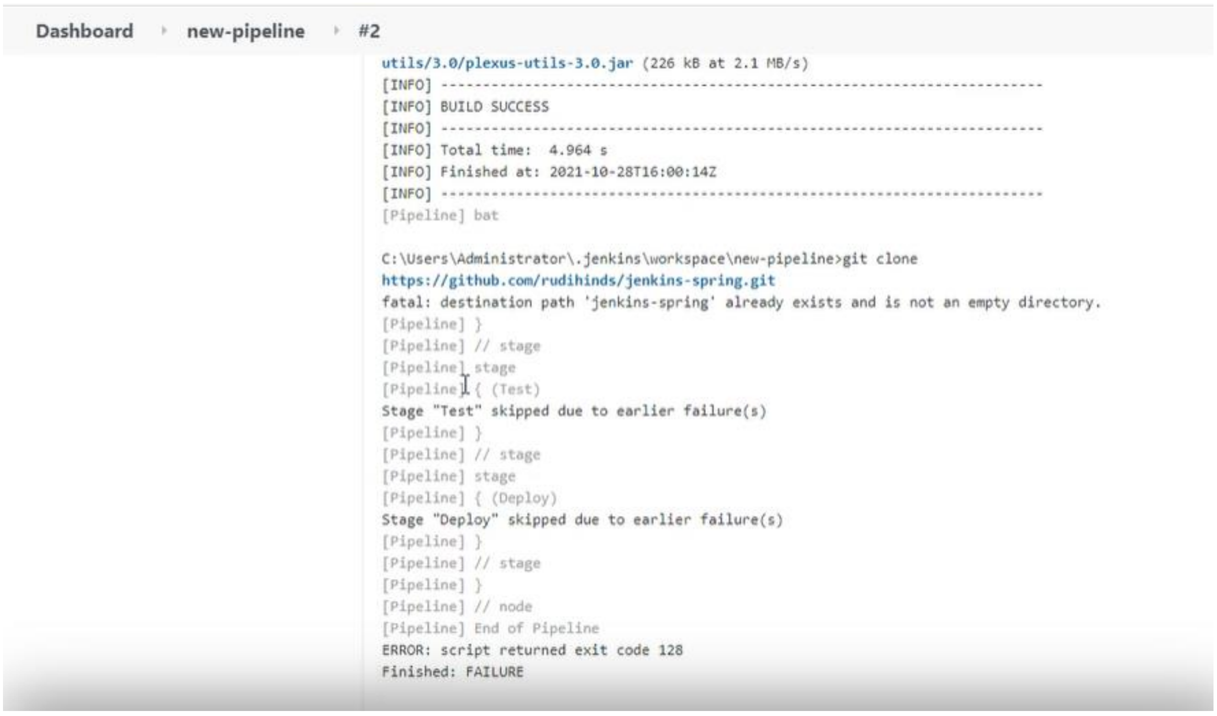
So, I gonna use it in this step:



```
    stages {
        stage('Build'){
            steps{
                bat 'mvn clean -f jenkins-spring'
                bat 'git clone https://github.com/rudihinds/jenkins-spring.git'
            }
        }
        stage('Test'){
            steps{
                bat 'mvn clean test -f jenkins-spring'
            }
        }
        stage('Deploy'){
            steps{
                bat 'mvn clean package -f jenkins-spring'
            }
        }
    }
```

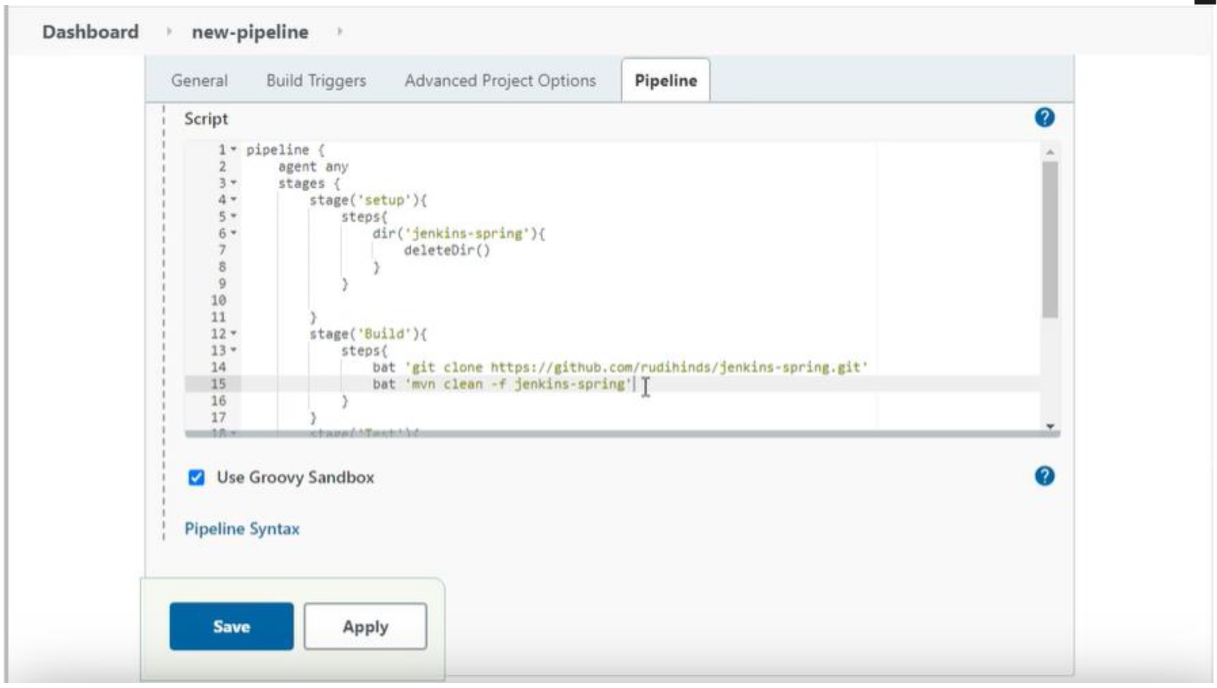
[✓ Use Groovy Sandbox]

And this is the result in the Console Output:



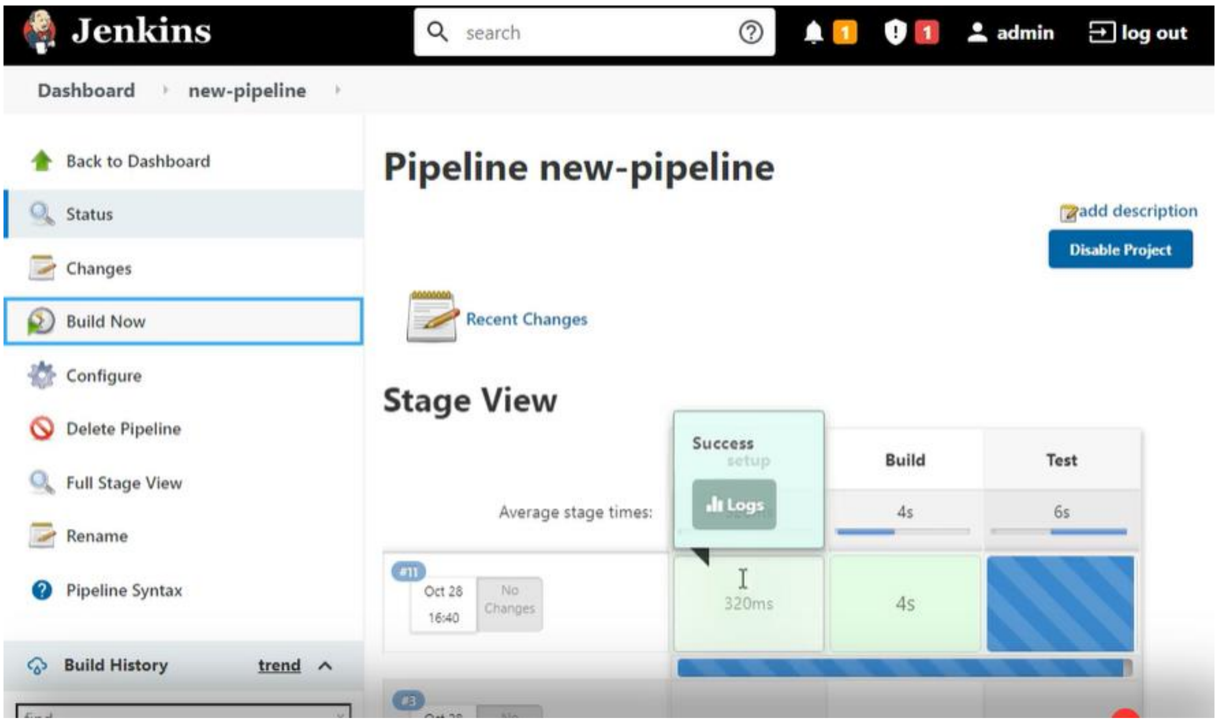
```
utils/3.0/plexus-utils-3.0.jar (226 kB at 2.1 MB/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.964 s
[INFO] Finished at: 2021-10-28T16:00:14Z
[INFO] ------------------------------------------------------------------------
[Pipeline] bat

C:\Users\Administrator\.jenkins\workspace\new-pipeline>git clone
https://github.com/rudihinds/jenkins-spring.git
fatal: destination path 'jenkins-spring' already exists and is not an empty directory.
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
Stage "Test" skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
Stage "Deploy" skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 128
Finished: FAILURE
```

The stage ‘test’ and ‘deploy’ skipped due to



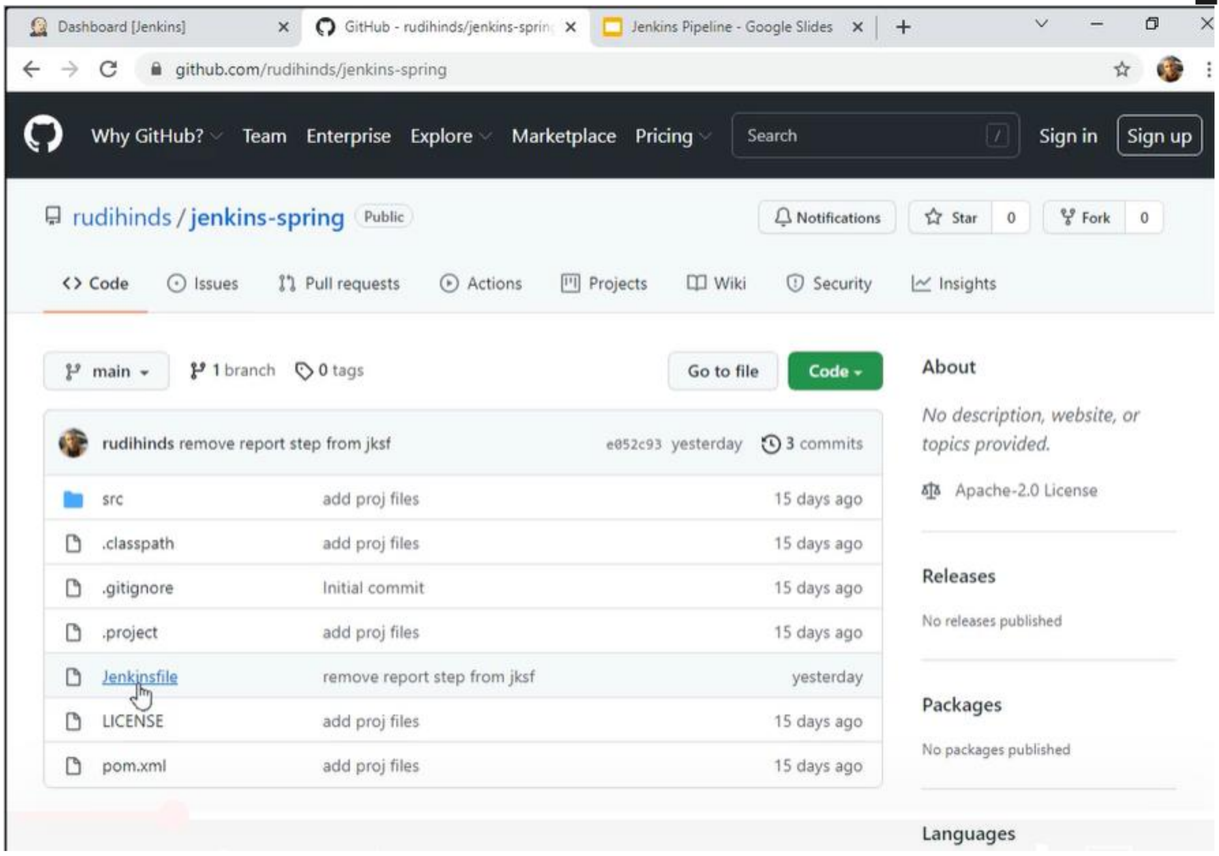
```
pipeline {
    agent any
    stages {
        stage('setup'){
            steps{
                dir('jenkins-spring'){
                    deleteDir()
                }
            }

        }
        stage('Build'){
            steps{
                bat 'git clone https://github.com/rudihinds/jenkins-spring.git'
                bat 'mvn clean -f jenkins-spring'
            }
        }
```

So we can see in the status that the stage setup was added:



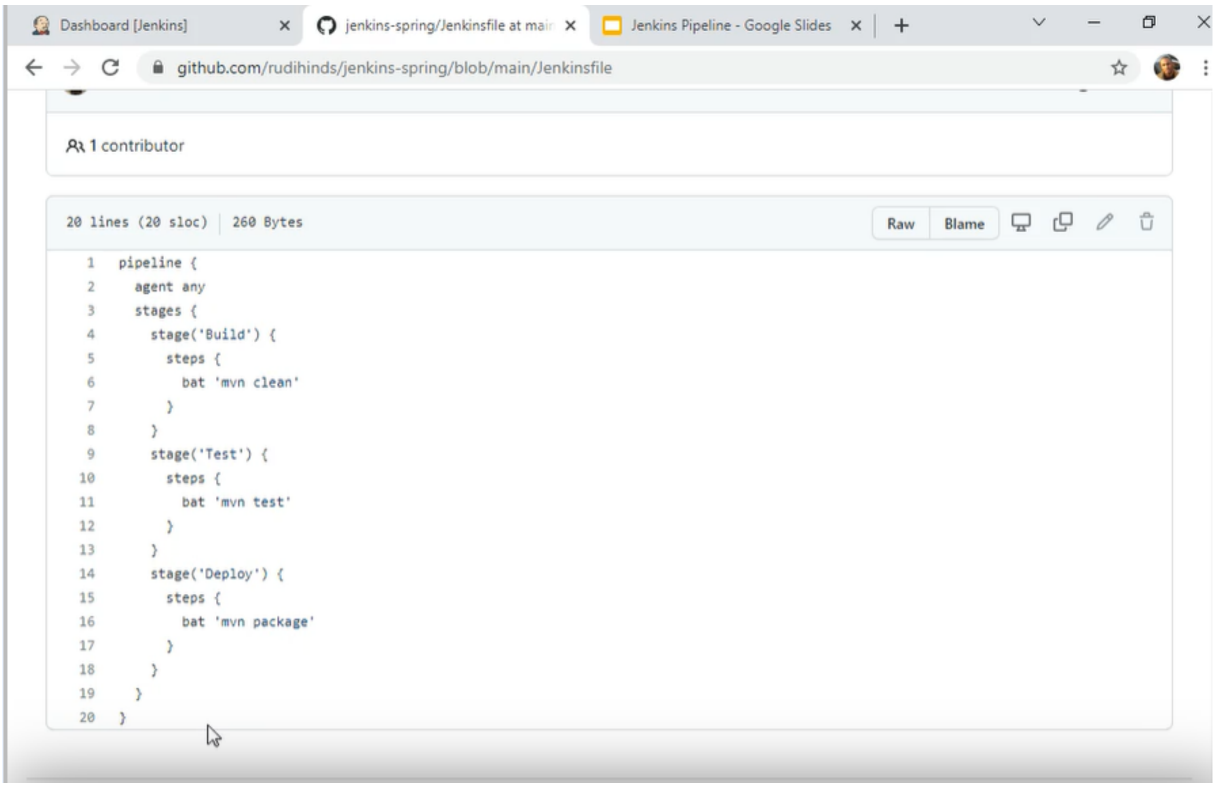
e. Task 5: Connect pipeline to SCM with Jenkins file

In this task, we need to have a Jenkins file uploaded in a source code management like Git


Dashboard [Jenkins]
GitHub - rudihinds/jenkins-sprin
Jenkins Pipeline - Google Slides
github.com/rudihinds/jenkins-spring
Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up
rudihinds / jenkins-spring Public
Notifications Star 0 Fork 0
Code Issues Pull requests Actions Projects Wiki Security Insights
main 1 branch 0 tags Go to file Code
About
No description, website, or topics provided.
Apache-2.0 License
Releases
No releases published
Packages
No packages published
Languages

rudihinds remove report step from jksf	e052c93 yesterday	3 commits
src	add proj files	15 days ago
.classpath	add proj files	15 days ago
.gitignore	Initial commit	15 days ago
.project	add proj files	15 days ago
Jenkinsfile	remove report step from jksf	yesterday
LICENSE	add proj files	15 days ago
pom.xml	add proj files	15 days ago

And, on opened this Jenkins file, we got:


Dashboard [Jenkins]
jenkins-spring/Jenkinsfile at main
Jenkins Pipeline - Google Slides
github.com/rudihinds/jenkins-spring/blob/main/Jenkinsfile
1 contributor
Raw
Blame

20 lines (20 sloc) | 260 Bytes

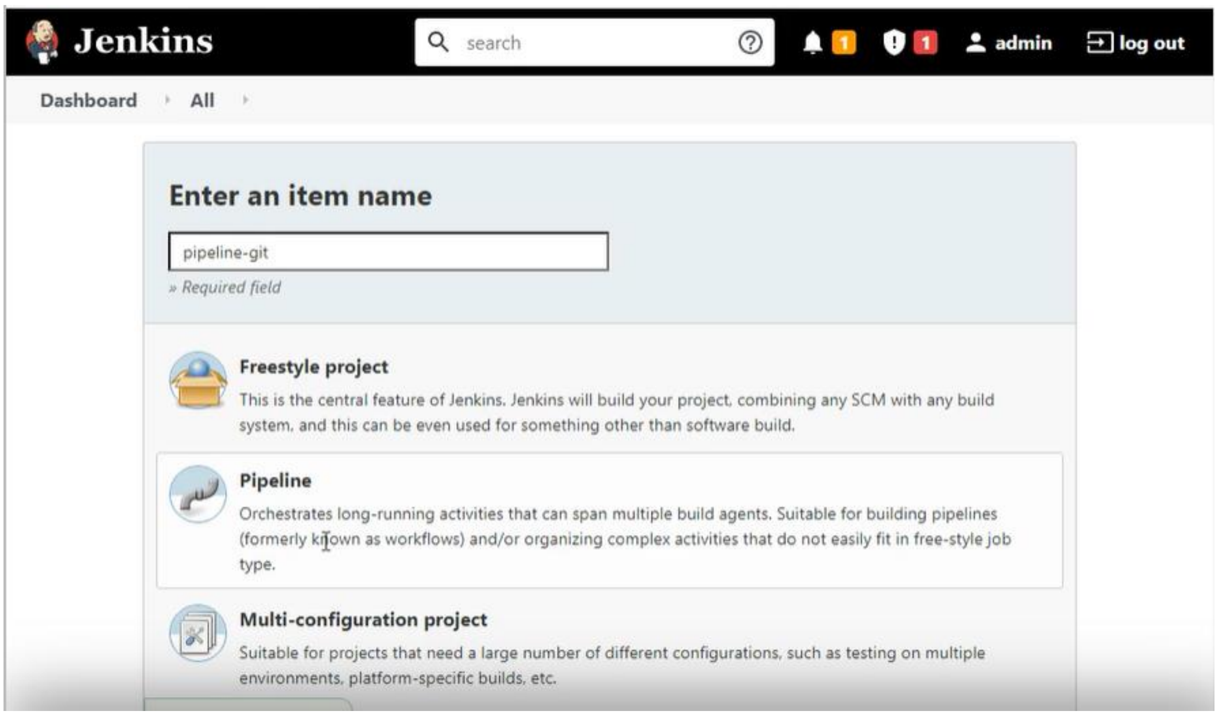
```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        bat 'mvn clean'
      }
    }
    stage('Test') {
      steps {
        bat 'mvn test'
      }
    }
    stage('Deploy') {
      steps {
        bat 'mvn package'
      }
    }
  }
}
```

Now, the part of the script where we had to clone the repo is not going to be used because getting this, we are gonna be telling Jenkins to get the repo directly from source code management and it's going to get the Jenkins file from there and it's gonna know what to do with that by itself.

So, we don't need to provide that as a separate step in and of itself.

So, we do simply have three separate steps where we provide a batch command and we are just doing a clean process here.

So, firstly we gonna create a new item:



Jenkins | search | admin | log out

Dashboard › All ›

Enter an item name

pipeline-git

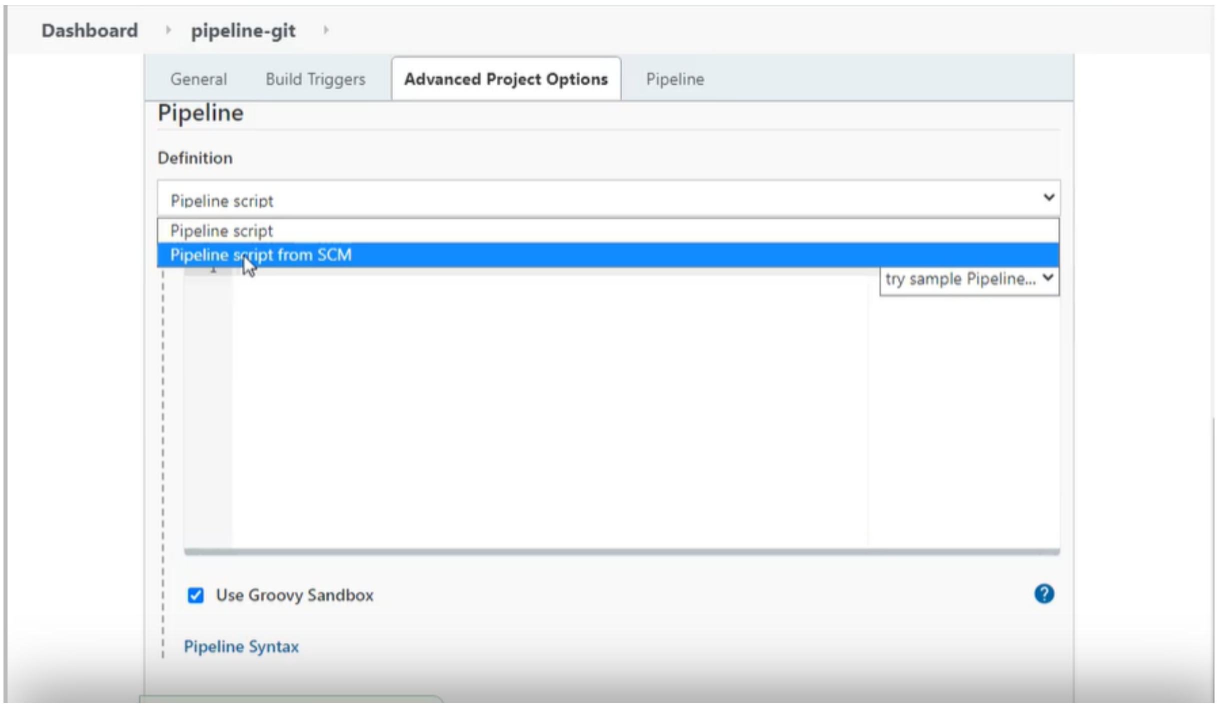
» Required field

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.

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.

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

And we gonna choose the second option the pipeline script:



Dashboard › pipeline-git ›

General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition

Pipeline script

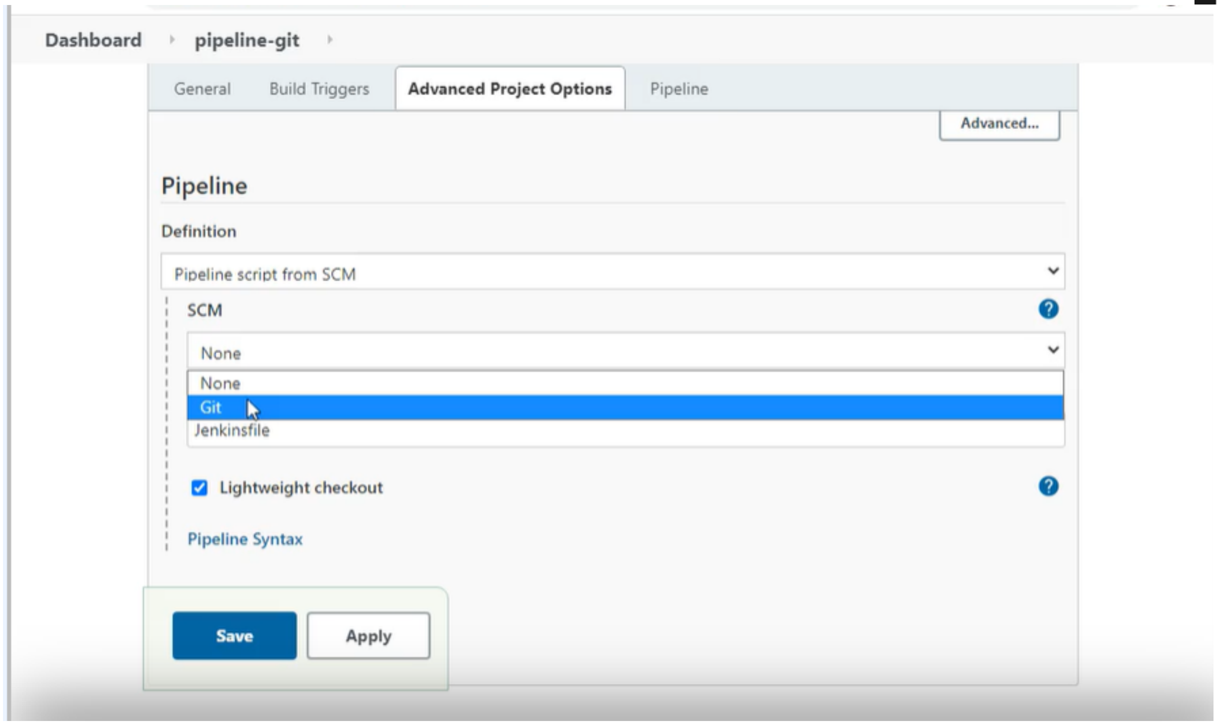
- Pipeline script
- Pipeline script from SCM

try sample Pipeline...

☑ Use Groovy Sandbox

Pipeline Syntax

And we select Git here:


Dashboard › pipeline-git ›

General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

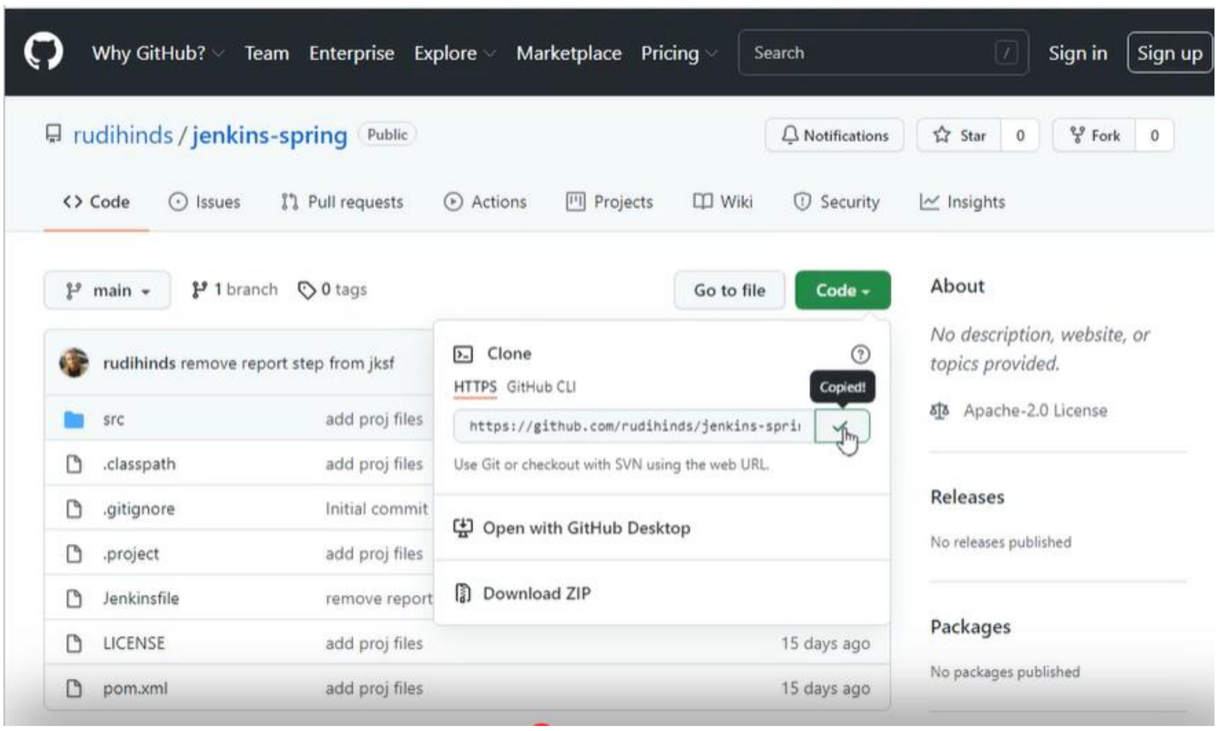
SCM: None

- None
- Git
- Jenkinsfile

☑ Lightweight checkout

Pipeline Syntax

And, let's copy the link from git:


rudihinds / jenkins-spring (Public)

main · 1 branch · 0 tags

Clone — HTTPS | GitHub CLI

```
https://github.com/rudihinds/jenkins-spri
```

Use Git or checkout with SVN using the web URL.

rudihinds remove report step from jksf		
src	add proj files	
.classpath	add proj files	
.gitignore	Initial commit	
.project	add proj files	
Jenkinsfile	remove report	
LICENSE	add proj files	15 days ago
pom.xml	add proj files	15 days ago

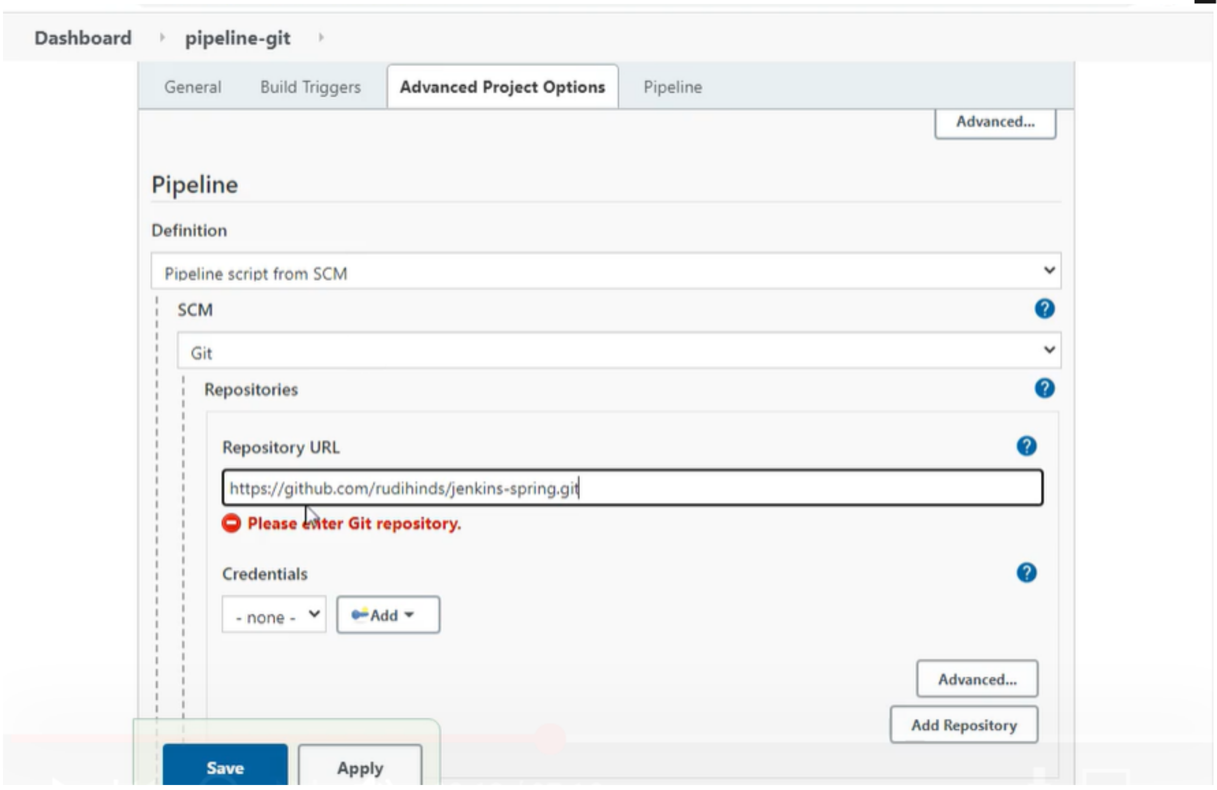
About: No description, website, or topics provided. Apache-2.0 License

Releases: No releases published

Packages: No packages published

Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up
Notifications Star 0 Fork 0
Code Issues Pull requests Actions Projects Wiki Security Insights
Go to file Code
Copied!
Open with GitHub Desktop
Download ZIP

And, we simply paste the URL:


Dashboard › pipeline-git ›

General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

SCM: Git

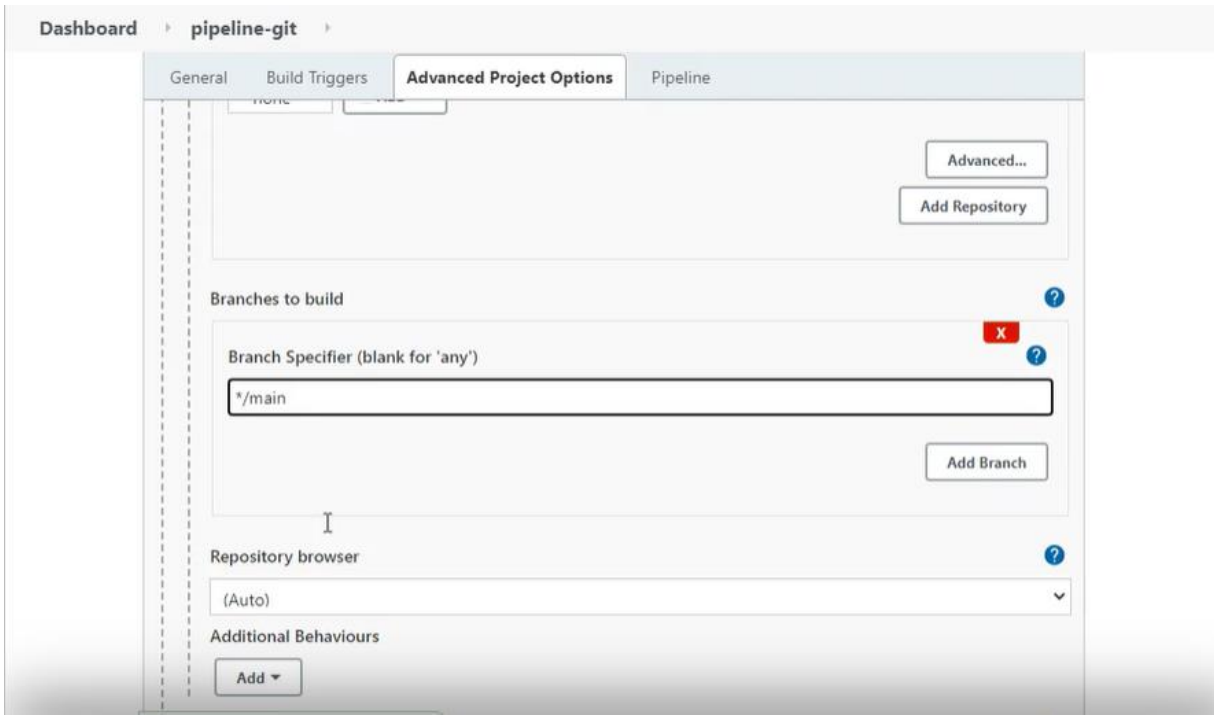
Repositories

Repository URL: https://github.com/rudihinds/jenkins-spring.git

Please enter Git repository.

Credentials: - none -

And, we change the name of the branch like we have in Git, here we have the branch called 'main':


Dashboard › pipeline-git ›

General | Build Triggers | Advanced Project Options | Pipeline

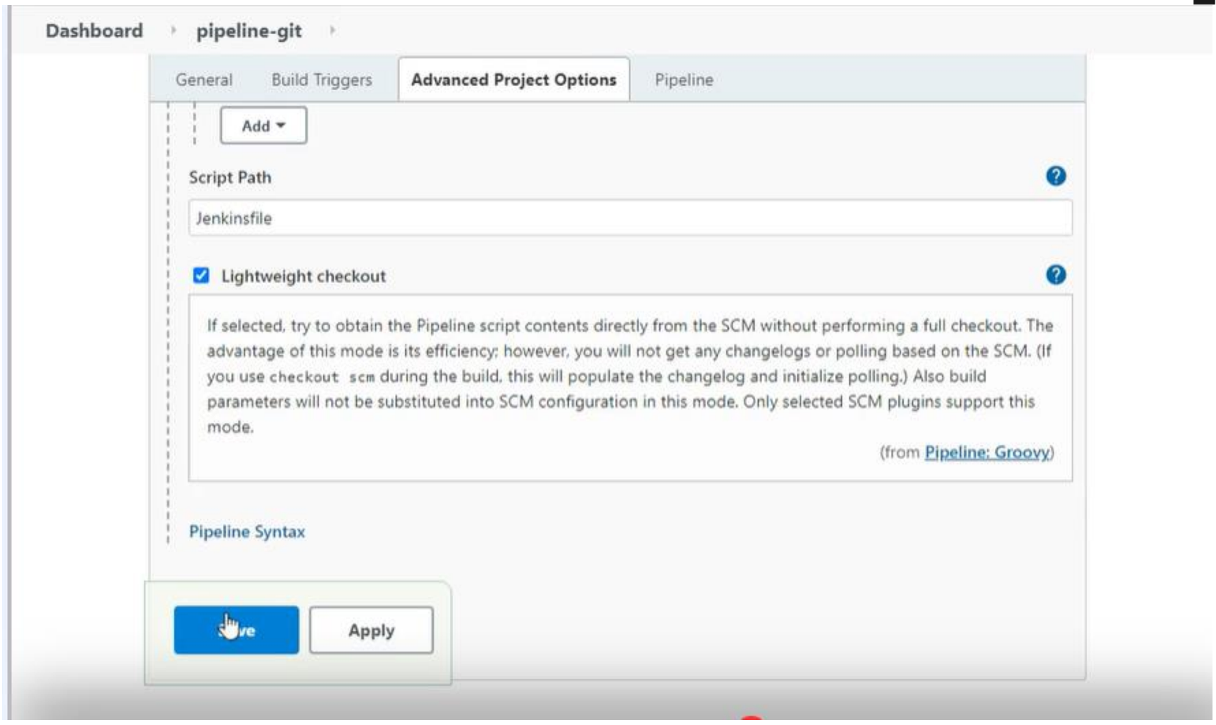
Branches to build

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

Repository browser: (Auto)

Additional Behaviours

We gonna simply click on 'save', because all the configuration is actually inside of the script.


Dashboard › pipeline-git ›

General | Build Triggers | **Advanced Project Options** | Pipeline

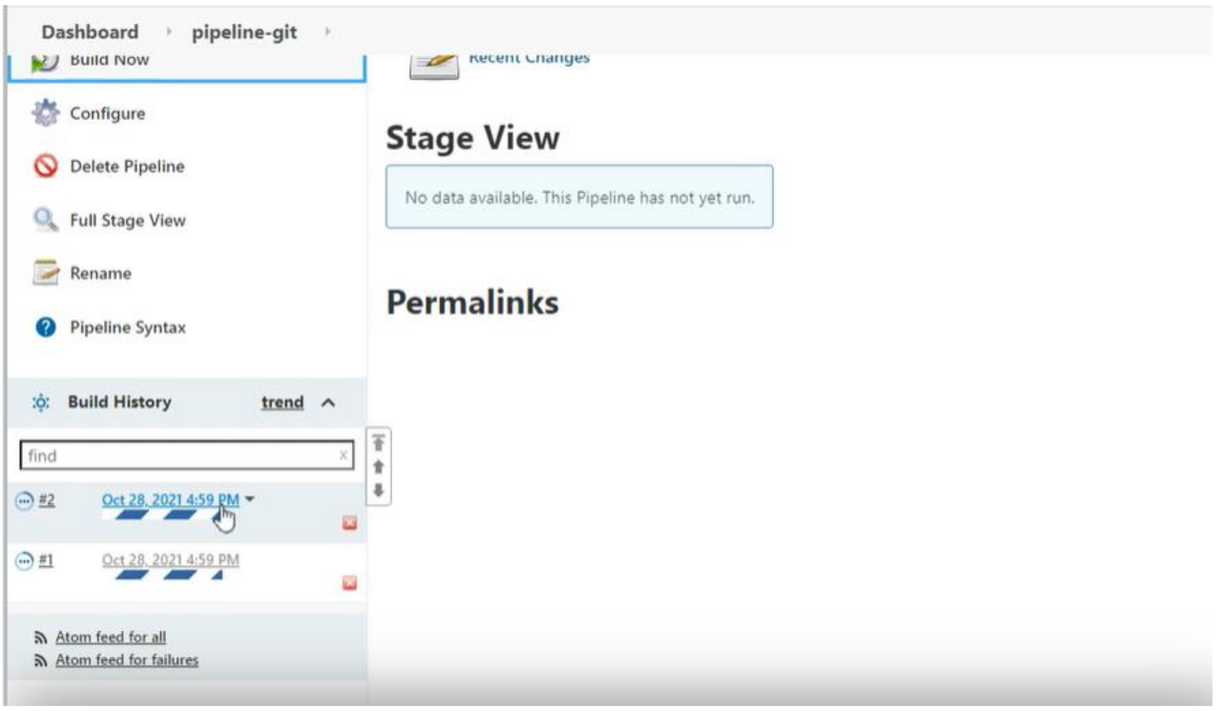
Script Path: Jenkinsfile

☑ Lightweight checkout

If selected, try to obtain the Pipeline script contents directly from the SCM without performing a full checkout. The advantage of this mode is its efficiency; however, you will not get any changelogs or polling based on the SCM. (If you use `checkout scm` during the build, this will populate the changelog and initialize polling.) Also build parameters will not be substituted into SCM configuration in this mode. Only selected SCM plugins support this mode.

(from Pipeline: Groovy)

And now, we gonna click on 'Build Now' for accomplishing the process:


Dashboard › pipeline-git ›

Stage View

No data available. This Pipeline has not yet run.

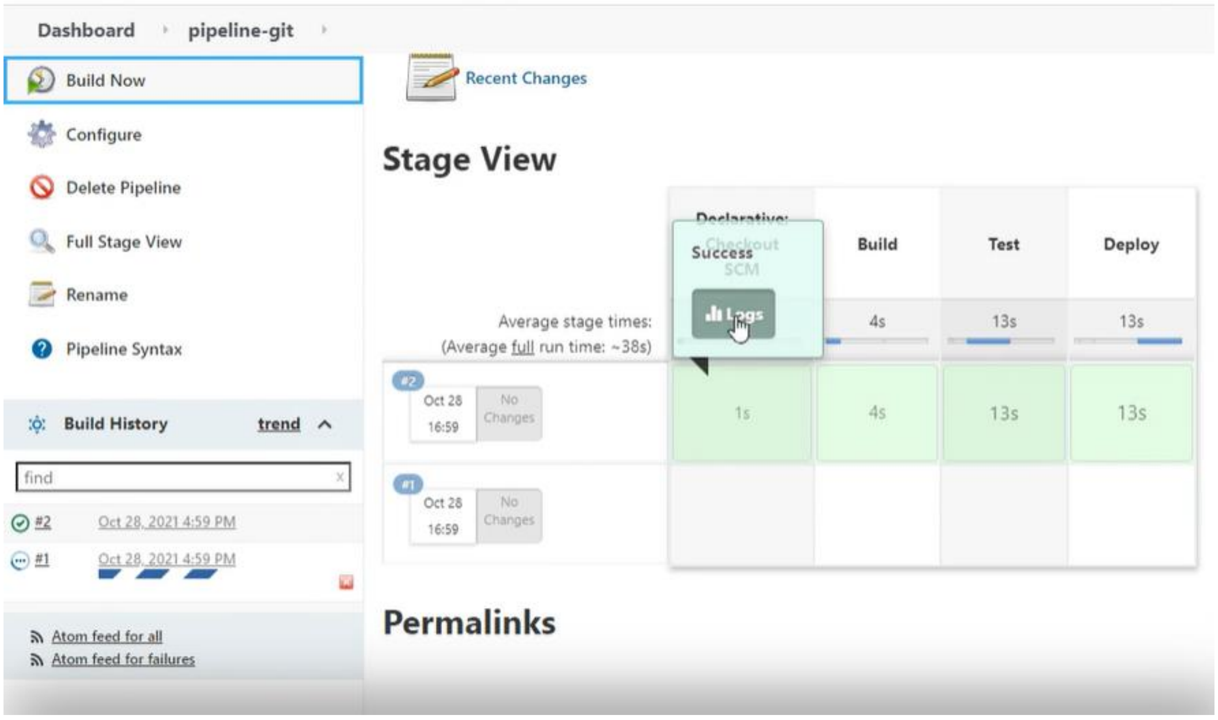
Permalinks

Build History

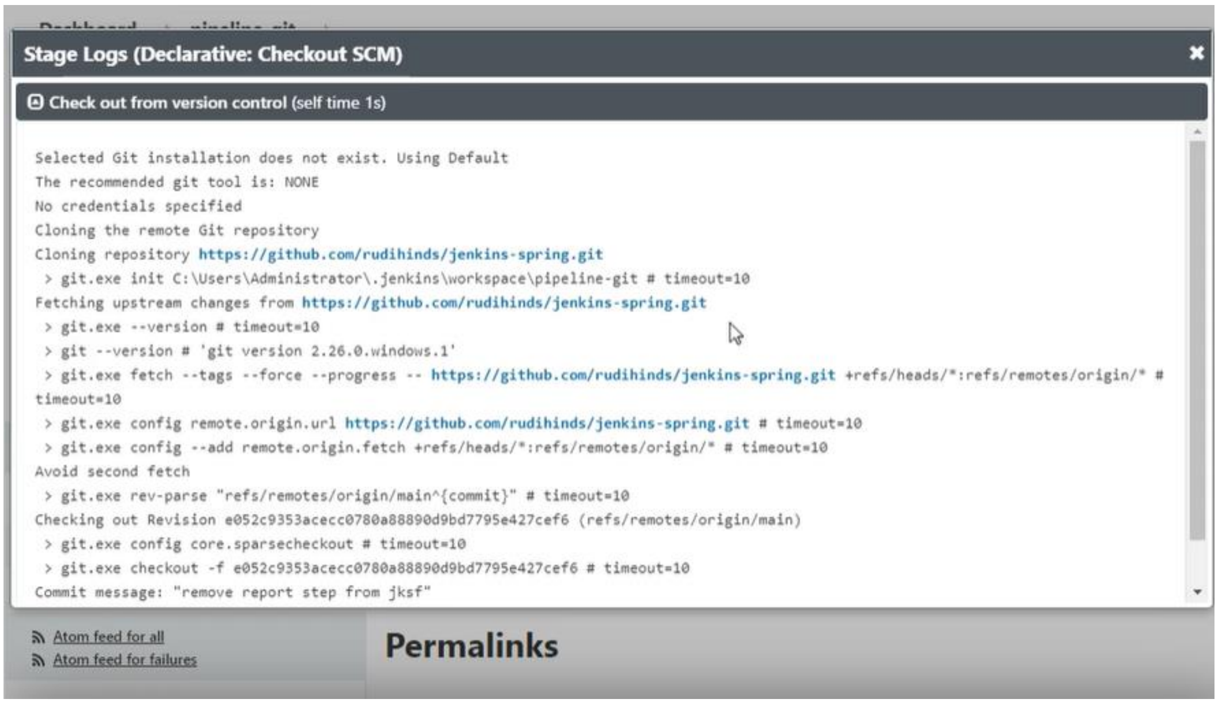
#2 Oct 28, 2021 4:59 PM

#1 Oct 28, 2021 4:59 PM

And, we can click on Logs for seeing the results of each stage:



and we be able to see clone the repository

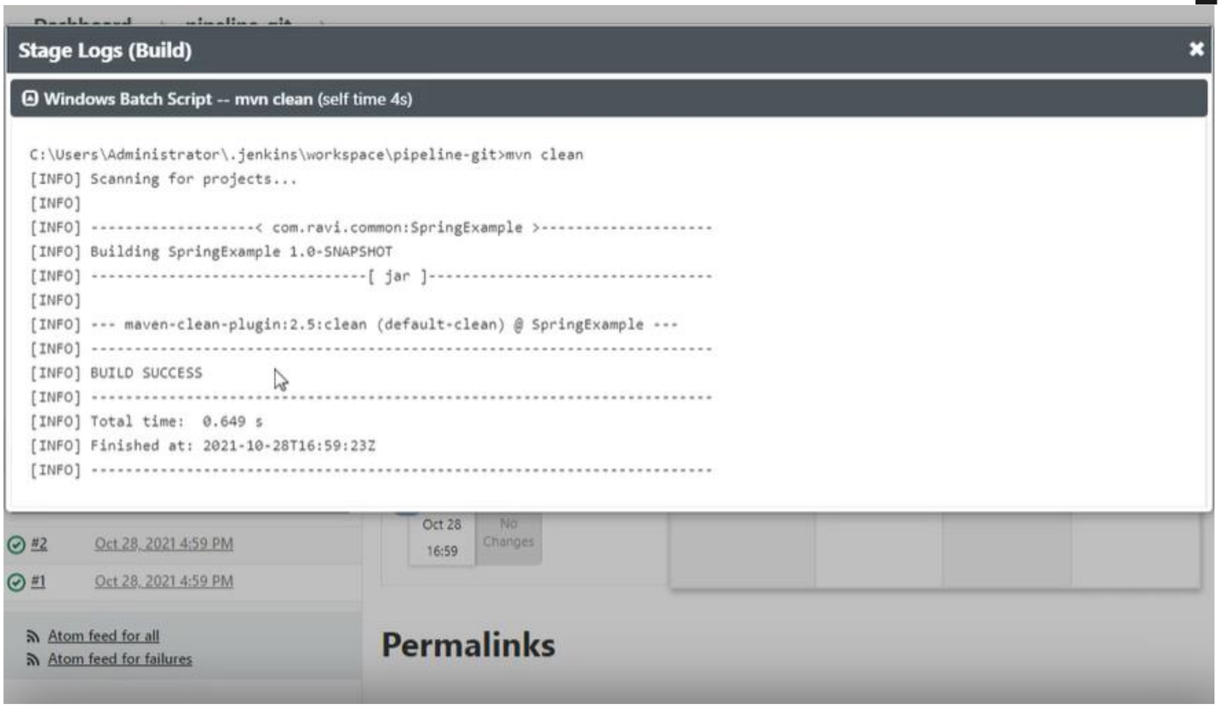


Stage Logs (Declarative: Checkout SCM)

Check out from version control (self time 1s)

```
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/rudihinds/jenkins-spring.git
 > git.exe init C:\Users\Administrator\.jenkins\workspace\pipeline-git # timeout=10
Fetching upstream changes from https://github.com/rudihinds/jenkins-spring.git
 > git.exe --version # timeout=10
 > git --version # 'git version 2.26.0.windows.1'
 > git.exe fetch --tags --force --progress -- https://github.com/rudihinds/jenkins-spring.git +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git.exe config remote.origin.url https://github.com/rudihinds/jenkins-spring.git # timeout=10
 > git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision e052c9353acecc0780a88890d9bd7795e427cef6 (refs/remotes/origin/main)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f e052c9353acecc0780a88890d9bd7795e427cef6 # timeout=10
Commit message: "remove report step from jksf"
```

Same thing for the build stage:



Stage Logs (Build)

Windows Batch Script -- mvn clean (self time 4s)

```
C:\Users\Administrator\.jenkins\workspace\pipeline-git>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< com.ravi.common:SpringExample >-------------------
[INFO] Building SpringExample 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ SpringExample ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.649 s
[INFO] Finished at: 2021-10-28T16:59:23Z
[INFO] ------------------------------------------------------------------------
```

This is the type of automation that we’re going to be waiting in our life projects.